Customizing the

Caucus 2.7[a]

User Interface

by Screen Porch

http://screenporch.com

Table of Contents

Chapter 1

Overview

1.1 Introduction

Caucus is the most flexible computer conferencing package available today. With Caucus, the entire user interface can be modified to meet the specific needs of your users. These changes might range from minor wording modifications to seamless integrations of Caucus with other software packages, or even complete translations to other languages.

This manual describes in detail how to modify the text-only version of the Caucus user interface. Please read it carefully before making any changes. For information about the World-Wide-Web version of Caucus, see the Caucus 3.0 manuals.

1.2 Typographical Conventions

Several different type styles are used in this guide to represent different concepts. Examples of each style are shown below.

Words that are emphasized are shown in *italics*, as in "*Never* change the original dictionary files supplied with Caucus".

Words that should be replaced by a specific instance of some object are also shown in *italics*. For example, "EDIT *filename* uses the default text editor to edit the file *filename*."

Caucus commands are always shown in upper case, as in "SHOW ITEM 1".

Names of specific files are shown in **boldface**, such as the dictionary file **america0**.

Sample text from a macro or dictionary string is usually indented and shown in Geneva. For example, the definition of the macro LAB is:

```
:M_lab
list all items brief
```

1.3 The Caucus Options File

The standard Caucus user interface includes a set of choices or options that control exactly how Caucus works and what the users can do. For example, one option governs whether a user may run other programs from inside Caucus. Another option tells Caucus if it should log the time a user spends in each conference.

The options are contained in an "option file" which can be changed only by the Caucus administrator. Chapter 2 describes how to use the options files.

1.4 The Caucus Dictionary

The key to modifying Caucus is its dictionary. The dictionary contains all of the words that Caucus "knows" — including the text of the Caucus commands, the standard menus, keywords, error messages, help text, macro commands, and other displays.

Most of the modifications you can make to Caucus require changing a dictionary. (The exceptions are the options file, described in chapter 2, and the attachments map, described in chapter 12.) Each dictionary is a plain text file and can be changed with an ordinary text editor or word processor.

There are three key features in Caucus which you can use in customizing your own user interface:

- a standard command-line interface
- a standard menu interface
- a macro programming language

The command-line interface is the original Caucus user interface. It is described in the Caucus User's Guide. Starting from a single prompt ("AND NOW?"), Caucus users type English-like sentences or commands to tell Caucus what they want to do. Caucus also may prompt them for other information along the way (for example, "RESPOND, PASS, or ?"). These commands are the basic building blocks for any other kind of interface.

A standard menu interface was added to Caucus in version 2.2, and is described in the Caucus Menu User's Guide. Menus make Caucus much easier to learn and to use. The Caucus menus are written in the macro programming language.

The macro programming language allows you to extend the capabilities of Caucus. With "macros" you can create new commands, change old ones, or even write programs *inside* Caucus. These programs can interact with the user, access files, or run other applications on your host computer.

1.5 Capabilities: What You Can Do With These Tools

You can change the Caucus user interface in hundreds of different ways. But most changes that are made to Caucus fall into a small number of categories. The paragraphs below describe these categories.

Change the way Caucus looks
The entire look of the command line or menu interface can be changed. It's easy to modify the text of menus and prompts to provide "power user" or novice user interfaces.

Change the way Caucus acts
New commands can be added to extend what a user can do, or provide more convenient ways of doing common actions. (For example, the macro LAB is a short-cut for LIST ALL ITEMS BRIEF.) New menus also can be added to extend the menu interface.

Turn off some features
Features can be disabled by renaming the relevant commands and removing the help text. Alternately, the feature can be restricted by requiring that the user know a special password.

Change default settings
The default values for the SET command can be changed to fit your users' expectations. The SET command, and how it controls the way Caucus acts, is described in chapter 9 of the Caucus User's Guide.

Translation to other languages
The entire user interface can be translated into another language simply by changing the text in the dictionary. Caucus has already been translated into French, Dutch, Italian, and Japanese.

Integration with other applications
A Caucus macro can run other programs or command scripts on your host computer, and can bring back the results as text to put into a conference, or even as commands for Caucus to perform.

File upload and download
This is a special case of the previous category. Any file transfer program can be run from a macro to reliably upload prepared text into a conference, download material from a conference into a file, or upload and download attached files.

Attachment Display
Caucus discussions may have "attached" files of any type. A unique action may be performed during the display of each type of attached file.

1.6 How to Use This Guide

The rest of this guide demonstrates exactly how to make the kinds of changes described in the categories in the previous section. You do not need to read the entire guide in order to make any changes to Caucus. Instead, scan the chapter descriptions below, and find the one(s) that

relate to the changes that you want to make. Note that chapter 3 is required reading for anyone contemplating changes to the dictionary. Similarly, reading chapter 7 is a must if you wish to write macros or menus.

Chapter 2 describes the Caucus option files, and how to use them. This chapter is independent of the rest of this guide.

Chapter 3 describes the format of the dictionary files, and how to modify them. You *must* read this chapter before making any changes to the dictionary.

Chapter 4 describes how to change the default settings used by the SET commands.

Chapter 5 describes the editor macros, which extend and control Caucus users' access to text editors and word processors on the host computer.

Chapter 6 describes how to change the display format of items, messages, responses, etc.

Chapter 7 is the reference guide for the Caucus macro programming language. Read this chapter when you need to know precisely how a particular feature works.

Chapter 8 describes macros that allow users to upload and download files to or from conferences using the Kermit file transfer protocol.

Chapter 9 discusses a general approach to integrating Caucus with other application programs, and describes some detailed examples.

Chapter 10 gives an example of how to turn off a Caucus feature — the Caucus electronic mail component.

Chapter 11 explains how to translate the Caucus dictionary into other languages.

Chapter 12 describes the attachment map. This file tells Caucus how to display files that are attached to items or responses.

Several companies have developed add-on software that extends the user interface and integrates Caucus with other applications. If you are considering making significant changes to the Caucus interface, you may wish to consider their products first. For more information, contact Screen Porch
.

Chapter 2

Invoking Caucus

2.1 Introduction

There are many ways of altering the way that Caucus looks and behaves. This chapter describes two of those ways. The first method involves changing the command line which invokes Caucus, and is covered in section 2.2. Sections 2.3 through 2.5 describe how to change the "Option Files". Finally, section 2.6 lists the command line options that duplicate, for the most part, the functionality of the Options Files.

2.2 The Command Line

This section assumes that you are familiar with writing operating system scripts. If you are not, it will be difficult to use these features to alter the way that Caucus behaves.

Users start Caucus in one of two ways: manually from the command line or automatically during their login procedure. Either way, a command script is invoked which actually executes the Caucus program. (A command script is sometimes called a command file, shell script, batch file, cpl file, EXEC, and so on, depending on your host operating system.) This command script is automatically created when you install Caucus. If you display this command script, it should look similar to one of the scripts in the table below, depending on which operating system you are running (yours may have different pathnames). This table lists each operating system, the name of the command script, and the command line it contains which executes the Caucus program.

Unix and Xenix:
cv2
```
/u/caucus/BIN2/caucus_x /u/caucus  $1   -y master.opt
```

VMS:
cv2.com
```
cau :== sys$user:[caucus.bin2]caucus_x.exe
cau  sys$user:[caucus 'p1' -y master.opt
```

Novell LAN:
caucus.inc

```
caucus_x %root%   $1   -y master.opt
```

Let's look at the (last line of the) command script.

In general, the first "word" in the command line tells the operating system how to find the Caucus program.

Typically, the second word is the path of the home directory of the Caucus account. These first two fields should not be changed, and must always be present.

The second word in the VMS command line does not end in a right box ']', and should not end in one.

The third word specifies the first argument passed to the command script. This argument is optional and indicates that you would like to start using Caucus in a specific conference. If you do not specify a conference in which to start, Caucus will prompt you with "Join which conference?".

For example, to run Caucus from a Unix prompt and automatically join the conference 'demonstration', enter

```
cv2 demonstration
```

If you would like different sets of users to run Caucus and automatically start in different conferences, create one **cv2** script for each group, and change the third word to the name of the conference in which you would like that group to start.

The fourth word is '-y', which is a "command line option". The command line options are described in section 2.6. Many of the command line options described there have become obsolete. Please use the "Characteristic Options", discussed in sections 2.3 through 2.5, whenever possible.

The '-y' option is the "characteristics file" option. It must be followed by a simple filename, one without any preceding directory names. This file must reside in the home directory of Caucus. It should contain a list of characteristics that control how Caucus behaves. This file is explained in great detail in sections 2.3 through 2.5. Caucus is delivered with two characteristics files: **master.opt** and **captive.opt**.

There are many other command line options. If you need to use one or more of them, they must be specified on the Caucus command line after the specification of the Caucus home directory. The safest way to do this is to add them to the end of the command line, for example:

```
/u/caucus/BIN2/caucus_x  /u/caucus  $1  -y   master.opt  -w
```

Again, however, we recommend that in this case you use the 'expiration' characteristics option.

2.3 Introduction to Option Files

As described briefly in section 1.3, Caucus uses a standard set of options to control exactly how Caucus works and what Caucus users can do. These options turn various special features on and off, such as logging the time spent in a conference, access to the host operating system, input and output redirection, and so on.

The options are contained in plain text files, called "option files". Caucus is distributed with two such files, called **master.opt** and **captive.opt**. Only the Caucus manager has access to these files.

These files are in turn referenced by the command scripts described above, and their Captive Caucus counterparts, which actually run the Caucus program. (See your Caucus Manager's Guide for more information on Captive Caucus.) The command scripts run Caucus with the options specified in **master.opt** or **captive.opt**. Examine the contents of these scripts to see how this is done on your operating system.

Never change the contents of the **master.opt** and **captive.opt** files. If you wish to change the options that Caucus uses, make a copy of one of the option files, and edit the copy. Then edit the command script that you use to run Caucus, and look for the name of the original option file. Change it to be the name of your new option file.

2.4 Format of the Option files

Each line of an option file contains an option specification or a comment. Comment lines must begin with a '#' character. An option specification is the name of an option followed by a value for the option. The name and value must be separated by at least one space. The value is either a number (0, 1, 2, etc.), one of the words "on" or "off", or a filename, depending on the option. Each option has a default value, which is used if the option is not specified by name in the option file.

2.5 List of Available Options

All of the options are listed below, with a brief description of each. A similar table is included in the comments in the **master.opt** file.

The format of this list is:

option_name value_type default_value
meaning

baudrate numeric 1200
MS-DOS dial-in baud rate, for PCs only. A value of 999 autobauds to 300 or 1200. This is only useful for Novell LAN and DOS verions.

caploop on/off off

ON means that Caucus does not automatically exit; when one user is finished, it prompts again for the name and userid of the next user. (You can exit Caucus at this point by pressing <CANCEL>.) This option requires that CAPTIVE be ON.

captive on/off off

ON tells Caucus to use the captive login shell. It is useful only for captive userids. This means that Caucus prompts the user for a name and password.

command on/off off

ON allows the Caucus user to execute operating system commands directly from the "AND NOW?" or menu prompts by prefixing them with an exclamation point ("!"). See Section 9.5 of the Caucus User's Guide for more on command escaping.

debug numeric 0

If this non-zero, Caucus will display internal debugging output. Our technical support staff will describe how to use this if you think that there is a problem with Caucus.

diskformat string ascii

This option describes the data representation which Caucus uses to store conference data on the computer's disk drive. If your users will only be using the ASCII data set, you do not need to specify this option. In Europe or Japan, you may wish to use one of the following values for diskformat:

euc
iso8859
sjis

diskfull on/off on

ON tells Caucus to check whether the disk is getting to full to run Caucus. The check is performed every time a conference is joined. OFF turns off disk checking.

echo on/off on

OFF tells Caucus not to echo any characters that are typed by the user. This is useful only for half-duplex transmissions.

editcom on/off on

This option controls use of the EDIT command. If EDITCOM is ON, users may edit files by typing EDIT at "AND NOW?" or a menu prompt. If EDITCOM if OFF, the EDIT command will not work, but will display a warning message instead.

editlist on/off off

ON tells Caucus to restrict the use of text editors. When this option is specified, only the text editors or editor macros listed in the "editnames" table of the dictionary file **macros0** may be used. See chapter 5 for more information.

eightbit on/off off

ON tells Caucus to allow the use of all eight bits for each character. This should be turned ON whenever any non-American dictionaries are used.

entryfile filename no default

Tells Caucus to display the contents of the file *filename* when it begins executing. If no value is given for *filename*, this option is ignored. *Filename* must be located in the Caucus home directory.

exitfile filename no default

Tells Caucus to display the contents of the file *filename* when a user leaves Caucus. If no value is given for *filename*, this option is ignored. *Filename* must be located in the Caucus home directory.

expiration on/off on

If you have a demonstration version of Caucus with an expiration date, ON instructs Caucus to display its expiration date.

fastprompt on/off off

ON tells Caucus to skip display of two short text input prompts (the default values for these are ">" and "?"). These two prompts have string indices of "ss_inprom" and "ced_Kquest", respectively, in the dictionary file **america0**. This is useful if your connection to the host computer is slow to turn around.

fileinput pathname no default

If specified, Caucus will read commands and other input from file *pathname* instead of from the keyboard. When Caucus encounters an end-of-file, it exits normally. If no value is given for *pathname*, this option is ignored. *Pathname* must be a full or relative pathname.

fileoutput pathname no default

If specified, Caucus will write all of its output to file *pathname* instead of to the user's terminal. If no value is given for *pathname*, this option is ignored. Fileoutput is used primarily for testing purposes. *Pathname* must be a full or relative pathname.

fullscreen on/off on

This option only applies to the VM/CMS version of Caucus. When it is ON, Caucus uses the fullscreen VM interface. When it is OFF, Caucus uses a line-oriented interface, suitable for "dumb" terminals.

hangup on/off on

This option controls hangup detection. If HANGUP is ON, and a dial-in user hangs up the phone, Caucus exits gracefully. If HANGUP is OFF, Caucus (and any child processes or applications) ignores hangups.

iocontrol on/off on

This option controls whether Caucus is allowed to change the state of the user's terminal. It defaults ON. Iocontrol OFF may be useful when Caucus is connected to another process via a pipe or similar mechanism.

japanese on/off off

ON turns on Japanese (Kanji) features. Specifically, this tells Caucus to recognize a double-width kana space as a word separator. It also tells Caucus to assume that the *first* name of a person's name should be the name by which they are alphabetized. This is only useful for Japanese Caucus.

logconf on/off off

ON tells Caucus to log the user's entry into and exit from each conference. The logfile is named **log*nnn*** for each conference, where *nnn* is the number of the conference. The logfile is automatically placed in the Caucus home directory.

logfeature on/off off

If this option is set ON, Caucus writes the logging information into a file called lgYYMMDD (where YY is replaced by the last two digits of the year, MM by the month number, and DD by the day number) in the user's directory in the Caucus database.

