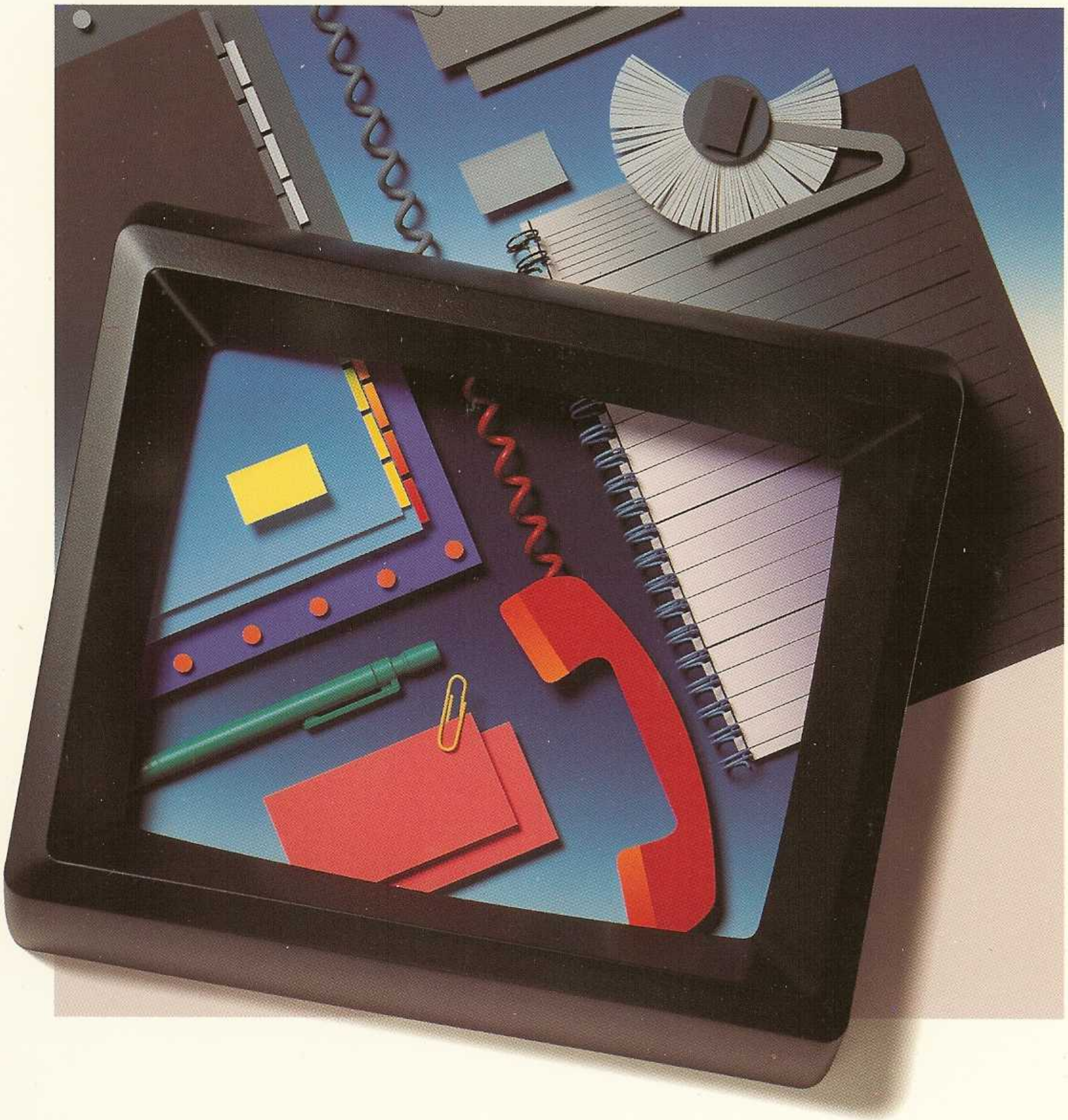
Lotus Agenda

Working with Macros



Release 2.0

Lotus Agenda
Working with Macros

Release 2.0

Contents

Introduction

This book provides information about Lotus Agenda® macros and macro commands. Macros automate and simplify working in Agenda®. You can also use macros to develop customized Agenda applications.

How this Book Is Organized

This book is organized into five chapters that provide the following information:

- **Chapter 1, "Using Macros,"** provides the procedures for working with macros, including creating, running, and editing macros. You don't have to be very familiar with macros to run them or to begin using them.

  Read this chapter if you want to learn basic information about using macros in Agenda.

- **Chapter 2, "About Macro Commands,"** introduces the Agenda macro commands, including information about macro command syntax, arguments, and variables.

  Read this chapter if you plan to create macros using macro commands.

- **Chapter 3, "Macro Command Descriptions,"** provides detailed descriptions and, in many cases, examples of each macro command and special variable.

  Read this chapter to learn about the macro commands offered in Agenda, or as a reference when creating macros that include macro commands.

- **Chapter 4, "Tips and Techniques,"** provides tips and trouble-shooting suggestions that can help you when you create macros.

  Read this chapter before you begin to create macros or if you encounter problems when running macros that you've created.

- **Chapter 5, "Sample Macros,"** provides several sample macros that use many of the macro commands described in this book.

  Read this chapter to get ideas about macros that you can create in Agenda. You can also use — or modify — these sample macros for your own use.

Before You Begin

Before you begin working with macros, you should be familiar with Agenda and its basic features. To use macros for developing customized Agenda applications, you should be familiar with Agenda and how it works.

Typographic Conventions

Working with Macros uses the following typographical conventions:

The names of Agenda function keys, accelerator keys, and special keys appear in small capital letters. Function keys are identified by the key number, followed by the Agenda name.

Example
F6 (PROPS)

Information that you type appears in a different typeface.

Example
```
Call Helen about this week's meeting
```

Commands, settings, and choices appear in boldface type.

Example
Press F10 (MENU) and select **File Attach**.

Chapter 1
Using Macros

A macro is a series of Agenda® keystrokes and/or macro commands that you create to perform an Agenda task. When you run the macro, Agenda reads through the keystrokes and commands in the macro and performs them automatically.

Macros automate and simplify working in Agenda. They reduce the time that users would otherwise spend performing long and repetitive tasks; macros also streamline complex procedures. For example, you can create a macro that formats and prints a report. Macros can also help users who are unfamiliar with Agenda work with specific Agenda applications. For example, you can use macros to develop an Agenda application with customized menus and function keys.

You can include macro commands in macros. For information about these commands and how to use them, see Chapters 2 and 3.

Note You cannot create macros that work while using **Print Setup** or previewing a document. You can, however, create macros (including those created using Learn mode) that take the user to **Print Setup** or **Print Preview**, and then continue executing once the user leaves **Print Setup** or **Print Preview**.

In addition, you can't use macros directly from an earlier release of Agenda in Agenda 2.0 if those macros contain menu commands, function keys, or other keystrokes from the earlier release. You can, however, edit these macros to work in Agenda 2.0. (See "Editing Macros" later in this chapter.)

In this Chapter

This chapter describes

- What a macro is
- How to create and use macros
- How to display or not display hidden macros
- How to import a macro in a text file into an Agenda file
- How to export a macro to a text file
- How to attach macros to keys
- How to store and work with macros in external macro files

Note Macros are often written for other people. In this chapter, the term "you" refers to the person writing the macro. The term "user" refers to the person running the macro.

About Macros

There are two ways to create macros. You can create:

- Simple keystroke macros using Learn mode
- Macros that include macro commands

With simple macros, you record keystrokes while you perform a task. The stored keystrokes can type characters (such as letters and punctuation marks) or perform an Agenda task (such as those performed by menu commands, and function, accelerator, arrow, and special keys). When a user plays back a simple macro, Agenda performs the keystrokes contained in that macro. For example, you can create a simple macro that formats and prints your weekly status report.

Macros can include macro commands as well as keystrokes. Macros that include macro commands provide a powerful programming language tool that can perform long and complicated tasks. Macro commands perform complex operations such as changing the appearance of a screen, displaying messages, looping, calling other macros, or prompting for user input. For detailed information about macro commands, see Chapters 2 and 3.

Each Agenda file can contain both simple macros and those that include macro commands. Macros are stored in the file in which you create them (unless you store a macro in an external macro file). Each Agenda file can contain up to 100 macros. (For information on using external macro files, see "About Macro Files" later in this chapter.)

The keystrokes and commands in a macro cannot exceed 2,000 characters. If a macro exceeds this limit, Agenda tells you while the macro is compiling and the macro aborts. The macro text (the macro name, the text from the keystrokes, the commands, and any comments) cannot exceed 10,000 characters. There is no size limit to a macro stored in an external macro file, but if it exceeds 10,000 characters, it cannot be modified. (See "About Macro Files" later in this chapter.)

Parts of a Macro

Agenda macros have two parts: the macro name and the macro instructions.

- The macro name is a word or phrase of up to 35 characters. You name a macro when you create it; the name should describe what the macro does. The name must be on the first line of the macro, enclosed in braces ({ }).
- The macro instructions are the keystrokes and/or macro commands that Agenda plays back when the user runs the macro.

  A keystroke can consist of a single character, such as a W or dollar sign ($), or a key name enclosed in braces ({ }), such as {LEFT}.

Creating Macros

You can create both simple macros and those that include macro commands in views, notes, and the category manager.

The process for creating a macro involves the following steps:

1. Planning the macro (for example, identifying the steps of the Agenda task that you are automating)
2. Naming the macro in the macro manager (giving the macro a descriptive name)
3. Recording the keystrokes in Learn mode and/or entering the macro commands

4. Documenting the macro with comments (describing the purpose of the macro and its commands)
5. Running the macro to see if it works correctly
6. Fixing any problems in the macro, if necessary, using the {DEBUGON} command to step through the macro to find the problem (see Chapter 3)
7. Specifying where the macro can be run from and whether the macro is protected by completing the settings in the Macro Properties box

When you create a macro, you indicate whether you want users to run the macro in views, notes, the category manager, and/or everywhere else in the file (for example, from boxes with settings). For example, if you create a macro that formats a column, it makes sense to allow users to run this macro only in views — not in notes, the category manager, or anywhere else in Agenda.

To specify where users can run a macro, select **Yes** for the appropriate settings under **Macro can be run from** in the Macro Properties box. See "Macro Properties Settings" later in this chapter.

You can also create a macro that executes each time a file is opened. To do this, name the macro `autoexec` when you create it. You can only have one autoexec macro per Agenda file.

The following sections describe how to create simple macros and macros that contain macro commands. Procedures for running and working with macros are provided later in this chapter.

Note If you press a key that has an attached macro while you are recording a macro in Learn mode, Agenda inserts a {CALL} command into the macro you are recording. This calls the macro that is *attached* to that key. (See Chapter 3.)

Creating Simple Macros

The easiest way to create a macro is to use Learn mode. In Learn mode Agenda records your keystrokes while you perform a task. You can begin recording a macro in a view, a note, or the category manager.

Keystrokes can include anything that you can type, press, or select using your keyboard: menu commands; function, accelerator, arrow, and special keys; text (letters and punctuation marks); as well as keys that have attached macros.

To create simple macros using Learn mode:

1. Place the highlight in a view, note, or the category manager (where you want to begin recording keystrokes for the macro).

2. Press ALT-F3 (MACRO).

   Agenda displays the macro manager (Figure 1-1).

3. Type a name of up to 35 characters and press ENTER.

4. Press F7 (LEARN).

   Agenda removes the macro manager and returns you to where you were in Agenda. When Agenda is in Learn mode, it displays the LEARN indicator (LEARN) in the upper right corner of the screen and records a keystroke each time you press a key. When you press a key, Agenda also beeps to remind you that keystrokes are being recorded.

5. Press each keystroke that you want to record. When you finish, press ALT-F3 (MACRO) again.

   Agenda stops recording keystrokes and displays a message telling you that Learn mode has been turned off. The LEARN indicator also disappears from the upper right corner of the screen.

6. Press any key to remove the message.

Notes When you create a macro, Agenda records only the keystrokes, not whether you are typing them in a view, note, or the category manager. For example, if you are recording a macro in the category manager and you press F7 (PROMOTE), Agenda records that keystroke as F7. If you run this macro in a view, Agenda interprets the keystroke as F7 (MARK) (the function of F7 in a view).

If you make a mistake while recording a macro, you can stop recording the macro by pressing ALT-F3 (MACRO). Press AFT-F3 (MACRO) again to display the macro manager. You can then make changes to the macro by pressing F2 (EDITMAC) on the macro name in the macro manager, or append or replace the macro using Learn mode again. (See "Editing Macros," "Replacing a Macro," and "Appending to a Macro" later in this chapter.)

Pressing ALT-Z while recording a macro in Learn mode turns Learn mode *off* (because ALT-Z runs the last-run macro).

Creating Macros that Include Macro Commands

You can create macros that include macro commands in a view, a note, or the category manager in either of two ways. You can:

- Edit an existing macro you already created using Learn mode
- Type all the macro instructions (including the keystrokes and macro commands) in the Macro edit screen. (For detailed information on macro commands, see Chapter 3.)

To create a macro that includes macro commands:

1. Press ALT-F3 (MACRO).

   Agenda displays the macro manager (Figure 1-1).

2. Do one of the following:

   - To edit an existing macro that was created in Learn mode, highlight the macro name.
   - To create a new macro, type a name of up to 35 characters. Then press ENTER.

3. Press F2 (EDITMAC) .

   Agenda displays the Macro edit screen.

4. Type any key names and/or commands that you want to include in this macro. You can type uppercase and/or lowercase letters. Make sure you enclose key names in braces. (For example, type `{F10}` to specify the F10 function key.) See "Representing Keys in Macros" later in this chapter and Chapter 3 for information about macro commands.

Note The set of characters in braces on the first line of the Macro edit screen is the macro name. While you can edit this name, do *not* delete the macro name or enter any macro commands or keystrokes before it. If you enter any characters before the macro name that are *not* enclosed in braces, Agenda displays <Nameless Macro> instead of the macro name in both the macro manager and the Macro Properties box.

5. Press F10 (MENU) and select **Return**.

Agenda saves the macro and returns to the macro manager.

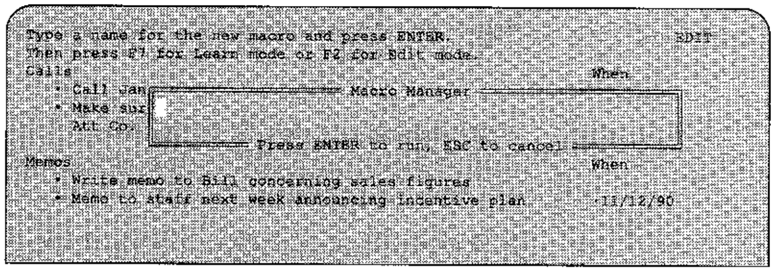
Notes If you type a key that is attached to another macro, the *original* Agenda function of that key is called in the current macro. To include a macro that's attached to a key in the current macro, use {CALL}. (See Chapter 3.)

In a macro, leading white space in a line is ignored but trailing white space is significant. This means that any spaces at the end of a line in a macro are typed as literal spaces.

To see how many spaces are at the end of each line in a macro, you can display carriage returns. To display carriage returns, select **Yes** for **Display carriage returns** using the **Utilities Customize** command.

The Macro Manager

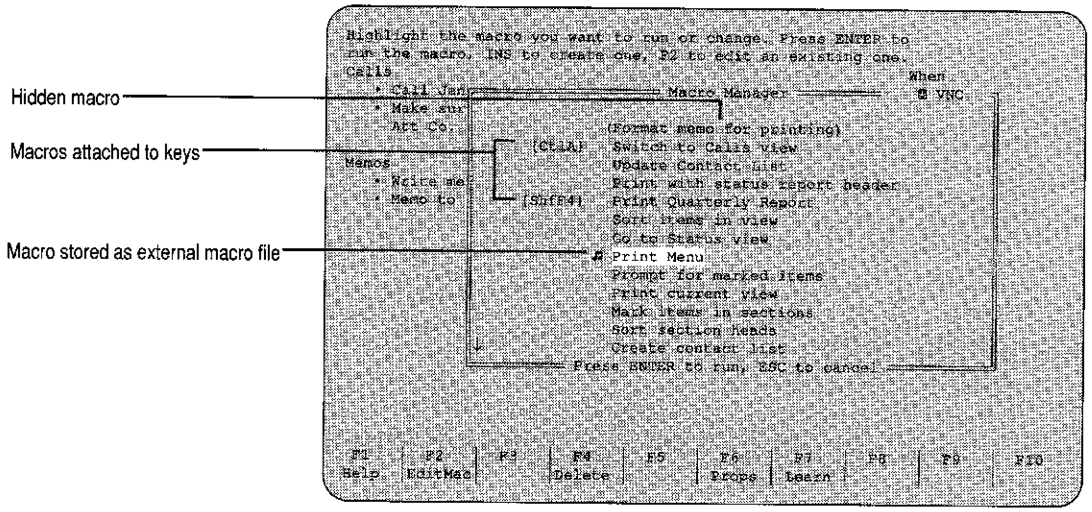
When you press ALT-F3 (MACRO) in a new Agenda file, the macro manager is empty (Figure 1-1).



```
Type a name for the new macro and press ENTER.                    EDIT
Then press F7 for Learn mode or F2 for Edit mode.
Calls                                                        When
   • Call Jan ============ Macro Manager ===================
   • Make sur|                                             |
     Att Co. |                                             |
             ======= Press ENTER to run, ESC to cancel =====
Memos                                                        When
   • Write memo to Bill concerning sales figures
   • Memo to staff next week announcing incentive plan      11/12/90
```

Figure 1-1 *The macro manager*

When you press ALT-F3 (MACRO) in an Agenda file that already contains macros, the macro manager lists the names of these macros (Figure 1-2). When you create a macro, the macro is added to the list of macros in the macro manager.



```
Highlight the macro you want to run or change. Press ENTER to
run the macro, INS to create one, F2 to edit an existing one.
Calls                                                        When
  • Call Jan═══════════════ Macro Manager ══════════      ▣ VNC
  • Make sur
    Att Co.                 (Format memo for printing)
                    [CtlA]   Switch to Calls view
Memos                        Update Contact List
  • Write me                 Print with status report header
  • Memo to        [ShfF4]   Print Quarterly Report
                             Sort items in view
                             Go to Status view
                           ♬ Print Menu
                             Prompt for marked items
                             Print current view
                             Mark items in sections
                             Sort section heads
         ↓                   Create contact list
         ══════ Press ENTER to run, ESC to cancel ══════

 F1  | F2      | F3 | F4     | F5 | F6    | F7    | F8 | F9 | F10
Help |EditMac  |    |Delete  |    | Props | Learn |    |    |
```

Figure 1-2 *The macro manager with a list of macros in a view*

The function key map at the bottom of the screen shows the function keys that you can use in the macro manager. Press ALT to display the alternate function keys that you can use in this box.

When you highlight a macro name in the macro manager, Agenda may display indicators in the upper right corner of the box. These indicators provide information about the highlighted macro including:

- Where the macro can be run from (views, notes, and so forth)
- Whether the macro is protected (see Appendix G in the *User's Guide*)
- Whether the macro can override protection (see Appendix G in the *User's Guide*)

The following table describes the indicators that may appear in the upper right corner of the macro manager when you highlight a macro.

Indicator	*Meaning*
¶	Macro is protected (see Appendix G in the *User's Guide*)
☺	Macro can override protection (see Appendix G in the *User's Guide*)
V	Macro can be run from views
C	Macro can be run from the category manager
N	Macro can be run from notes
E	Macro can be run from everywhere else (for example, from boxes with settings)

Under the following conditions, the macro manager also displays the following information about the macros in the current file:

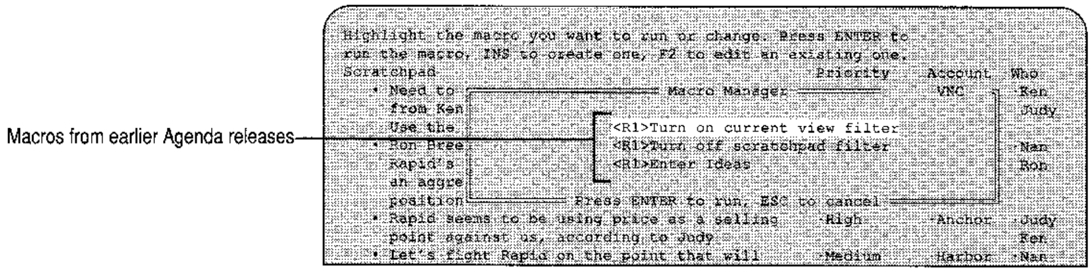
- If you press ALT-F7 (DISPLAY), the macro manager displays the names of any hidden macros in parentheses. For example, Figure 1-2 shows the macro manager in a view. In this example, the "Format memo for printing" macro is hidden because it can't be run in a view. (See "Hidden Macros" later in this chapter.)

- If any macros are attached to keys, the names of these keys appear in braces beside the macros to which they are attached. For example, in Figure 1-2, the "Print Quarterly Report" macro is attached to SHIFT-F4 and the "Switch to Calls view" macro is attached to CTRL-A.

  In addition, if you attach a macro to a function key, SHIFT function key, ALT function key, or CTRL function key, the first seven characters of the macro name appear in the function key map at the bottom of the screen. (See "Attaching Macros to Keys" later in this chapter.)

- If any macros are stored in external macro files, double note symbols (♫) appear beside those macros. For example, in Figure 1-2, the "Print Menu" macro is stored in an external macro file. (See "About Macro Files" later in this chapter.)

- If the current file was created in an earlier release of Agenda, the <R1> symbol appears beside the names of macros in that file after it is opened in Agenda 2.0. For example, in Figure 1-3, the macros in the macro manager are all from an earlier release of Agenda.



```
Highlight the macro you want to run or change. Press ENTER to
run the macro, INS to create one, F2 to edit an existing one.
Scratchpad                                 Priority    Account  Who
  • Need to ══════════ Macro Manager ════════      VNC    ╗ ·Ken
    from Ken                                              ║  Judy
    Use the       <R1>Turn on current view filter         ║
  • Ron Bree      <R1>Turn off scratchpad filter          ║ ·Nan
    Rapid's       <R1>Enter Ideas                         ║  Ron
    an aggre                                              ║
    position ════ Press ENTER to run, ESC to cancel ══════╝
  • Rapid seems to be using price as a selling  ·High    ·Anchor  ·Judy
    point against us, according to Judy                            Ken
  • Let's fight Rapid on the point that will    ·Medium  ·Harbor  ·Nan
```

Figure 1-3 *The macro manager with macros from an earlier Agenda release*

Note You can't use macros directly from an earlier release of Agenda in Agenda 2.0 if the macros contain menu commands, function keys, or other keystrokes from the earlier release. You can, however, edit these macros or keystroke them again in Learn mode so that they work in Agenda 2.0. (See "Editing Macros" later in this chapter.)

You specify most of the information shown in the macro manager for a highlighted macro in the Macro Properties box for that macro. (See "The Macro Properties Box" later in this chapter.)