(For example: for a userid "xyz" on a Unix system, on April 2, 1993, the logging file would be **USER001/xyz/lg930402**.) For more information about the format of the logging file, and which features Caucus automatically logs, see section 2.7.

loguser logfile no default

Keeps a running log of who has used Caucus in the file *logfile* in the **MISC** directory in the Caucus userid. Each successive entry into and exit from Caucus is appended to the end of this file. Do *not* specify a full pathname for this file.

newuser on/off on

ON means that anyone who can run Caucus in captive mode can register as a new user. OFF means that only people who already have login names may use captive Caucus. The Caucus Manager can add new login names by using the **cv2pass** program. Please see the Caucus Installation Guide for your system for more information.

oneuser on/off off

ON means that only one user at a time may access Caucus from a particular userid. OFF means that Caucus only prints a warning if multiple users share the same userid.

port1 or 2 1

Selects port for dial-in use of MS-DOS Caucus. 1 means COM1, 2 means COM2. This is only useful for Novell LAN and DOS versions.

print 0, 1, or 2 0

Controls whether and how users can print text on your computer's attached printer. A value of 0 means no printing may be done. If the print option is not specified, 0 is assumed. A value of 1 or 2 allows the user to print text with the Caucus PRINT modifier. For example,

typing SHOW ITEM 5 PRINT at the "AND NOW?" prompt would print the text and responses to item 5 on your computer's printer.

A choice of value 1 or 2 controls who defines how the printing is done. A '1' means that the instructions for how to print text are taken from the **printcod** file in the **MISC** directory in the Caucus userid. This file can be modified only by the system manager or qualified Caucus manager. This is the most secure choice when Caucus is accessed from a captive userid. (For users with their own individual userids, the security aspect is irrelevant.)

A value of '2' means that each user may define their own printing instructions. The instructions can be modified by the user with the Caucus SET PRINT command. This is the least secure choice for a captive userid.

The choice of the value affects security because the print instructions contain operating system commands. Thus, a user who can change the print instructions can effectively execute any operating system command available to that userid.

For more information, see PRINT and SET PRINT in the Caucus User's Guide.

redirection on/off off

ON allows input/output redirection. See Sections 9.4.5 through 9.4.7 in of the Caucus User's Guide for more on file transfers and redirection. Redirection lessens security for captive userids.

termformat string ascii

This option describes the data representation which Caucus expects from the keyboard. This is also the data representation which Caucus will use when writing text to the screen. If your users will only be using the ASCII data set, you do not need to specify this option. In Europe or Japan, you may wish to use one of the following values for diskformat:

euc
iso8859
sjis

underscore on/off off

ON means an underscore ("_") is added to the end of each person's userid for the purposes of accessing Caucus from an individual userid. This option is provided solely to ease the transition for sites that formerly used "captive" Caucus.

2.6 Command Line Options

Command line options are processed first, then characteristic file options are processed. This means that the characteristics file overrides the command line options.

These are the other command line options available besides the "-y" option. Most of these duplicate the behavior of characteristics in the "characteristics file" and are only included for compatibility with older versions of Caucus. The following table matches each of the old

command line options with the new characteristics by name. The options that do not duplicate features in the characteristics file are explained after the table.

Old Option	Characteristic
-a	entryfile
-b	baudrate
-c	command
-d	debug
-i	fileinput
-l	loguser
-n	newuser
-o	port
-p	print
-q	caploop
-r	redirection
-s	captive
-t	pctype
-u	logconf
-v	autologin
-w	expiration
-x	exitfile
-z	fullscreen

The option "-v", the autologin feature, may only be used with the CAPTIVE ON characteristic, or command line option "-s".

The "-v" option must appear in the form:

```
-v name:password
```

where "name" is the person's Caucus userid, and "password" is their password.

The option "-h" turns off Caucus's checking for setuid or "sysprv" privilege. It is meant for use with on-line debuggers such as "ups".

2.7 LOGFEATURE logging file

When the LOGFEATURE characteristic is turned ON, Caucus writes records about which features were used into a “logging file”.

The logging file is a plain ASCII file, containing one line per log record. Each line of the file contains 9 fields, separated by a blank. The field sizes and names are shown below, in order:

size	name	description
12	userid	user's userid
8	feature	name of feature
12	value	value of feature, eg. conference name
5	ivalue	integer value of feature, eg. item number
6	date	shown as YYMMDD
6	time	shown HHMMSS
6	duration	time spent in Caucus, in seconds
3	count	number of things...
7	size	size, eg. size of text of new item

When LOGFEATURE is ON, Caucus automatically records the features listed below, with the fields shown:

ITEMADD (user added a new item)
value: conference name
ivalue: new item number
size: size of new item text

RESPREAD (user read responses)
value: conference name
ivalue: item number
count: number of responses read

MAILREAD (user read their mail)
count: number of messages read

CAUCUS (user accessed Caucus)
duration: time spent in this conference

CONF (user accessed a specific conference)
value: conference name
duration: time spent in this conference

MAILSENT (user sent an original mail message)
value: recipient's name or address
size: size of message text

MAILREPL (user sent a reply mail message)
value: recipient's name or address
size: size of reply text

MAILRRCT (user requested a return receipt message)
count: number of return receipts requested

For more information about logging custom designed features, see section 7.13.

Chapter 3

The Caucus Dictionary

3.1 Introduction

Caucus is distributed with a standard or "master" American English . This dictionary begins with the file **diction0** in the Caucus **DIC2** directory. **Diction0** in turn includes the files **america0, exec0, help0, long0, menus0, macros0, macros0.nam, macros0.ptr**, **net0**, **standar0**, and **sysmac0**. (See section 3.2.4 below for a description of "include".) All dictionary files must reside in the **DIC2** directory.

Only the Caucus userid should be given permission to access these files unless your site has special needs. Since these files determine much of the "look-and-feel" of Caucus, it may be unsafe to allow other users to access these files.

The dictionary is broken up into these files for convenience, modularity, and to make it easier to find specific text. **America0** contains most of the commands, prompts, and text displayed by Caucus. **Help0** contains the help text. **Menus0, long0, standar0,** and **exec0** contain the menus. **Macros0** contains the Caucus macros, **net0** contains CaucusLink text, and **sysmac0** contains the operating system dependent parts of the macros.

A single Caucus host computer can have many different dictionaries, each of which is assigned a number. Caucus users can select the dictionary they want with the SET DICTIONARY command (see section 9.2 in the Caucus User's Guide). The master dictionary is normally dictionary number 0. In fact, that's why the file names all end in 0; it implies that these are the files that make up dictionary 0.

Starting with version 2.5, the **DIC2** directory also contains the files **diction987**, **flibchg987**, **flibmac987**, **flibmac987.nam**, and **flibmac987.ptr**. These files automatically are compiled when Caucus is installed to create dictionary 987, which supports file libraries.

3.2 Dictionary Format

The format of a Caucus dictionary file is very straightforward, and all dictionary files have the same format. Each file is an ordinary text file with no more than 80 characters per line. (A line may be longer than 80 characters, but it will be wrapped at 80 characters when displayed by

Caucus.) As distributed, the dictionary files contain no control or other non-printing characters. (They may be added if desired.)

There are five kinds of lines in a dictionary file:

- comment
- index
- string
- include
- control

You may wish to examine a dictionary file, such as **america0**, as you read the sections that follow.

3.2.1 Comments

A line begins with a "" as the first character on the line. Comment lines are completely ignored by Caucus. Comments may be placed anywhere in the dictionary, but each comment line must always begin with a "#". Each dictionary file begins with comments that describe the purpose, contents, and revision date of the file.

If you want a line of text in a string to start with "#" and *not* be a comment, escape it by preceding it with a "\". See section 7.9.

3.2.2 Indices

An line begins with a colon (":"), followed immediately by a name. Caucus uses the index to identify the string that begins on the next line. Some indices are followed by another word on the same line (such as "table" or "res") that conveys a special meaning to Caucus. Indices should *never* be changed.

3.2.3 Strings

A is made up of all of the lines between one index and the next. This includes blank lines (but not "#" comment lines). Each string is "owned" or identified by the index that precedes it.

This pairing of indices and strings is the whole purpose of the dictionary. Caucus uses the index to look up a string in the same way that you look up a word in a regular dictionary to find its definition.

For example, here are two strings taken from the **america0** file:

```
:mai_Pandhelp  res

AND NOW?  (? for help)  $
#
:mai_Pandnow  res

AND NOW?  $
#
```

When Caucus prompts you to enter your next command, it knows to look for the index "mai_Pandhelp" in the dictionary. Caucus takes everything between that index and the next index and displays that string on the user's terminal. In this example, Caucus would display one blank line followed by the words "AND NOW? (? for help)". If you were to edit the dictionary, you could make this string anything you want, and Caucus would faithfully display it — so long as the index remained unchanged.

Note that the final newline or carriage return is not considered part of the string. This makes it possible to display a string that will stop in the middle of a line. For example, when Caucus displays "AND NOW? (? for help)", the cursor on the user's terminal stops to the right of the "help)" and does not go on to the next line. If you want a string to end with a newline, you must put an extra one in — that is, the last line of the string must be a blank line.

Also note the "$" at the end of the "AND NOW" lines in the example above. A "$" at the end of a line is removed when the dictionary is compiled. (See section 3.5 for an explanation of compiling the dictionary.) Some computer systems and text editors automatically strip off any blanks at the ends of lines. So the "$" is a place-holder that keeps the trailing blanks from being removed.

Caucus normally uses "mai_Pandhelp" as the index for the command prompt. If SET VERBOSE is OFF, Caucus uses "mai_Pandnow" instead.

Strings with the name format of xxx_Pyyy are "prompt strings" and can use a "prompt macro". The prompt macro string has the same name as the prompt string, with a ".pm" appended.

The most commonly used prompt strings are:

mai_Pandnow	AND NOW?
mai_Pandhelp	Verbose form of AND NOW?
res_Fitemp	RESPOND, PASS, ? for more options
mes_Fdisp	REPLY, PASS, DELETE, ? for more options
mes_Foksend	OK to send to ?

For more information about how to use prompt macros, see section 7.13.

3.2.4 Include

An line begins with the characters ":<" and is followed immediately on the same line by the name of a file. The include line means that the contents of the named file are included at this

spot in the dictionary, just as though they had been typed there. The **diction0** file, for example, contains only "#" comments and include lines. In addition, Caucus allows the extension of dictionary strings with multiple include files. For example, to include the contents of "file1" and "file2" in the dictionary string "macronames":

```
:macronames
      text
:<file1
:<file2
```

3.2.5 Control

A control line begins with a colon (:) and a space, and is followed by control options. Currently, the only control option is "language=x", where "x" can be "ascii", "euc", "iso", or "sjis". This option selects the default language for the rest of the file. If there is no language control option, "ascii" is assumed.

3.3 Types of Strings

The strings in a dictionary are loosely divided into six categories, depending on their function within the Caucus program. These categories are: interaction strings, short strings, help strings, tables, macro definitions, and menu strings. Each of these types is described in more detail below.

3.3.1 Interaction Strings

Interaction strings make up the largest category, and these strings take up most of the space in **america0**. The naming of the indices for these strings follows a particular convention. Each index is made up of three pieces: a prefix, a type, and a name. The three-character prefix designates the related function: for example, "add" for the add command, "reg" for the new user registration process, and so on. The type is a single letter code that shows how this string is used. The types are:

- P for a user prompt
- H for the help text about a prompt
- E for an error message
- T for miscellaneous text
- K for a keyword typed by the user
- A for a table of options, modifiers, or keywords
- F for a C language "sprintf" format string

The last piece of the index, the name, tells the purpose of the string. Here are some examples of interaction string indices and their purposes:

del_Presign
The DELETE PERSON command (prefix del_) prompts (type P) if the user really wants to resign from the conference.

del_Hresign
If the user asks for help (type H) at the del_Presign prompt, Caucus displays the text of del_Hresign.

del_Tpergone
When the organizer deletes a person, Caucus displays a message (type T) confirming that the person is deleted (name "pergone")

Interaction strings may be of any length. Each line of an interaction string should be no more than 80 characters long.

3.3.2 Short Strings

Short strings are the punctuation strings that Caucus uses frequently, such as ">". These strings are loaded into memory when Caucus starts up, rather than being pulled from the dictionary each time they are needed. The indices for the short strings all begin with the characters "ss_". These strings must be kept very short, typically no more than 8 characters. The short strings are also contained in **america0**.

3.3.3 Help Strings

Help strings contain the text of one kind of on-line help. There are two kinds of on-line help available in Caucus. The first kind is the text that you see when you type "?" or "HELP" at a Caucus prompt. That text is found in the type H interaction strings. There is only one help message for each prompt.

The help strings contain the second kind of help. You see these when you type "HELP", followed by a keyword, at the "AND NOW?" prompt. There is only one help message for each such keyword recognized by Caucus. The index for each help string begins with the characters "h_" and usually ends with the name of the keyword. For example, when you type HELP ADD, Caucus looks for the help string owned by the index "h_add".

Help strings may be of any length, although each line should be less than 80 characters long. The help strings are contained in the file **help0**.

3.3.4 Tables

A table is a special category of string that contains an ordered list of words. Caucus uses a table when it needs to associate a word with a number. For example, the list of the standard Caucus commands is kept in a table called "commands". In **america0** this table looks something like:

```
#COMMANDS:  Regular Caucus commands recognized at AND NOW?
#
:commands  table
echo    help/?     add    delete    show  list  change    forget
search  send       status customize set   join  freeze    version
```

The position (or number) of a word in this table determines the function that Caucus performs. The association between position and function is embedded in the Caucus program and cannot be

changed. For example, suppose you switched "add" and "delete", the third and fourth words in this table. The ADD command would then delete objects and the DELETE command would then add objects.

A slash ("/") in a table separates synonyms for that word. For example, in the "commands" table, "help/?" means that either "help" or "?" is recognized as the help command.

3.3.5 Paired Tables

Pairs of tables are used when Caucus needs to associate a keyword in one table with an index in another table. Caucus looks up the keyword in the first table and then finds the index in the corresponding position in the second table.

The Caucus HELP command is a good example of an application of paired tables. HELP uses two dictionary indices, "helpwords" and "helpptrs". Short versions of these two tables are shown below.

```
:helpwords  table
novice        commands        verbs objects     add   delete      show
change
#
#
:helpptrs  table
h_novice     h_commands       h_verbs     h_objects   h_add h_delete
h_show       h_change
```

Each keyword in the first table is matched with the name of an index in the second table. For example, suppose a user types "HELP SHOW". Caucus finds "show" in the seventh position in the table "helpwords". It then looks for the word in the seventh position in "helpptrs" and finds "h_show". Caucus proceeds to display the dictionary string indexed by "h_show".

First of all, this means that you can easily change the names of the keywords recognized by the HELP command. This is particularly useful when translating the Caucus dictionary into another language.

Secondly, you can also add new keywords and new help text. For example, suppose you wish to add a "HELP MACROS" command that displays information about new macros you have created. Start by adding the keyword "macros" to the end of the "helpwords" table. (Words in a table must be separated by at least one blank space. They need not line up in neat columns as shown in the example.) Then add a matching index, say "h_macros", to the end of the "helpptrs" table. Finally, create a new index "h_macros" followed by the text you want Caucus to display.

3.3.6 Macro Definitions

Caucus macros add to the set of commands available at the "AND NOW?" prompt. A macro is a new command that is built out of existing commands (and possibly other macros).

Each macro has its own name, index, and string. When a user types the name of a macro at "AND NOW?", Caucus looks up the index for that macro, and performs the commands given in its string. The pair of tables ":macronames" and ":macroptrs" connect the name of each macro to its index, much like the paired tables in the HELP example in the previous section. The index for a macro "xyz" is typically called "M_xyz", although this is a convention and is not required.

Macro definitions may be any size, and are stored in the file **macros0**.

Section 3.4 explains how to add your own new macros.

3.3.7 Menu Strings

Each Caucus menu has a one-word name that identifies it. Each menu also has four strings. The indices for these strings are the name of the menu followed by the suffixes "SM", "CH", "MM", and "RP". Therefore a menu called "mail" would have four strings called "mailSM", "mailCH", "mailMM", and "mailRP".

All menu strings are contained in the files **menus0** and **exec0**. Their purpose and function are described in section 7.11.

3.4 Changing the Dictionary

Customizing the Caucus user interface means changing the dictionary. Any change — whether adding new commands, editing the text of a menu, or making new default SET options means changing the text of the dictionary.

There are some very important rules that you *must* follow when changing the dictionary:

〈 **Never** change the dictionary files supplied with Caucus. If you do, your changes will be erased the next time you install a Caucus software upgrade.

〈 To change the text of a dictionary string, copy the string (including the ":index" part) into the file **local000** and edit it there. This file is automatically included at the beginning of **diction0**, and it is *not* erased when you upgrade Caucus.

〈 Note that this means there will be two definitions for the string in the dictionary: yours and the original. If Caucus finds multiple definitions of a string, it uses the first one, and ignores the rest. This means that your strings in **local000** will override any later definitions in the dictionary.

〈 Adding a new macro requires three steps. First put the name of the new macro at the end of the file **local000.nam**. (If this file does not exist, you must create it. Like **local000**, this file is not erased when you upgrade Caucus.) Next put the name of the index for your macro at the end of the file **local000.ptr**. Third and last, put the definition of the macro (including the ":index" part) in the **local000** file.

To understand this procedure better, carefully read through the files **macros0.nam** and **macros0.ptr**. **local000.nam** and **local000.ptr** act as extensions to the "macronames" and "macroptrs" tables, respectively. You may think of these files as being "inside" their respective tables.

Remember that your computer can have multiple dictionaries. You may prefer to leave dictionary 0 unchanged, and create a separate dictionary 1 that contains your changes. In that case, copy **diction0** to a file called **diction1**, and change all occurrences of "local000" in **diction1** to "local001". Then put your changes into **local001**, **local001.nam**, and **local001.ptr**.

Here are some other guidelines you should consider when making changes to the dictionary:

〈 If you change the name of a Caucus command (or object, or keyword), remember to change all occurrences of that command in the dictionary. For example, if you decide to rename the SHOW command to READ, you must edit copies of the "commands" table, the "helpwords" table, and any of the help text and help strings where the word SHOW appears.

〈 If you have multiple dictionaries available to your users, you may wish to create some new macros that switch between dictionaries. If you have a novice user dictionary and a power user dictionary, you may find the commands "NOVICE" and "POWER" more mnemonic than "SET DICTIONARY 1" and "SET DICTIONARY 0".

〈 Older versions of Caucus kept dictionary 0 in a file called **mdtext0**. If you have made significant changes to this file and wish to use them, follow these steps:

〈 First, use a program like diff or merge2 (available from Screen Porch) to find the differences between your changed **mdtext0** and the original file. Then place the changed strings, including the ":index" lines, into your **local000** file.

〈 If you are making changes that affect the operation of Caucus, as opposed to the way it looks, you may need to add your changes to the file **local987** as well as **local000**. Local987 is used by the Caucus file libraries.

For example, if you are changing the string sys_Tset0 (see section 4.2) to change user's default settings, this affects Caucus' overall operation. This change should be included in **local987**.

3.5 Compiling the Dictionary

Once you have changed a dictionary, you must compile it. This translates the dictionary text into a binary format that can be read by the Caucus program.

To compile a dictionary, you must be logged into the Caucus userid. All dictionary files must reside in the Caucus **DIC2** directory. For example, to recompile the standard dictionaries, 0 (for conferences) and 987 (for file libraries), type:

```
cv2mkmd   diction0     0
cv2mkmd   diction987  987
```

To compile a dictionary file called **diction1** and assign it dictionary number 5, type:

```
cv2mkmd   diction1     5
```

The details of how to run the cv2mkmd program may vary from computer to computer. See your Caucus Installation and Manager's Guide for details.

The cv2mkmd program will place the compiled dictionary in the **MISC** directory, which is in the Caucus home directory. The file name of the compiled dictionary begins with **dicti** and ends with a three digit number which is equal to the dictionary number. In the second example, the compiled dictionary would have a file name of **dicti005**. The only management concern involves disk space — if you have many dictionaries on your system, you may want to delete old dictionaries that are no longer used. Once cv2mkmd has finished, your new dictionary is in place and ready to use.

Usually cv2mkmd does its work silently. If you would like it to report each dictionary file as it is compiled, add the **-v** flag to the compile line in the cv2mkmd shell script. It must go at the end of the line which has “makemd_x” in it. For example, in Unix, the line would look like this:

```
/u/caucus/BIN2/makemd_x /u/caucus/MISC/dicti $1 $2 -v
```

Chapter 4

Changing Default Settings

4.1 Introduction

The first time a person uses Caucus, the software goes through a standard registration procedure which asks the person for their name, phone number, and a brief introduction. As part of this procedure, Caucus makes certain assumptions about how the person will use the software. These assumptions, or "settings", include which text editor the user prefers, the number of lines on the user's terminal, and so on. All of these settings can be changed by the user with the SET command. (See chapter 9 of the Caucus User's Guide.)

These assumptions are called the "default settings" and are contained in a single string in the dictionary. Changing the values in this string is the single most common modification made to a Caucus installation.

4.2 Changing Default Settings

To change the default settings, copy the string "sys_Tset0" from the **america0** file to your **local000** file. Edit the string in **local000**; your changes will override the definitions in **america0**. Each word in the string is the default value for a particular SET option. Each option is described below in detail. The numbers correspond to the position in the string: that is, 1 means the first word, 2 the second, and so on.

For Caucus version 2.5 or higher, you must also add these changes to the file **local987** so that the changes will be incorporated in the Caucus file libraries.

(If you are using Caucus on a VM/CMS system, the name of the default settings string is "sys_Tsetcms0".)

1. DEBUG controls debugging output. Must be "ON" or "OFF". The default is "OFF".

2. SCREENSIZE is the number of lines on a user's terminal screen. After each screenful, Caucus pauses and prompts the user to press <RETURN>. A value of 0 means "don't pause". The default is 23.

3. SCREENWIDTH is the width of the user's screen in characters. The default is 79.

4. FORMAT controls reformatting of the user's text. Must be "ON" or "OFF". "ON" means the user's text is reformatted into paragraphs. The default is "OFF".

5. EDIT is the name of the user's preferred text editor. The default is "caucus". See Chapter 5 for more information on making other editors available.

6. TAB is a number that controls tab expansion. During text entry, Caucus translates tabs into spaces. TAB is the distance in characters between tab stops. The default is 8.

7. PAGE is a decimal number, the value of the ASCII "top-of-page" character for your printer. It is used by the PRINT modifier. The default is 12.

8. PRINT is either "default" or "edit". It controls how text is printed for a user. "Default" means the Caucus printcode instructions are used. "Edit" means the user's personal printcode instructions are used. The default is "default". (See section 2.3 for a complete description of how to let users choose their own print instructions or prevent them from doing so.)

9. VERBOSE is either "ON" or "OFF". Certain prompts, such as "AND NOW?", have a long form and a short form. VERBOSE controls which is used. The default is "ON".

10. DICTIONARY is the number of the current dictionary. The default is 0.

11. EOT is the user's end-of-text marker. It can be any string that does not contain blanks. It can also be either: "1eof", which means one (the digit "1") local end-of-file character; or "2return", which means 2 <RETURN>s in a row. The default is ".", which means the user types a dot (period) at the start of a new line followed by a <RETURN>.

12. EXPIRATION controls the display of the expiration warning for demonstration copies of Caucus. It must be "ON" or "OFF". The default is "OFF".

13. TEXT_ENTRY controls how Caucus gets text for new items, responses, and messages. It must be either "terminal", "wordwrap", or "editor". "Terminal" means the user is prompted for text, one line at a time. "Wordwrap" means the user is prompted for text, but Caucus automatically wraps words around the right margin as they are typed. "Editor" means Caucus immediately starts the user's preferred text editor. The default is "terminal".

14. RECEIPTSIZE is used by return-receipt messages. A receipt includes the author and subject of the original message, plus RECEIPTSIZE lines of text from that message. The default is 0 (zero).

15. MYTEXT controls when a user's own text appears as new to that user. It must be "now", "never", or "later". "Now" means the material appears as new, immediately. "Never" means it is considered seen. "Later" means (for responses only) it appears new only after other users have added subsequent responses. The default is "later".

16. STARTMENU is the name of a macro that is executed on entry to a conference. It is traditionally used to start a set of menus. See section 7.10.6 for details. The default is "execmenu".

17. IMPORT_MAIL is used to determine whether or not to import a user's operating system mail into their Caucus mailbox. It can have one of three values: OFF, IMPORT, COPY. If the field is OFF, Caucus doesn't try to access the user's operating system (OS) mailbox. COPY tells Caucus that it should copy each of the messages in the user's OS mailbox into the user's Caucus mailbox. If the user chooses IMPORT, the messages will be copied into the Caucus mailbox and deleted from the OS mailbox.

    This feature is only available on Unix and VMS systems. For all other systems it should be set to OFF. Furthermore, on some Unix systems, the OS mail cannot be copied and left in the OS mailbox. Instead, if the user has chosen COPY, Caucus will copy the messages into the Caucus mailbox and leave a copy of the messages in the file "old_mail" in the user's home directory.

18. EXPORT_MAIL is used to determine whether or not to export a user's Caucus mail to their system mailbox. When another Caucus user sends you Caucus mail, if EXPORT_MAIL is OFF, the mail will be delivered to your Caucus mailbox. If EXPORT_MAIL is set to EXPORT, such mail will be delivered to your Unix or Vax system mailbox.

19. TERMINAL defines the user's terminal type, and is used to determine how to display file attachments to items and responses. The list of terminal types is in the file **attach_map** in the MISC directory.

20. SHOWATTACH controls what Caucus should do when displaying an item or response that has an attachment. OFF (0) ignores the attachment, ON (1) always displays the attachment, and ASKME (2) asks the user if they wish to display the attachment.

21. SET TRANSFER determines which file transfer protocol to use when downloading or uploading a file attachment for an item or response. Valid options are: none, copy, kermit, xmodem, ymodem, and zmodem.

Note that changes made to sys_Tset0 affect only those users who register after the change is made. It does not change the settings of your existing users. If you absolutely must change the settings for existing users, consider placing the appropriate SET command in the auto-fire macro _enter_caucus. See section 7.12 for details.

4.3 Example: Changing the Default Value

When a user finishes entering an item, response, or message, he or she must type a special character or characters, called the end-of-text string or EOT. Normally, EOT is ".". This means that to finish entering text, the user must type a dot (period) at the beginning of a new line and press <RETURN>.

Suppose that you want your users to type ".s" instead of just ".". Copy the string "sys_Tset0" from **america0** to **local000**. (If there already is a copy in **local000**, do not copy it again, but edit your existing copy. Include the change in **local987**.) It should look something like this:

```
:sys_Tset0 table
off 23 79 off caucus 8 12 default on 0 . off terminal
0 later execmenu
```

Look up EOT in the table in section 4.2. It is the eleventh word. As you can see in the sample sys_Tset0 above, the eleventh word is ".". Change it to ".s", save your changes to **local000** and **local987** and recompile dictionary 0 and 987. From now on, when new users register with Caucus their default EOT will be ".s".

Chapter 5

Caucus Editor Macros

5.1 Introduction

As a rule, each Caucus user has the ability to select a favorite text editor or word processor for use in editing text inside Caucus. This is the purpose of the SET EDITOR command. (See chapter 9 of the Caucus User's Guide for more details.) In addition, chapter 4 of this guide describes how the Caucus manager can select a default text editor for all Caucus users.

Both of these features make Caucus more friendly and familiar to users who need to edit conference text. There are times, however, when you may want more sophisticated editing tools. That is the purpose of the Caucus editor macros.

Normally, when a person types SET EDITOR XYZ, it means that they want XYZ to be the default editor inside Caucus. XYZ must be the name of a command a person would type at an operating system prompt to edit a file. For example, if the operating system command is "emacs filename", then the Caucus command is SET EDITOR EMACS.

Alternatively, the Caucus administrator may redefine XYZ to be an editor macro. The definition of this macro can include *any* host operating system commands. When a person uses the XYZ editor, Caucus performs the commands chosen by the Caucus administrator.

5.2 Defining an Editor Macro

These are the four steps required to define and use an editor macro:

1. Turn on the EDITLIST option.
   Edit the option file (eg. master.opt or captive.opt) that is used by the script that runs Caucus, and add the line "EDITLIST ON". You do this only once, even if you have multiple editor macros.

2. Add your macro to editnames and editptrs.
   Copy these strings from **macros0** to **local000**, and edit them there. Add the name of the editor macro (such as "xyz") onto the end of the "editnames" table, and the name of the macro definition (such as "M_xyz") onto the end of "editptrs".

3. Define the macro.
   In **local000**, create a string (such as "M_xyz") that contains the commands to be performed. Each line in the string may contain one host operating system command. Unlike other Caucus macros, each line should *not* start with a "!". Each editor macro takes exactly one argument, called "$2", which is the name of the file containing the text to be edited. In other words, when the commands in your macro are being performed, any "$2"s will be replaced by the name of that file. The next section gives examples of this.

4. Copy all of the changes you just made to **local000**, and add them to the end of the **local987** file. (This file is used by Caucus file libraries.) Recompile dictionary 0 *and* dictionary 987.

5.3 Sample Editor Macros

The example below defines four editor macros, called "caucus", "vi", "vi5", and "wp". While the example assumes the host operating system is Unix, similar examples could be created for any operating system.

The "caucus" macro just starts the built-in Caucus text editor. "vi" invokes the Unix 'vi' editor. "vi5" is the same as "vi", but it tells the editor to use a 5-line window on the screen. (This is useful for people editing text over a slow connection such as a telephone line.) "wp" runs a hypothetical word processor called "wordproc", in combination with a filter program called "wpfilter".