Sorting the List of Macros

You can sort the list of macros in the macro manager alphabetically. You may want to do this if you have many macros and want to quickly scan the list.

To sort the list of macros alphabetically:

1. Press ALT-F3 (MACRO).

   Agenda displays the macro manager.

2. Press ALT-F5 (SORT).

Agenda sorts the list of macros alphabetically.

Rearranging the List of Macros

You can rearrange the position of macro names in your list of macros. For example, you might want to change the order in which Agenda displays macros to appear in order of the tasks they perform.

To rearrange the list of macros:

1. Press ALT-F3 (MACRO).

   Agenda displays the macro manager.

2. Highlight the macro you want to move and press ALT-F10 (MOVE).

   Agenda displays a symbol (») beside the macro you select.

3. Move the symbol to the macro name above or below which you want to place the macro in the list.

4. Do one of the following:

   - To insert the macro name above the symbol, press CTRL-ENTER.
   - To insert the macro name below the symbol, press ENTER.

Agenda rearranges the list of macros and removes the symbol.

Deleting Macros

You can delete a macro if you no longer want it.

To delete a macro:

1. Press ALT-F3 (MACRO).

   Agenda displays the macro manager.

2. Highlight the name of the macro you want to delete and press F4 (DELETE) or DEL.

   Agenda asks if you want to delete the current macro. Select **Yes**.

Agenda deletes the macro from the macro manager in this file.

Note You cannot delete a macro if it is protected. (See Appendix G in the *User's Guide*.)

Hidden Macros

You can control where a macro can be run from. For example, you might specify that users can run a macro from notes, but not from views or the category manager. In this case, the name of the macro *does not* appear (or is displayed in parentheses) in the macro manager in views or the category manager. This macro is a hidden macro in views and category manager. The name of this macro, however, *does* appear in the macro manager in notes.

You specify where users can run a macro by completing the **Macro can be run from** setting in the Macro Properties box. (See "Macro Properties Settings" later in this chapter.)

Hidden macros do *not* appear in the macro manager when it is first displayed. However, once you set Agenda to display hidden macros, Agenda will continue to display them enclosed in parentheses until you set Agenda to *not* display hidden macros.

Users can switch between displaying or not displaying the names of hidden macros in the macro manager.

To display or not display hidden macros in the macro manager:

1. Press ALT-F3 (MACRO).

   Agenda displays the macro manager.

2. Press ALT-F7 (DISPLAY).

Agenda displays or does not display any hidden macros in the macro manager.

In Figure 1-2, when the macro manager displays in a view, users can press ALT-F7 (DISPLAY) to display the name of the hidden macro, "Format memo for printing." Agenda displays the names of hidden macros in parentheses. If a file does not include any hidden macros, pressing ALT-F7 (DISPLAY) has no effect. If you highlight a hidden macro and press ENTER to run it, Agenda tells you that the macro can't be run in the current mode.

Note You cannot switch between displaying and not displaying the names of hidden macros if the file is sealed. (See Appendix G in the *User's Guide*.)

Representing Keys in Macros

Text characters (a, b, c, and so forth) are recorded in a macro as the characters themselves. Function keys, accelerator keys, and special keys are represented by abbreviations of key names, enclosed in braces.

If you add function, accelerator, and/or special keys to a macro when editing it, you must include the braces as well as the key name abbreviation.

To include a function, accelerator, or special key in a macro, do one of the following:

- Press ALT = (hold down ALT, press EQUAL, then release both keys), then press a key listed in the following table.

  Agenda automatically encloses the key name in braces.

- Type the abbreviation for the key name as identified in the following table and enclose it in braces, for example, `{CtlW}`.

  You can type the abbreviation in uppercase and/or lowercase. If you misspell a key name or forget to include the braces, the macro types the characters instead of performing the keystroke.

The following table shows how Agenda records keys in macros. For example, if you press ALT and = to record a key and then press ALT and F2, Agenda records this key as {AltF2}.

Key	*Agenda Key*	*Alt*	*Ctrl*	*Shf*
F1 – F10	{F*n*} where *n* is the function key number, such as {F2}	{AltF*n*} where *n* is the function key number, such as {AltF5}	{CtlF*n*} where *n* is the function key number, such as {CtlF4}	{ShfF*n*}, where *n* is the function key number, such as {ShfF5}
A – Z	The lowercase letter, such as a	{Alt*n*} where *n* is the key letter, such as AltA	{Ctl*n*} where *n* is the key letter, such as CtlB	The uppercase letter, such as A
0 – 9	The number, such as 8	*		The corresponding punctuation character, such as &
TAB	{TAB}			{ShfTAB}
BACKSPACE	{BS}		⌂	{BS}
ENTER	{ENTER}		{CtlEnter}	{ShfEnter}
ESCAPE	{ESC}		{ESC}	{ESC}
HOME	{HOME}		{CtlHome}	{HOME}
END	{END}		{CtlEnd}	{END}
INSERT	{INS}			{INS}
DELETE	{DEL}			{DEL}

continued

Key	*Agenda Key*	*Alt*	*Ctrl*	*Shf*
↑	{UP}			{UP}
↓	{DOWN}			{DOWN}
←	{LEFT}		{CtlLeft}	{LEFT}
→	{RIGHT}		{CtlRight}	{RIGHT}
PGDN	{PGDN}		{CtlPGDN}	{PGDN}
PGUP	{PGUP}		{CtlPGUP}	{PGUP}
PRINTSCRN			{CtlPrtSc}	
BREAK			{CtrlBrk}	
GREY	{GREY-}			{ALT-}
GREY+	{GREY+}			{ALT=}
[			{ESC}	
H			{BS}	
I			{TAB}	
J			{CtlEnter}	
M			{ENTER}	

*ALT-0 through ALT-9 represent international characters. See Appendix D in the *User's Guide.* In addition, on many systems, ALT-*n* (where *n* is a digit on the numeric keypad) is interpreted as a control character.

Notes You cannot represent the ALT-F1 (COMPOSE) key in a macro. Simply type the compose character instead. (See Appendix D in the *User's Guide.*) Also, you cannot represent the ALT-Z key in a macro.

If you attach a macro to a key, you should keep in mind that the CTRL-[, CTRL-H, CTRL-I, CTRL-J, and CTRL-M keys are recorded as other keys (as shown in the preceding table). For example, if you attach a macro to CTRL-M, each time the user presses ENTER, Agenda runs the attached macro. See "Attaching Macros to Keys" later in this chapter.

SPACE BAR does not have a special key name representation. It is simply recorded as a space character.

The Macro Properties Box

You use the Macro Properties box to specify and display information about a macro. From this box you can:

- Display the name of the macro
- Display the contents of and edit the macro
- Attach an external macro file to a macro
- Attach the macro to a key
- Identify where users can run the macro in Agenda
- Indicate whether the macro is protected (see Appendix G in the *User's Guide*)
- Indicate whether the macro can perform protected operations, if you have specified one or more protection settings for this file (see Appendix G in the *User's Guide*)

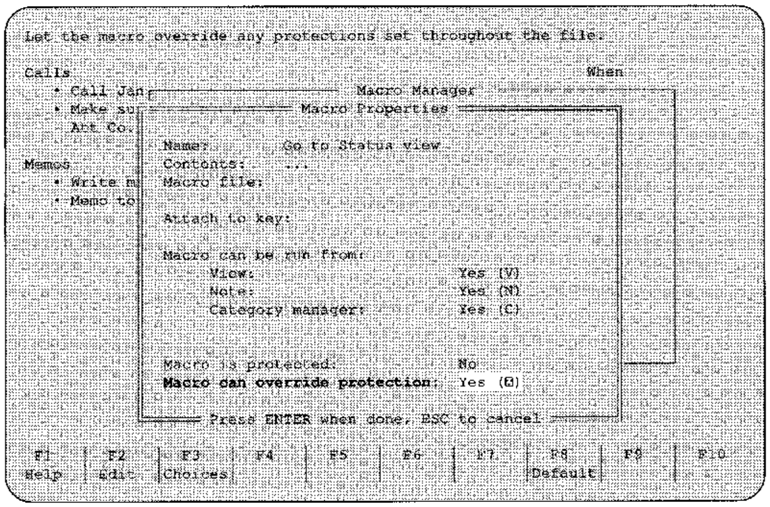
To display the Macro Properties box:

1. Press **ALT-F3 (MACRO)**.

   Agenda displays the macro manager.

2. Highlight the macro for which you want to display the Macro Properties box.
3. Press **F6 (PROPS)**.

Agenda displays the Macro Properties box (Figure 1-4).



```
Let the macro override any protections set throughout the file.

Calls                                                 When
   • Call Jan        Macro Manager
   • Make su         Macro Properties
     Art Co.
               Name:         Go to Status view
Memos          Contents:     ...
   • Write m   Macro file:
   • Memo to
               Attach to key:

               Macro can be run from:
                    View:                      Yes (V)
                    Note:                      Yes (N)
                    Category manager:          Yes (C)


               Macro is protected:             No
               Macro can override protection:  Yes (O)

                    Press ENTER when done, ESC to cancel
```

Figure 1-4 *The Macro Properties box*

Macro Properties Settings

You use the Macro Properties box to specify and display information about a macro, and to enter and edit the contents of the macro.

Name Displays the name of the macro. This name also appears in the macro manager.

You cannot move the highlight to or edit the name of a macro from this setting. (For information on editing a macro name, see "Changing Macro Names" later in this chapter.)

Contents Displays the contents of the macro. (Press **SPACE BAR** to display the Macro edit screen for the macro.) You can edit the macro name from here. (See "Changing Macro Names" later in this chapter.)

This setting appears only if the **Macro is protected** setting is set to **No**.

Macro file Specify the name of the external macro file to which you want to attach the current macro. If you do not specify an extension, Agenda automatically gives the extension .MAC to the file. If the contents of this macro already exist, (in other words, if the macro already contains keystrokes and/or macro commands), Agenda asks if you want to delete the existing macro. (You can also use the **File Attach** command from the Macro edit screen to attach an external macro file to a macro. See "About Macro Files" later in this chapter.)
)

Attach to key Specify the key to which you want to attach the macro. Make sure you enclose the key name in braces. The key name also appears enclosed in braces beside the macro name in the macro manager. (See "Representing Keys in Macros" earlier in this chapter and "Attaching Macros to Keys" later in this chapter.)

Agenda displays the **Everywhere else** setting under **Macro can be run from**.

Macro can be run from Specify where you want to let users run the macro from (**View**, **Note**, **Category manager**, and/or **Everywhere else**). The **Everywhere else** setting appears only when you attach a macro to a key in the **Attach to key** setting.

The **Macro can be run from** settings determine whether a macro is hidden. For example, if you do not let users run a macro in views, this macro is a hidden macro in a view. For more information, see "Hidden Macros" earlier in this chapter.

For each of the **Macro can be run from** settings, select one of the following choices.

Choice	*Result*
No	Prevents users from running this macro in views, notes, the category manager, or from everywhere else in Agenda (for example, from boxes with settings). The macro name does not appear in the macro manager when the macro manager is displayed in that part of Agenda. If **ALT-F7 (DISPLAY)** is pressed, Agenda displays the name of hidden macros in parentheses (unless the file is sealed).
Yes	Lets users run this macro in views, notes, the category manager, or from everywhere else in Agenda (for example, from boxes with settings). Users can run the macro from the macro manager or, if this macro is attached to a key, by pressing the appropriate key. The macro can be run only from anywhere else in Agenda (other than views, notes, or the category manager) if it is attached to a key.

Note **Yes** is the default choice for the **View**, **Note**, and **Category manager** settings. **No** is the default for the **Everywhere else** setting.

Even if you select **No** for any or all of these settings, other macros can still call or run this macro from anywhere in Agenda. You use the macro commands {CALL} and {GOTO) to do this. (See Chapter 3.)

Macro is protected Specify whether you want to let users display the contents of and edit the current macro, or change any of the settings in the Macro Properties box for the current macro. The choices are **Yes** and **No** (default). (See Appendix G in the *User's Guide.*)

If you select **Yes**, Agenda does *not* display the **Contents** setting in the Macro Properties box.

Macro can override protection Specify whether you want the current macro to perform operations that the user is protected against performing. (See Appendix G in the *User's Guide.*) The choices are **Yes** and **No** (default).

For example, suppose you develop an Agenda application in which you want to prevent users from arbitrarily modifying a certain view, while still allowing some changes to be made to this view under macro control.

Select **View Properties** and set **Full protection** for the **View protection** setting. At this point you can write a macro that performs the desired changes (for example, that sorts items, or hides done items). By selecting **Yes** for the **Macro can override protection** setting for this macro, the macro makes the changes to the view, even though this view is protected.

Note Unless this file is sealed, users can change the **Macro can override protection** setting and other protection settings. For more information about sealing files, see Appendix G in the *User's Guide.*

Running Macros

Macros make doing your work easier. To run macros, users don't need to know much about them; they can simply run the macros that already exist in a file.

Agenda provides two procedures for running a macro. A macro can be run from the macro manager or by pressing a key (if a macro is attached to it). Both of these procedures are described in this section.

Macros can be run from views, notes, the category manager and/or from everywhere else in Agenda (for example, from boxes with settings or in Edit mode). You specify where you want to let users run a macro with the **Macro can be run from** settings in the Macro Properties box.

Note The **Everywhere else** setting appears only if you attach a macro to a key in the **Attach to key** setting in the Macro Properties box. Selecting **Yes** for this setting allows users to run macros that are attached to keys from anywhere else in Agenda other than views, notes, and the category manager (for example, from boxes with settings or in Edit mode).

If you develop an application that includes macros that users can run from places other than views, notes, and the category manager, you might want to attach these macros to function, ALT function, SHIFT function, or CTRL function keys. Agenda then displays the names of these macros in the function key map at the bottom of the screen. (Hold down the ALT, SHIFT, and CTRL keys to see the names of macros attached to these function keys.) (See "Attaching Macros to Keys" later in this chapter.)

Note Macros that are hidden (macros that appear in parentheses in the macro manager when you press ALT-F7 (DISPLAY)) cannot be run.

To run a macro from the macro manager:

1. Press ALT-F3 (MACRO).

   Agenda displays the macro manager.

2. Do one of the following:

   - Highlight the name of the macro you want to run and press ENTER.

   - Type the name of the macro you want to run and press ENTER *twice*.

Agenda tells you that the macro is compiling and then executes the keystrokes and commands contained in that macro.

To run a macro that's attached to a key:

- Press the key to which the macro is attached.

To run the macro you last ran during the current Agenda session:

- Press ALT-Z.

Editing Macros

You can edit a macro name as well as the keystrokes and/or macro commands that the macro contains to correct mistakes or to change what the macro does. You can also edit a macro if it contains menu commands, function keys, or other keystrokes from an earlier release so that you can use it in Agenda 2.0. (In the macro manager, Agenda displays <R1> beside the names of macros from earlier releases of Agenda.)

When you edit a macro, you can use many of the editing features offered in Agenda. (See Chapter 14.) You *cannot*, however, insert markers into macros. Also, press F1 (HELP) for comprehensive lists of the keys that you can use to edit macros.

Note If a macro from an earlier release was created in Learn mode, it may be easier to simply record the new keystrokes for this macro in Agenda 2.0 using F7 (LEARN).

Changing Macro Names

Once you create a macro, you might want to change its name to better describe its contents.

To change a macro name:

1. Press ALT-F3 (MACRO).

   Agenda displays the macro manager.

2. Highlight the macro whose name you want to change and press F2 (EDITMAC).

   Agenda displays the Macro edit screen for the macro you selected. The name of the macro appears in braces on the first line of the macro.

3. Make any changes to the macro name. Make sure that the name does not exceed 35 characters and that it is enclosed in braces ({ }).

4. Press F10 (MENU) and select **Return** to return to the macro manager.

Agenda displays the macro in the macro manager with the new name.

Note Do *not* delete the braces or enter any macro instructions before the macro name in the Macro edit screen. If you enter any characters before the macro name that are *not* enclosed in braces, <Nameless Macro> appears instead of the macro name in both the macro manager and the Macro Properties box.

Editing the Contents of Macros

You can edit the contents of a macro to correct mistakes or to change what the macro does. You can also edit the contents of a macro if it contains menu commands, function keys, or keystrokes from an earlier release of Agenda so that you can use it in Agenda 2.0.

Note Agenda 2.0 supports all macro commands from earlier releases of Agenda. Note that the {INPUT} from earlier releases of Agenda is {INPUTTEXT} in Agenda 2.0. (See Chapter 3.)

To edit the contents of a macro:

1. Press **ALT-F3 (MACRO).**

   Agenda displays the macro manager.

2. Highlight the name of the macro you want to edit and press **F2 (EDITMAC).**

   Agenda displays the Macro edit screen for the macro you selected (Figure 1-5). The set of characters in braces on the first line is the macro name. Subsequent characters are the macro instructions.

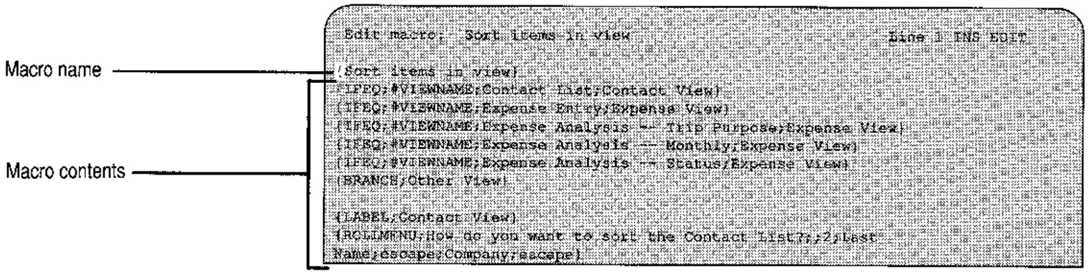
3. Make any changes to the contents of this macro. (For general information about editing, see Chapter 14 in the *User's Guide.*)

   To discard any changes that you made to the macro before saving it, press **ESC**. Agenda asks if you want to discard the changes. Select **Yes** and press **ENTER**.

4. When you finish editing the macro, press **F10 (MENU)** and select **RETURN.**

Agenda redisplays the macro manager.

Note If you edited the contents of a macro from an earlier release of Agenda so that it works in Agenda 2.0, you can delete the <R1> beside the macro name by editing the macro name.



```
Edit macro:  Sort items in view                          Line 1 INS EDIT

{Sort items in view}
{IFEQ;#VIEWNAME;Contact List;Contact View}
{IFEQ;#VIEWNAME;Expense Entry;Expense View}
{IFEQ;#VIEWNAME;Expense Analysis -- Trip Purpose;Expense View}
{IFEQ;#VIEWNAME;Expense Analysis -- Monthly;Expense View}
{IFEQ;#VIEWNAME;Expense Analysis -- Status;Expense View}
{BRANCH;Other View}

{LABEL;Contact View}
{RCLIMENU;How do you want to sort the Contact List?;;2;Last
Name;escape;Company;escape}
```

Figure 1-5 *Macro edit screen*

You can also edit a macro from the Macro Properties box:

1. Press ALT-F3 (MACRO).

   Agenda displays the macro manager.

2. Highlight the name of the macro you want to edit and press F6 (PROPS).

   Agenda displays the Macro Properties box.

3. Highlight **Contents** and press SPACE BAR.

   Agenda displays the Macro edit screen for the macro you selected (Figure 1-5).

4. Follow Steps 3 and 4 in the previous procedure to edit the macro.

Macro Edit Commands

You can use the menu commands that display in the Macro edit screen to work with the current macro.

To display the Macro edit screen menu:

1. Press ALT-F3 (MACRO).

   Agenda displays the macro manager.

2. Highlight the name of the macro you want to work with and press F2 (EDITMAC).

   Agenda displays the Macro edit screen for the macro you selected.

3. Press F10 (MENU).

Agenda displays the macro edit menu commands. The following table describes each command.

Command	*Function*
File	Displays the **Attach**, **Detach**, and **Erase** commands. **Attach** attaches an external macro file to the current macro. (You can also complete the **Macro file** setting in the Macro Properties box to attach an external macro file to the current macro.) **Detach** removes the attachment of the macro from the external macro file. **Erase** removes the attachment of the macro from the external macro file *and* erases the file from your disk. (See "About Macro Files" later in this chapter.)
Print	Displays the **Final**, **Preview**, **Layout**, **Setup**, and **Named** print commands so that you can print the contents of the current macro. If you select one of these commands, Agenda first saves any changes made to the macro. (See Chapter 17 in the *User's Guide*)
Import	Imports the contents of a text file into a macro. (See "Importing Text Files into Macros" later in this chapter.)
Export	Exports the contents of a macro to a text file. (See "Exporting Macros to Text Files" later in this chapter.)
Clear	Clears the contents of the current macro.
Return	Returns to the macro manager or Macro Properties box, saving any changes made to the current macro.
Quit	Ends the current Agenda session.

Replacing a Macro

If you make errors, particularly while recording a macro using Learn mode, it may be easier to re-enter keystrokes and replace the macro than to edit the macro extensively. You might also want to replace a macro that was created in Learn mode in an earlier release of Agenda.

To replace a macro:

1. Press ALT-F3 (MACRO).

   Agenda displays the macro manager.

2. Highlight the macro you want to replace and press F7 (LEARN).

Agenda tells you that the macro already exists and asks if you want to replace the existing macro or append to it. Select **Replace** and press ENTER.

Agenda removes the macro manager and returns you to where you were in Agenda. When Agenda is in Learn mode, it displays the LEARN indicator (LEARN) in the upper right corner of the screen and records a keystroke each time you press a key. When you press a key, Agenda also beeps to remind you that keystrokes are being recorded.

3. Press the keystrokes you want to record. When you are finished, press ALT-F3 (MACRO) again.

   Agenda stops recording keystrokes and displays a message that Learn mode has been turned off. The LEARN indicator also disappears from the upper right corner of the screen.

4. Press any key to remove the message.

Note You cannot replace a protected macro. (See Appendix G in the *User's Guide.*)

Appending to a Macro

You can add keystrokes to the end of an existing macro. You might do this if you want to create a long macro one segment at a time. You can enter and test a set of keystrokes and then extend the macro by appending additional keystrokes.

The procedure for appending keystrokes to an existing macro is the same as for replacing a macro except that you select **Append** in Step 2 in the preceding procedure ("Replacing a Macro").

Transferring All Macros to a New File

When you create a new Agenda file, it contains no macros. You can, however, create a *new* file that contains macros from another file.

The **File Transfer Template** command creates a new file and copies the structure of one file (its category hierarchy, views, conditions and actions, file properties, as well as its macros) to a new file.

Note The new file does not contain items or their assignments from the original file, although you can use the **File Transfer Import** command to import items.

To transfer all macros from one file to another file:

1. Open the file whose structure you want to copy.
2. Press F10 (MENU) and select **File Transfer Template**.

   Agenda displays the Select File box.
3. Press INS and type the name of the new file to which you want to copy the structure of the current file. Then press ENTER.

Caution Do *not* type the name of an existing file, unless you want to erase that file and reuse the name.

Agenda creates the file that you specified in Step 3 and copies the structure of the current file to it.