Here is the text of the editnames and editptrs strings, along with the macro definitions:

```
:editnames   table
caucus   vi    vi5    wp
#
:editptrs    table
M_caucus M_vi  M_vi5  M_wp
#---------------------------
:M_vi
vi  $2
#
:M_vi5
vi  -w5  $2
#
:M_wp
wordproc  $2
wpfilter  <$2  >$(userid).wp
mv  $(userid).wp  $2
```

The "vi" and "vi5" macros are very straightforward. They both invoke the Unix 'vi' editor, and tell it to work with the "$2" file. "vi5" also specifies a command line option "-w5" that tells 'vi' to use a five line window.

The "caucus" macro is even simpler; it has no macro definition! Since the Caucus editor is built-in, no operating system commands are required. You must, however, reserve a place for it in the editnames and editptrs tables.

The "wp" macro is more complicated, and demonstrates more of the capabilities of editor macros. Many word processors, such as the hypothetical 'wordproc', store files or documents in their own particular format. These formats often contain non-printable characters and other binary information that Caucus does not understand. Caucus automatically attempts to strip out this information as it reads back the contents of the edited file, but this can still result in truncated lines, strangely formatted paragraphs, or gibberish added to the text.

Most of these word processors do have a "save file as ascii (or text)" option. One solution to the problem described above is to require people to use this option. In practice, this is hard to enforce.

The "wp" macro takes another approach. After the user has finished editing with 'wordproc', the "wp" macro runs yet another program, here called 'wpfilter'. This program filters out the binary information from the edited file, producing a plain text file that Caucus can understand. Many word processors include a program or other means of performing this filtering. In some cases, the filter program may simply be the word processor itself, run with a script or other special options.

The "wp" macro defined above edits the "$2" file with 'wordproc'. Then the macro runs the program 'wpfilter' and uses the file that 'wordproc' created as its input. The output of the filter is placed into a temporary file called "$(userid).wp". (For more information on "<", ">", "$2", and "$(userid)", see sections 7.10, 7.4, and 7.5.)

Finally, the Unix 'mv' command moves the contents of the temporary file back into the "$2" file, where they are read by Caucus. This last command demonstrates that the final text-only version of the edited file must be stored in the file referenced by the $2 argument. This is implied in the other macros.

This technique of combining multiple programs in a single editor macro offers many possible uses. These include: integrating text editors and spelling checkers; using previously prepared forms for Caucus text entry; and restricting or controlling access to word processor functions. The last idea is the basis of the next section.

5.4 Editor Macros and Security

Many Caucus hosts are set up such that the majority of the users can only run Caucus, and do not have access to the host operating system. This may be done as a security precaution or for strictly business reasons. There are several Caucus options (see chapter 2) that are designed specifically for this purpose.

At first glance, the Caucus editor macros and the SET EDITOR command may appear to compromise such restrictions. A clever person may find a way to use certain text editors to gain access to the operating system.

For that reason, when the EDITLIST option is turned on (see section 5.2), *only* the editors listed in "editnames" may be used. This is an important point. It means that the Caucus administrator can define precisely which editors may (and may not) be run from inside Caucus.

Depending on the editor or word processor, it also means that the administrator may be able to restrict or limit some of the editor's functions. For example, some editors have options that permit or do not permit access to the host operating system. By using an editor macro to run such an editor, you can turn off such access in the same way that the "vi5" example macro restricted 'vi' to a five line window.

Of course, any user who has access to the dictionary files can change the "editnames" and "editptrs" tables. This is another reason why access to the dictionary files must be tightly controlled. Typically, only the Caucus userid should have access to the dictionary files. Please remember this when you edit or create new dictionary files.

Chapter 6

Changing the Display Format

6.1 Introduction

When a person reads a Caucus item, response, or message, the text appears on their screen in a particular manner or "format". It may be indented, or preceeded by an item number or author name or a variety of other information.

Like many other things in Caucus, this manner of display may be changed. A set of Caucus dictionary strings, called the "format strings" control how the messages, items and responses are displayed. This chapter describes each of the format strings and how to change them to suit your needs.

Please remember that when making changes to the Caucus dictionary, you must copy the strings you want to change into new files (such as **local000**) and edit the copies. Do not change the originals stored in **america0**. (Please see section 3.1 of this manual for a description of each of the original dictionary files, and section 3.4 for more information on the process of creating dictionaries.)

6.2 Default Displays

When someone asks Caucus to show an item, Caucus normally displays it like this:

```
Item 1    22-SEP-89    2:41    Charles Roth (roth)
Introduction to this demonstration conference.
Text entered as  Item 1 shows here.
```

This display is made up of a number of parts called *fields*. The first line contains four fields:

- an item identifier that tells the reader what item they are reading
- a date stamp and a time stamp that tells the reader when the item or response was entered
- the author of the item and userid (in parentheses)

The second line contains one field, the title of the item. The final field is the text of the item and consists of the rest of the lines in the item.
The format of the display is controlled by a dictionary format string. When Caucus is installed, the text contained in this dictionary string tells Caucus to display items in the format shown above. You can change this format by changing the string associated with item display.

The item is followed by all of its responses. The display of responses is controlled by another format string. The following sections describe each of the format strings and what they control.

6.3 Item Display

The index of the string that controls how an item is displayed is "res_Fshowitem". The default definition for this string is shown below. Note that the index for the string is followed by the word "res". This must not be changed. It means that the string is memory resident.

```
:res_Fshowitem  res

Item &i    &9d    &5t    &a (&u &h)
&s

&x
```

Each of the fields described in the section 6.2 are represented in the string by variable names that begin with the symbol "&". The variables are interspersed with plain text. Any text can be added to these strings to make them more informative. For example, when Caucus interprets res_Fshowitem, it displays a blank line, followed by the word "Item", followed by a space, the item number, and so on.

Each item has its own number, which is represented by the variable "&i". Similarly, the date and time have variable names "&d" and "&t". While &u and &h represent the user's userid and hostname, "&a" is the author of the item. The next line is "&s", the title (subject) of the item. This is followed by a blank line, and finally the item text, "&x". Once you understand these few rules and the names of the variables you will be able to alter the display of items and responses to suit your needs and those of your users.

The "9" in "&9d" is called the "display width" and is the number of characters that Caucus uses to display this field. In this case, even if the date that the item was entered is more than nine characters, Caucus shortens it to the first nine characters. For positive numbers, Caucus will right-justify this text. If instead the date format was specified as "&-9d", Caucus would have left-justified the text.

You can specify a number larger or smaller than the text you expect. For example, you may find that very long names of authors tend to confuse the display. To change this, you can enforce a particular field length for authors' names by inserting a number between the "&" and the "a". If you decide that 40 characters is long enough for an author's name, change the "&a" to "&40a". Since this is a positive number, Caucus will display the author's name against the right margin. It will pad the rest of the 40-character field with spaces.

If you prefer the author's name to appear as far left as possible, use "&-40a". Since Caucus uses the full display width in the format string, it displays the date and time, followed by four spaces, followed by the name of the author and enough spaces to make 40 characters. You may wish to restrict the total within the fields on one line to be less than 80 characters, as this is the width of most terminals. Caucus will "wrap" any line longer than 80 characters.

6.4 Response Display

There are two format strings that control the display of responses. They are described in the following two sub-sections.

6.4.1 SHOW RESPONSE

For an example of how to use format strings, look at "res_Fshowresp", the format string that controls the display of responses. The default definition is:

```
:res_Fshowresp  res
&i:&r) &a (&u&h) &55z &9d &5t
&1x
```

which yields the familiar display:

```
7:12) Jeff Victor (jvictor)         12-AUG-90  10:48
 Kermit: the first file transfer amphibian.
```

Suppose that you want to give each of the fields a label when Caucus displays a response. You might decide that a very informative display would look like this:

```
Item 7                  Response 12
Author: Jeff Victor (Userid:jvictor)
Date Entered: 12-AUG-90 Time Entered: 10:48
   Kermit: the first file transfer amphibian.
```

To accomplish this, you would use the following dictionary string:

```
:res_Fshowresp  res

Item &-28i Response &r
Author: &a (Userid: &u)
Date Entered: &-19d Time Entered: &t
&2x
```

This example also demonstrates the one special case of the field width modifier. In the text variable "&x" *only*, a field width modifier specifies the number of spaces to indent the display of text from the left side of the screen. Caucus only supports one or two spaces of indentation.

6.4.2 SHOW RESPONSE BRIEF

The other format string controls the display of the SHOW RESPONSE BRIEF command. The default definition is:

```
:res_Fshowbrief res
&i:&r) &a (&u&h) &9d &5t
```

Which has the result of displaying only the response header, not the text.

A common alteration to this definition is to add one or both of the &b and &n fields to display the size of the response. See the table of fields in section 6.10.

6.5 Separators

In addition to item and response headers and text, Caucus displays other information during a SHOW ITEM or SHOW RESPONSE. After displaying the item text, but before displaying the first response, Caucus tells the user how many responses there are. The format string for this is res_Fresp, which looks like this:

```
:res_Fresp
 %d responses selected, out of %d total.
```

"%d" is a special case of plain text which *must* appear in this format string. This is where Caucus puts the number of responses. If you change this format, you must use "%d" somewhere in the string to specify the position of this number. Caucus may behave unexpectedly if the "%d" is absent.

Caucus uses a series of hyphens to separate one response from the next. The string that controls what is displayed is "res_Tsep" and looks like this:

```
:res_Tsep
- - - - -
```

If you prefer a different separator, you might change this to:

```
:res_Tsep  res

*****

#
```

This adds a blank line above the text of the separator and one below it.

If there have been any responses deleted from an item, Caucus displays a line like this after the last response, but before prompting for new responses:

```
(3 deleted responses)
```

This is controlled by this string:

```
:res_Fdeld
 (%d deleted responses)
```

As with res_Fresp, the two characters "%d" specify the position of a value. This must be included in the format string.

6.6 List Items

Many Caucus commands, such as LIST ITEMS or DELETE RESPONSES, display the "header" of an item or items. Different commands use different dictionary strings to display these headers. The table below shows which commands use which strings.

Command	String
CHANGE ITEM	res_Fshowhead1
CHANGE RESPONSE	res_Fshowhead1
CHANGE TITLE	res_Fshowhead1
ADD RESPONSE	res_Fshowhead1
LIST ITEM BRIEF	res_Fshowhead3
DELETE ITEM	res_Fshowhead3
DELETE RESPONSE	res_Fshowhead4
LIST RESPONSE BRIEF	res_Fshowhead4
LIST ITEM	res_Fshowhead5
LIST RESPONSE	res_Fshowhead6
SHOW RESPONSE	res_Fshowhead7
SHOW RESPONSE BRIEF	res_Fshowhead8

6.7 Message Display

The SHOW MESSAGE command uses four different strings to control the precise display of messages. The default values for each of these strings is shown below:

```
:mes_Fshowhd
```

```
(&m)  --------------------------------
 SUBJECT: &-64s
```

The string "mes_Fshowhd" displays the message header: the message number ("&m"), and the message subject ("&s").

```
#
:mes_Fshowpt
 &H from &a (&u&h) &50z &9d &5t
&1x
#
```

Following the header, Caucus displays each part of the message according to "mes_Fshowpt". A Caucus message may have several parts: an original message, a reply, a reply to a reply, and so on. For each part, "&H" is the message keyword (taken from string mes_Atype, usually "MESSAGE", "REPLY", or "FORWARD"). "&a" is the name of the author of that message part, “&u” is the author's userid, and "&h" is the author's hostname (if any). The "&50z" is a special code that pads or truncates the text on the current line to column 50. The date and time ("&9d" and "&5t") appear starting in column 52. Finally, the text of the message part appears beginning on the next line, indented one column ("&1x").

A divider is displayed between each pair of message parts, according to "mes_Fshowdv". The default text is just a blank line:

```
:mes_Fshowtr
#
```

Lastly, a trailer is displayed after the entire message: that is the purpose of "mes_Fshowtr". The default text is completely empty:

```
:mes_Fshowdv
```

There is also a dictionary string which controls the preface to the text of the subject. The index to this string is "mes_Treplypref". The default is an empty string.

The most common use of this is to put the text "Re: " into this string. If you do this, the subject field of all replies will be prepended wih "Re: "

This is done only once, so a reply to a reply will have only one "Re: " at the beginning of the subject field.

6.8 List Messages

The output of the LIST MESSAGE command has two formats: verbose and brief. Caucus uses the verbose form unless you use the BRIEF keyword. The verbose format looks like this:

```
:mes_Flistlg
&-2w&3m) &H from &a (&u&h) &56z &9d &5t
```

```
            SUBJECT: &-63s
```

The brief form looks like this:

```
:mes_Flistbr
&-2w&3m) &a (&u&h) &26z &9d &-42s
```

The only new format code introduced by these strings is "&w", which refers to the "newness" of the message. If the message is old, &w just becomes blank spaces. If the message is new, &w displays the text of the string "mes_Tnewmsg". The default value of that string is just "*".

6.9 _LIST_STATUS

The Caucus command "_list_status", described in section 7.14, can be used to display summary status about the current conference according to a dictionary format string. Information about which special codes are meaningful to _list_staus is contained in the summary table below. (Note: the "&a", "&u", and "&h" codes refer to the current user instead of an author of some text.)

6.10 Summary

This section describes each of the variables that can be used in the display format strings. Each of them begins with the "&" symbol and ends in a letter. A number may be inserted between the "&" and the letter, to indicate a field size and text justification as described in sections 6.3 and 6.4. If you want a single "&" to appear in the text, use two of them: "&&".

The codes "I R M L" in the "Used With" column say which strings the code may be used with: I is for item display, R for response display, M for message display, and L for the _list_status command.

Code	Purpose	Used With
&a	author name	I R M L
&b	number of characters in text	I R
&c	conference name	I R M L
&d	date	I R M L
&e	extent (size) of attached file	I R
&f	name of attached file	I R
&h	hostname of author	I R M L
&H	message keyword	M
&i	item number	I R
&I	number of new items in conference	L
&k	name of CANCEL key	L
&l	number of responses to this item	I R
&m	message number	M
&M	number of new messages	L
&n	number of lines of text	I R
&N	number of items with any new responses	L
&o	object type of attached file	I R
&o	organizer of conference	L
&r	response number	R
&R	"-" followed by second response number	I R
&R	number of new responses in conference	L
&s	subject or title	I R M
&t	time text was entered	I R M
&T	displays elapsed time since start of Caucus session, in the form mm:ss	L
&u	userid of author	I R M L
&w	"newness" of message	M
&x	text	I R M
&X	first line only of text	I R M
&z	truncate or pad to this column number	I R M L

Remember that the string **res_Fdeld** requires a special sub-string "%d". If this string is empty, it is ignored and no error message is produced when deleting messages.

Chapter 7

Macro Language Reference

7.1 Introduction

The Caucus macro language is a collection of features that extend the capability of the Caucus commands in specific ways. The complete set of these features is described in this chapter.

The macro language is not a complete programming language by itself. However, in combination with other programs and command scripts at the host operating system level, it can provide a powerful programming environment.

7.2 Macros

The basic tool in the macro language is called, not surprisingly, the "". A macro has three parts:

〈 a
〈 an
〈 a or

When you type the name of a macro at "AND NOW?", Caucus looks up the matching index, and finds the definition in the dictionary. (This process is described in detail in section 3.3.6.) Caucus proceeds to perform the commands in the text of the definition as though they had been typed by the user.