Importing Text Files into Macros

You can import a macro from a text file into a macro in another Agenda file. You might do this if you want to bring a macro from another Agenda file into the current Agenda file. When you import the contents of a text file, Agenda copies the contents of that text file into the current macro.

To import a text file into a macro in an Agenda file:

1. Open the file into which you want to import the macro.
2. Press ALT-F3 (MACRO) to display the macro manager and type a name (up to 35 characters) of a new macro. Press ENTER and then press F2 (EDIT MAC) to display the Macro edit screen for that macro.
3. Press END to go to the end of the macro name. Then press F10 (MENU) and select **Import**.

   Agenda displays the File box.

4. Highlight **File name** and do one of the following:
   - Type the name of the text file whose contents you want to import (including the path, if necessary) and press ENTER.
   - Press F3 (CHOICES) and highlight the file whose contents you want to import. Then press ENTER.
5. Highlight **Strip carriage returns**. Select **No** (default) to keep single carriage returns in the macro when you import it, thus preserving the original formatting of the contents of the macro. Select **Yes** to remove single carriage returns from the macro when you import it, thus rewrapping the contents of the macro to 79-character lines.

Agenda displays the imported contents of the text file in the current macro. This macro is now part of the current Agenda file.

If you import a macro from a text file into a macro that already contains commands and/or keystrokes, Agenda inserts the contents of the file in the current macro after the cursor.

Caution When you import a macro to a second file, the macro may have two names: the name that it had in the original macro and the name that you gave the macro in the second file (Step 1). Be sure to delete one of these names when you complete this procedure. (Make sure that the macro name is on the first line of the Macro edit screen and is enclosed in braces ({ }).

Exporting Macros to Text Files

You can export the contents of a macro to a text file. You might do this to use the macro in another Agenda file. In this case, import the text file that contains the macro into the other Agenda file.

To export the contents of a macro to a text file:

1. Do one of the following:
   - In the macro manager, highlight the macro you want to export and press F2 (EDITMAC).
   - In the Macro Properties box for the macro you want to export, highlight **Contents** and press SPACE BAR.

Agenda displays the Macro edit screen.

2. From the Macro edit screen, press F10 (MENU) and select **Export**.

   Agenda displays the File box.

3. Do one of the following:

   - Type the name of a file to which you want to export the contents of the current macro (including the path, if necessary) and press ENTER *twice.*

   - Press F3 (CHOICES) to display a list of files. Highlight the name of the file to which you want to export the contents of the current macro and press ENTER *twice.*

   If the file already exists, Agenda asks whether you want to append or replace the contents of the file. Select **Append** to add the exported contents of the macro at the end of the text already in that file. Select **Replace** to delete the text in that file before placing the contents of the macro in it. Then press ENTER.

Note To specify a different file name, press ESC and repeat Steps 2 and 3.

Agenda creates a text file that contains the contents of the current macro. The contents of the macro still remain in the current Agenda file. You can now import the text file that contains the contents of the macro into another Agenda file.

Attaching Macros to Keys

You can attach a macro to most keys including function, CTRL, ALT, SHIFT function, and special keys (such as INS and GREY +). Attaching a macro to a key simplifies writing customized Agenda applications, and lets users carry out complex tasks in an application with a single keystroke. For example, you can attach a macro to CTRL-S that sorts items in a view. The user simply presses CTRL-S to sort the items.

You can attach a macro to a key, even if this attachment overrides existing key attachments or the functions of Agenda keys. This lets you customize Agenda to meet your needs.

For each file, you can attach up to 100 macros to keys.

Note You cannot attach a macro to ALT-F1, ALT-F3, or ALT-Z. In addition, certain keys in Agenda are recorded as other keys. You should be aware of these keys before attaching macros to them. See the table in "Representing Keys in Macros" earlier in this chapter.

To attach a macro to a key:

1. Press ALT-F3 (MACRO).

   Agenda displays the macro manager.

2. Highlight the macro that you want to attach to a key and press F6 (PROPS).

   Agenda displays the Macros Properties box for the current macro.

3. Highlight the **Attach to key** setting and do one of the following:

   - Press ALT = (hold down ALT, press EQUAL, then release both keys), then press the key to which you want to attach the macro, and press ENTER. (See "Representing Keys in Macros" earlier in this chapter.)

     Agenda automatically encloses the key name in braces.

   - Type the name of the key in braces, for example, `{F4}`, and press ENTER. (See the table in "Representing Keys in Macros" earlier in this chapter.)

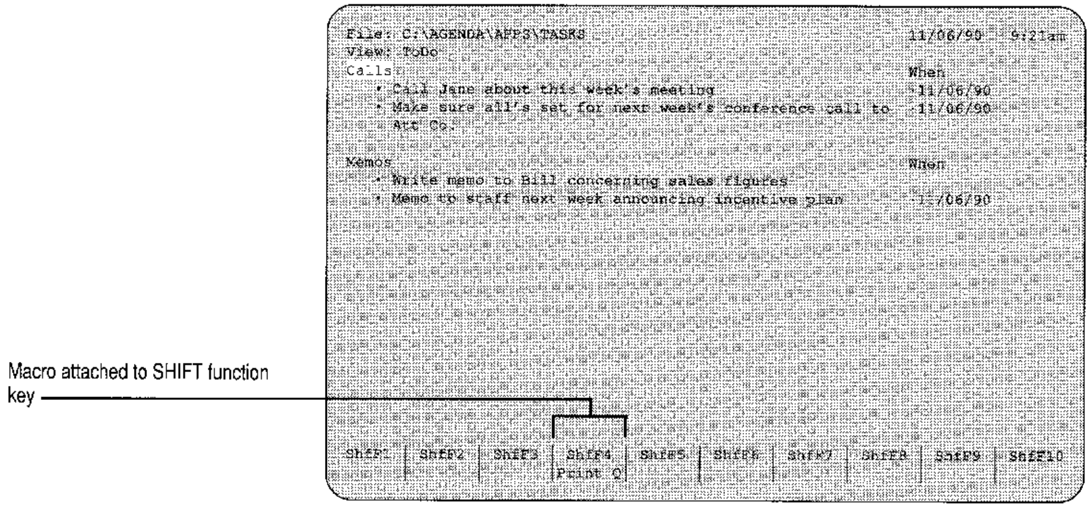
4. Press ENTER.

Agenda displays this key in braces beside the macro name in the macro manager.

If you attached the current macro to a function, CTRL , ALT, or SHIFT function key, the first seven characters of the macro name appear in the function key map at the bottom of the screen (Figure 1-6). Press ALT, CTRL, or SHIFT to see the key map for macros attached to those function keys.

To remove an attachment of a macro to a key:

- From the Macro Properties box, highlight **Attach to key** and press CTRL-ENTER.



```
File: C:\AGENDA\APP3\TASKS                                      11/06/90  9:21am
View: ToDo
Calls                                                           When
    • Call Jane about this week's meeting                       11/06/90
    • Make sure all's set for next week's conference call to    11/06/90
      Att Co.

Memos                                                           When
    • Write memo to Bill concerning sales figures
    • Memo to staff next week announcing incentive plan         11/06/90

ShfF1 | ShfF2 | ShfF3 | ShfF4  | ShfF5 | ShfF6 | ShfF7 | ShfF8 | ShfF9 | ShfF10
      |       |       |Print Q |       |       |       |       |       |
```

Figure 1-6 *Function key map with macro attached to a key*

Use these guidelines when attaching macros to keys:

- If a macro contains a reference to a key that is attached to another macro, the *original* Agenda function given to that key is called by that macro.

  For example, if a macro references the F1 key, the macro calls Agenda Help, which is the normal function of the F1 key in Agenda (even though another macro may be attached to F1). Instead, use the **{CALL}** command to call the macro attached to the key. You can thus develop application-specific Help screens (that are attached to the F1 **(HELP)** key) but still give users access to the Agenda Help system.

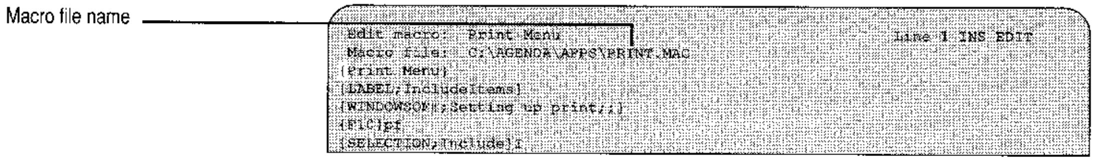
- If you press a key that has a macro attached to it while you are in Learn mode, Agenda inserts a **{CALL}** command into the macro you are recording to call the macro that is attached to that key.

For more information about the {CALL} command, see Chapter 3.

About Macro Files

You can store macros in separate, external macro files. This lets you create one macro that can be used with more than one Agenda file. Macro files are not part of your Agenda file; Agenda stores only the file name and directory in your file.

When you display the Macro edit screen for a macro file (by pressing F2 (EDITMAC) on the macro name in the macro manager, or by pressing SPACE BAR on **Contents** in the Macro Properties box), Agenda displays the contents of the macro file. Agenda also displays the name of the macro file in the upper left corner of the Macro edit screen (Figure 1-7). In the macro manager, Agenda displays a double note symbol (♫) beside all macros stored as external macro files. (See "The Macro Properties Box" earlier in this chapter.)



```
 Edit macro:  Print Menu                                    Line 1 INS EDIT
 Macro file:  C:\AGENDA\APPS\PRINT.MAC
{Print Menu}
{LABEL;IncludeItems}
{WINDOWSOFF;;Setting up print;;}
{F10}pf
{SELECTION;Include}I
```

Figure 1-7 *Macro attached as external macro file*

You work with macros in external macro files the same way you work with macros in your Agenda file. When you leave the macro, Agenda saves any changes that you've made to the macro file.

You can either:

- Attach a new or existing macro file to a macro and then enter or edit the contents of the macro
- Store an existing macro in an external macro file (rather than store it as part of your Agenda file)

Note If you copy your Agenda file to another disk, you should also copy any external files attached to that file — including macro files. (The **File Maintenance MakeCopy** command doesn't copy external files, but it does copy the attachments to these files.)

For information on printing a list of external files attached to an Agenda file, see Chapter 22 in the *User's Guide*.

Attaching a Macro File

You can attach a new or existing file to a macro and then enter or edit the contents of the macro. You might do this if you want to create a macro that you plan to use with several Agenda files.

- If you attach a new macro file to a macro, Agenda displays a blank Macro edit screen; you can then type the contents of the macro that you want to store in this external macro file.
- If you attach an existing macro file to a macro, Agenda displays the contents of the macro contained in that file. You can then edit the contents of that macro to meet your needs, or use the macro as it is. Any changes that you make to the macro are saved to the macro file.

You can attach a macro file to a macro from either the Macro edit screen or the Macro Properties box.

To attach a macro file to a macro from the Macro edit screen:

1. Press ALT-F3 (MACRO) and do one of the following:
   - Type a name (up to 35 characters) of a new macro and press ENTER.
   - Highlight the name of an existing macro.
2. Press F2 (EDITMAC).

   Agenda displays the Macro edit screen for the macro you specified.
3. Press F10 (MENU) and select **File Attach**.

   Agenda displays the File box. If you highlighted the name of an existing macro in Step 1, Agenda asks if you want to delete this macro. Select **Yes** and press ENTER to delete the contents of the macro.
4. Do one of the following:
   - Type the name (up to 8 characters) of a new macro file. If you do not specify a file extension, Agenda gives the file the extension .MAC. Then press ENTER.
   - Press F3 (CHOICES) to display a list of .MAC files. (If you had previously entered a file name with an extension other than .MAC, Agenda displays all the files with that extension.) Highlight the macro file to which you want to attach the current macro and press ENTER.
5. Press ENTER.

If you specified the name of an existing file in Step 4, Agenda displays the contents of this file. Make sure that the name of the macro appears in braces on the *first* line of the macro.

If you specified a new file, Agenda displays a blank macro file. Enter the name of the current macro in braces (up to 35 characters) on the *first* line of the Macro edit screen. You can now type the contents of the macro.

To attach a macro file from the Macro Properties box:

1. Press ALT-F3 (MACRO) and do one of the following:

   - Type a name (up to 35 characters) of a new macro and press ENTER.

   - Highlight the name of an existing macro.

2. Press F6 (PROPS).

   Agenda displays the Macro Properties box for the macro you specified.

3. Highlight **Macro file** and do one of the following:

   - Type a name (of up to 8 characters) of a new macro file and press ENTER. If you do not specify a file extension, Agenda gives the file the extension .MAC.

   - Press F3 (CHOICES) to display a list of .MAC files. (If you had previously entered a file name with an extension other than .MAC, Agenda displays all files with that extension.) Highlight the macro file to which you want to attach the current macro and press ENTER.

     If you highlighted the name of an existing macro in Step 1, Agenda asks if you want to delete this macro. Select **Yes** and press ENTER to delete the macro.

     If the macro already has an attached macro file, Agenda asks if you want to delete the attachment from the current macro file. Select **Yes** and press ENTER.

4. Do one of the following:
   - If you specified the name of an existing macro file in Step 3, do one of the following:
     - If you specified a macro file that already has a name, this name displays in the **Name** setting in the Macro Properties box. Highlight **Contents** and press SPACE BAR to display the contents of the current macro.
     - If you specified a macro file that had no name (or the name does not appear on the first line of the macro), <Nameless Macro> appears as the macro name in the **Name** setting in the Macro Properties box. Highlight **Contents** and press SPACE BAR to display the contents of the current macro. Enter the name of the macro (up to 35 characters) in braces on the *first* line of the Macro edit screen.
   - If you specified the name of a new macro file in Step 3, <Nameless Macro> appears as the macro name in the **Name** setting. Highlight **Contents** and press SPACE BAR. Enter the name of the current macro (up to 35 characters) in braces on the *first* line of the Macro edit screen. You can now type the contents of the macro.
5. Press F10 (MENU) and select **Return**.

Agenda returns to the Macro Properties box and saves any changes you made to this macro.

Note You can attach macro files to an Agenda file that resides on a Local Area Network (LAN). If more than one user is sharing an Agenda file with these attached macro files, or if more than one user has an Agenda file with one or more of these macro files attached to it, all users can see the contents of the attached macro files (as long as the macros in those files aren't protected).

However, only one user at a time can modify a macro file. Thus, if a macro file is being modified, other users can see its contents, but cannot make changes to it. Changes made to the macro file cannot be seen by others on the LAN until the user making the changes leaves the macro (which saves the changes to the macro file). (See Appendix F in the *User's Guide*.)

Storing an Existing Macro in a Macro File

You may have already entered a macro in your Agenda file and then decide that you want to store it in an external macro file so that you can use it with your other Agenda files. In this case, export the contents of the macro to a file and then attach it as a macro file.

To store an existing macro in a macro file:

1. Press ALT-F3 (MACRO) and highlight the name of the macro you want to store in a macro file. Then press F2 (EDITMAC).

   Agenda displays the Macro edit screen for the macro you specified.

2. Press F10 (MENU) and select **Export**.

   Agenda displays the File box.

3. Type the name (up to 8 characters) of the macro file that you want to create and type the file extension .MAC; then press ENTER *twice*. If you don't specify an extension for the file you are exporting, Agenda gives the file the extension .MAC.

   If the macro file you specified exists, Agenda asks if you want to **Append** or **Replace** the file. Select **Append** to add the macro to the end of the contents of the specified file. Select **Replace** to replace the file. Then press ENTER.

   To specify a different file name, press ESC and repeat Step 2.

4. Press F10 (MENU) and select **File Attach**.

   Agenda asks if you want to delete the existing macro. You *must* delete the existing macro before you can attach an external macro file. Select **Yes** and press ENTER to delete the existing macro. (Make sure that you have completed Step 3 before doing this.)

   Agenda displays the File box. Specify the name of the file that you exported your macro to in Step 3.

5. Press ENTER *twice*.

Agenda attaches the macro file that you created to the current macro.

Detaching Macro Files

You can remove the attachment of a macro from an external macro file. This procedure removes the attachment of the current macro from the external macro file but does *not* delete the macro file from your disk. This means that you can still attach this macro file to other Agenda files.

You can remove the attachment of a macro from an external macro file from either the macro manager or the Macro Properties box.

To remove the attachment of a macro from an external file from the macro manager:

1. Press ALT-F3 (MACRO).

   Agenda displays the macro manager.

2. Highlight the name of the macro that you want to detach from the macro file and press F2 (EDITMAC).

   Agenda displays the Macro edit screen for the macro you selected.

3. Press F10 (MENU) and select **File Detach.**

Agenda detaches the macro file from the current macro, but does *not* delete the file from your disk. This means that the macro file still exists. You can attach it to the current or any other Agenda file.

To remove the attachment of a macro from an external macro file from the Macro Properties box:

1. Press ALT-F3 (MACRO).

   Agenda displays the macro manager.

2. Highlight the name of the macro that you want to detach from the macro file and press F6 (PROPS).

   Agenda displays the Macro Properties box for the macro you specified. For the **Macro file** setting, Agenda displays the name of the macro attached to this macro.

3. Highlight **Macro file** and press CTRL-ENTER to detach the macro from this file.

   Agenda asks if you want to detach the existing macro. Select **Yes** and press ENTER *twice.*

Erasing Macro Files

You can erase a macro file from your disk if you no longer want to attach this macro file to the current, or any, Agenda file. This procedure removes the attachment of the current macro from the external macro file *and* also erases this file from your disk.

To erase a macro file from your disk:

1. Press ALT-F3 (MACRO).

   Agenda displays the macro manager.

2. Highlight the name of the macro attached to the macro file that you want to erase and press F2 (EDITMAC).

   Agenda displays the Macro edit screen for the macro you specified.

3. Press F10 (MENU) and select **File Erase**.

   Agenda asks if you want to delete the macro file from disk. Select **Yes** and press ENTER to erase this file.

Agenda removes the attachment of the current macro to this macro file *and* erases this file from your disk.

Chapter 2
About Macro Commands

Agenda macro commands provide you with a powerful programming language tool. You can use these commands to write macros that perform complex operations and simplify working in Agenda.

Chapter 1 provides information about working with macros, including procedures for creating them. Agenda lets you create simple macros by recording keystrokes in Learn mode; you can also create macros that include macro commands. This chapter introduces the macro commands that you can include in macros. Chapter 3 describes in detail each macro command.

In this Chapter

This chapter describes

- The types of macro commands that Agenda offers
- Macro command syntax
- Arguments and variables used with macro commands
- The difference between global and local variables

Note Macros are often written for other people. In this chapter, the term "you" refers to the person writing the macro. The term "user" refers to the person running the macro.

Macro Commands

You can create macros that include macro commands to perform complex operations. For example, you can create macros that

- Change the appearance of a screen
- Display messages
- Prompt for user input
- Loop within a macro
- Call other macros

To create a macro that includes macro commands, follow the procedure in "Creating Macros that Include Macro Commands" in Chapter 1.

Types of Macro Commands

Agenda macro commands fall into the following groups:

- Variable commands define (and undefine) integer, floating-point, and string variables. Some of these commands also assign values to numeric variables and character strings to string variables.
- String-manipulation commands let you make revisions to string variables.
- Flow-of-control commands direct the path of macro execution so that, for example, you can include loops within a macro, or branch to other macros.
- Input/output commands let you request information from, and display information to, the user.
- Highlight-control commands control the movement of the highlight on the screen.
- Miscellaneous commands control macro execution and change different parts of the screen display.

The following tables list Agenda macro commands by group and briefly describe the purpose of each command. For more complete descriptions of these commands and how they work, see Chapter 3.

Variable Commands	*Purpose*
{CLEAR}	Sets the value of some (or all) string variables to null or numeric variables to zero (0)
{DEFFLOAT}	Defines a numeric floating-point variable
{DEFINT}	Defines a numeric integer variable
{DEFSTR}	Defines a string variable
{LET}	Assigns a character string to a string variable and/or a numeric value to a numeric variable
{UNDEF}	Undefines some (or all) numeric and string variables

String-manipulation Commands	*Purpose*
{APPEND}	Appends a string to the end of a string variable
{FIND}	Searches for a string within another string
{LEFTSTR}	Puts into a string variable the leftmost *n* number of characters from another string
{LENGTH}	Assigns the number of characters in a string to a numeric variable
{MIDSTR}	Puts into a string variable a certain number of characters from a point within another string
{RIGHTSTR}	Puts into a string variable the rightmost number of characters from another string

Flow-of-control Commands	*Purpose*
{BRANCH}	Transfers control of macro execution to another point within the same macro
{CALL}	Transfers control of macro execution to another macro. The current macro regains control when the called macro executes {RETURN}
{FOR}	Initiates a loop that performs a task a specified number of times. {LABEL} is the end of a {FOR} loop
{GOTO}	Transfers control of macro execution to another macro. Control does not return to the current macro when the other macro finishes executing
{IF}	Causes the macro to branch to {LABEL} within the same macro if the expression in the command is true
{IFEQ}	Compares two strings and, if they are equal, causes the macro to branch to {LABEL} within the same macro
{IFKEY}	Causes the macro to branch to {LABEL} within the same macro if the user presses a key
{IFNOTEQ}	Compares two strings and, if they are *not* equal, causes the macro to branch to {LABEL} within the same macro
{LABEL}	Creates a reference point in a macro. Lets {BRANCH}, {IF}, {IFEQ}, {IFKEY}, {IFNOTEQ}, and {FOR} transfer control of macro execution to this point within the same macro
{ONBREAK}	Transfers control of macro execution to another macro if the user presses **CTRL-BREAK** during macro execution. The current macro regains control when the other macro executes {RETURN}
{ONERROR}	Transfers control of macro execution to another macro if an error occurs during macro execution. The current macro regains control if the other macro executes {RETURN}
{RETURN}	Returns control of macro execution to the macro that called the current macro using {CALL}, {ONBREAK}, or {ONERROR}

Input/Output Commands	*Purpose*
{ALERT}	Causes a macro to display a box with one or two messages and then pause and wait for the user to press a key
{GETKEY}	Causes a macro to pause and wait for the user to press a key. {GETKEY} stores the keystroke in a variable
{INPUTCAT}	Causes a macro to pause and wait for the user to enter a category name. {INPUTCAT} stores the category name in a variable and creates a new category if a new category is specified
{INPUTFILE}	Causes a macro to pause and wait for the user to enter a file name. {INPUTFILE} stores the file name in a variable
{INPUTTEXT}	Causes a macro to pause and wait for the user to enter text. {INPUTTEXT} stores the text in a variable
{LARGEBOX}	Displays a screen of information and pauses for the user to press **ENTER**
{LOTUSMENU}	Creates a Lotus-style menu across the top of the screen that lets the user perform various operations
{ROLLMENU}	Creates a box with choices that lets the user perform various operations

Highlight-control Commands	*Purpose*
{BS}	Moves the highlight the number of times specified with this command
{CTLEND}	Same as above
{CTLHOME}	Same as above
{CTLLEFT}	Same as above
{CTLPGDN}	Same as above
{CTLPGUP}	Same as above
{CTLRIGHT}	Same as above
{DEL}	Same as above
{DOWN}	Same as above

continued

Highlight-control Commands	*Purpose*
{END}	Same as above
{ENTER}	Same as above
{ESC}	Same as above
{HOME}	Same as above
{INS}	Same as above
{LEFT}	Same as above
{PGDN}	Same as above
{PGUP}	Same as above
{RIGHT}	Same as above
{SELECTION}	Moves the highlight to a specific setting within a box
{SHFTAB}	Moves the highlight the number of times specified with this command
{TAB}	Same as above
{UP}	Same as above

Miscellaneous Commands	*Purpose*
{COMMENT}	Lets you add explanatory comments to a macro. Does not execute
{DEBUGOFF}	Turns off single-step macro execution
{DEBUGON}	Turns on single-step macro execution, which runs a macro one keystroke or instruction at a time
{QUIT}	Terminates macro execution
{SPEED}	Slows down macro execution to one of three speeds that are slower than normal keystroke execution
{TYPE}	Types a string, the contents of a variable, or the result of a numeric expression on the screen
{WINDOWSOFF}	Freezes the screen display while a macro is running, optionally displaying a "Please wait" box

continued

Miscellaneous Commands	*Purpose*
{WINDOWSON}	Unfreezes a screen that has been previously frozen by {WINDOWSOFF} and updates the screen display
{WINDOWSUPD}	Updates the screen display that has been previously frozen by {WINDOWSOFF} while a macro is running

Macro Command Syntax

Each macro command consists of a keyword (the command name) and, in most cases, one or more arguments. For each command, the keyword and any arguments must be enclosed in braces (Figure 2-1).

Macro commands have the following syntax:

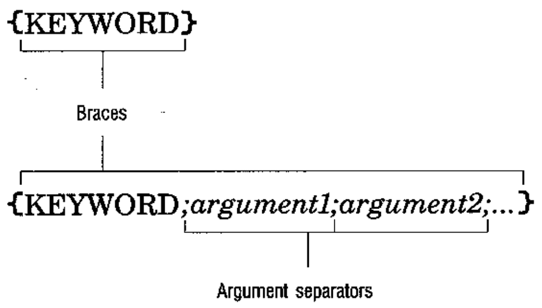


Figure 2-1 ***Macro command syntax***

The keyword tells Agenda what action to perform.

Arguments provide the information that Agenda needs to complete the command: the what, where, and when of a particular action. Arguments can be string constants, numeric constants, string variables, numeric variables, special variables, and/or expressions. (See also "About Arguments" later in this chapter.)

Use these guidelines when working with macro commands:

- Enclose the entire macro command (the keyword and its arguments, if any) in braces ({ }).
- Separate the keyword and each argument with a semicolon.
- You can type uppercase and/or lowercase letters when you enter a macro command. (Macros are not case sensitive.)
- The compiled keystrokes and commands in a macro cannot exceed 2,000 characters. If the macro exceeds this limit, Agenda tells you while the macro is compiling and the macro aborts. The macro text (the macro name, the text from the keystrokes, the commands, and any comments) cannot exceed 10,000 characters.
- Blank lines between commands in a macro are ignored.

About Arguments

Agenda supports six types of arguments that you can include in macro commands. The following table briefly describes each of these arguments. The rest of this chapter provides more detailed information about these arguments.

Argument Type	*Description*	*Example*
String constant	A constant that is a character string (such as the text of a message). You must enclose a string constant in quotes if you want Agenda to recognize leading or trailing white space or if you want to include semicolons (;), braces ({ }), or percents (%) in the string (if the % symbol is the first character in the string)	In the command **{INPUTTEXT;Please enter your name;%name}**, "Please enter your name" is a string constant

continued

Argument Type	*Description*	*Example*
Numeric constant	A constant that is a number	In the command **{ROLLMENU;Do you want to quit?;choices;2;Yes;Labelone;No; Labeltwo}**, 2 is a numeric constant.
String variable	A variable representing a character string (including any sequence of letters, numbers, and symbols)	In the command **{INPUTTEXT;Please enter your name;%fullname}**, %fullname is a string variable.
Numeric variable	A variable representing a numeric value	In the command **{DEFINT;%year}{LET;%year;1990}**, %year is a numeric variable.
Expression	Formulas that you can use to perform mathematical operations or string comparisons on variables and constants	In the command **{TYPE;(5+7)}**, (5+7) is an expression.
Special variable	A variable that returns a specific value, describing the current state of the Agenda file. Special variables cannot be given any other value by the user. Some special variables return strings; others return numbers. Special variables are always preceded by a pound sign (#)	In the command **{IF;(#MODE=NOTE);Start}**, #MODE is a special variable equal to NOTE only if the cursor is in a note.

Some macro commands have arguments that expect either strings or numbers.

- For commands for which a number is expected, you can include numeric variables, numeric constants, some special variables, or expressions for the arguments.
- For commands for which a string is expected, you can include string variables, string constants, and some special variables for the arguments.

If you use the wrong type of argument in a command, Agenda tries to convert the variable to the appropriate argument type.

- If you use a numeric value where a string is expected, Agenda converts the numeric value to a string representation.
- If you use a string where a number is expected, Agenda tries to convert the string to a numeric value. If the string cannot be converted to a number, Agenda uses a value of zero (0).

For example, the command {FOR;%i;1;10;1;LOOP} expects numbers for the second, third, and fourth arguments. If, on the other hand, the command was {FOR;%i;1;%a;1;LOOP}, and %a was previously defined in the macro as a string, Agenda tries to convert this string to a numeric value.

About String Constants

You must enclose string constants in quotes if you want the string to include

- Semicolons (;), braces ({ }), and percents (%) (if the % symbol is the first character in the string)
- An expression operator, such as a hyphen (-), or anything that looks like an expression, such as (Profit - Expenses), so that Agenda interprets this as a string and *not* an expression. For example, if you specify {IF;(%a="Joanna James-Smith");LOOP}, Agenda will not interpret the hyphen as a minus sign (see also "About Expressions" later in this chapter)
- White space (spaces, tabs, and carriage returns) before and/or after string constants

  Agenda strips white space before and after string constants. If you want Agenda to recognize white space preceding and/or following a string, you must enclose the string in quotes. For example, the command {LET;%d;" John Smith "} recognizes a space preceding and following the string John Smith.

You can enclose the string in either single or double quotes, but both the open and close quotes must match.

If you need to enclose a string in quotes for one of the above reasons and the string already contains quotes, enclose the string with the other type of quote. For example, to specify a string that already contains double quotes, type `'Employee number "1110"; John Smith'`

Notes You cannot specify a string constant that contains, as text, both a single and double quote.

If you write a macro that includes a slash (/), this character produces different results, depending on where the macro executes the keystrokes that include the slash. For example, if a macro containing a slash is run in a view, the slash brings up the menu. If a macro containing a slash is run while editing an item or a note, the macro produces the slash character.

Tip To bring up the menu, regardless of what part of Agenda the macro is run in, include {F10} in the macro. To produce the slash (/) character, include this character in the {TYPE} command, for example {TYPE; /}.

About Variables

As shown in "About Arguments" earlier in this chapter, Agenda supports both string variables and numeric variables. (For information on special variables, see Chapter 3.)

String variables represent specific character strings and can include any sequence of letters, numbers, and symbols.

Numeric variables represent numeric values.

Agenda offers two types of numeric variables:

- Integer numeric variables

  Integer numeric variables provide loop counters and string indexes that you can use, for example, in string-manipulation commands and to perform numeric operations. Each integer numeric variable can hold numbers between ± 2 billion.

- Floating-point numeric variables

  Floating-point numeric variables let you perform numeric operations using floating-point numbers. Each floating-point numeric variable can hold numbers between ± 1 E 70. Floating-point numeric variables evaluate to two decimal places.

Agenda performs the following mathematical operations on numeric integer and floating-point variables:

- Addition
- Subtraction
- Multiplication
- Division
- Comparison

About Expressions

Expressions are formulas that you use to perform mathematical operations or string comparisons on variables and constants. You can use an expression anywhere that you use a numeric variable. Expressions always return a numeric value (either integer or floating point). Expressions that are used for comparisons return a value of 1 (True) or 0 (False). You *must* enclose expressions in parentheses.

Valid operators for expressions are:

+, -, *, /, <, >, <=, >=, <>, =

Expressions are evaluated from left to right with operator precedence in the following order:

- Unary operators (for example, -1 or +1)
- Multiplication (*) and division (/)
- Addition (+) and subtraction (-)
- Relational operators (>, <, >=, <=, =, and <>)

Thus, for the following expression:

1 + 2 > 4/2

4/2 is evaluated first, 1 + 2 is evaluated second, and the comparison of the results, 3 > 2, is evaluated last, showing the expression to be true.

Note You cannot use exponents with expressions.

You can also perform comparisons of strings. Both parameters being compared must be strings.

Valid operators for string comparisons are:

<, >, <>, =, >=, <=

Note Comparison operators are *not* case sensitive. For example, (lynne = LYNNE) is true.

Global and Local Variables

Agenda lets you define variables as either global or local.

A global variable is a variable that is accessible by *all* macros in a file. A global variable is also permanent; it remains in the file until you undefine it using {UNDEF}.

You identify a variable as global by preceding the variable name with *two* percent (%) symbols. For example, %%name and %%guest are both global variables.

You define a global variable if you want information to be shared among macros and used in a variety of places. For example, you might define a global variable and use {INPUTTEXT} to input a person's name and then use that variable (%%name) in other macros.

You might also define a global variable if you want a macro to store information from one execution of the macro to the next. You must use a global variable if you want one macro to provide a value to another macro (in either direction).

In contrast to global variables, local variables are accessible to the current macro *only*; other macros — including called macros — do *not* have access to local variables and their values. Local variables are also temporary; as soon as the macro finishes executing, Agenda automatically undefines any local variables in that macro. (In other words, local variables do not remain in the file.)

You identify a local variable by preceding the variable name with *one* percent (%) symbol. For example, %name and %guest are both local variables.

Sometimes, you might have two macros, each of which define a local variable with the same name. Because these are local variables, these macros each access two different variables (even though they have the same name). If, on the other hand, these macros both define a global variable with the same name, they each access and change the value of the same variable (Figure 2-2).

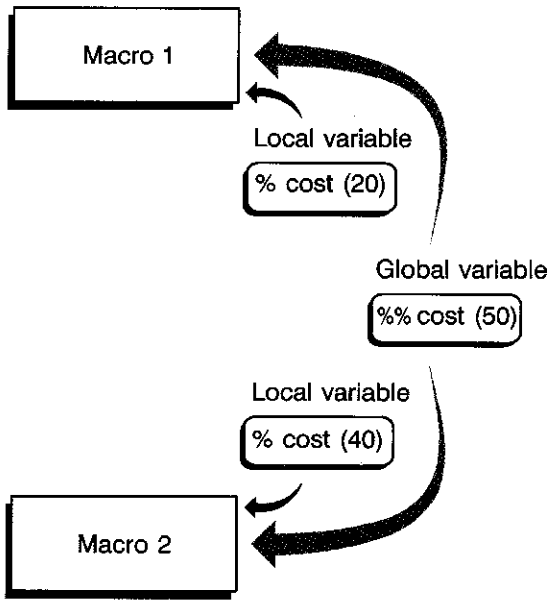


Figure 2-2 *Global and local variables*

In this figure, the first macro (Macro 1) includes the local variable %cost and gives it a value (20). The second macro (Macro 2) also includes a local variable by the same name (%cost) and gives it a value (40). Because each of these variables is a local variable, Macro 1 uses 20, the value assigned to the %cost variable in that macro and Macro 2 uses 40, the value assigned to the %cost variable in that macro. Neither macro can access or change the value of the other's %cost variable.

On the other hand, Macro 1 and Macro 2 might each define a global variable (%%cost) and give it a value (for example, 50). By definition, this variable, and its value, is accessible by all macros in the file. Thus, both Macro 1 and Macro 2 access the same global variable (%%cost) and its value of 50. In addition, if either macro changes the value of %%cost, the value of that variable is changed and *all* macros now access the new value for that variable.

Guidelines for Working with Variables

Use the following guidelines when working with variables in macro commands:

- Both local and global variable names can be up to 24 characters long. Variable names can include special characters *except* semicolons (;), braces ({ }), or percents (%).
- Local variable names must be preceded by a single percent (%), for example, %a or %number.
- Global variable names must be preceded by two percent signs, for example, %%plan or %%average.
- Each string variable (regardless of whether it's a local or global variable) represents a string of up to 1,000 characters. If a string variable is longer than the maximum length of characters for a particular argument, Agenda uses as much of the variable as possible.
- Each Agenda file can contain up to 100 macros and an unlimited number of variables (both local and global).
- Macro variables are stored in the Agenda file. This has different implications, depending on whether a variable is global or local.

  If a command in one macro gives a value to a *global* variable, all macros in that file use the value assigned to that global variable until it is changed with a macro command or undefined (with the {UNDEF} command).

  Make sure to undefine all global variables when you no longer need them because global variables remain in your file. (See {UNDEF} in Chapter 3.)

  If a command in one macro gives a value to a *local* variable, only that macro uses the value assigned to that local variable. Once a macro completes processing, Agenda automatically undefines all local variables in that macro.
- You should not combine a (numeric or string) variable and a character string into a single argument. Use {APPEND} to make a single string; then use the result.

Chapter 3
Macro Command Descriptions

Agenda offers many macro commands that you can include in macros. These commands provide a powerful programming language tool that performs long and complicated tasks. Macro commands can also perform complex operations such as changing the appearance of a screen, displaying messages, looping, calling other macros, and prompting for user input.

In this Chapter

This chapter describes

- The syntax and purpose of each macro command
- The purpose of each special variable

In most cases, this chapter also includes one or more examples of how each macro command works. The commands are organized alphabetically by keyword.

For basic information about macro commands, see Chapter 2. For examples of more complex macros that include many of these commands, see Chapter 5. See also *Starter Applications* for sample applications that include many of the commands described in this chapter.

Note Optional arguments for the commands that follow are enclosed in brackets ([ ]). Examples that include ellipses (...) indicate that macro commands preceding or following the commands in the example have been omitted.

{ALERT}

{ALERT} causes a macro to display a box with one or two messages and then pause and wait for the user to press a key.

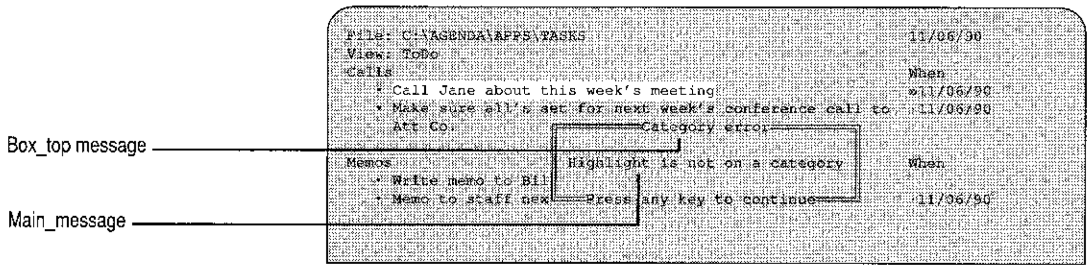
Syntax {ALERT;*boxtop_message;main_message*}

boxtop_message represents the message at the top of the box. *main_message* represents the message displayed in the box. *box-top_message* and *main_message* can each be character strings or variables of up to 76 characters.

The following example produces a sample alert box (Figure 3-1).

Example

{ALERT;Category error;Highlight is not on a category}	displays an alert box



```
File: C:\AGENDA\APPS\TASKS                                    11/06/90
View: ToDo
Calls                                                         When
   • Call Jane about this week's meeting                      »11/06/90
   • Make sure all's set for next week's conference call to    11/06/90
     Att Co.        ╔════════Category error═════════╗
Memos               ║ Highlight is not on a category ║        When
   • Write memo to Bil║                              ║
   • Memo to staff nex╚═══Press any key to continue═══╝        11/06/90
```

Figure 3-1 *Sample alert box created using {ALERT}*

The main message is centered in the box. The message "Press any key to continue" always displays along the bottom border of the box.

Note To display only one of the messages, omit one of the arguments (for example, {ALERT;;main_message}) or use {ALERT;boxtop_message;}. Because {ALERT} requires two arguments, you must include two semicolons even if only one argument is provided.

{APPEND}

{APPEND} appends a string to the end of a string variable.

Syntax {APPEND;*strvar*;*string*}

string represents the string that you are appending to the end of *strvar*. *strvar* represents the string variable to which you are appending *string*.

Example

`{INPUTTEXT;Enter first name;%first}`	prompts user for first name
`{INPUTTEXT;Enter last name;%fullname}`	prompts users for last name
`{APPEND;%fullname;", "}`	Appends ", " (a comma and space) to last name
`{APPEND;%fullname;%first}`	Appends first name to last name (separated by a comma and space)
`{TYPE;%fullname}`	types out lastname, firstname

This macro produces the following output:

```
Lastname,  Firstname
```

{BRANCH}

{BRANCH} transfers control of macro execution to another point within the same macro.

Syntax {BRANCH;*labelname*}

labelname represents the point to which macro execution transfers control. *labelname* can be a character string or a variable. You use {LABEL} to establish labelname.

If there is no {LABEL} command with a matching *labelname*, the macro proceeds to the command after {BRANCH} (that is, {BRANCH} is ignored).

Note Matching is *not* case sensitive.

The following is an example of a subroutine called by a macro to return the user to a view.

Example

`{LABEL;ToView}`	
`{BRANCH;#MODE}`	branches to {LABEL} designed to match the possible values of #MODE
`{ESC}`	escapes out of every mode in Agenda except view, note, or category manager
`{BRANCH;ToView}`	branches to top of loop to try again
`{LABEL;Note}`	if #MODE = NOTE
`{F5}`	F5 exits note
`{BRANCH;ToView}`	loops to try again
`{LABEL;CatMgr}`	if #MODE = category manager
`{F9}`	F9 goes to the view
`{LABEL;View}`	if #MODE = VIEW, macro successfully executes
`...`	
`{RETURN}`	returns to the macro that executed {CALL}

Note Whereas {BRANCH} transfers control of macro execution to another point within the *same* macro; {CALL} transfers control of macro execution to *another* macro.

{CALL}

{CALL} transfers control of macro execution to another macro. The current macro regains control when the called macro executes {RETURN}.

Syntax {CALL;*macroname*[;*parameter1*;*parameter2*];...}

macroname represents the name of the macro being called. *parameter1*, *parameter2*... represent optional arguments that the caller macro passes to the called macro. These arguments are applied to the first, second, and so forth {INPUTTEXT}, {INPUTCAT}, or {INPUTFILE} commands encountered in the called macro.

The purpose of this technique is to provide a macro that serves as a subroutine for other macros. In this case, the macro gets its information directly from the macro that called it, without prompting for user input. (See {INPUTTEXT}, {INPUTCAT}, and {INPUTFILE} later in this chapter.)

The macro that executed {CALL} regains control when the called macro executes {RETURN}. If the called macro does *not* include {QUIT} or {RETURN}, Agenda returns to the calling macro when the called macro finishes executing.

Note The special variable #ARGCOUNT equals the number of parameters in {CALL} left to be executed. (See #ARGCOUNT later in this chapter.)

In the following example, Macro 1 (the caller macro) calls Macro 2 (the called macro). *parameter1* (John Smith) "pairs up" with the first {INPUTTEXT} command in the called macro, {INPUTTEXT;Enter the name;%name}.

parameter2 (555-1234) "pairs up" with the second {INPUTTEXT command}, {INPUTTEXT;Enter the phone number;%phone}. Thus, %name contains John Smith and %phone contains 555-1234, without the user ever being prompted for this input.

Example

Macro 1 (The caller macro)

`{CALL;Add Name and Number;John Smith; "555-1234"}`	calls Macro 2, Add Name and Number, and passes the name and telephone number to the variables %name and %phone in Macro 2
`. . .`	

Macro 2 (The called macro, Add Name and Number)

`{Add Name and Number}`	the called macro
`{INPUTTEXT; Enter the name;%name}`	passes the name and phone number used as parameters in Macro 1 to the variables %name and %phone without prompting for user input
`{INPUTTEXT; Enter the phone number;%phone}`	
`{F9}`	switches to the category manager
`{INS}`	adds the category using the contents of %name
`{TYPE;%name}`	
`{ENTER}`	
`{F5}`	enters the note
`{ENTER}`	puts the phone number on line 2 of the category's note

continued

{TYPE;%phone}	
{F5}	exits the note, accepting the changes made to it
{F9}	exits the category manager
{RETURN}	returns control to Macro 1, the macro which executed the {CALL} command

{CLEAR}

{CLEAR} sets the value of some (or all) string variables to null or numeric variables to 0. This command is equivalent to the macro command {LET;%*var*;" "} or {LET;%*var*;0}.

Syntax {CLEAR;*var1*;[*var2*];...} or {CLEAR;ALL}

var represents the name of the string or numeric variable that you want to clear. You can use one {CLEAR;*var*} command to clear as many global and/or local string and numeric variables as you want.

{CLEAR;ALL} clears all the global variables in a file and all the local variables in the current macro at once.

Whereas {UNDEF} removes all traces of a variable, {CLEAR} leaves the variables that it clears in the file so that they retain their status as being either string or numeric variables. {CLEAR} also sets the values of string variables to null and numeric variables to 0. {CLEAR} is useful for resetting variables that you want to reuse in a macro. (See also {UNDEF} later in this chapter.)

{COMMENT}

{COMMENT} lets you add explanatory comments to a macro. Comments are especially helpful if other people use your macros. (Comments do not execute.)

Syntax {COMMENT;*text*}

text represents the explanatory comment that you add to a macro. Comments are visible only when you are editing a macro; they do not affect a macro when it runs. Everything between the COMMENT keyword and the closing brace is ignored when the macro is executed. There is no maximum length for comment text.

You can also "comment out" macro instructions. When you do, the macro does not execute these instructions. You might comment out one or more macro instructions that you don't plan to use at this time, but that you don't want to delete. The following example illustrates this point.

Note You must include both an opening *and* closing brace when commenting out macro instructions.

Example

Comments out macro instructions

`{INPUTTEXT;First name?;%fname}`	asks user for first name
`{INPUTTEXT;Last name?;%fullname}`	asks user for last name
`{APPEND;%fullname; ", "}`	adds a comma (,) and space ( ) to the full name
`{APPEND;%fullname; %fname}`	adds first name to the full name
`{COMMENT;For debugging`	"comments out" the following macro instructions
`{ALERT;Full name; %fullname}`	shows full name
`{COMMENT; (Done debugging)}`	this is a regular comment
`}`	ends "comment out"

{DEBUGOFF}

{DEBUGOFF} turns off single-step macro execution. You set single-step execution using {DEBUGON}.

Syntax {DEBUGOFF}

{DEBUGON}

{DEBUGON} turns on single-step macro execution, which lets you run a macro one keystroke at a time.

Syntax {DEBUGON}

With {DEBUGON}, you (or the user) press any key to step through macro execution, one keystroke or command at a time. Agenda displays the {DEBUGON} indicator (DEBUG) in the upper-right corner of the screen to indicate that you are in this mode. {DEBUGON} stays in effect until the macro ends or encounters a {DEBUGOFF} command.