The definition of a macro can contain any Caucus commands, including other macros. A macro should not, however, call itself, either directly or indirectly through another macro. A macro can perform any number of Caucus commands, although the commands must be separated by semicolons or placed on different lines. There is no limit to the size of a macro.

7.3 Echo

Macros often need to display arbitrary text on the user's terminal. For example, the Kermit macros described in Chapter 8 tell the user to instruct the local Kermit to receive or send a file.

The ECHO command is provided for this purpose. ECHO is one of several commands that are intended for use only in macros, and are therefore not documented in the Caucus user's guide. The general form of the ECHO command is:

```
echo  text
```

which simply displays a line of text (all of the words after ECHO) on the user's terminal, followed by a carriage return.

ECHO may be used with the Caucus input and output redirection modifiers >, >>, and <. (See section 9.4 of the Caucus User's Guide.) This is very useful if you want a macro to display many lines of text. Rather than use an ECHO command for each line of text, you could put all of the text into a file, and say:

```
echo  <file
```

There are several options that you may use with the ECHO command that change the way it works. The options are:

-c	Clear screen
-f	Same as -p unless full-screen mode is turned off, in which case entire command is ignored (VM/CMS only)
-h	User help window (VM/CMS only)
-i	Display text, ignoring <CANCEL> key
-n	Display text without final carriage return
-p	Display text and wait for <RETURN>
-q	Do not print user's key presses while waiting for <RETURN>
-s	Display named dictionary string
-u	Display text, without wrapping long lines, and without pausing after each screenful

To use an option, put it after the word ECHO, and before any text to be displayed. To use several options, separate them with blank spaces. For example:

```
echo  -n -p Press RETURN to continue...
```

displays "Press RETURN to continue...", leaves the cursor at the end of the line, and waits for the user to press <RETURN>.

The -s option must be followed immediately by the index of a dictionary string. It displays the text of the definition of that dictionary string.

The -q option is only useful when used with the -p option.

On some computers, the -c option may do nothing. Check with Screen Porch LLC.for more details.

7.4 Macro Arguments

When you type the name of a macro at "AND NOW?", you can type other words on the same line. (Just as you can type other words after a regular Caucus command, as in "SHOW ITEM 1".) These words are called "arguments", and can be used by the macro. In the definition of the macro, the arguments are represented by the special words "$1" through "$9". These words are called "variables".

When Caucus performs the commands in the macro definition, it replaces all of the variables with the appropriate argument. "$1" becomes the first word on the line, which is the name of the macro. "$2" is replaced with the second word on the line (the first word after the macro name), and so on up to "$9". Any variables in the definition that do not have a matching word are removed.

For example, here are the index and definition of a simple macro called "who":

```
:M_who
show  person  $2  $3  $4
list  items  author  "$2  $3  $4"  brief
```

Suppose that a user types "WHO JOE SMITH" at the "AND NOW?" prompt. Caucus displays the personal information about Joe Smith, plus the item numbers and titles of any items added to the conference by Joe Smith. In this case, since there are only three words in the command line, the "$4" is replaced by nothing and effectively disappears.

7.5 Predefined Variables

Besides the argument variables $1 through $9, Caucus also provides a set of predefined variables. Each of these variables is replaced with specific text, as shown in the table below:

Variable name	Replaced with
$(cancel)	name of the <CANCEL> key
$(confid)	Caucus directory pathname
$(confname)	name of this conference
$(confnum)	number of this conference
$(dict)	current dictionary number
$(eot)	the user's end of text string
$(isorg)	1 if this user is an organizer, else 0
$(isread)	1 if this is a read-only member, else 0
$(name)	the user's name
$(showattach)	whether to show file attachments
$(start)	the user's STARTMENU
$(terminal)	the user's terminal type
$(thisitem)	number of most recent item
$(thisresp)	number of most recent response

$(transfer)	file transfer protocol
$(udnum)	USERnnn directory number
$(userid)	the user's userid

To see an example of how these variables work, try typing the following line at "AND NOW?":

```
echo  I am $(name), my userid is $(userid).
```

All Caucus variables, including the predefined variables, may be in upper or lower case letters. This means that "echo $(eot)" and echo $(EOT)" produce the same results.

7.6 Macro Functions

In addition to the macro variables defined above, Caucus provides "macro functions", which provide added power and flexibility. Macro functions are similar to variables, except that they take arguments which further specify the value which replaces the function.

All macro functions begin with "$(", just as the predefined variables do. These two characters are followed by the name of the function, a colon character, the arguments, which also are separated by colons, and a final right parenthesis. An example of a complete macro function is:

```
$(attname:1:2)
```

There presently are three macro functions and they relate to file attachments. The name of the first function is "attname". It takes two arguments: an item number range and a response number range. This function and its arguments are replaced by the name of the file attached to the response specified by the two arguments. The previous example of a macro function would be replaced by the name of the file which is attached to item 1, response 2 of the conference you are in. If that response does not have a file attachment, the text of the function is removed, but not replaced by anything. See the examples below for more details.

The second macro function is "atttype". It takes two arguments - the same two arguments as "attname". The function and its arguments are replaced by the format ('type') of the file attachment. Attachment formats can be ASCII, PostScript, or one of many others.

The third macro function is "attnum". It takes three arguments: item number range, response number range, and file attachment name. The text of this function is replaced by the item number and response number of the response which matches the three arguments. The item number and response number which replace the function text are separated by a colon.

Here are some examples:

```
echo $(attname:1:last)      name of attachment to last response of item 1

echo $(attnum:1-5:all:file) item:response number of attachment named file
```

Note that you may use one of the following keywords as an item or response number argument: "this", "all", "last". (See Caucus User's Guide section 4.4 for more information on using keywords as instances.)

Caucus evaluates macro functions in two steps. First, it replaces all of the keywords and macro variables with their number ranges. Then Caucus evaluates the macro function itself. Macro functions are evaluated left to right, therefore they may not be nested.

The above examples are simple for the purpose of explanation. The power of macro functions may be seen in this version of the Caucus File Library 'get' command:

```
show  attach $(attnum:$(thisitem):all:$2)>$(attname:$(thisitem):all:$2)
$(transfer)send $(attname:$(thisitem):all:$2)
delfile $(attname:$(thisitem):all:$2)
```

The first line copies the attachment out to a temporary file. The second command transfers the file to the user's personal computer. The last line deletes the temporary file.

7.7 Memory Variables

Memory variables are the third kind of variable supplied by Caucus. Unlike the argument variables and the predefined variables, the value of a memory variable can be set inside a macro.

The memory variables are named V0 through V9. Each memory variable can hold up to 100 characters of text.

A memory variable may be used anywhere in a macro definition, just like the other variables. To set the value of a memory variable, use the _ASSIGN command. This is another command supplied specifically for use in macros, like ECHO. The general form of this command is:

```
_assign  variable  text
```

Consider this example:

```
_assign  v0  To be or not to be?
_assign  v1  $(v0)  That is the question.
_assign  v0
```

The first line puts the text "To be or not to be?" into the memory variable V0. The second line, after replacing $(V0) with the contents of V0, puts the text "To be or not to be? That is the question." into V1. The last line clears V0. This means that V0 now contains *no* text.

The contents of memory variables are not retained across Caucus sessions. This means that if you exit Caucus and start it again, the contents of the memory variables are lost. All memory variables start off as empty, that is containing no text. The contents of user variables, explained in the next section, are retained across Caucus sessions.

7.8 User Variables

User variables are the fourth kind of Caucus variable. They are identical to memory variables, except they have one special property: their contents are remembered by Caucus from one session to the next.

User variables are unique to each user, in the same way that each user has a unique personal introduction. Just like the memory variables, they may be used in a macro, or given a value with the _ASSIGN command.

The user variables are named U0 through U9. Each user variable can hold up to 80 characters of text. This text cannot contain the "new-line" character.

User variables are often combined with auto-fire macros (see section 7.11) to add new capabilities to Caucus. In particular, you can reserve some of the user variables to act like additional predefined variables that are relevant to your site, but still unique to each user.

For example: in practice, most Caucus users have a "home" conference. It's the first conference they join when they start a Caucus session, and often where they do their most important communication. You can let each user define a home conference by making the changes listed below:

1. Reserve user variable U0 for the name of a person's home conference. Don't use it for anything else.

2. Define a new macro called HOME. The definition of HOME is:

```
:M_home
_assign  u0  $2
```

3. Create a new definition for the _over_caucus auto-fire macro:

```
:M_over_caucus
join  $(confname)  $(u0)
```

To have a "home" conference, a user must, at AND NOW or a menu prompt, type the word HOME followed by the name of a conference. That conference is now permanently remembered as the name of the user's home conference. Caucus will automatically start each new session in the home conference.

A significant change was made to the way variables are evaluated in version 2.5 of Caucus. As of this version, "multiple pass substitution" is performed on all variables. This means that if a variable expands into the name of another variable, that variable in turn is evaulated.

For example:

```
_assign v0  \$(confname)
```

puts the characters "$(confname)" into variable v0. But,

```
echo $(v0)
```

displays the conference name, since v0 expands into $(confname) which, in turn, expands into the conference name.

7.9 Escape Character

Several characters have special meanings inside a Caucus macro. A semicolon (";") separates two commands, while a dollar sign ("$") means the rest of this word is a variable.

If you need to use one of these characters without its special meaning, place a backslash ("\") immediately in front of it. This is called "escaping" the character. Thus, for example, typing

```
echo What is my; status
```

displays the text "What is my", followed by the output from the status command. On the other hand,

```
echo  Roses are red\; violets are blue
```

displays a familiar bit of doggerel. Backslashes may also be escaped. To get a single backslash, use two together ("\\").

7.10 Access to the Operating System

Caucus provides four tools for accessing the operating system from within a macro. The first is the "!" command prefix, described in section 9.5 of the Caucus User's Guide. The second is the standard input and output redirection operators <, >, and >>, described in section 9.4 of the same guide. The third is the use of < at "AND NOW?" to redirect command input. The fourth is the use of << at "AND NOW?" to redirect comand *and* data input. These tools are described below in more detail.

7.10.1 Command escape "!"

To execute an operating system command at "AND NOW?" or from within a macro, prefix it with "!". This works even if the user is not allowed to use "!" at "AND NOW?". (See the COMMAND option in chapter 2 for more information.) Starting an operating system command with "!" is fairly slow. If your macro needs to execute several operating system commands, try to pack them together in a command script to minimize the number of "!"s.

7.10.2 Redirection with <, >, >>

The input and output redirection operators, like "!", function in a macro whether or not the user is allowed to use them at "AND NOW?". (See the REDIRECTION option in chapter 2.)

Output redirection may also be "thrown away" by redirecting it to what is humorously known as the "bit bucket". The syntax for this is ">@". Thus, the command:

SHOW ITEM 5 >@

doesn't display anything at all, but does have the effect of marking item 5 as "seen".

7.10.3 Command redirection with "<file"

The next tool is a special instance of the "<" input redirection operator. Typing "<file" at "AND NOW?" or a menu prompt, or using "<file" as a command in a macro, redirects the source of Caucus command input to **file**. This means that the *next* time Caucus is about to prompt the user for a command, the commands are instead read from **file**. Once all the commands in **file** have been processed, Caucus prompts the user again with "AND NOW?".

Note that this special form of input redirection applies only to *commands*. Any input required by the command must still come from the user. For example, if **file** in "<file" contains the CHANGE NAME command, the user must still type their new name.

You may also use "<" without a file name to force Caucus to return to reading command input from the keyboard.

These three these tools are used in the example that follows. Consider the DOWNLIST, DOWNLOAD, DOWNNEW, and DOWNKER macros defined below:

```
:M_downlist
echo  join $2 \$(confname)    >$(userid).dwn
echo  downnew                 >>$(userid).dwn
echo  join $3 \$(confname)    >>$(userid).dwn
echo  downnew                 >>$(userid).dwn
echo  join $4 \$(confname)    >>$(userid).dwn
echo  downnew                 >>$(userid).dwn
echo  downker                 >>$(userid).dwn
echo  stop                    >>$(userid).dwn
#
:M_download
show  new  messages            >$(userid).t
<$(userid).dwn
#
:M_downnew
show new resp                 >>$(userid).t
show new items                 >>$(userid).t
#
:M_downker
!ckermit -s $(userid).t
```

The DOWNLIST macro creates a permanent list of up to three conferences selected by the user. The DOWNLOAD macro copies all of the new material in those conferences into a temporary file, without stopping for a response from the user. It then invokes DOWNKER to download the file via Kermit.

To finish the example, suppose a person with userid "smith" once types "DOWNLIST A B C". Thereafter, smith need only type "DOWNLOAD" each day to download all of the new material entered in conferences A, B, and C since the previous day.

Note the use of "\$(confname)". The "$" is escaped by the "\" and is not treated as the beginning of a variable. This means that, when **smith.dwn** is being written, it contains "$(confname)" instead of the name of the current conference. Later, when commands are read from **smith.dwn**, the variable "$(confname)" *is* replaced by the name of the current conference.

The purpose behind the "\$(confname)" is to ensure that Caucus acts reasonably if the user typed "DOWNLIST A B", with no third conference name. In that case, "$4" disappears, and the JOIN command merely re-joins the same conference.

Caution: the "<file" command may be used only once per macro, typically at the very end of the macro. When "<file" is encountered in a macro, it is not executed immediately. Rather, Caucus waits until the next time it would ordinarily prompt the user for a command—and *then* starts reading commands from **file**. You can, however, chain macros and "<file"s, one after another. For example, macro A could use "<file_a", and **file_a** could in turn execute macro B, which could use "<file_b", and so on.

7.10.4 Command and data redirection with "<<file"

The last tool, the "<<" input redirection operator, redirects *all* input (commands and data) from a file. It otherwise acts exactly like the "<" operator, and may be used at "AND NOW?", a menu prompt, or as a command in a macro. When the end of the file is reached, input immediately reverts to the keyboard.

You may also use "<<" without a file name to force Caucus to return to reading command and data input from the keyboard.

This tool is extremely useful for writing macros that are "self-contained"; that is, that operate without asking the user any questions. A simple example is the KILL macro defined below. It acts just like the Caucus DELETE command, only it does not ask the user for confirmation. It just goes ahead and deletes the specified object.

```
:M_kill
echo  delete $2 $3 $4 $5 $6 $7 $8 $9  >$(userid).ans
echo  yes                            >>$(userid).ans
<<$(userid).ans
```

Caution: like "<", the "<<file" command does not take effect immediately. Commands or data will be read from the file *the next time keyboard input is needed.*

7.10.5 Combining "<" and "<<"

It is possible to combine "<" and "<<" in one macro in powerful (and complicated) ways. If, for example, a macro contains both "<abc" and "<<xyz", commands are read from file **abc**, but any input data needed for those commands are read from **xyz**.

The MAKEITEM macro defined below uses both "<" and "<<" to create a new item without asking the user any questions. Furthermore, if item creation is not allowed (if the user has readonly permission to the conference, or the organizer has disallowed the creation of new items), then MAKEITEM fails gracefully.

MAKEITEM has the syntax:

MAKEITEM textfile This is the Title

where **textfile** is a file containing the text of the new item, and "This is the Title" is the title for the new item.

```
:M_makeitem
 echo -s M_makeitem.T2          >$(userid).t1
 echo    $(userid).t2          >>$(userid).t1
 echo    $3 $4 $5 $6 $7 $8 $9  >>$(userid).t1
#
 echo -s M_makeitem.T1          >$(userid).t2
 echo    $2                    >>$(userid).t2
 echo -s M_makeitem.T3         >>$(userid).t2
#
 <<$(userid).t1
#
:M_makeitem.T1
 add item <
:M_makeitem.T2
<
:M_makeitem.T3
<<
#
```

This macro looks pretty foreboding at first, but it makes more sense upon examining the command file that it builds. Assuming that the userid is "xyz", that $2 is "textfile", and that the title ($3, $4, etc) is "This is the title", then the macro produces:

in file xyz.t1

```
<xyz.t2
This is the title
```

in file xyz.t2

```
add item <textfile
< <
```

The makeitem macro starts by redirecting all input from xyz.t1. This in turn redirects command input from xyz.t2. The xyz.t2 file tries to add a new item, taking the text from textfile. If it succeeds, it reads the item title from xyz.t1, which is still open. If the "add item" fails (for instance, if user xyz is not allowed to add items), then the "<<" line immediately closes xyz.t1, and the title is ignored.

7.11 Caucus Menus

Support for menus was added in Caucus version 2.2 with two new commands, _CALLMENU and _RETURN. Like ECHO and _ASSIGN, these commands are meant only for use in macros.

7.11.1 _CALLMENU

The _CALLMENU command starts the execution of a menu. It must be followed on the same line by a menu name and a memory variable name, in that order. When a menu is executed, it follows these four steps:

1. perform a command (typically ECHO some text)
2. prompt the user to type a choice
3. perform the set of commands selected by that choice
4. go back to step 1 and repeat

These four steps are repeated indefinitely or until a _RETURN is executed in the third step. This "returns" from the menu, breaking the cycle.

7.11.2 A Sample Menu

The complete definition for a menu called "Sample" follows, exactly as it would appear in a dictionary file. The rest of this section references the Sample menu to describe how Caucus menus work in general.

```
:sampleSM
 echo -i -s sample.txt
#
:sample.txt
Sample Menu
– – – – – – – –
 1 - Display status
 2 - Show the new responses
 3 - Exit Caucus

Enter your choice: (1)
#
:sampleCH table
1 2 3
#
:sampleMM table
M_cancel M_status M_nochoice
M_status M_new M_exit
#
:sampleRP table
1 1 1 1 1 1
#
:M_cancel
_return
:M_nochoice
echo I don't understand '$(v0)'.
```

```
:M_status
status
:M_new
show new responses
:M_exit
stop
```

To start the "sample" menu from "AND NOW?", type:

```
_callmenu sample v0
```

7.11.3 SM, CH, MM and RP Strings

When a menu is started, Caucus looks for four specific dictionary strings, called "nameSM", "nameCH", "nameMM", and "nameRP", where "name" is the name of the menu.

SM means "start macro". The first thing the menu does is perform the commands in the text of this string. This corresponds to step 1 in section 7.10.1. In the "sample" menu, sampleSM uses ECHO to display the text of string sample.txt.

Once the start macro is finished, Caucus waits for the user to type a line of text and press <RETURN>. Caucus takes the first word on the line typed by the user, and tries to match it against each of the words in the CH string. CH is short for "choices"—it contains the list of menu choices.

The MM string contains the "macro map", the list of actions (macros) the menu may perform. MM and CH are paired tables. When the user types a choice from the CH table, Caucus performs the corresponding macro from the MM table. Note, however, that the MM table has three more entries than the CH table. The first three entries in MM provide actions for three special cases:

- The first entry is performed if the user presses <CANCEL> instead of typing a word.
- The second entry is performed if the user just presses <RETURN>.
- The third entry is performed if the user typed a word that is not in the CH table.

The remaining entries in MM correspond directly to the CH table. That is, if the user types the first word in CH, Caucus performs the fourth entry in MM. The second word in CH corresponds to the fifth word in MM, and so on.

The RP or "repetition" string is also a paired table. It contains a number for each entry in MM. Each number controls how many times the matching MM macro can be performed before the SM start macro is executed. Typically, RP contains all 1's. This means that after any MM macro is performed, Caucus immediately re-executes the SM start macro. A value of 2 for any entry in RP would mean that the user could select the matching entry in MM twice before the SM start macro was re-executed.

The CH, MM, and RP strings must all have the dictionary "table" attribute.

7.11.4 Stepping Through the Sample Menu

When the Sample menu is started, it immediately executes sampleSM, which displays the text of string sample.txt. Caucus then waits for the user to type a line of text. If the user types "1", Caucus matches the CH "1" to the sampleMM entry M_status, and performs the STATUS command. If the user types "2", Caucus performs M_new, which displays the new responses. After both "1" and "2", sampleSM is re-executed, displaying the menu text once again. On the other hand, if the user types "3", Caucus immediately exits with the STOP command.

If the user presses <CANCEL> instead of typing a choice, Caucus performs M_cancel, the first entry in sampleMM. This macro does an _RETURN, which leaves the menu and returns to "AND NOW?".

If the user just presses <RETURN>, Caucus uses the second entry in sampleMM, which is M_status. Note that M_status appears twice in sampleMM: once as choice "1", and once as the default choice.

If the user types something other than "1", "2", or "3", Caucus uses the third entry, M_nochoice. This is where the memory variable named in the _CALLMENU command comes into play. When the user types a choice that's not in the CH table, the entire line of text is placed in the menu's memory variable. For example, if the user types "4", M_nochoice uses this feature to say "I don't understand '4'".

Finally, unless the user exits Caucus or _RETURNs to "AND NOW?", the menu performs sampleSM again, repeating the cycle.

7.11.5 Menu Details

There are several details about how menus work that do not appear in the Sample menu. These details are described below.

Menus can be "stacked" or "layered". This means that a menu can invoke a sub-menu, the sub-menu a sub-sub-menu, and so on. To make a menu choice invoke a sub-menu, that choice must perform a macro from the MM table that contains the _CALLMENU command that starts the sub-menu. When this choice is selected by the user it immediately starts the sub-menu. When the sub-menu _RETURNs, the original menu continues from where it left off.

The _RETURN command may be followed on the same line by a one-word argument or object that tells Caucus how far to return. _RETURN by itself, or _RETURN followed by -1, means "return back 1 level". _RETURN followed by a menu name always returns to that menu, regardless of how many layers of menus there are in between. _RETURN followed by a negative number returns that many levels back from the current menu. _RETURN followed by a positive number N returns to menu level N. (The very first menu is level 1, its sub-menus are level 2, and so on.)

For example, suppose you have a menu A that has a sub-menu B. B in turn has a sub-menu C. C can have a menu choice that returns directly to A, by executing either "_RETURN A" or "_RETURN -2". If menu A is always the very first menu, "RETURN 1" would also work.

You may also use _RETURN to leave the menus completely by returning to a level that does not exist. For example, _RETURN followed by a very large negative number (such as -1000) will return through *all* of the layers of menus. "_RETURN X" where menu X does not exist, will do the same thing.

The SET DICTIONARY command also has this effect, as a safeguard. If the user's dictionary is changed, Caucus automatically _RETURNs through all layers of menus. This is required, since the new dictionary may not contain any menus at all.

Memory variables are also more powerful than the "Sample Menu" suggests. The rule for a menu's memory variable is this: if the user presses <CANCEL> or <RETURN>, the memory variable is emptied. If the user types a word that is not in the CH table, the entire line typed by the user is placed in the memory variable. If the user types a word that *is* in the CH table, all of the text *after* the first word is placed in the memory variable.

Finally, note that the macros in the MM table have indices and strings, but not names. All references to these macros are done by the index. Unlike regular macros, they do not have entries in the "macronames" table.

7.11.6 SET STARTMENU

In the previous sections, the sample menu was started by typing an _CALLMENU command at "AND NOW?" This is impractical for day-to-day use.

The proper way to start a menu is to define a named macro that performs the _CALLMENU command. This macro may then be typed by the user at "AND NOW?"—or it can be started automatically.

The latter is the purpose of SET STARTMENU, a new form of the standard Caucus SET command. The value of SET STARTMENU must be the name of a macro, or the word "OFF". Each time the user enters a conference, Caucus executes the macro specified by SET START-MENU. ("OFF" means no macro is executed.) By starting a menu this way, the user may never even need to see the "AND NOW?" prompt!

Note that the macro is executed each and every time the user enters a conference. It is a fundamental assumption of Caucus that commands happen *in* a conference. JOINing another conference automatically _RETURNs through all layers of menus in use. If STARTMENU is not "OFF", the specified macro is started all over again on entering the new conference.

STARTMENU has a default value for new users that is defined in the dictionary string "sys_Tset0". See chapter 4 for information about changing the default.

7.11.7 Input Redirection with Menus

The previous sections all assume that menu input comes from the user. That is, when _CALLMENU processes a menu, it takes its input (the menu choice) from the user's keyboard.

It is also possible to redirect menu input to come from a file. For example, the command:

```
_callmenu sample v0 <source
```

processes the menu "sample" as before, but this time the menu choice is read from the "redirection" file **source**.

There are some important rules to consider when using input redirection with a menu:

1. When menu input is redirected, it *stays* redirected until the end of the file is reached, the top level menu is exited, or the _CALLCLOSE command is executed. This applies even through subsequent _CALLMENU commands! Thus, if the "sample" menu in the example above had a submenu, the submenu would read the next line from **source**, and so on.

2. When the end of the redirection file is reached, or the top level menu is exited, or _CALLCLOSE is executed, menu input reverts to the user's keyboard.

3. Each _CALLMENU reads exactly one line from the input redirection file.

4. If the redirect file does not exist, or if the _CALLMENU command did not name a file, input is taken from the user's keyboard.

5. While menu input is being redirected, it may not be redirected again. Consider the example below:

   ```
   _callmenu sample  v0 <source
   ...
   _callmenu submenu v1 <junk
   ```

   Suppose **source** contains at least two lines of text. The "_callmenu sample" command reads the first line from **source**. Then the "_callmenu submenu" reads the second line from **source**, *ignoring the request to redirect input from **junk***. When in doubt, use a _CALLCLOSE after the last _CALLMENU that you expect to read from the redirect file.

6. While menu input is redirected, the "SM" part of a menu is *not* executed. This is very important. Since the SM menu macro is usually used to display the text of the menu, this feature means that redirecting menu input can be used to silently navigate through a set of stacked or layered menus.

Menu input redirection can also be used to perform "if—then—else" decision-making in Caucus macros. For example, the macro code fragment below uses the conference name to decide which banner file to display on entry to a conference.

```
:M_enter_conference
  echo $(confname)     >temp
 _callmenu B_choose v0 <temp
#
:B_chooseSM
:B_chooseCH  table
```

```
  demonstration  salon/cafe
:B_chooseMM  table
 B_nothing
 B_nothing
 B_banner3
 B_banner1
 B_banner2
#
:B_chooseRP
 1 1 1 1
#
:B_nothing
#
:B_banner1
 echo -s B_banner1.txt
_return
#
:B_banner2
 echo -s B_banner2.txt
_return
#
:B_banner3
 echo -s B_banner3.txt
_return
#
```

M_enter_conference is an auto-fire macro that executes automatically on entry to any conference. It writes the conference name into file **temp**, and then immediately executes the menu B_choose, reading the menu input from **temp**.

If the menu input (the conference name) is "demonstration", Caucus executes B_banner1, which displays the dictionary string B_banner1.txt. If the conference name is either "salon" or "cafe", Caucus executes B_banner2, which displays the string B_banner2.txt. For any other conference, Caucus executes B_banner3.

7.12 Auto-Fire Macros

To start a macro in earlier versions of Caucus, you typed its name at "AND NOW?". Version 2.2 adds a set of predefined "auto-fire" macros. Each of these macros is automatically executed (or "fired") at a specific time. By writing your own definitions for these macros, you can considerably extend the actions that Caucus performs at those times.

Each auto-fire macro has a specific fixed name that describes when it is executed. The names and execution times of these macros are listed below.

_begin_caucus is executed just once at the very beginning of Caucus, just before **_over_caucus** but after **_register_user**.

_over_caucus fires once just before the very first "Join which conference?", and also after a user leaves a conference with STOP.

_enter_caucus fires when entering a conference, just before the greeting and status displays. This macro is executed only once per Caucus session.

_enter_conference fires immediately on entering a conference, and before the START-MENU macro is executed. The default definition of **_enter_conference** displays the greeting and status information.

_exit_ambconf fires when a user tries to join a conference by typing an ambiguous conference name.

_exit_cantjoin first when a user tries to join a conference, but is excluded from becoming a member.

_exit_caucus fires just before Caucus exits. It is the very last thing that Caucus does.

_exit_conference fires when leaving a conference, just before logging the user off that conference (if logging is enabled). The default definition of **_exit_conference** displays the "You are now leaving ..." message.

_exit_noconf fires when a user tries to join a non-existent conference.

_exit_nojoin fires when a user is asked if s/he wants to join a conference, and declines.

_beg_help fires when the user has asked for help, before displaying the help text.

_end_help fires after displaying all of the help text.

_beg_add fires when adding an item, response, or message, and before selecting the object to be added.

_end_add fires after the new item, response, or message has been added, and just before the next "AND NOW?" or menu prompt. **_beg_add** and **_end_add** do not apply to responses added from the "RESPOND, PASS, or ?" prompt.

_beg_showitem fires when Caucus displays an item for the user, just before displaying the item header.

_mid_showitem fires just after item text is displayed, but before any responses are shown.

_end_showitem fires after the user has selected PASS at the "RESPOND, PASS, or ?" prompt.

_beg_showmess fires when showing a message, just before displaying the header.

_end_showmess fires when showing messages, after each message is displayed, but before the "REPLY, PASS, DELETE..." prompt.

_beg_searchitem fires when searching items or responses, just before asking for the range of items to search.

_end_searchitem fires after a search, just before returning to "AND NOW?".

_beg_entertext fires just before getting the first line of text from a user for an item, response, or message.

_end_entertext fires after the user has finished entering text (and if SET FORMAT is ON, after it is reformatted).

_register_user fires after a new user registers with Caucus, but before joining the first conference. This happens before **_begin_caucus**.

_delete_person fires after deleting a person from a conference, but just before confirming that the person has been deleted.

_new_user fires after a user has joined a conference for the first time, but before **_enter-_caucus**.

In general, please restrict the commands used by these macros to STOP, SET, STATUS, ECHO, !, _ASSIGN, or _CALLMENU. Other combinations of commands may confuse Caucus. For example, if **_beg_add** uses the ADD command, Caucus may loop until it runs out of memory and then fail.

The **_begin_caucus** and **_over_caucus** macros may also use the JOIN and _CHECK commands. **_Enter_caucus**, **_enter_conference**, and **_exit_conference** may use any command.