{DEBUGON} can help you debug a long or complicated macro; it lets you step through the macro to help you determine where the problem is.

Note If you use {DEBUGON}, {WINDOWSOFF} has no effect. In addition, {DEBUGON} will execute {WINDOWSON}, if {WINDOWSOFF} was previously executed.

{DEFFLOAT}

{DEFFLOAT} defines global and/or local floating-point variables.

Syntax {DEFFLOAT;*numvar1*[;*numvar2*];...}

numvar represents the numeric floating-point variable you're defining.

You must use {DEFFLOAT} to define a numeric floating-point variable. Agenda does not default to floating-point numeric variables. Display of floating-point variables default to two decimal places.

You might define a floating-point variable, for example, to collect expense information. (For an example of a macro that uses {DEFFLOAT} for this purpose, see Chapter 5.)

If you use {DEFFLOAT} to define a variable that already exists, the command is ignored.

You can use one {DEFFLOAT} command to define several local and/or global numeric floating-point variables. For example, you can use {DEFFLOAT;%%rate;%i;%j} to define one global numeric floating-point variable (%%rate) and two local numeric floating-point variables (%i and %j).

Note You should not use floating-point variables as loop counters or string indexes in string-manipulation commands. Doing so may produce unpredictable results. Use an integer instead. (See {DEFINT} later in this chapter.)

The following example illustrates how {DEFFLOAT} defines a floating-point variable.

Example

`{DEFFLOAT;%f}`	defines a floating-point variable
`{LET;%f;(4.11 + 0.62)}`	sets %f equal to the sum of (4.11 + 0.62)
`{TYPE;%f}`	types out the contents of %f

This macro produces the following output:

```
4.73
```

{DEFINT}

{DEFINT} defines global and/or local numeric integer variables. You can use numeric integer variables as loop counters and string indexes in, for example, string-manipulation commands.

Syntax {DEFINT;*numvar1*[;*numvar2*];...}

numvar represents the numeric integer variable you're defining.

You can use {DEFINT} to define as many local and/or global numeric integer variables as you want. For example, you can use {DEFINT;%%count;%i;%j} to define one global numeric integer variable (%%count) and two local numeric integer variables (%i and %j).

If you use {DEFINT} to define a variable that already exists, the command is ignored.

You do not have to define a numeric integer variable using {DEFINT}. You use this command only to define a numeric integer variable if it is not obvious from the macro that a variable is a numeric integer variable. In many cases, Agenda can determine whether a variable is a string or numeric integer variable based on its use in a macro.

For example, in {FOR;%i;1;10;1;LOOP}, Agenda recognizes that the variable %i is a numeric integer variable because arguments in {FOR} must be numeric. If, in the macro command {LET;%x;%y}, the variable %y has a numeric integer value, Agenda assumes that the variable %x is also a numeric integer variable because %x = %y.

{DEFSTR}

{DEFSTR} defines a local or global string variable.

Syntax {DEFSTR;*strvar1*[;*strvar2*];...}

strvar represents the string variable you're defining.

You can use this command to define as many local and/or global string variables as you want. For example, you can use {DEFSTR;%%name;%first;%last} to define one global string variable (%%name) and two local string variables (%first and %last).

If you use {DEFSTR} to define a variable that already exists, the command is ignored.

You do not have to define a string variable using {DEFSTR}. You use this command only to define a string variable if it is not obvious from the macro that a variable is a string variable. In many cases, Agenda can determine whether a variable is a string or numeric variable, based on its use in a macro.

For example, in the macro command {RIGHTSTR;%s;%name;1}, Agenda recognizes that the variable %s is a string variable because the value assigned to it is a string.

{FIND}

{FIND} searches for *string2* in *string1*, beginning at a specific point in *string1*. The search is not case sensitive. {FIND} returns the *position* in *string1* where *string2* begins.

Syntax {FIND;*numvar;string1;string2;position*}

numvar represents the integer variable in which the result is stored. *string2* represents the string you are searching for in *string1*. *string1* represents the string in which you are searching for *string2*. *position* is a number that represents the point at which the search begins.

Starting its search at *position*, {FIND} finds *string2* in *string1*. If found, it puts the number of the first character of the match in *numvar*. If it is not found, Agenda gives *numvar* the value –1.

Example

`{FIND;%string;one two three;three;1}`	finds the string "three" in the string "one two three", beginning with the first character
`{TYPE;%string}`	types the number of the position of the first character of the match

This macro produces the following output:

```
9
```

{FOR}

{FOR} initiates a loop that performs a task a specified number of times.

Syntax {FOR;*numvar;initial val;final val;incr;labelname*}

numvar represents the variable that *initial val, final val,* and *incr* control the value of. *initial val* sets the starting value of *numvar* the first time the loop is executed. *final val* represents the value for *numvar* that causes the loop to end. *incr* represents the number by which *numvar* is incremented (or decremented). *labelname* represents the label that designates the last instruction in the {FOR} loop. All commands before the corresponding {LABEL} command are executed for each loop.

Note The arguments in {FOR} must be integers.

In the following example, the {FOR} loop creates ten new items, each numbered from 1 to 10. After the tenth item is entered, the loop ends.

Example

`{FOR;%Item;1;10;1;ENDLOOP}`	initiates a {FOR} loop that enters ten items in sequence
`{TYPE;%Item}`	types a number (the current value of %Item) into item
`{ENTER}`	accepts the item
`{LABEL;ENDLOOP}`	indicates the end of the {FOR} loop

Note You can also decrement the loop counter using a negative number for *incr*. For example, if the {FOR} command in the above example read {FOR;%ITEM;10;1;-1;ENDLOOP}, it would create ten items numbered from 10 to 1, ending with 1.

{GETKEY}

{GETKEY} causes a macro to pause and wait for the user to press a key. {GETKEY} stores the keystroke in a variable. You can test or use that variable in subsequent macro commands.

Syntax {GETKEY;*strvar*}

strvar represents the name of the string variable you want to use to store the keystroke.

{GETKEY} does not prompt the user for the keystroke. You should use other macro commands to provide a prompt for user input.

Note You may have a macro that displays a box that requires the user to press a key to clear the box (such as {ALERT}, *followed by* {GETKEY}. In this case, the first key pressed clears the alert box; the *second* key pressed is stored as a {GETKEY} variable.

The following example shows a macro that repeats each keystroke twice (for example, if the user presses a, Agenda produces aa on the screen) until the user presses TAB.

Example

`{LABEL;Get More}`	top of loop
`{GETKEY;%key}`	waits for next key pressed

`{IF;(%key={TAB});`
`  Stop get}`
	if TAB pressed, gets out of loop

`{TYPE;%key}`	outputs the key twice
`{TYPE;%key}`	
`{BRANCH;Get More}`	loops up to wait for next key
`{LABEL;Stop get}`	
`....`	

{GOTO}

{GOTO} transfers control of macro execution to another macro. When the macro runs, it executes all keystrokes and commands before the {GOTO} command, and then begins executing the second macro. Control of macro execution does not return to the calling macro when the called macro finishes executing.

Syntax {GOTO;*macroname*}

macroname represents the name of an existing macro. If *macroname* is not found, macro execution continues with the command after {GOTO} ({GOTO} is ignored). When the called macro is finished executing, control of macro execution is *not* returned to the macro that executed {GOTO}. (See {CALL} earlier in this chapter.)

Note Whereas {GOTO} transfers control of macro execution to another macro; {BRANCH} transfers control of macro execution to *another* point within the same macro.

{IF}

{IF} causes a macro to branch to {LABEL} within the same macro if *expression* is true.

Syntax {IF;*expression;labelname*}

expression represents the numeric or string comparison that is evaluated when {IF} executes. *labelname* can be a character string or variable.

Expressions *must* be enclosed in parentheses ( ). *labelname* represents the label that the command branches to. If *expression* is false, the macro proceeds to the next macro command after {IF}. (For more information on expressions, see Chapter 2.)

In the following example, the macro compares the variables %a and %b. If their values are equal, the macro displays the message, "The values are equal." If their values are not equal, the macro displays the message, "The values are not equal."

Example

`{IF;(%a=%b);EQUAL}`	compares %a and %b to see if they are equal
`{ALERT;;The values are not equal}`	displays this message if the values are not equal
`{BRANCH;SKIP}`	skips over the equal message
`{LABEL;EQUAL}`	
`{ALERT;;The values are equal}`	displays this message if the values are equal
`{LABEL;SKIP}`	resumes macro execution here

{IFEQ}

{IFEQ} compares two strings and, if they are equal, causes a macro to branch to another point ({LABEL}) within the same macro.

Syntax {IFEQ;*value1;value2;labelname*}

value1 and *value2* represent the two strings that are compared. *value1* and *value2* can be numeric or string constants, numeric or string variables, special variables, or expressions.

labelname represents the point to which macro execution transfers control within the same macro. *labelname* can be a character string or variable. You use {LABEL} to establish *labelname.*

{IFEQ} compares *value1* to *value2* (this comparison is not case sensitive). One of the following happens:

- If the values are equal, the macro transfers control to the {LABEL;*labelname*} command in the same macro.
- If the values are not equal, execution continues with the command that follows {IFEQ}.

{IFEQ} is equivalent to {IF;(*value1=value2*);labelname}.

If there is no {LABEL} command with a matching *labelname,* execution continues with the command after {IFEQ} ({IFEQ} is ignored).

{IFKEY}

{IFKEY} causes a macro to branch to another point within the same macro ({LABEL}) if the user presses a key.

Syntax {IFKEY;*labelname*}

labelname represents the point to which macro execution branches within the same macro. *labelname* can be a character string or variable. You use the {LABEL} command to establish *labelname.*

You can use {IFKEY} when a macro initiates an activity that you want to continue until the user interrupts it by pressing a key. When the user presses a key, macro execution continues at the {LABEL;*labelname*} command within the same macro.

{IFKEY} does not store the keystroke. If necessary, you can follow this command with {GETKEY} or {INPUTTEXT} to capture the keystroke and test it.

If there is no {LABEL} command with a matching *labelname*, macro execution continues with the command that follows {IFKEY} ({IFKEY} is ignored).

In the following example, the contents of %a are typed repeatedly until the user presses a key. The keystroke that caused the loop to terminate is placed in %b by {GETKEY} to get the key out of the buffer so that it does not clear the Alert box. If {GETKEY} was not included in this macro, the key that caused the loop to terminate would cause the Alert box to be cleared before it could be read by the user.

Example

`{LABEL;loop}`	
`{LET;%a;a}`	
`{TYPE;%a}`	types the contents of the variable
`{IFKEY;end loop}`	aborts LOOP if key pressed
`{BRANCH;loop}`	if no key pressed, branches to LOOP and continues looping
`{LABEL;end loop}`	
`{GETKEY;%b}`	gets the key that was pressed (include to get key out of buffer)
`{ALERT;;Loop ended.}`	displays the message "Loop ended."
`{QUIT}`	

{IFNOTEQ}

{IFNOTEQ} compares two strings and, if they are *not* equal, branches to another point ({LABEL}) within the same macro.

Syntax {IFNOTEQ;*value1;value2;labelname*}

value1 and *value2* represent the two strings that are compared. *value1* and *value2* can be numeric or string constants, numeric or string variables, special variables, or expressions.

labelname represents the point to which macro execution transfers control within the same macro. *labelname* can be a character string or a variable. You use {LABEL} to establish *labelname*.

{IFNOTEQ} compares *value1* and *value2* (this comparison is not case sensitive). One of the following happens:

- If the values are not equal, the macro transfers control to the {LABEL;*labelname*} command in the same macro.
- If the values are equal, execution continues with the command that follows {IFNOTEQ}.

{IFNOTEQ} is equivalent to {IF;(*value1<>value2*);*labelname*}.

If there is no {LABEL} command with a matching *labelname*, macro execution continues with the command after {IFNOTEQ} ({IFNOTEQ} is ignored).

{INPUTCAT}

{INPUTCAT} causes a macro to pause and wait for the user to enter a category name. {INPUTCAT} stores the category name in a variable and creates a new category if a new category is specified. You can test or use that variable in subsequent macro commands.

Syntax {INPUTCAT;*prompt_message;var*}

prompt_message represents the message in the top border of the box. It can be a character string of up to 76 characters or a variable. (Any string longer than 76 characters will be truncated.)

var represents the name of the variable in which Agenda stores the category name that the user enters. The user can enter up to 69 characters.

If the user presses ENTER without specifying a category name or presses ESC, one of the following happens:

- If a string variable is used, the value of *var* is set to null.
- If a numeric variable is used, the value of *var* is set to 0 (zero). Or, if a string is specified for a numeric variable, the value of *var* is set to 0 (zero).

You'd use {INPUTCAT} rather than {INPUTTEXT} to let the user use automatic completion or press F3 (CHOICES) to display the list of categories in the current file.

{INPUTCAT} displays a box with the *prompt_message* in the top border of the box and the message "Press ENTER to accept, ESC to cancel" in the bottom border of the box. This command displays the highlight in the box where the user types characters.

The user can do either of the following:

- Type the name of a new category and press ENTER to create a new category.

  Agenda automatically adds the category as a child of MAIN.

- Press F3 (CHOICES) to display the category hierarchy and select an existing category, or add a child of another category.

To add a category as a child of another category:

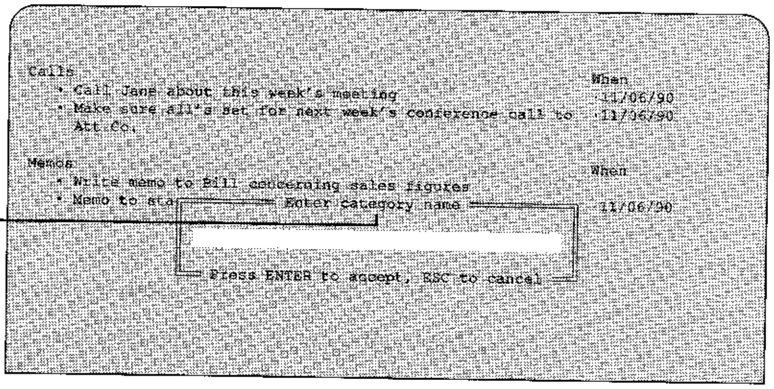
1. Press F3 (CHOICES) to display the category hierarchy.
2. Highlight the category under which you want to add a child category.
3. Press ALT-R and type the name of the category you want to add. Then press ENTER *three times.*

Agenda adds the category as a child of the category you specify.

In the following example, Agenda prompts the user to enter the name of the category to which to assign the current item (Figure 3-2). This macro assumes that the highlight is on an item.

Example

`{INPUTCAT; Enter category name;%catname}`	prompts user for category name
`{F6}`	displays the Item Properties box
`{SELECTION; Assigned to}`	moves the highlight to Assigned to setting
`{INS}`	adds a category
`{TYPE;%catname}`	types the category name
`{ENTER;2}`	accepts the category name and Item Properties box



```
Calls                                                          When
   • Call Jane about this week's meeting                       11/06/90
   • Make sure all's set for next week's conference call to    11/06/90
     Att.Co.

Memos                                                          When
   • Write memo to Bill concerning sales figures
   • Memo to sta╔═══════ Enter category name ═══════╗          11/06/90
                ║                                   ║
                ╚═ Press ENTER to accept, ESC to cancel ═╝
```

Figure 3-2 *Sample prompt for category name using {INPUTCAT}*

You can also pass values to {INPUTCAT} commands (or to {INPUTTEXT} or {INPUTFILE}) in a called macro using the arguments in {CALL}. In this case, the macro gets its information directly from the macro that called it, without prompting for user input. (See {CALL} and #ARGCOUNT in this chapter.)

{INPUTFILE}

{INPUTFILE} causes a macro to pause and wait for the user to enter a file name. The {INPUTFILE} command stores the file name in a variable. You can test or use that variable in subsequent macro commands.

Syntax {INPUTFILE;*prompt_message;var*}

prompt_message represents the message in the top border of the box. It can be a character string of up to 76 characters or a variable. (Any string longer than 76 characters is truncated.)

var represents the name of the variable in which Agenda stores the file name that the user enters. The user can enter up to 79 characters.

If the user presses ENTER without specifying a file name, or presses ESC, one of the following happens:

- If a string variable is used, the value of *var* is set to null.
- If a numeric variable is used, the value of *var* is set to 0 (zero). Or, if a string is specified for a numeric variable, the value of *var* is set to 0 (zero).

You'd use {INPUTFILE} rather than {INPUTTEXT} to let the user use automatic completion or press F3 (CHOICES) to display a list of files.

{INPUTFILE} displays a box with the *prompt_message* in the top border of the box and the message "Press ENTER to accept, ESC to cancel" in the bottom border of the box. This command displays the highlight in the box where the user types characters.

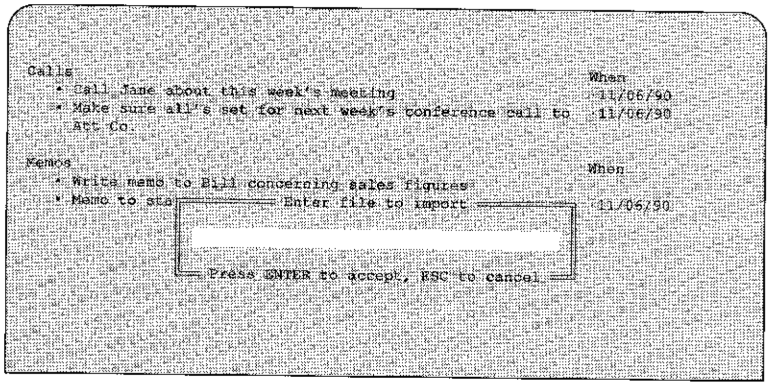
The user can do either of the following:

- Type the name of a new file and press ENTER to specify a new file name.
- Press F3 (CHOICES) to display a list of files and select an existing file.

In the following example, Agenda prompts the user to enter the name of a file to import into the current Agenda file (Figure 3-3). This macro assumes that the highlight is in a view.

Example

`{INPUTFILE; Enter file to import;%filename}`	prompts user for file to import
`{F10}FTI`	displays Import Structured file box
`{TYPE;%filename}`	types the file name
`{ENTER;2}`	accepts the file name and imports the structured file



```
Calls                                                          When
   • Call Jane about this week's meeting                       11/06/90
   • Make sure all's set for next week's conference call to    11/06/90
     Art Co.

Memos                                                          When
   • Write memo to Bill concerning sales figures
   • Memo to sta╔═══════ Enter file to import ════════╗        11/06/90
                ║                                     ║
                ║                                     ║
                ╚═ Press ENTER to accept, ESC to cancel ═╝
```

Figure 3-3 *Sample prompt for file name using {INPUTFILE}*

You can also pass values to {INPUTFILE} commands (or to {INPUT-TEXT} or {INPUTCAT}) in a called macro using the arguments in {CALL}. In this case, the macro gets its information directly from the macro that called it, without prompting for user input. (See {CALL} and #ARGCOUNT in this chapter.)

{INPUTTEXT}

{INPUTTEXT} causes a macro to pause and wait for the user to enter text. {INPUTTEXT} stores this text in a variable. You can test or use that variable in subsequent macro commands.

Syntax {INPUTTEXT;*prompt_message;var*}

Note In previous releases of Agenda, {INPUTTEXT} was {INPUT}.

prompt_message represents the message in the top border of the box. It can be a character string of up to 76 characters or a variable. (Any string longer than 76 characters will be truncated.)

var represents the name of the variable in which Agenda stores the text the user enters. The user can enter up to 79 characters.

If the user presses ENTER without specifying any text, or presses ESC, one of the following happens:

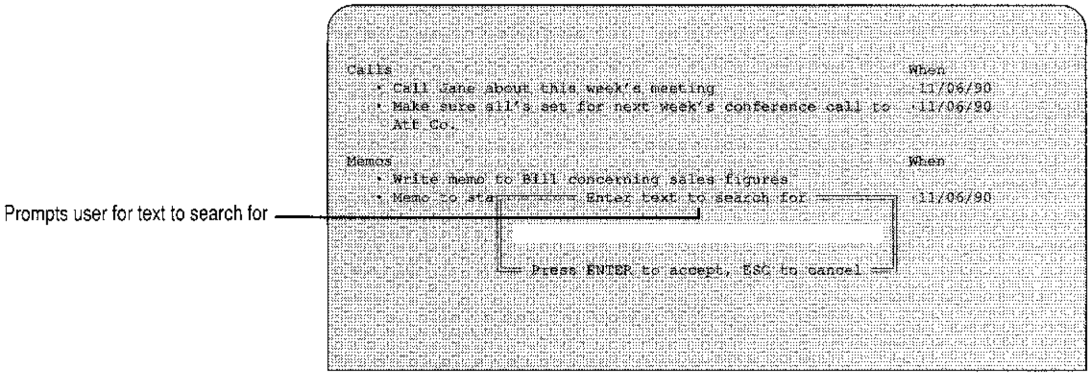
- If a string variable is used and the user leaves the highlight empty and presses ENTER, the value of *var* is set to NULL.
- If a string variable is used and the user presses ESC, the ESC character is produced.
- If a numeric variable is used and the user leaves the highlight empty or presses ESC, the value of var is set to 0 (zero).

{INPUTTEXT} displays a box with the *prompt_message* in the top border of the box and the message "Press ENTER to accept, ESC to cancel" in the bottom border of the box. This command displays a highlight in the box where the user types characters. The user can edit the string of characters before pressing ENTER.

In the following example, Agenda prompts the user for text to search for within item text in the current view (Figure 3-4). This macro assumes that the highlight is in a view.

Example

`{INPUTTEXT; Enter text to search for;%search}`	prompts user for text to search for
`{ALTF6}`	displays Search box
`{Type;%search}`	types text to search for
`{ENTER;2}`	accepts entry and starts search



```
Calls                                                        When
    • Call Jane about this week's meeting                     11/06/90
    • Make sure all's set for next week's conference call to  11/06/90
      Att Co.

Memos                                                        When
    • Write memo to Bill concerning sales figures
    • Memo to sta═══════ Enter text to search for ═══════╗    11/06/90

                 ╚═ Press ENTER to accept, ESC to cancel ═╝
```

Figure 3-4 *Sample prompt created using {INPUTTEXT}*

You can also pass values to {INPUTTEXT} commands (or to {INPUT-CAT} or {INPUTFILE}) in a called macro using the arguments in {CALL}. In this case, the macro gets its information directly from the macro that called it, without prompting for user input. (See {CALL} and #ARGCOUNT in this chapter.)

{LABEL}

{LABEL} creates a reference point in a macro. {LABEL} lets {BRANCH}, {IF}, {IFEQ}, {IFKEY}, {IFNOTEQ}, and {FOR} transfer control of macro execution to the reference point within the same macro.

Syntax {LABEL;*labelname*}

labelname represents the reference point in a macro. It can be a character string or a variable. Use the same character string or variable in the commands from which you want to transfer control as you use for *labelname*.

In the following example, {BRANCH} transfers control to a point later in the macro, that is, to {LABEL}.