When Caucus is run, the autofire macros are executed in this order: **_register_user** (if a new participant), **_begin_caucus**, **_over_caucus**, **_enter_caucus**, **_enter_conference**, and then the user's STARTMENU if they have one defined.

7.13 Prompt Macros

The notion of "auto-fire macros" was extended in version 2.5 of Caucus to include all of the standard Caucus prompts. This means that whenever Caucus is about to issue one of its standard prompts and then ask the user for input, you may direct it to run a specific macro first. These macros are called "prompt macros".

For example, to run a macro just before each "AND NOW?" prompt, follow the steps below. As usual, make all changes or additions to your local000 or equivalent file.

1. Change the first word of the string dic_Aproperties to be "1". (This "turns on" the use of prompt macros for the current dictionary. Since there is some processing overhead when prompt macros are used, the default condition is "off", i.e., prompt macros are not used.)

2. Add a string called "mai_Pandnow.pm". (This is the name of the prompt string used to display the "AND NOW?", plus a ".pm" at the end to indicate that this is the prompt macro string.)

3. Put the commands that you want executed in the mai_Pandnow.pm string.

(If you actually are adding a prompt macro to run at "AND NOW?", you should repeat steps 2 and 3 for the string "mai_Pandhelp", the verbose form of "AND NOW?".)

Note that after the prompt macro is executed, Caucus will still display the usual prompt string and request input from the user. If the prompt macro displays text, you may wish to "empty out" the regular prompt string so that it displays nothing.

In general, any prompt string of the form "xxx_Pyyy" may have an associated prompt macro. You also may add prompt macros for the strings res_Fitemp, mes_Fdisp, and mes_Foksend.

7.14 Other Commands for Macros

There are several other commands that are intended specifically for use in macros or menus. Like _ASSIGN and _CALLMENU, they all begin with a leading "_" to distinguish them from commands meant for the user.

7.14.1 _LISTCONF

_LISTCONF has only one function: it lists the conferences available to the user without leaving the current conference. Prior to version 2.2 of Caucus, the only way to see this list was to leave a conference to use the JOIN LIST command.

7.14.2 _WELCOME

_WELCOME displays the text of the conference greeting. This is the text created by the conference organizer with the CUSTOMIZE GREETING command.

7.14.3 _LOAD_STATUS

Each time Caucus displays the "AND NOW?" prompt, it also automatically updates its knowledge about new messages, items, and responses that were entered by anyone since the previous "AND NOW?" prompt. This knowledge is called the conference status. The status is not, however, automatically updated at each *menu* prompt. The _LOAD_STATUS command forces Caucus to update the conference status.

In version 2.3, the macro code for Caucus main (level 1) menus contain a _LOAD_STATUS command. This means the conference status is updated each time the user sees the main menu.

7.14.4 _SHOW_STATUS

_SHOW_STATUS displays the traditional Caucus conference status information, as shown in the example below:

```
New items are:           5
Unseen items are:        1
New responses on items:  2  4
```

7.14.5 _LIST_STATUS

_LIST_STATUS displays a summary of Caucus conference information. It has two forms:

```
_list_status

_list_status  stringname
```

The first form displays a one-line summary of the Caucus conference status information. It is used in the Caucus executive menus to display a status line before the text of each menu. The format of this display is controlled by the dictionary string sys_Foneline.

The second form displays text and status information according to the tailorable format string *stringname*. Tailorable format strings are described in more detail in chapter 6.

7.14.6 _CHECK

_CHECK displays, for each conference the user belongs to, the number of new items and responses in that conference. It has exactly the same effect as the caucuscheck (**cauchk_x**) program, except that the functionality has been embedded directly in Caucus.

7.14.7 _LOG

The _LOG command extends Caucus' "LOGFEATURE" logging capability, which is described in sections 2.5 and 2.7. The _LOG command allows the macro writer to log custom "features" beyond those included in Caucus.

The syntax of the _LOG command is:

_LOG feature value ivalue starttime count size

The meanings of these fields are described in section 2.7. The "starttime" field is the time at which the feature began, in absolute number of seconds. It is assumed that the feature ended when the _LOG command was executed.

7.14.8 _CMI

The _CMI command forces Caucus to go and look at the user's SET IMPORT_MAIL option. If it is set to COPY or IMPORT, Caucus immediately goes out and imports the mail from their system mailbox.

Caucus normally imports mail only when it starts up. (The auto-fire macro _enter_caucus executes _CMI.) If your users want to check their system mailbox more frequently, you may wish to add _CMI to other auto-fire macros, or add another macro that explicitly runs _CMI.

(See section 9.2.6 of the Caucus User's Guide for more on SET IMPORT_MAIL.)

7.14.9 _BREAK

The _BREAK command is used to cease execution of a macro. In the macro below, for example, only "A" is displayed.

```
:M_example
echo A
_break
echo B
```

7.14.10 _RESTRICT

The _RESTRICT command is followed immediately (on the same line) by any Caucus command. _RESTRICT tells Caucus to execute the command with all of the current restrictions for this user, i.e. as if the user had typed the command at "AND NOW?".

_RESTRICT is a good way to close possible security loopholes in macros. Commands inside a macro normally can do things that the user may not be allowed to do, such as redirect output to a file or execute shell (operating system) commands.

If a command inside a macro gets built out of user text or user-typed arguments to a macro, the resulting command may do things that the macro writer did not intend (such as run an arbitrary operating system command).

By prepending such "built" commands with _RESTRICT, the macro writer can close all such possible loopholes.

For an example of the use of _RESTRICT, see the executive menus in the dictionary file **exec0**.

Chapter 8

The Kermit Macros

8.1 Introduction

Caucus is distributed with a set of macros that automate uploading and downloading text between a conference and a user's personal computer. One group of these macros uses Kermit, a public domain file transfer program, to ensure error free transfer of text.

The Kermit macros are also good examples of how to use the principles and tools described in the previous chapter to create effective macros. To that end, the rest of this chapter describes the use and implementation of these macros.

8.2 Using the Kermit Macros

The Kermit macros are called KERMITSHOW, KERMITADD, and KERMITSEND. KERMITSHOW is analogous to the Caucus SHOW command. Instead of displaying text on the screen, it downloads the selected text to the user's personal computer. KERMITADD and KERMITSEND are like ADD and SEND, respectively. They move text in the opposite direction, from files already prepared on the user's personal computer into items, responses, or messages on the Caucus host computer.

8.2.1 Using Kermitshow

To download conference text with KERMITSHOW, the user must:

1. Have the Kermit program or a terminal emulation package that supports the Kermit file-transfer protocol on the personal computer.

2. Type KERMITSHOW followed on the same line by a list of *objects* and *instances*, just like the Caucus SHOW command.

3. Switch the personal computer from terminal emulation to Kermit receive mode (see the documentation for the terminal emulation program).

4. Wait for the file transfer to finish.

5. Switch the personal computer back to terminal emulation mode.

8.2.2 Using Kermitadd and Kermitsend

To upload previously prepared items, responses, or messages, the user must:

1. Type KERMITSEND, KERMITADD ITEM, or KERMITADD RESPONSE followed by an item number.

2. Send the prepared file from the personal computer via Kermit (see the documentation for the user's terminal emulation program).

3. Wait for the file transfer to finish.

4. Switch the personal computer back to terminal emulation mode.

8.3 Kermit Macro Definitions

The definitions of the Kermit macros are contained in the dictionary file **sysmac0**. The implementations of KERMITSHOW, KERMITADD, and KERMITSEND for a typical Unix system are described in detail below.

8.3.1 Kermitshow Definition

Here is the dictionary string M_ckermitshow, followed by a description of each line of the macro:

```
:M_ckermitshow
show $2 $3 $4 $5 $6 >$(userid).t
echo Switch to your local Kermit, and
echo tell it to receive a file.
downasciikermit      $(userid).t
_sys_delete          $(userid).t
```

`show $2 $3 $4 $5 $6 >$(userid).t`
The macro arguments $2 through $6 (see section 7.4) are replaced by the rest of the words in the KERMITSHOW command typed by the user. The resulting output is not displayed on the screen, but is placed in the temporary file $(userid).t.

The expression "$(userid)" is a predefined variable (see section 7.5) that is replaced by the person's userid. For example, if the person's userid is "pat", then the temporary file is named "pat.t". If Caucus is running in "captive" mode (see chapter 2), then many people may be creating temporary files in the same directory. This means that the names of these files *must* be unique to each user.

`echo Switch to your local Kermit...`
This tells the user to begin receiving the file on the personal computer.

`downasciikermit $(userid).t`

The KERMITSHOW macro uses yet another macro, called "downasciikermit", to transfer the temporary file. KERMITSHOW is defined this way, so that it can be constant across many different operating systems. Only the subsidiary macros, such as "downasciikermit", need to be different from system to system.

`_sys_delete $(userid).t`
Finally, when the transfer is complete, another subsidiary macro called "_sys_delete" is used to delete the temporary file. Macros should always clean up after themselves.

Here are typical Unix definitions for the subsidiary macros "downasciikermit" and "_sys_delete":

```
:M_downasciikermit
!$(confid)/BIN2/ckermit -s $2
#
:M_sys_delete
!rm -f $2 $3 $4 $5 $6 $7 $8 $9
```

Note that "downasciikermit" uses the predefined variable "$(confid) to form the full Unix pathname of the ckermit program distributed with Caucus. The "-s" option tells ckermit to send the file named in the "$2" argument.

In "_sys_delete", "rm" is the Unix command to remove (delete) a file, and the "-f" option means "do it silently".

8.3.2 Kermitadd Definition

Here is the dictionary string M_kermitadd, followed by a description of each line of the macro:

```
:M_kermitadd
_sys_delete $(userid).t
echo Switch to your local Kermit, and
echo send your file now...
upasciikermit       $(userid).t
add $2 $3 $4 $5 $6 <$(userid).t
_sys_delete         $(userid).t
```

`_sys_delete $(userid).t`
This macro begins by deleting the temporary file, to be absolutely certain that the transfer starts with a clean slate.

`echo Switch to your local Kermit...`
This tells the user to begin sending the file from the personal computer.

`upasciikermit $(userid).t`
Another subsidiary macro actually receives the file.

`add $2 $3 $4 $5 $6 <$(userid).t`
The macro arguments $2 through $6 are replaced by the rest of the words typed by the user in the KERMITADD command. The ADD command then reads the text in from the temporary file, and places it in a new item or response.

```
_sys_delete $(userid).t
```
Once again, clean up afterwards.

KERMITADD uses one new subsidiary macro, which looks like this:

```
:M_upasciikermit
!$(confid)/BIN2/ckermit -ra $2
```

Note that "upasciikermit" is just like "downasciikermit", except that is uses the "-ra" option. This tells Kermit to receive a file and give it the name supplied in the $2 argument.

8.3.3 Kermitsend Definition

Finally, here is the dictionary string M_kermitsend:

```
:M_kermitsend
_sys_delete $(userid).t
echo Switch to your local Kermit, and
echo send your file now...
upasciikermit $(userid).t
send $2 $3 $4 $5 $6 <$(userid).t
_sys_delete    $(userid).t
```

The KERMITSEND macro works just like KERMITADD, except that it uses SEND instead of ADD.

8.4 About Kermit

Kermit was developed at Columbia University, and is distributed with Caucus, free, with their permission. Kermit is available for a wide variety of computers for a nominal media fee from Columbia University. For more information, contact your local computer users' groups or write to:

Kermit Distribution
Columbia University Center for Computing Activities
612 West 115th Street
New York, NY 10025

Columbia University notes, "Please use Kermit only for peaceful and humane purposes".

Chapter 9

Integrating Caucus with Other Applications

9.1 Introduction

By itself, Caucus is a very powerful tool for handling conferencing and electronic mail. As the previous chapters have shown, you can customize how Caucus performs these functions in a wide variety of ways.

You can also extend the range of the *kinds* of functions Caucus can perform by integrating it with other software applications. There are two approaches to doing this:

Caucus as master. This means that Caucus is the top-level program, the one that the user sees and interacts with. Caucus in turn may tell the operating system to run another application and then come back. This is the most common approach.

Caucus as slave. This means that some other application interacts with the users. That application may in turn run Caucus with a prepared script, and bring the results from Caucus back to the user. This is rarer, but occasionally useful.

9.2 Caucus as master: the Vote Counter

This section describes a fairly simple application, called a vote-counter, that could be integrated under Caucus (that is, with Caucus as the master). In this example, the vote-counter "application" is built from a text editor and some operating system scripts—but it could just as easily have been a sophisticated parser or database program.

Here is the basic idea: we want Caucus users to be able to discuss a topic in an item, and vote on a conclusion. Caucus will handle getting the votes and displaying the results, and the outside application will actually count up the votes.

For the purpose of this example, a vote must be either the word YES, or the word NO, all in capital letters. Presumably the author of the item will describe the question to be voted on, and describe just what a YES or NO vote means. Each member of the conference may then add responses discussing the question. To vote, they must add a response containing the word YES or NO. (Any other text in the response is ignored by the vote-counter.)

To add up the results of the voting, a new macro called TALLY is created and added to the common dictionary. (See sections 3.3.6 and 3.4 for details on how to add a new macro to the

dictionary.) The definition of TALLY is shown below. The example assumes that the operating system is Unix, but a similar example could be written for most operating systems.

```
:M_tally
show resp $2 >$(userid).r
!$(confid)/tally.sh  $(userid)  $2
<$(userid).scr
```

To start this macro, you would type TALLY followed by the item number at "AND NOW?" or a menu prompt. TALLY places the text of the responses into a temporary file, invokes the vote-counting application (**tally.sh** in the Caucus directory) on that file, and then processes the Caucus commands that the application put in the file $(userid).scr.

The real work is done by **tally.sh**. It scans the temporary file for all lines that contain YES or NO, counts the lines, and then creates Caucus commands that display the vote summary and add the summary as a response to the item. Here is a Unix shell script for **tally.sh**:

```
:
y='grep YES $1.r/wc/ (read a b; echo $a)'
n='grep NO   $1.r/wc/ (read a b; echo $a)'
if test $y = 0 -a $n = 0; then
   echo echo There are no votes on item $2. >$1.scr
else
   echo "Yes votes: $y"      >$1.t
   echo "No  votes: $n"     >>$1.t
   echo echo             \<$1.t   >$1.scr
   echo add resp $2 \<$1.t  >>$1.scr
fi
```

Tally.sh is passed two arguments, called $1 and $2 in the shell script shown above. $1 is the user's userid. It is used in the script as part of the name of some temporary files. For example, if the userid is "camber", then $1.t is **camber.t**, and $1.scr is **camber.scr**. $2 is just the item number.

The first "grep" line searches for all lines that contain "YES", and counts them, placingthe result in the variable "y". The next line puts the number of "NO" votes in N.

The rest of **tally.sh** writes Caucus commands into a Caucus script file $1.scr. The "if" line checks if both Y and N are 0. If so, it uses the Unix "echo" command to write

```
   echo There are no votes on item $2.
```

into $1.scr. Otherwise, **tally.sh** writes the vote summary into the temporary file $1.t, and writes Caucus commands that display the text in $1.t and add it as a new response.