Example

`{BRANCH;enter_name}`	macro execution skips to {LABEL}
`. . .`	skips commands after {BRANCH}
`{LABEL;enter_name}`	macro execution resumes here

{LARGEBOX}

{LARGEBOX} lets you display a screen of information. You might use this command to display a copyright message, informational messages, or other information to the user.

Syntax {LARGEBOX;*header;string*}

header represents the box header or title; it can be a character string of up to 76 characters or a variable. (If *header* is more than 76 characters in length, it will be truncated.)

Note If you don't want to display a header, you need to include an argument separator when you omit *header*. The semicolon (;) serves as the argument separator. If you omit the argument separator, the macro may produce unexpected results.

string represents the contents of the box; it can contain approximately 1,000 characters. If the string is large, Agenda automatically makes line breaks and scrolls, as needed. It also sizes the box appropriately and centers it on the screen. If the box is larger than one screen of characters in length, Agenda displays arrows in the upper and/or lower corner of the box. The user can use ↑, ↓, PGUP, PGDN, HOME, and END to see the rest of the screen.

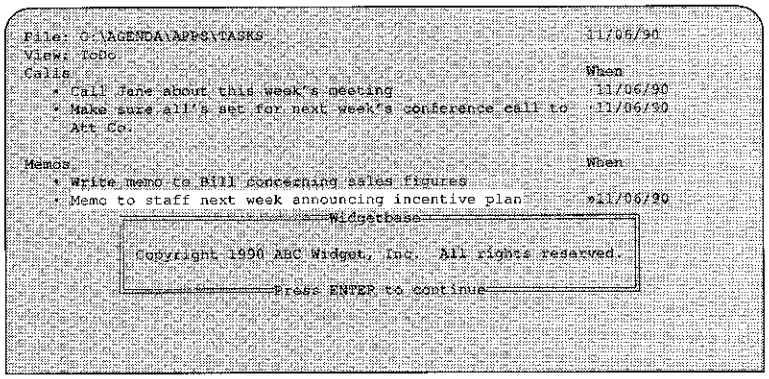
Agenda always displays the message "Press ENTER to continue" in the bottom border of the box.

Note To include semicolons (;), braces ({ }), or a percent (%) (if the % symbol is the first character in the string) in *string* as literals, you must enclose the string in quotes.

Example

`{LARGEBOX; Widgetbase; Copyright 1990 ABC Widget, Inc. All rights reserved.}`	displays a screen of information

In this example, the macro displays a copyright screen for the ABC Widget Inc. company.



```
File: C:\AGENDA\APPS\TASKS                                        11/06/90
View: ToDo
Calls                                                             When
    • Call Jane about this week's meeting                         11/06/90
    • Make sure all's set for next week's conference call to      11/06/90
      Att Co.

Memos                                                             When
    • Write memo to Bill concerning sales figures
    • Memo to staff next week announcing incentive plan           »11/06/90
              ════════════════════Widgetbase═══════════════════════
              Copyright 1990 ABC Widget, Inc.  All rights reserved.
              ══════════════Press ENTER to continue════════════════
```

Figure 3-5 *Sample screen created using {LARGEBOX}*

{LEFTSTR}

{LEFTSTR} puts into *strvar* the leftmost *n* number of characters (*width*) from *string*.

Syntax {LEFTSTR;*strvar;string;width*}

strvar represents the string variable in which the leftmost number of characters from *string* is stored. *string* represents the string from which the leftmost number of characters is taken. *width* represents the number of characters taken from *string*.

Example

`{LEFTSTR;%strvar; Press enter;5}`	puts the leftmost five characters from string into strvar
`{TYPE;%strvar}`	types the string variable

This macro produces the following output:

```
Press
```

{LENGTH}

{LENGTH} assigns the number of characters in a *string* to *numvar*.

Syntax {LENGTH;*numvar;string*}

numvar represents the number of characters in *string*. *string* represents the string, whose length is assigned to *numvar*.

Example

`{LENGTH;%numvar; Agenda}`	puts the length of string into numvar
`{TYPE;%numvar}`	types the numeric variable

This macro produces the following output:

```
6
```

{LET}

{LET} assigns a character string to a string variable or a numeric value to a numeric variable.

Syntax {LET;*var;string*} or {LET;*var;number*}

var represents the string or numeric variable to which you want to assign a numeric or string value. *string* represents the character string that you want to assign to *var*. *number* represents the numeric value that you want to assign to *var*.

The following example illustrates how {LET} handles a numeric variable. The parentheses enclosing the numeric variable ensures that Agenda evaluates the argument as an expression.

Example

`{LET;%var;(3+4)}`	sets var equal to 7
`{TYPE;%var}`	types out the contents of var

When this macro executes, {LET} gives the variable %var a value of 3 + 4 or 7. Because Agenda can perform operations on numeric variables, in this example {TYPE} produces the results of the operation 3 + 4:

```
7
```

The following example illustrates how {LET} handles a string variable.

Example

`{LET;%str;Lotus Agenda}`	sets str equal to Lotus Agenda
`{TYPE;%str}`	types out the contents of str

This macro produces the following output, because {TYPE} types out the contents of the string variable:

```
Lotus Agenda
```

Note If you want Agenda to recognize leading or trailing white space (spaces, tabs, and carriage returns) or include semicolons (;), braces ({ }), or percents (%) in string constants, you must enclose the string in quotes.

{LOTUSMENU}

{LOTUSMENU} lets you create a Lotus-style menu across the top of the screen. The user can select menu choices in either of two ways:

- Type the first letter of the desired choice
- Use ← and → to highlight a menu choice and then press ENTER on the desired choice

Syntax {LOTUSMENU;*number;choice1;string1;label1* [;*choice2;string2;label2;*]...}

number represents the number of menu choices. *choice1,choice2...* represent the actual menu choices, or commands, that appear across the top of the screen. *string1,string2...* represent the long prompts that appear as the user highlights each *choice*. *label1,label2...* represent the labels that the macro branches to if the user selects the corresponding menu *choice*.

Use the following guidelines when using {LOTUSMENU} to create a menu:

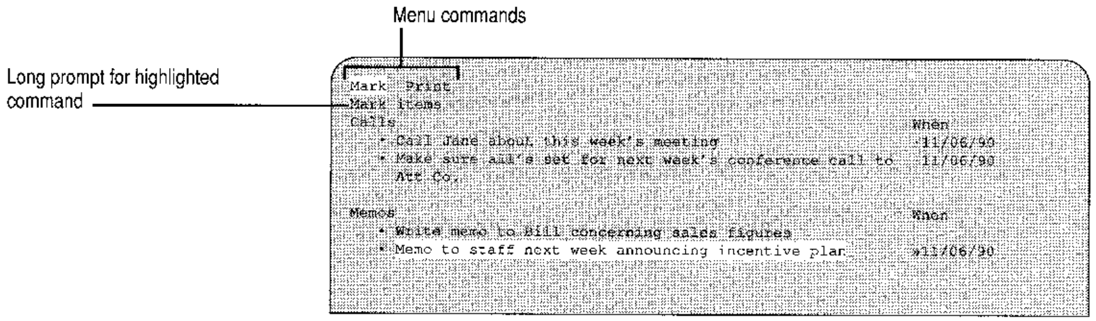
- Each menu can have up to 20 menu command choices (15 characters each). The maximum total length of all menu choices is 76.
- Begin each menu choice with a different character so that users can select each choice by typing the unique first character.
- Uppercase and lowercase letters are equivalent when making a menu selection; for example, the user can select Two by typing `t` or `T`.
- If more than one choice starts with the same letter, when the user types that letter Agenda selects the first choice, reading from left to right, whose first character matches the character that was typed.
- {LOTUSMENU} lets you display long prompts describing each highlighted menu choice. If you have a color monitor, Agenda displays by default long prompts in yellow on blue, menu choices in white on blue, and highlighted menu choices in white on red. (These are the default colors for the **Color** choice for the **Color** setting that you specify using the **Utilities Customize** command. To change these colors, see Chapter 1 in *Setting Up Agenda*.)

If you want to display other menu choices (rather than a long prompt) for a highlighted menu choice, precede *string* with a dollar sign ($). This lets you display these submenu choices in the same color as other menu choices. For an example of how Agenda displays menu choices versus long prompts with a color monitor, move the highlight across the menu choices in the File menu.

This following macro produces a Lotus-style menu with two menu command choices.

Example

{LOTUSMENU;2;Mark; Mark items; labelone;Print; Print marked items; labeltwo}	indicates the choices and prompts that appear in the menu when this macro executes
{QUIT}	if the user presses ESC, ends macro execution
{LABEL;labelone} ... **{QUIT}**	if the user selects Mark, executes the macro commands at labelone, and ends macro execution
{LABEL;labeltwo} ... **{QUIT}**	if the user selects Print, executes the macro commands at labeltwo, and ends macro execution



```
Mark  Print
Mark items
Calls                                                       When
   • Call Jane about this week's meeting                    11/06/90
   • Make sure all's set for next week's conference call to 11/06/90
     Att Co.

Memos                                                       When
   • Write memo to Bill concerning sales figures
   • Memo to staff next week announcing incentive plan      »11/06/90
```

Figure 3-6 *Sample Lotus-style menu created using {LOTUSMENU}*

Macro processing branches to a different point, depending on what the user does:

- If the user selects the first menu choice (Mark), macro processing branches to {LABEL;labelone}.
- If the user selects the second menu choice (Print), macro processing branches to {LABEL;labeltwo}.

Note If the user presses ESC, or if {LOTUSMENU} references labels that don't exist, the macro continues executing at the next macro command after {LOTUSMENU}.

{MIDSTR}

{MIDSTR} puts the number of characters specified in *width*, starting with the character identified in *position*, from *string* into *strvar*. For example, {MIDSTR;strvar;abc;2;1} results in strvar = b.

Syntax {MIDSTR;*strvar;string;position;width*}

strvar represents the string variable that stores the number of characters. *string* represents the string from which the number of characters is taken. *position* is a number that represents the point in *string* from which characters are taken. *width* represents the number of characters, starting with *position*, that are taken from *string* and put into *strvar*.

Example

`{LET;%string;Personal Information Management}`	sets string equal to the string "Personal Information Management"
`{MIDSTR;%strvar; %string;10;11}`	beginning with the tenth character, puts eleven characters into strvar
`{TYPE;%strvar}`	types the string variable

This macro produces the following output:

```
Information
```

{ONBREAK}

{ONBREAK} lets you transfer control of macro execution to another macro if the user presses CTRL-BREAK during macro execution.

Syntax {ONBREAK;*macroname*} or {ONBREAK}

macroname represents the name of the break handler macro.

A break handler macro runs when the user presses CTRL-BREAK during macro execution. Macro processing is returned to the executing macro (to the point at which the user pressed CTRL-BREAK) by executing {RETURN}. If the break handler macro does not include {QUIT} or {RETURN}, macro processing automatically returns to the executing macro when the break handler macro finishes executing. If there is no break handler macro, pressing CTRL-BREAK causes a macro to terminate execution.

The ability to maintain macro execution, even if the user presses CTRL-BREAK, is very useful for application developers who are providing turnkey Agenda applications.

Note You can reference only one break handler macro at a time. If you reference another macro using {ONBREAK} during macro execution, Agenda uses the most recently executed {ONBREAK} command in the macro. You should put the {ONBREAK} command at the beginning of a macro.

If a macro that includes an {ONBREAK} command calls one or more macros, the {ONBREAK} command remains in effect for all called macros unless a new {ONBREAK} command is executed. Therefore, if the user presses CTRL-BREAK during macro execution, Agenda transfers control to the break handler macro, regardless of when the user pressed CTRL-BREAK.

{ONBREAK} used without the *macroname* argument removes the current break handler macro.

The following example shows two macros, the second of which serves as a break handler for the first. In this example, a break handler macro (Macro 2) is provided because Macro 1 takes a long time to complete and there is a chance that the user may inadvertently press CTRL-BREAK during macro execution.

Example
Macro 1 (The executing macro)

`{ONBREAK;`
`  Break Handler}`
	establishes Break Handler as the break handler macro (Macro 2)

`{WINDOWSOFF;`
`  Long Macro;`
`  Press Ctrl-Break to`
`  Abort}`
	freezes the display and puts up a "Please wait" box

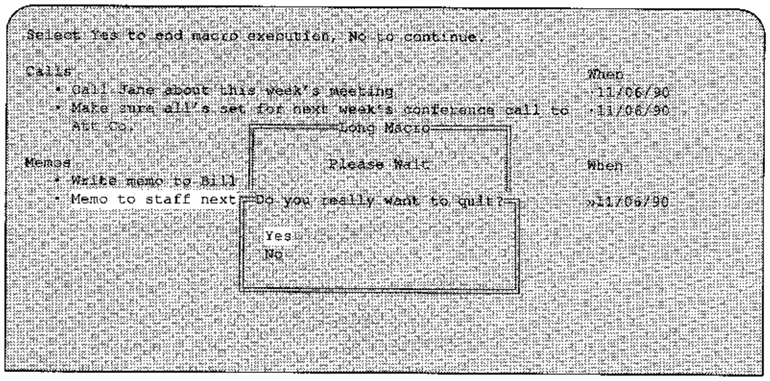
`. . .`	the body of the long macro
`{WINDOWSON}`	restores the display
`{ONBREAK}`	removes the break handler

Macro 2 (The break handler macro)

`{Break Handler}`	name of break handler macro
`{ROLLMENU; Do you really want to quit?;Select Yes to end macro execution, No to continue.;2;Yes; Yeslabel;No; Nolabel}`	asks if the user wants to quit
`{LABEL; Nolabel}` `{RETURN}`	if the user selects N or ESC, macro processing resumes where it left off
`{LABEL; Yeslabel}` `{QUIT}`	if the user selects Y, all macro processing terminates

If the user presses CTRL-BREAK during macro execution, the break handler macro displays (in a simple roll menu) a message confirming whether the user really intended to quit (Figure 3-7). If the user types N or presses ESC to indicate that the user did not intend to quit, the break handler macro allows the original macro (Macro 1) to continue executing.

In contrast, if {ONBREAK} wasn't present and the user pressed CTRL-BREAK when the macro was processing, macro execution would terminate.



```
Select Yes to end macro execution, No to continue.

Calls                                                        When
   • Call Jane about this week's meeting                     ·11/06/90
   • Make sure all's set for next week's conference call to  ·11/06/90
     Att Co.               ═══════Long Macro═══════

Memos                            Please Wait                 When
   • Write memo to Bill
   • Memo to staff next ═Do you really want to quit?═        »11/06/90

                          Yes
                          No
```

Figure 3-7 *Sample message created using {ONBREAK}*

{ONERROR}

{ONERROR} lets you transfer control of macro execution to another macro if specific errors occur during macro execution.

Syntax {ONERROR;*macroname*} or {ONERROR}

macroname represents the name of the error handler macro.

An error handler macro is one that is called whenever specific errors occur in Agenda during macro execution. Macro processing is returned to the executing macro (to the point at which the error occurred) by executing {RETURN}. If there is no error handler macro and an error occurs in Agenda during macro execution, macro processing is terminated.

If appropriate macro instructions are provided, macros can handle many errors that occur in Agenda during macro execution. For example, you might have a macro that prints marked items, prompting the user to mark items to print. If the user does not mark items prior to printing, an error occurs. If there was no error handler macro, this situation would cause macro execution to terminate. You can, however, provide an error handler macro to handle this type of error. (For a list of Agenda errors that macros can handle, see Appendix D in the *User's Guide.*)

Note Macros cannot handle all errors. For example, macros cannot handle DOS errors such as "Disk is full."

{ONERROR} works in conjunction with the special variables #ERRNO and #ERRMSG. When an error in Agenda occurs during macro execution and an error handler macro is present, the error handler macro checks to see if the error that occurred is one that it can handle. (Errors are identified by #ERRNO, a number associated with each Agenda error.) If the macro can handle the error (it can undo or safely ignore the error), the error handler macro can execute {RETURN} to continue macro processing at the point at which the error occurred.

If {ONERROR} does not recognize and thus, cannot handle the error, macro processing terminates. If the error handler macro cannot handle the error, the error handler macro should execute either:

- {QUIT}, which terminates macro processing
- {QUIT;ERROR} which terminates macro processing and displays the error message associated with the error that occurred

(See {QUIT}, #ERRNO, and #ERRMSG, later in this chapter. See also Appendix D in the *User's Guide* for descriptions of Agenda error messages that macros can handle and the number (#ERRNO) associated with each.)

The ability to maintain macro execution, even if an error occurs in Agenda during macro execution, is very useful for application developers who are providing turnkey Agenda applications.

Note You can reference only one error handler macro at a time. If you reference another macro using {ONERROR}, Agenda uses the most recently executed {ONERROR} command in the macro. You should use {ONERROR} at the beginning of a macro.

If a macro that includes an {ONERROR} command calls one or more macros, the {ONERROR} command remains in effect for all the called macros unless a new {ONERROR} command is executed. Therefore, if an error occurs during macro execution, Agenda transfers control to the error handler macro, regardless of when the error occurred.

{ONERROR} used without the *macroname* argument removes the current error handler macro.

The following example shows how an error handler macro works. In this example, the purpose of Macro 1 is to make modifications to all notes attached to categories in a file. The modifications themselves are indicated by ellipses (...) in Macro 1. (It is assumed that this macro is being run from the category manager.)

An error might occur during the execution of Macro 1: A note might be attached as an external note file that is greater than 10,000 characters. Notes greater than 10,000 characters can only be read; they cannot be modified.

In this case, modifications cannot be made to such notes. This means that as soon as one of these notes is displayed, an error occurs. Rather than displaying the message that Agenda usually displays for this error, and terminating macro execution, Agenda calls the error handler macro (Macro 2). The error handler macro sets a flag (called %%SkipThisNote) in Macro 2. This flag tells Macro 1 to skip making the modification to this note and continue execution.

Example

Macro 1 (The executing macro)

`{ONERROR; Error Handler}`	establishes Error Handler as the error handler macro
`{DEFINT; %%SkipThis-Note}`	defines the error flag variable
`{CLEAR; %%SkipThis-Note}`	ensures the error flag is reset
`MAIN{ENTER}`	moves the highlight to the category Main
`{F5}`	goes to the note attached to Main
`{LABEL; Loop}`	
`{IF; (%%SkipThisNote= #TRUE);Skipit}`	if the error handler has set the error flag to TRUE, skips the note
`. . .`	the body of the macro
`{LABEL; Skipit}`	
`{Clear; %%SkipThis-Note}`	resets the error flag before moving to the next note
`{AltN}`	moves to the next note
`{IF; (#mode <> Dialog); NoChangesMade}`	if changes were made, Agenda asks "Discard changes?"
`N`	enters "N" for no
`{LABEL; NoChangesMade}`	
`{IF; (#mode = Note); Loop}`	in new note, loops to see if note is read-only
`{ONERROR}`	removes the error handler

Macro 2 (The error handler macro)

`{Error Handler}`	the name of the error handler macro
`{IF; (#ERRNO = 6200); ReadOnlyNote}`	6200 = Notes this long can only be viewed (read-only)
`{QUIT;ERROR}`	if an error occurs that the macro can't handle, displays the error message and ends the macro
`{LABEL; ReadOnlyNote}`	
`{LET; %%SkipThisNote; #TRUE}`	sets the error flag to TRUE which tells other macro not to modify this note
`{RETURN}`	returns to the previous macro at the point at which the error occurred

{QUIT}

{QUIT} terminates macro execution.

Syntax {QUIT} or {QUIT;ERROR}

If a macro that is running is called by another macro (via {CALL}), {QUIT} does not return to the caller macro; it terminates all macro processing.

If {QUIT} executes (by pressing a key which has an attached macro) while Learn mode is on, Learn mode is turned off. To continue, turn Learn mode on again and append to the macro. (See Chapter 1.)

{QUIT} and {QUIT;ERROR} also work in conjunction with an error handler macro. If an error occurs during macro execution and an error handler macro is present, the error handler macro checks to see if the error that occurred is one that it can handle. For those cases when it cannot handle the error, include {QUIT} or {QUIT;ERROR} in the error handler macro to execute either {QUIT} (to terminate macro execution) or {QUIT;ERROR} (to terminate macro execution and display the error message associated with that error). (See {ONERROR} earlier in this chapter.)

{RETURN}

{RETURN} returns control of macro execution to the macro from which it was called (by {CALL}).

Syntax {RETURN}

{RETURN} also returns control to the macro from which the current macro was called by either {ONBREAK} or {ONERROR}. (If the called macro is a break handler macro, it would be called if the user pressed CTRL-BREAK during macro execution. If called macro was an error handler macro, it would be called if an error occurred during macro execution.)

Macro processing resumes at the next command to be executed, which is typically the command after {CALL} (or the point at which CTRL-BREAK was pressed or an error occurred).

If a macro does not include {RETURN} (or {QUIT}) and macro execution continues until the end of the macro, at that point Agenda assumes there's a {RETURN} and returns control of macro execution to the macro that called it.

{RIGHTSTR}

{RIGHTSTR} puts into *strvar* the rightmost number of characters from string.

Syntax {RIGHTSTR;*strvar;string;width*}

strvar represents the string variable in which the rightmost number of characters from *string* is stored. *string* represents the string from which the rightmost number of characters is taken. *width* represents the number of characters taken from *string*.

Example

`{RIGHTSTR;%strvar;`
`  Initial View;4}`
	puts the rightmost four characters from string into strvar

`{TYPE;%strvar}`	types the string variable

This macro produces the following output:

```
View
```

{ROLLMENU}

{ROLLMENU} lets you create a box with choices in a vertical list.

This box lets users select a choice in either of two ways:

- Use ↑ and ↓ to highlight a choice and then press ENTER on the desired choice.
- Use Agenda's automatic completion feature by typing the name of the desired choice and then pressing ENTER when the highlight is on that choice.

For more information on using boxes in Agenda and automatic completion, see Chapter 2 in the *User's Guide*.

Syntax {ROLLMENU;*header;long prompt;number;* *choice1;label1*[*;choice2;label2;*]...}

header represents the text that appears at the top of the box and identifies the box; it can be up to 76 characters long. *long prompt* represents the text that appears at the top of the screen; *long prompt* can be up to two lines of 70 characters each. *number* represents the total number of choices in the box; *number* can be up to 40 (the maximum number of choices allowed in each box). *choice1,choice2...* represent the actual choices in the box. *label1,label2...* represent the labelnames that the macro branches to if the user selects the corresponding menu choice.

Note If you omit the *header* or *prompt* arguments, you need to enter an argument separator for each argument you omit. The semicolon (;) serves as the argument separator. If you omit the argument separators, the macro may produce unexpected results.

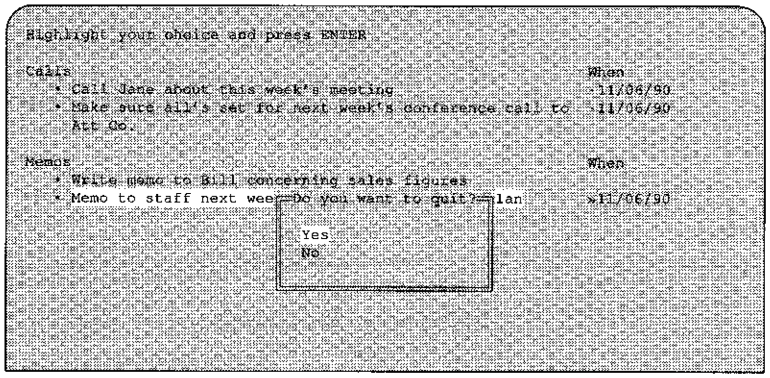
Use the following guidelines when using {ROLLMENU} to create a box with choices:

- You can include up to 40 choices for each box. If the box exceeds the length of the screen, the user can use the arrow keys to scroll to see the choices.
- Uppercase and lowercase letters are equivalent when selecting a choice using automatic completion. The user can select No by typing n or N.
- If the user presses ESC, or if {ROLLMENU} references labels that don't exist, the macro continues executing at the next macro command after {ROLLMENU}.
- To specify a two-line long prompt, include a vertical bar in *long prompt* at the point at which you want the text to break. Each line can be up to 70 characters in length.

The following macro creates a box with two choices.

Example

`{ROLLMENU;Do you want to quit?;Highlight your choice and press ENTER;2; Yes;labelone; No;labeltwo}`	indicates the choices that appear in the box when the macro executes
`{QUIT}`	if the user presses ESC, ends the macro
`{LABEL;labelone}` `...` **`{QUIT}`**	if the user selects Yes, executes the macro commands after labelone and ends macro execution
`{LABEL;labeltwo}` `...` **`{QUIT}`**	if the user chooses No, executes the macro commands after labeltwo and ends the macro



```
Highlight your choice and press ENTER

Calls                                                  When
   • Call Jane about this week's meeting               11/06/90
   • Make sure all's set for next week's conference call to  11/06/90
     Att Co.

Memos                                                  When
   • Write memo to Bill concerning sales figures
   • Memo to staff next wee╔Do you want to quit?╗lan   »11/06/90
                           ║                    ║
                           ║ Yes                ║
                           ║ No                 ║
                           ║                    ║
                           ╚════════════════════╝
```

Figure 3-8 *Sample box created using {ROLLMENU}*

Macro processing branches to a different point, depending on what the user does.

- If the user selects the first choice (Yes), macro processing branches to {LABEL;labelone}.
- If the user selects the second choice (No), macro processing branches to {LABEL;labeltwo}.

{SPEED}

{SPEED} slows down macro execution to one of three speeds that are slower than normal keystroke execution. You might use {SPEED}, for example, if you've developed a training application and want to have the file execute slowly so that users can see the keystrokes execute.

Syntax {SPEED;*argument*}

argument represents the speed of macro execution. It can be SLOW, MED, FAST, or MAX. SLOW, MED, and FAST are all slower than normal playback speed. MAX returns playback to normal (MAX) speed.

Macro execution always starts at normal (MAX) speed. When a macro finishes executing, Agenda macro execution automatically returns to normal speed.

For all speeds *except* MAX, the speed with which a macro executes is based on the system clock (not the processing speed of your computer). This means that if, for example, a macro is running at {SPEED;MED} on both an IBM® XT™ and a Compaq® 386 computer, this macro runs at the same speed on both computers.

{TYPE}

{TYPE} types the contents of a variable as if the user had typed it on the keyboard.

Syntax {TYPE;*parameter*}

parameter represents the constant, expression, variable, or special variable whose value Agenda types.

In the following example, {LET} defines the variable %a; {TYPE} types out the contents of this variable on the screen.

Example

`{LET;%a;MEMO}`	defines the variable %a
`{TYPE;%a}`	displays "MEMO" on the screen as if the user typed it

{UNDEF}

{UNDEF} undefines some (or all) numeric and/or string variables.

Syntax {UNDEF;*var1*[;*var2*];...} or {UNDEF;ALL}

var represents the name of the existing string or numeric variable that you want to undefine. You can use {UNDEF;*var*} to undefine as many string and/or numeric variables as you want.

{UNDEF;ALL} undefines the global variables in a file and all local variables in the current macro at once.

Typically, you use this command to undefine a global variable so that you can use a global variable with the same name, but a different value, for another purpose. You also use this command to undefine a variable that you no longer plan to use so that your file is not filled with unused global variables.

Note Although you can also use this command to undefine a local variable, it is not necessary to do so because Agenda automatically undefines local variables in a macro when the macro completes processing.

Whereas {CLEAR} leaves the variables that it clears in the file, {UNDEF} leaves no trace of the variables that it undefines. Thus, they do not retain their status as being either string or numeric variables. (See {CLEAR} earlier in this chapter.)

{WINDOWSOFF}

{WINDOWSOFF} freezes the screen display while a macro is running, optionally displaying a "Please wait" box. You use this command in conjunction with {WINDOWSON} and/or {WINDOWSUPD}. ({WINDOWSON} unfreezes a screen that was previously frozen by {WINDOWSOFF}; {WINDOWSUPD} updates a screen that was previously frozen by {WINDOWSOFF}.) (See {WINDOWSON} and {WINDOWSUPD} later in this chapter).

Syntax {WINDOWSOFF[;*header;footer*]} *header* represents the text in the top border of the "Please wait" box. *footer* represents the text in the bottom border of the "Please wait" box. *header* and *footer* can each be 76 characters in length. The default *footer* is "Macro running."

Without any arguments, {WINDOWSOFF} does not display a "Please wait" box but it does freeze the entire screen display until the macro encounters {WINDOWSON} or {WINDOWSUPD}.

If *header* or *footer* is specified, the screen freezes and Agenda displays the specified header and footer in a box with the message "Please wait." {WINDOWSUPD} does not refresh the screen if a "Please wait" box is displayed.

Macro commands that display messages for the user still appear when the screen is frozen.

Notes Both *header* and *footer* are optional arguments. If you include only one of these arguments, you need to enter an argument separator for the argument you omit. The semicolon (;) serves as the argument separator. If you omit the argument separator, the macro may produce unexpected results. You do not need to include argument separators if you omit *both* the *header* and *footer* arguments.

If you include {DEBUGON} in a macro, {WINDOWSOFF} is ignored. If {DEBUGON} is executed while windows are off, windows are turned on. (See {DEBUGON} and {DEBUGOFF} earlier in this chapter.)

In the following example, {WINDOWSOFF} freezes the screen and displays the message "Please wait" with the header "Section being added." As the new section head and items are added to the view, the changes are not shown until {WINDOWSON} is executed and the screen is redisplayed.

Example

`{WINDOWSOFF;Section being added;}`	freezes the window display and displays a "Please wait" box with the header "Section being added" and the default footer "Macro running"
`{AltD}`	
`New{ENTER}`	inserts the new section head, New, below the current section, but does not display it at this time
`item1{ENTER}`	adds five items to the section
`item2{ENTER}`	
`item3{ENTER}`	
`item4{ENTER}`	
`item5{ENTER}`	
`{WINDOWSON}`	unfreezes and updates the screen

{WINDOWSON}

{WINDOWSON} unfreezes a screen that has been previously frozen by {WINDOWSOFF}. If a "Please wait" box displays while the screen was frozen, {WINDOWSON} clears the box and updates the display to reflect any changes that have been made to the information. (See {WINDOWSOFF} earlier in this chapter.)

Syntax {WINDOWSON}

{WINDOWSUPD}

{WINDOWSUPD} updates the screen display while a macro is executing and then keeps the screen frozen with the new display. This command cannot be used if the {WINDOWSOFF} command that originally froze the screen includes a "Please wait" box. (See {WINDOWSOFF} and {WINDOWSON} earlier in this chapter.)

Syntax {WINDOWSUPD}

Using {WINDOWSUPD} in the following macro allows the user to see the changes being made when {WINDOWSUPD} is executed.

Example

`{WINDOWSOFF}`	freezes the window display
`{AltD}`	
`New{ENTER}`	inserts the new section head, New, below the current section but does not display it at this time
`{WINDOWSUPD}`	updates the display, showing the new section head and then freezes the display again
`item1{ENTER}`	adds five items to the section, but does not display them
`item2{ENTER}`	
`item3{ENTER}`	
`item4{ENTER}`	
`item5{ENTER}`	
`{WINDOWSON}`	unfreezes the screen and displays the updated information (the new items)

Highlight-control Commands

Highlight-control commands move the highlight the number of times that you specify in the *number* argument. Where you are in Agenda determines where the highlight moves. For example, if you are in a note, {DOWN;4} moves the highlight down four lines. If you omit the *number* argument, the highlight moves once. Highlight-control commands work for all highlight-movement keys, such as PGUP, DOWN, and CTRL-PGDN.

If you move the highlight somewhere that it is not allowed, Agenda beeps. For example, if you are in the category manager and you write a macro that includes the {LEFT} command, Agenda beeps because you cannot move the highlight to the left in the category manager.

The following list identifies the Agenda highlight-control commands. These commands repeat the given keystroke the number of times specified in *number*.

Note The *number* argument must be a numeric constant. Variables are not valid arguments for highlight-control commands.

Syntax

- {BS[;*number*]}
- {CTLPGUP[;*number*]}
- {CTLPGDN[;*number*]}
- {CTLEND[;*number*]}
- {CTLHOME[;*number*]}
- {CTLLEFT[;*number*]}
- {CTLRIGHT[;*number*]}
- {DEL[;*number*]}
- {DOWN[;*number*]}
- {END[;*number*]}
- {ENTER[;*number*]}
- {ESC[;*number*]}
- {HOME[;*number*]}
- {INS[;*number*]}
- {LEFT[;*number*]}
- {PGUP[;*number*]}
- {PGDN[;*number*]}
- {RIGHT[;*number*]}
- {SHFTAB[;*number*]}
- {TAB[;*number*]}
- {UP[;*number*]}

{SELECTION}

In a box that contains settings, {SELECTION} moves the highlight directly to the setting whose prompt is *string*.

Syntax {SELECTION;*string*}

string represents the setting in the box to which the highlight is moved. (If the setting is hidden, or does not match, the command is ignored and the macro may produce unexpected results.) You can use either uppercase or lowercase when specifying *string*. When the highlight is on a setting that {SELECTION} moved to, *string* is the string that #PROMPT returns. (See #PROMPT later in this chapter.)

After you use {SELECTION} to move to a particular setting, you can also include macro instructions that complete the setting. If the setting is one with choices, you can include macro instructions that:

- Use F3 (CHOICES) to display the choices for the setting
- Type out the first letter of a choice if the first letters of the choices are unique.

{SELECTION} is a very useful command because some selections you make in Agenda boxes affect whether or not Agenda displays additional settings in these boxes. In other words, you cannot specify a certain number of keystrokes and be certain that Agenda will place the highlight on the setting that you want.

Because Agenda displays some settings based on choices made for other settings, you may want to confirm where the highlight is before you provide macro instructions that act on that setting. You use #PROMPT to confirm where the highlight is.

For example, you might create a macro that includes the instruction {SELECTION;File already exists:}R. In the Export Structured File box, if the file that is specified already exists, Agenda displays the **File already exists** setting. This macro selects **R** to replace the specified file.

If the file that is specified does not already exist, however, the instruction {SELECTION;File already exists:} does not move to this setting and typing **R** will produce unexpected results.

To ensure that the highlight is on the **File already exists** setting before the macro types **R**, you can include the command {IF;(#PROMPT = "File already exists:");replace}.

If you use #PROMPT to confirm where the highlight is in the Print Layout and/or Header and Footer boxes, the value of #PROMPT is not simply the text of the current setting. For more information, see #PROMPT later in this chapter.

In addition, there are some settings in Agenda that have the same names (for example, there are two **Sort on** settings in the Item Sorting in All Sections box, one under **Primary sort key**, and one under **Secondary sort key**). For these settings, the value of #PROMPT is typically the heading under which the setting displays and the name of the setting. Thus, before using {SELECTION} to move the highlight to one of these settings, you should first verify the value of #PROMPT and then specify that value in {SELECTION}.

Note In order for {SELECTION} to work, #MODE must equal DIALOG. (See #MODE later in this chapter.)

The following example shows how you can use {SELECTION}. The macro in this example uses {SELECTION} to move the highlight to the appropriate settings and select choices to print the items in a view.

Example

`{F10}PF`	displays the Final Print box
`{SELECTION;Print:}`	moves the highlight to the **Print** setting and
`   {F3}View`	selects to print the view
`   {ENTER}`	
`{SELECTION;Include:}`	moves the highlight to the **Include** setting
`   {F3}Item`	and selects to print items in that view
`   {ENTER}`	
`{SELECTION;Print to:}`	moves the highlight to the **Print to** setting
`   {F3}Printer`	and selects a printer
`   {ENTER}`	
`{ENTER}`	accepts the choices in this box and begins printing

Special Variables

Special variables are variables that return a specific value, describing the current state of the Agenda file. For example, if a user is in a note, the value of the special variable #MODE is NOTE. Special variables cannot be given any other value by the user. You can include special variables in macro commands.

#ARGCOUNT

The value of #ARGCOUNT contains the number of arguments passed by {CALL} that are left to be matched by the {INPUTCAT}, {INPUTFILE}, or {INPUTTEXT} commands in a called macro. (See {INPUTCAT}, {INPUTFILE}, {INPUTTEXT}, and {CALL} earlier in this chapter.)

For example, you can include #ARGCOUNT in a called macro to control how many times a loop executes (because #ARGCOUNT equals the number of arguments in {CALL} left to be matched to the {INPUTCAT}, {INPUTFILE}, or {INPUTTEXT} commands.)

The following example shows how you can use #ARGCOUNT to control the number of times a loop executes. In this example, Macro 1 calls Macro 2.

Example
Macro 1 (The caller macro)

**{CALL;Add names;
Helen Voss;
Floyd Hastings;
Norbert Thomas}**
	calls Macro 2, Adds names, and passes the names to the variable %name in Macro 2

Macro 2 (The called macro, Add names)

{Add names}	the called macro
{LABEL;Test #ARGCOUNT}	

**{IF;(#ARGCOUNT <> 0);
Enter name}**
	tests the value of #ARGCOUNT to determine if there are any names to be passed by Macro 1

{RETURN}	if the value of #ARGCOUNT=0, control of macro execution returns to Macro 1
{LABEL;Enter name}	
{INPUTTEXT;Please enter employee name;%name}	assigns the names used as parameters in Macro 1 to the variable %name
{TYPE;%name}{ENTER}	types the contents of the variable %name

**{BRANCH;
Test #ARGCOUNT}**
	branches back to the label test #ARGCOUNT to determine if there are any more parameters to pass from Macro 1

{LABEL;Continue}	
...	

When Macro 1 (the caller macro) first calls Macro 2, (the called macro), #ARGCOUNT=3 (the number of arguments in the caller macro). Since #ARGCOUNT does not equal 0, the macro executes the loop (Enter name) and the first argument (Helen Voss) in {CALL} "pairs up" with the first {INPUTTEXT} command in Macro 2, {INPUTTEXT; Please enter employee name;%name}.

After it completes the first loop, #ARGCOUNT=2 (the number of arguments in {CALL} left to be executed). Processing continues until there are no arguments in {CALL} left to be processed. At this point, #ARGCOUNT=0 and {RETURN} is executed, passing control back to the first macro.

#ASCII(nnn)

The value of #ASCII(*nnn*), where *n* is a decimal number, results in the character with the ASCII value *nnn*. For example, the value of #ASCII(027) equals ESC.

#CLIPBOARD

The value of #CLIPBOARD is a string that contains the current contents of the clipboard. (When a user is editing an Agenda file and cutting or copying text, Agenda places the cut or copied text in the Clipboard. The user can then paste this text elsewhere in the current file or in another Agenda file during the same session.) If the Clipboard contains more than 1,000 characters, the special variable #CLIPBOARD contains only the first 1,000 characters.

Note If the Clipboard contains text with markers, the value of #CLIPBOARD used in a command (such as {TYPE}), will *not* contain any markers.

#DATE

The value of #DATE is a string containing the current system date in the default date format, which you set using the **File Properties** command.

#DEPTH

The value of #DEPTH is the depth or level of the highlighted category in the category manager. #DEPTH has a value only when #MODE=CATMGR. #DEPTH of the category MAIN equals 0. (See #MODE later in this chapter.)

#ERRMSG

The value of #ERRMSG is a string containing the error message of the last error that occurred. #ERRMSG corresponds to the value of #ERRNO, the number of this error message. If you create a macro that handles errors, you can use #ERRMSG to use the text of the error message of the error that occurred.

(See {ONERROR} and {QUIT} earlier in this chapter. For a list of Agenda error messages and their associated numbers, see also Appendix D in the *User's Guide*.)

#ERRNO

The value of #ERRNO is the number of the last error that occurred. (The text of the message is in #ERRMSG, as described earlier in this chapter.)

If you create a macro to handle errors that occur while other macros execute, you can use #ERRNO within the error handler macro so that you can include instructions in the macro to handle the error. (See

{ONERROR} earlier in this chapter.) For a list of Agenda error messages and their associated numbers, see Appendix D in the *User's Guide*.

#FALSE

The value of #FALSE is 0 (zero). You can use #FALSE to test #KEYHIT with the {IF} command. (See {IFKEY}, {IF}, and #KEYHIT in this chapter.)

#FILENAME

The value of #FILENAME is the name of the current Agenda file, including the extension, without the path.

#FILEPATH

The value of #FILEPATH is the path of the current file, without the filename.

#HIGHLIGHT_ TYPE

The value of #HIGHLIGHT_TYPE is the highlighted element; for example, an item, column head, and so forth. (For a complete list of #HIGHLIGHT_TYPE values, see #MODE later in this chapter.)

#HIGHLIGHT_ VALUE

The value of #HIGHLIGHT_VALUE is the highlighted text; for example, the text of an item, column head, and so forth. #HIGHLIGHT_VALUE has no value if the value of #MODE=OTHER. (For a complete list of #HIGHLIGHT_VALUE values, see #MODE later in this chapter.)

#KEYHIT

The value of #KEYHIT returns either one of the following numbers: 1 (true) or 0 (false). The command {IF;(#KEYHIT=#TRUE);loop} is the same as {IFKEY;loop}. (See {IFKEY} and {IF} earlier in this chapter.)

The following example shows a macro that types the contents of the variable %a continually, until the user presses a key.

Example

```
{LABEL;LOOP}
{LET;%a;a}
{TYPE;%a}
{IF;(#KEYHIT=#FALSE);      if user doesn't press a key, keeps looping
   LOOP}
{QUIT}
```

#MARK_COUNT, #MARKED_IN_ VIEW

The values of #MARK_COUNT and #MARKED_IN_VIEW are the number of marked items in the current file and the number of marked items in the current view, respectively.

Tip You can use this special variable together with the ALT-J (jump to marked) accelerator key to write macros that can perform complex operations on all marked items.

#MODE

The value of #MODE is a string that identifies what part of Agenda you are in. Values for #MODE can be any of the following:

- View
- Catmgr
- Note
- Macroedit
- Itemedit
- Catedit
- Fieldedit
- Dialog
- Select
- Menu
- Other

The commands in the following macro let the macro continue executing only if the user is in a view.

Example

`{IF;(#MODE=View);continue}`	tests the contents of the special variable #MODE
`{QUIT}`	terminates all macro processing if #MODE is not equal to VIEW
`{LABEL;continue}`	if the user is in a view, continues macro processing
...	

Note You can test a value for #MODE using either uppercase or lowercase.

The special variables #MODE, #HIGHLIGHT_TYPE, #HIGHLIGHT_VALUE, #PROMPT, and #DEPTH are related to one another.

- #MODE always has a value.
- #HIGHLIGHT_TYPE, #HIGHLIGHT_VALUE, #PROMPT, and #DEPTH all depend on the value of #MODE for their respective values.

The value of #HIGHLIGHT_TYPE is the highlighted element; for example, an item, column, and so forth.

The value of #HIGHLIGHT_VALUE is the highlighted text; for example, the text of an item, column head, and so forth. This special variable has no value if the value of #MODE=OTHER.

The value of #PROMPT is the setting that the user is currently responding to in a box with settings. This special variable has a value only when the value of #MODE=DIALOG. (See #MODE=DIALOG later in this chapter.)

The value of #DEPTH is the depth or level of the highlighted category in the category hierarchy. This special variable has a value only if #MODE=CATMGR. (See #DEPTH earlier in this chapter and #MODE=CATMGR later in this chapter.)

The following sections identify the values of #HIGHLIGHT_TYPE, #HIGHLIGHT_VALUE, #PROMPT, and #DEPTH given the values of #MODE.

MODE=VIEW

When #MODE=VIEW, the highlight is in a view. #PROMPT and #DEPTH have no values.

If the highlight is on	*#HIGHLIGHT_ TYPE*	*#HIGHLIGHT_VALUE*
Item	Item	Text of the item
Calculation label	Label	Text of the calculation label
Section head	Section	Text of the section head category name

continued

If the highlight is on	*#HIGHLIGHT_TYPE*	*#HIGHLIGHT_VALUE*
Column head	Column	Text of the column head category name
Numeric column entry	Numeric	Text of the Numeric column entry
Date column entry	Date	Text of the Date column entry
Unindexed category column entry	Unindexed	Text of the Unindexed column entry
Standard category column entry	Standard	Text of the Standard column entry

#MODE=CATMGR

When #MODE=CATMGR, the highlight is in the category manager on a category. In this case, #PROMPT has no value. #DEPTH is the depth or level of the highlighted category in the category hierarchy. (See #DEPTH earlier in this chapter.) #HIGHLIGHT_TYPE is always the type of category that the highlight is on (standard, numeric, date, or unindexed).

If the highlight is on	*#HIGHLIGHT_TYPE*	*#HIGHLIGHT_VALUE*
Numeric category	Numeric	Text of the numeric category name
Date category	Date	Text of the date category name
Unindexed category	Unindexed	Text of the unindexed category name
Standard category	Standard	Text of the standard category name

- **#MODE=NOTE**
- **#MODE=MACROEDIT**
- **#MODE=ITEMEDIT**
- **#MODE=CATEDIT**
- **#MODE=FIELDEDIT**

When #MODE=NOTE, MACROEDIT, ITEMEDIT, CATEDIT, or FIELDEDIT the highlight is in a note, macro, item, category, or text setting (such as the **File description** setting), respectively. In any of these cases, #PROMPT and #DEPTH have no values.

If the highlight is on	*#HIGHLIGHT_ TYPE*	*#HIGHLIGHT_VALUE*
A single character (if there is no marked text and the cursor is not on a marker)	Character	Character under the cursor
Marked text	Marked	Text that is marked
Attribute, font, or special marker	Marker	Long description of the marker

Notes The #HIGHLIGHT_VALUE of marked text does *not* include any markers that are contained in that marked region of text.

When #MODE=CATEDIT, #HIGHLIGHT_TYPE cannot equal Marker.

#MODE=DIALOG

When #MODE=DIALOG, the highlight is in a box that contains settings, at the point at which the user could press ENTER to accept the box. (For example, the user has not pressed F3 (CHOICES) to display choices for a setting.) In this case, the value of #PROMPT is the text of the setting that is currently highlighted. #DEPTH has no value.

Note #MODE=DIALOG does not tell you which box the highlight is in.

If the highlight is on	*#HIGHLIGHT_ TYPE*	*#HIGHLIGHT_VALUE*
A setting that displays choices in a box (when the user presses F3 (CHOICES))	Choice	Text of the highlighted choice in a box that contains settings
A text setting (such as **View name**)	Text	Text of the highlighted text setting
A numeric setting (such as **Line spacing**)	Number	Text of the highlighted number setting
A file name setting (such as **Export to file**)	Filename	Text of the highlighted file name setting
A date setting (such as **Start date**)	Date	Text of the highlighted date setting
A list of category names (such as **Sections**)	Listfield	Text of the highlighted category in the list
A setting that displays an ellipses (such as **Header/Footer...**)	Continuation	Null
A setting that displays a category name (such as **Section head**)	Category	Text of the highlighted category setting

#MODE=SELECT

When #MODE=SELECT, the highlight is in the View Manager, Macro Manager, the Select File box, or in boxes with choices. In this case, #PROMPT and #DEPTH have no value.

For the following box	*#HIGHLIGHT_ TYPE*	*#HIGHLIGHT_VALUE*
View Manager	Viewname	Text of the highlighted view name
Macro Manager	Macroname	Text of the highlighted macro name
Select File	Filename	Text of the highlighted file name
Choices	Choice	Text of the highlighted choice

#MODE=MENU

When #MODE=MENU, the highlight is on a menu command. In this case, #PROMPT and #DEPTH have no value.

If the highlight is on	*#HIGHLIGHT_ TYPE*	*#HIGHLIGHT_VALUE*
a menu command	Menu	Text of selected menu choice

#MODE=OTHER

When #MODE=OTHER, the highlight is anywhere in Agenda that is not covered by the other values for #MODE. For example, the value of #MODE=OTHER in Help and where part of the category hierarchy is displayed, such as in the **Section** setting after pressing F3 (CHOICES)). When #MODE=OTHER, #PROMPT and #DEPTH have no value.

#NULL

The value of #NULL is null (empty). You can use #NULL for making string comparisons. For example, you can use #NULL to test for a string variable that contains no data, as shown in the example below. You can also use #NULL to test any occurrences of null values (for example, when #MODE=DIALOG and #HIGHLIGHT_TYPE= Continuation, #HIGHLIGHT_VALUE=NULL).

Example

`{IF;(%x=#NULL);LOOP}`	branches to LOOP if %x is empty

The two commands {LET;%a;#NULL} and {CLEAR;%a} have the same effect.

Note You can use #NULL only for making *string* comparisons. #NULL does not work for numeric comparisons.

#PROMPT

If the highlight is in a box with settings, the value of #PROMPT contains the name of the current setting. For example, if the **Special actions** setting is highlighted in the Category Properties box, the value of #PROMPT is Special actions:. In addition, in {SELECTION;*string*}, *string* is the text that #PROMPT returns. (See #MODE and {SELECTION} in this chapter.)

The settings in the Print Layout box that control separators, spacing, fonts, attributes, and alignment have values for #PROMPT that are not simply the name of the setting. This is also true for the settings in the Header and Footer box that contain the header and footer text.

The value of #PROMPT for each of these settings, is equal to the name of the setting and the particular heading under which this setting displays. For example, in the Print Layout box in a view, the value of #PROMPT for the **After items/notes** setting under **Separators** is After items/notes Separators.

You determine the value for #PROMPT in the Header and Footer box (in views, notes, and the category manager) in a similar manner. For example, the value of #PROMPT for the first line of text under **Header** is Header Line 1 Left.

In addition, there are some settings in Agenda that have the same names (for example, there are two **Sort on** settings in the Item Sorting in All Sections box, one under **Primary sort key**, and one under **Secondary sort key**). For these settings, the value of #PROMPT is typically the heading under which the setting displays and the name of the setting.

Tip If you plan to use {SELECTION} to move the highlight to one of these settings, you should first verify the value of #PROMPT and then specify that value in {SELECTION}.

#TIME

The value of #TIME is the current system time in the default time format, which you set using the **File Properties** command in the Global Date Settings box.

#TRUE

The value of #TRUE is 1 (true). You can use #TRUE to test #KEYHIT with the {IF} command. (See {IFKEY}, {IF}, and #KEYHIT earlier in this chapter.)

#VIEWNAME

The value of #VIEWNAME returns the name of the view.

- If #MODE=VIEW, the value of #VIEWNAME is the name of the current view.
- If #MODE equals anything *except* VIEW, the value of #VIEWNAME is the name of the last-displayed view.

See #MODE earlier in this chapter.

Chapter 4
Tips and Techniques

This chapter provides some guidelines and troubleshooting suggestions that you can use when you create macros or when you encounter problems running macros you've created.

In this Chapter

This chapter provides

- Suggestions for creating macros that choose menu commands
- Suggestions for creating macros in the correct mode
- Troubleshooting techniques

Note Macros are often written for other people. In this chapter, the term "you" refers to the person writing the macro. The term "user" refers to the person running the macro.

Choosing Menu Commands in Macros

If you include instructions in a macro that select Agenda menu commands, it is a good idea to select those commands by their first letter, rather than by highlighting the command and pressing ENTER. For example, in a view, both of the following macro instructions select the **Print** command:

```
{F10}{RIGHT;4}{ENTER}
```

```
{F10}P
```

The second form is better because it is shorter, easier to understand, and does not depend on the order of commands in the menus.

The same is true of selecting choices for settings in boxes. It's better to select a choice for a setting by pressing F3 (CHOICES) and then typing the full name of the choice. This is a more precise action than pressing SPACE BAR to cycle through the choices for a setting, pressing F3 (CHOICES) and then using ↑ and/or ↓ to highlight the choice, or typing the first letter of the desired choice (since some choices begin with the same first initial letter).

Note Do *not* use keystrokes to move to a specific choice in a box. Instead, use {SELECTION} to go directly to the choice. For guidelines for using this command, see "{SELECTION}" in Chapter 3.

In addition, if you write a macro that includes a slash (/), this character produces different results, depending on where the macro executes the keystrokes that include the slash. For example, if a macro containing a slash is run in a view, the slash brings up the menu. If a macro containing a slash is run while editing an item or a note, the macro produces the slash character.

Tip To bring up the menu, regardless of what part of Agenda the macro is run in, include {F10} in the macro. To produce the slash (/) character, include this character in the {TYPE} command, for example {TYPE; /}.

Starting in the Correct Part of Agenda

You can start recording a macro in a view, the category manager, or a note. If you record a macro in a view, and play it back in a note, you are likely to get unexpected results.

If you are creating macros for your own use, this may not be a problem because you can usually remember or tell from the macro name where you have to be in Agenda to run the macro. If you create a macro that you want to share with others, you might want to make sure that the macro is run in the correct part of Agenda (in a view, note, and so forth).

You use the Macros Properties box to specify the part of Agenda in which a macro can be run (see Chapter 1). You can also use macro commands to select the correct part of Agenda to run a macro, as illustrated in the following example. The macro in this example ensures that the user is in a view before the macro is executed.

The following example shows a macro with commands that switches to the current view from anywhere in Agenda. You might use this macro as a subroutine.

Example

`{LABEL; ToView}`	
`{BRANCH; #MODE}`	branches to label designed to match the possible values of #MODE
`{ESC}`	escapes from any part of Agenda except the note, view, or category manager
`{BRANCH; ToView}`	branches to top of loop to try again
`{LABEL; Note}`	if MODE=NOTE, F5 exits note
`{F5}`	
`{BRANCH; ToView}`	loops to try again
`{LABEL; CatMgr}`	if MODE=CATMGR, F9 to go to the view
`{F9}`	
`{LABEL; View}`	if MODE=VIEW, in correct part of Agenda to run the macro

Troubleshooting

You may encounter problems when working with macros. This section lists in alphabetical order some of these problems and what to do to correct them.

Running a macro in the wrong part of Agenda

The most obvious evidence that a macro is not working is that it types unexpected characters. If the macro is simply a set of recorded keystrokes, and it types keystrokes where they are not expected (such as in a menu, or in a box that contains settings), it usually means that the macro was recorded in one part of Agenda and run in another part. You might encounter this problem if you create a macro in a view and run it in a note. The Macro Properties box identifies where a macro can be run.

Missing a brace around a macro command

If the macro types characters that look like incomplete macro commands, it usually means that one or both braces enclosing the macro commands are missing. For example, consider the following macro commands:

```
{LABEL;first_step
{IFEQ;%a;%b;first_step}
```

In this example, the right brace is missing from the {LABEL} command. When a macro with these lines is executed, {IFEQ} does not execute. Instead, Agenda types the following characters:

```
%a;%b;first_step}
```

Enclosing a macro command in parentheses (( )) or brackets ([ ])

If the macro types characters that look like incomplete macro commands, it usually means that you did not enclose a macro command in braces ({ }), but used one or more parentheses (( )) or brackets ([ ]) instead.

Missing a semicolon between arguments

You may write a macro command that requires multiple arguments but you leave out a semicolon (or add an extra one). For example:

```
{IFEQ;%a;%blabelname}
```

{IFEQ} requires three arguments separated by semicolons. The line above includes only two arguments and the semicolon before *labelname* is missing. {IFEQ} will not execute. Instead, Agenda types:

```
{IFEQ;%a;%blabelname}
```

Missing a semicolon if omitting arguments

Some commands have optional arguments. For example, for {ALERT;boxtop_message;main_message}, if you don't want to display a boxtop_message in the Alert box, you need to enter an argument separator for the argument you omit. The semicolon serves as the argument separator. Thus, {ALERT;;main_message} displays an Alert box with just a main message.

Incorrect spacing between commands

A space is interpreted as a literal character (except at the beginning of a line). This means that inadvertent placement of a space between commands may produce unexpected results. For example, in a box with settings, pressing SPACE BAR may switch between choices for settings.

Not enclosing character strings in quotes

You must enclose string constants in quotes if you want the string to include:

- Semicolons (;), braces ({ }), or percents (%) (if the % symbol is the first character in the string)
- An expression operator, such as a hyphen (–), or anything that looks like an expression, such as (Profit – Expenses) so that Agenda interprets this as a string and *not* an expression. For example, if you specify {IF;(%a="Joanna James-Smith");LOOP}, Agenda will not interpret the hyphen as a minus sign. (See also "About Expressions" in Chapter 2.)
- White space (spaces, tabs, and carriage returns) before and/or after string constants

  Agenda strips white space before and after string constants. If you want Agenda to recognize white space preceding and/or following a string, you must enclose the string in quotes. For example, the command {LET;%d;" John Smith "} recognizes a space preceding and following the string John Smith.

Not enclosing expressions in parentheses

You must enclose expressions in parentheses in order for them to be recognized as expressions and not strings. (See "About Expressions" in Chapter 2.)

Chapter 5
Sample Macros

This chapter provides sample macros that include many of the macro commands described in this book. You can use these macros as they are presented here, or modify them before using them in your own work. For additional macros that include many of these commands, see *Starter Applications.*

In this Chapter

This chapter provides sample macros that

- Export items in a section using #PROMPT and #HIGHLIGHT_VALUE
- Parse a name using string-manipulation commands
- Search for any of several strings using {FIND}
- Perform calculations across several numeric columns in a view using expressions and floating-point variables

Sample Macro 1: Exporting Items in a Section

You can use the following macro to guide a new user through exporting all items in a section to another file. This macro prompts the user to specify a file name, checks whether the file exists, prompts for user action based on whether the file exists, and informs the user when the export is complete.

This macro also illustrates the use of the {SELECTION} command, and the #HIGHLIGHT_VALUE and #PROMPT special variables in a macro.

Example

`{Export the Current Section}`	the macro name
`{WINDOWSOFF}`	hides the Export Structured File box from the user
`{F10}FTE`	**Menu, File, Transfer, Export**
`{LET;%ExFile;#HIGHLIGHT_VALUE}`	gets default file name from screen
`{LABEL;Input Name}`	
`{INPUTFILE;Enter an Export File Name; %ExFile}`	asks user to enter file name
`{IF;(%ExFile=#NULL);Input Name}`	if no response, asks again
`{IF;(%ExFile={ESC});EscPressed}`	if user presses ESC, aborts
`{TYPE;%ExFile}`	puts user's response into the **Export to file** setting
`{ENTER}`	
`{SELECTION;File already exists}`	moves the highlight to the **File already exists** setting
`{IF;(#prompt <> File already exists:); Export it}`	if the file does not already exist, export the items
`{ROLLMENU; File already exists; Please choose:;3;Append;Append;Replace; Replace;Choose another file;Input Name}`	if the file already exists, asks the user what action to take
`{BRANCH;Input Name}`	if user presses ESC, asks the user to enter a file name
`{LABEL;Append}`	
`A`	selects **Append** and exports the items
`{BRANCH;Export it}`	
`{LABEL;Replace}`	
`R`	selects **Replace** and exports
`{LABEL;Export it}`	
`{SELECTION;Items}`	moves the highlight to the **Items** setting
`{F3}Items in section`	makes sure **Items in Section** is selected, and, if it is, proceeds with the export
`{ENTER;2}`	
`{LET;%Complete_Msg;"Exported File: "}`	once export is complete, displays a message to let the user know that the export was successful
`{APPEND;%Complete_Msg;%ExFile}`	
`{ALERT;;%Complete_Msg}`	
`{BRANCH;End}`	
`{LABEL; EscPressed}`	branches here if the user pressed ESC from the file name box. Cancels entire operation
`{ALERT;User Pressed Escape;Operation Can-celled}`	
`{ESC;4}`	
`{LABEL; End}`	
`{RETURN}`	unfreezes the screen display

Sample Macro 2: Parsing a Name

The following macro prompts for a user's name and displays it as Last name, First name Middle name. This macro illustrates string manipulation; you can also use it as a subroutine for other macros.

Example

```
{Parsing a Name}                            the macro name
{DEFINT;%space;%space_two;%start_last}      defines integer variables
{LABEL;Input Name}
{INPUTTEXT;Please enter your                prompts for user's name
   full name;%name}
{IF;(%name=#NULL);Input Name}               if no response, asks again
{IF;(%name={ESC});Done}                     if user presses ESC, aborts
{FIND;%space;%name;" ";1}                   finds the first space
{FIND;%space_two;%name;" ";(%space+1)}      checks if there is a middle name or initial by
                                               looking for another space starting at one
                                               position beyond the last value of %space
{LENGTH;%length;%name}                      finds the length of the string
{IF;(%space_two=-1);NoMid}                  determines whether there is a middle name
{LET;%start_last;%space_two}                if there is a middle name, the last name starts
                                               after it
{BRANCH;Continue}
{LABEL;NoMid}                               if there is no middle name, the last name starts
                                               after the first name
{LET;%start_last;%space}
{LABEL;Continue}
{RIGHTSTR;%last;%name;                      extracts the last name
   (%length - %start_last)}
{TYPE;%last}                                types the last name
{LEFTSTR;%first;%name;(%space-1)}           extracts the first name
{TYPE;", "}                                 types a comma and a space
{TYPE;%first}                               types the first name
{IF;(%space_two=-1);Done}                   if there is no middle name, ends macro
{MIDSTR;%middle;%name;                      if there is a middle name, extracts it
   %space;(%space_two-%space)}
{TYPE;%middle}                              types middle name or initial, if there is one
{LABEL;Done}
{QUIT}                                      ends macro
```

Sample Macro 3: Searching for Strings

This macro is an expansion of {FIND}. It is intended as a subroutine for other macros. This macro is used when you are searching for several strings at the same time.

You might use this macro to find white space within a string, where white space is defined as SPACE BAR, TAB, or ENTER. For example, you might want to locate white space so that you can manipulate part of a string (such as the last name from a string that contains both a first and last name).

This macro receives a source string, a string position, and any number of search strings. It returns (in the global variable %%offset) the first place in the source string that matches any of the search strings.

You might call this macro from another macro with the following macro instruction:

```
{CALL;Find OneOf;%srcstr;%n;" ";{TAB};{ENTER}}
```

where %scrstr is the string to be searched for, %n is the starting position of the search, and the " ", {TAB}, and {ENTER} arguments are match strings.

Example

```
{Find OneOf}                          the macro name
{IF;(#ARGCOUNT>2);ArgsOK}             checks to make sure this macro is called with at
                                         least 3 arguments.
{ALERT;Macro Bug;                     displays a message if not enough arguments
   Bad Args Passed to Find Oneof}        are passed from the previous macro and
   {QUIT}                                terminates macro execution
{LABEL;ArgsOK}                        goes here if the arguments are correct
{DEFINT;%%offset;%n;%f}               defines the global and local numeric variables
                                         in this macro
{DEFSTR;%match}                       defines the local string variable %match to
                                         store the arguments to be matched
{INPUTTEXT;;%src}                     first argument passed from first macro is
                                         source string
{INPUTTEXT;;%n}                       second argument passed from first macro is
                                         start position
{LET;%%offset;-1}                     no matches found yet
{LABEL;Loop}                          loops through all remaining arguments
{IF;(#ARGCOUNT=0);DONE}               if no arguments are left, ends macro
{INPUTTEXT;;%match}                   gets the next argument from the first macro
{FIND;%f;%src;%match;%n}              tries to find it in the source string and saves its
                                         location in %f
{IF;(%f=-1);Loop}                     if no match found, goes to next match string
{IF;(%%offset=-1);UseIt}              finds and uses first match
{IF;(%f>=%%offset);Loop}              if this match is not earlier than others, does not
                                         use it
{LABEL;UseIt}                         if this match is earliest found, saves its loca-
   {LET;%%offset;%f}                     tion, then tries next match string
   {BRANCH;Loop}
{LABEL;DONE}                          finished. Returns value already in variable
                                         %%offset
{RETURN}
```

Sample Macro 4: Performing Calculations Across Numeric Columns

The following macro shows how you can use expressions and floating-point variables to calculate totals for four numeric columns in a view. In this example, the macro prompts the user to enter the purpose of the trip and then travel expenses. Next, the macro types the expenses that the user enters into the appropriate numeric expense column in the view. Finally, it calculates the total expenses and types these results in the Total column.

This macro is based on a view with numeric columns as shown in Figure 5-1. In this view, Trip is the section head and Hotel, Meals, Airfare, Trans, and Total are numeric column heads.

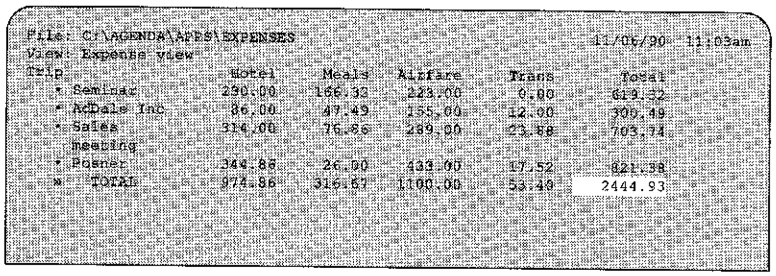
Example

`{Enter Travel Expenses}`	the macro name
`{WINDOWSOFF}`	freezes screen display
`{LABEL;Start}`	
`{DEFSTR;%trip;%temp}`	defines all string and floating point variables
`{DEFFLOAT;%hotel;%meals;%air;%trans;%ttl}`	
`{CLEAR;all}`	resets all variables to 0 or NULL
`{LABEL;Enter Trip}`	
`{LET;%temp;%trip}`	resets temp variable
`{INPUTTEXT;Enter purpose of trip;%temp}`	prompts user to enter purpose of trip
`{IF;(%temp={ESC});Escape}`	if user presses ESC, asks if user wants to quit
`{LET;%trip;%temp}`	stores user's input to a string variable
`{LABEL;Input Hotel}`	
`{LET;%temp;%hotel}`	resets temp variable
`{INPUTTEXT;Enter hotel expenses;%temp}`	prompts user to enter hotel expenses
`{IF;(%temp={ESC});Enter Trip}`	if user presses ESC, returns to previous entry
`{LET;%hotel;%temp}`	stores user's input to floating-point variable
`{LABEL;Input Meals}`	
`{LET;%temp;%meals}`	resets temp variable
`{INPUTTEXT;Enter meal expenses;%temp}`	prompts user to enter meal expenses
`{IF;(%temp={ESC});Input Hotel}`	if user presses ESC, returns to previous entry
`{LET;%meals;%temp}`	stores user's input to floating-point variable
`{LABEL;Input Airfare}`	
`{LET;%temp;%air}`	resets temp variable
`{INPUTTEXT;Enter airfare expenses;%temp}`	prompts user to enter airfare expenses
`{IF;(%temp={ESC}); Input Meals}`	if user presses ESC, returns to previous entry
`{LET; %air; %temp}`	stores user's input to floating-point variable
`{LABEL;Input Trans}`	

continued

`{LET; %temp; %trans}`	resets temp variable
`{INPUTTEXT;Enter other transportation expenses; %temp}`	prompts user to enter other transportation expenses
`{IF;(%temp={ESC}); Input Airfare}`	if user presses ESC, returns to previous entry
`{LET; %trans; %temp}`	stores user's input to floating-point variable
`{LABEL;Type Expenses}`	
`{CtlHome}{CtlLeft}`	moves cursor to top-left corner of view
`{TYPE;%trip}{ENTER}{RIGHT}`	types explanation of trip into an item
`{TYPE;%hotel}{ENTER}{RIGHT}`	types hotel expense into Hotel column
`{TYPE;%meals}{ENTER}{RIGHT}`	types meal expense into meals column
`{TYPE;%air}{ENTER}{RIGHT}`	types airfare expense into Air column
`{TYPE;%trans}{ENTER}{RIGHT}`	types transportation expense into Trans column
`{LABEL;Total}`	
`{LET;%ttl; (%hotel + %meals + %air + %trans)}`	calculates total expenses
`{TYPE;%ttl}{ENTER}`	types total expenses into Total column
`{BRANCH;End}`	branch to label which ends macro
`{LABEL;Escape}`	
`{ROLLMENU;Do you want to quit?; Select No to continue, Yes to end macro; 2;No;Continue;Yes;Quit} {BRANCH;End}`	if the user presses ESC while entering the trip explanation, asks if the user wants to continue or quit
`{LABEL;Quit}`	branch to label which ends
`{BRANCH;End}`	macro
`{LABEL;Continue}`	
`{BRANCH;Enter Trip}`	branch to label which continues entering expenses
`{LABEL;End}`	
`{ALERT;;Done entering expenses}`	displays message saying macro is done
`{WINDOWSON}`	updates the screen display with changes
`{QUIT}`	ends macro

This macro may produce the following output:



```
File: C:\AGENDA\APPS\EXPENSES                                    11/06/90  11:03am
View: Expense view
Trip                       Hotel     Meals   Airfare      Trans       Total
   • Seminar              230.00    166.32    223.00       0.00      619.32
   • AcDale Inc            86.00     47.49    155.00      12.00      300.49
   • Sales                314.00     76.86    289.00      23.88      703.74
     meeting
   • Posner               344.86     26.00    433.00      17.52      821.38
   »   TOTAL              974.86    316.67   1100.00      53.40     2444.93
```

Figure 5-1 *Expenses calculated across numeric columns*

Index