This version of the TALLY macro is rather primitive, and has several limitations. It only understands "YES" and "NO" votes, and does not ensure that each person only votes once. It cannot distinguish "YES" and "NO" from, say, "YESTERDAY" or "NOTE". It does, however, illustrate some of the basic principles of integrating a "slave" application with Caucus:

1. Put any Caucus data to be analyzed into a file. (Small amounts of data can be passed as arguments—the way the item number is passed to **tally.sh** in M_tally.)

2. Use a macro (or menu choice or auto-fire macro) to invoke the application on that file.

3. Use if/then capabilities in the application, the operating system, or in a Caucus _CALLMENU to make decisions based on the data.

4. Put the application's results in a file.

5. Based on the results, the application can also make up commands for Caucus to perform, by putting them in another file.

Note, in particular, paragraph 3. The **tally.sh** script uses the "if/then" capability of Unix shell scripts to test how many votes were added to the item. It performs one action if no votes were added, and a different action if at least one vote was found.

9.3 Caucus as master: Caucus Mail Integration

This section describes how outgoing Caucus mail is passed on to Unix or Vax mail.

When a Caucus user sends mail to "=*address*", the mail is sent, not to a Caucus user, but to the system mailbox (or internet address) *address*.

In order to actually send the mail, Caucus performs the following steps:

1. Puts the text of the mail in a temporary file.

2. Builds a command string by concatenating the two dictionary strings mes_Fcmisend1 and mes_Fcmisend2.

3. Substitutes the message subject for the first "%s" in the command string. Substitutes the address for the second "%s". Substitutes the name of the temporary file for the third "%s".

4. Executes the command string as an operating system command.

5. Deletes the temporary file. (For this reason, the command should not exit or finish until the mail has been sent, i.e. do not "background" the mail program.)

All of these steps are built in to Caucus, i.e. this is not something that the local Caucus manager has to do. The default values for mes_Fcmisend1 and mes_Fcmisend2 build a command line that runs the script **cmi_send**, which also is distributed with Caucus.

The point of this example, however, is that Caucus is running cmi_send as a "slave" application. Caucus packages up the mail to be sent, and hands it off to cmi_send (which in most cases will in turn run the local mail program). The Caucus manager may choose to make Caucus do

something else with outgoing mail by changing the definitions of mes_Fcmisend1 and mes_Fcmisend2. The manager may even choose to have different Caucus dictionaries perform *different* actions on outgoing mail.

9.4 Caucus as Slave: an Update Mailer

This section describes an example of the opposite situation, in which Caucus is the "slave", and does not directly interact with the user.

For this scenario, imagine a need to deliver new material in a conference to someone on another computer. Perhaps this person is a former conference member who has moved away to another organization, and now can only be reached by remote electronic mail. Or perhaps we wish to post new material from this conference to an electronic mail newsgroup. For the sake of simplicity, we'll ignore the question of whether replies from the person or newsgroup should come back to the conference.

The goal is to regularly gather up the new material in a particular conference *for* a specific person, and then e-mail this material off to that person. To further simplify the example, we'll assume that this person has a "captive" userid—that is, a userid and password that Caucus controls. (See chapter 2 for information about the option file, and the CAPTIVE option in particular.)

To implement this example, define a command script called **updater** that is the "master". It uses Caucus and the operating system's remote electronic mail package as "slaves". Once again, this example uses Unix, although a similar script could be written for most other operating systems.

The **updater** script is run from the regular operating system prompt, as shown below:

```
updater  userid  password  conference  address
```

In other words, to run the script, type **updater**, followed on the same line by the captive userid, the password, the conference name, and the remote electronic mail address of the person receiving the update.

The Unix shell script **updater** might look like this:

```
:
echo "captive  on"             >updater.opt
echo "redirect on"            >>updater.opt
echo "fileinput updater.inp"  >>updater.opt
echo $1                        >updater.inp
echo $2                       >>updater.inp
echo $3                       >>updater.inp
echo "show resp new pass       >updater.out" >>updater.inp
echo "show item new pass      >>updater.out" >>updater.inp
echo "stop"                   >>updater.inp
/usr/caucus/BIN2/caucus_x /usr/caucus -y master.opt \
      -y updater.opt >/dev/null
```

```
mail $4 <updater.out
rm          updater.out  updater.inp  updater.opt
```

Updater begins by creating a secondary option file **updater.opt**. This file tells Caucus to use captive mode, to allow input/output redirection, and (most importantly) to read commands not from the user, but from the file **updater.inp**.

Next, **updater** creates the **updater.inp** script for Caucus. The script contains the lines that would normally be typed by a Caucus user. Each line corresponds to a Caucus prompt: $1 is the captive Caucus userid (corresponds to "newuser/login:" prompt); $2 is the password ("Password:"); and $3 is the conference name ("JOIN which conference?"). The next two lines, in response to "AND NOW?" (or a menu prompt) display the new material into **updater.out**. Finally, "stop" exits Caucus.

After all the preparation is complete, **updater** actually runs Caucus. The line beginning "/usr/caucus..." was taken from the usual Unix **cv2** script that runs Caucus, and modified slightly. It instructs Caucus to use the options in **master.opt** *and* **updater.opt**, and to throw away all terminal output into the "bit bucket" or "black hole", called /dev/null. As soon as Caucus is finished, **updater** runs the 'mail' program, taking the output generated by Caucus in **updater.out**, and mailing it to the address given as $4. Finally, the last line cleans up, by removing all the files that were just generated.

Note that the **updater** script must be run from the Caucus userid, since for security reasons all option files must live in the Caucus directory. (In this example it is /usr/caucus.) Once this script is set up, one of the Unix utilities such as 'cron' or 'at' could then be used to automatically run the script at regular intervals.

The principles behind running Caucus as a "slave" are similar to those for running Caucus as a "master", although in the reverse order:

1. The master takes data from the user or an application, and uses it to write a set of Caucus commands.

2. The master then runs Caucus with these commands, instructing it to place any important output into a temporary file. Any other output is usually thrown away, as in the **updater** example.

3. The master may proceed to test the data produced in step 2, and take different actions depending on the result. For example, **updater** could be modified to mail the output file only if it really contained any new items or responses.

4. Finally, the master actually does something with the data produced by Caucus.

Chapter 10

Turning Off a Caucus Feature

10.1 Why Remove a Feature

Adding new features is not the only reason to customize Caucus. For some applications, you may wish to remove one of the standard Caucus features.

For example, suppose that everyone on your computer system already uses an existing electronic mail package. Adding yet another e-mail program might confuse your users, so you may decide to remove the electronic mail ("MESSAGES") feature from Caucus. The rest of this chapter describes how to do this.

(Note that Screen Porch LLC.does not recommend removing the MESSAGES feature from Caucus. Version 2.3 of Caucus contains improved methods for integrating outside electronic mail with Caucus MESSAGES. As always, however, the choice of how to customize Caucus is yours.)

10.2 Removing MESSAGES from Menus

Caucus is distributed with four different user interfaces: the executive menus, the long and short teaching menus, and the command line interface. To remove the MESSAGES feature, you must remove it from each of the four interfaces. This section describes removing MESSAGES from the menu interfaces.

Removing MESSAGES from the executive menu interface is straightforward. Use your favorite text editor to make the following changes to the file **local000**. (For more details on the structure of menus, see section 7.10.)

- Copy the string O_MainMenu.T from **exec0** to **local000**. Delete the "5 - Messages Menu" choice, and renumber the remaining choices to fill in the gap. (The "People" menu becomes choice 5, and so on.)

- Copy the string O_MainMenu.H from **exec0** to **local000**. Delete the description of choice 5, and renumber the rest to fill in the gap.

- Copy the string O_MainMenuCH from **exec0** to **local000**. Without the Message menu selection, there are now only six choices remaining. Therefore, remove the seventh one, choice "7".

〈 Copy the string O_MainMenuMM from **exec0** to **local000**. Delete the "M_O_MainMenu.5" line.

Removing the MESSAGES feature from the teaching menus is very similar. You must make the same kind of changes to the "Main" menu. This means copying the strings MainTxt (and MainTxt.S), HelpMainTxt, MainCH, and MainMM from the file **menus0** to **local000**. Edit them in the same way, respectively, as you did the O_MainMenu strings.

The teaching menus also have "sideways" references to the messages menu. That means that you can get to the messages menu from the Items or Persons menus as well as from the main menu. Look at the "Disc" and "Persons" menus in the file **menus0**, and edit them in exactly the same way as you did the "Main" menu.

10.3 Removing MESSAGES from Command Line Interface

In the Caucus command line syntax, a MESSAGE is an object, like ITEMS, RESPONSES, or PERSONS. The name of each kind of object is kept in the dictionary string "objectable" in the file **america0**. You can effectively remove the MESSAGES feature by hiding it—that is, rename the word MESSAGES in that string to some other, preferably obscure, word.

To make this change, copy the entire "objectable" string from **america0** to the file **local000**. Edit **local000**, and change the phrase "messages/send" to, for example, "_xyzzy". This removes the word MESSAGES from the list of objects recognized by the Caucus verbs SHOW, LIST, DELETE, and so forth. (Warning: do not just delete the "messages/send" phrase. You must replace it with some other word.)

There is one Caucus verb that applies only to messages: the verb SEND. You must also "hide" this verb by renaming it. Copy the string "commands" from **america0** to **local000**, and change the word "send" to something obscure.

Once you have made these changes, no one will be able to send or receive messages. To reduce the potential for confusion or frustration, you should also remove all references to MESSAGES from the Caucus on-line help. Copy the strings:

```
add_Enoarg        del_Enoarg
des_Enoarg        sho_Enoarg
```

from **america0** to **local000**, and edit out the references to MESSAGES. Similarly, copy the strings:

```
h_commands        h_verbs
h_objects         h_quick
h_add             h_list
h_send
```

from **help0** to **local000**, and edit them appropriately.

To summarize the previous sections, removing a feature involves three key steps:

1. Identify the the dictionary string or strings that let Caucus recognize the name of the particular feature. Change the string to hide the feature.

2. Remove access to the feature from the executive and teaching menus by removing the feature's entry from the appropriate CH and MM tables.

3. Edit the on-line help text to remove any references to the feature.

10.4 Protecting a Feature with a Password

Instead of completely removing a feature, you may wish to restrict it. You can protect it so that only the people who know a special password can use the feature. Password protected features can be accessed only through one of the menu interfaces.

To protect a feature, you must first hide it from the command line interface, as described in section 10.3. Then change the relevant menu to use the "hidden" name of the feature. Finally, place a guard menu in front of the original menu to check for a password before calling the original menu. An example of this process is described in more detail below.

10.5 Protecting the MESSAGES feature

To password protect the MESSAGES feature, begin by hiding MESSAGES from the command line interface as described in section 10.3. Then edit the menu interfaces to use the new, obscure, name for the object MESSAGES. In the executive menus, this means changing all of the menu macros that use SEND to use _XYZZY, and all of the menu macros that use LIST MESS, SHOW MESS, or DELETE MESS to use LIST _XYZZY, SHOW _XYZZY, and DELETE _XYZZY instead. (M_O_Message.4 and M_O_ListMess.2 are examples of such menu macros.) As usual, this means *copy* those strings from **exec0** to **local000** and edit them there.

The effect of this step is to hide MESSAGES from the command line interface, but allow them to be used from the executive menus.

The second step inserts the guard menu that checks for the password. Change the string M_O_MainMenu.5 so that it calls the menu PassMess instead of the menu O_Messages. Then add the definition of PassMess as shown below:

```
:PassMessSM
echo -n Enter the Messages password: $
#
:PassMessCH table
secre secret
#
:PassMessMM table
M_return1
M_return1
M_return1
M_return1
AllowMess
#
:PassMessRP
1 1 1 1 1
#
:AllowMess
_callmenu  O_Messages  v0
return -1
```

When the user is at the main menu and selects the Messages menu, Caucus first calls PassMenu and prompts for a password. In this example, the required password is the word "secret". If the user presses <RETURN> or <CANCEL>, or types the wrong password, Caucus returns to the main menu. Typing any of the substrings of "secret", that is "s", "se", "sec", "secr", or "secre", matches the first word in PassMessCH and Caucus *still* returns to the main menu. Only if the user types "secret" does Caucus match the second word in PassMessCH and invoke AllowMess, which in turn calls the real Messages menu.

Chapter 11

Translating Caucus into Other Languages

11.1 Introduction

Translating the Caucus interface into another language is both simple, and challenging. Simple, because it requires no programming — just translating the text of the dictionary files with an ordinary text editor. Challenging, because there are over 200,000 characters of text to translate, and all changes must be internally consistent.

This chapter describes the rules to consider when translating Caucus into another language. The same rules apply to any large-scale rewriting of the text in the same language. Caucus has already been translated into several languages: contact Screen Porch LLC.for more information.

11.2 Character Sets

The American English version of the Caucus dictionary contains only printable ASCII (or EBCDIC) characters. This is not, however, a requirement. You can put *any* characters (except a zero byte) into the dictionary and Caucus will display it faithfully. Be aware that your users may have different kinds of terminals that may react in different ways to certain non-printing characters, such as ASCII "ESC".

If your translation uses characters outside the standard printable ASCII set (character values octal 40 through 176 inclusive), you will need to use the EIGHTBIT option in chapter 2.

If you are using the Japanese version of Caucus, the "language" feature can be used in two forms. They both control the way that the dictionary compiler constructs individual characters. These two forms can be mixed within one dictionary, even within one dictionary file.

11.2.1 The String Directive

The first form is the "string directive". This is a modifier to the usual way of specifying a string. It tells the compiler to construct the string according to a particular data representation.
For example, to tell the compiler to generate this string in the EUC representation, use:

```
:sys_Tmeans10 language=euc
How to stop entering text
```

For both the "string directive" and the "global directive" the word after the "=" must be one of:

```
ascii
euc
sjis
```

11.2.2 The Global Directive

The second form is the "global directive". It is a stand-alone directive to the dictionary compiler. It instructs the dictionary compiler to construct *all* of the strings in this dictionary file, after the directive, according to the specified data representation.

For example, to tell the compiler to generate all of the following strings in this file in SJIS form, use:

```
: langauge=sjis
```

The colon is necessary in the first column. It must be followed by one space before the word "language".

There are two other rules about the "global directive":

1) This data representation *will* be used in dictionary files "included" by this file.

2) If an "included" file uses this form to change the data representation, that representation will *not* be applied after returning from the "include". For example, here are the listings from the two files, **A** and **B**. File **A** includes
File B.

A:
:stringA
This is string A.

:<B

:stringD
This is string D.

B:
: language=sjis

:stringB
This is string B.

:stringC
This is string C.
In the above example, strings "stringA" and "stringD" will be in the default data representation. Strings "stringB" and "stringC" will be in the SJIS representation.

There is no limit to the number of "string directives" and "global directives" you can use.

11.3 Preparing for Translation

This section describes how to prepare a set of Caucus dictionary files for translation. Please follow the steps below carefully.

First and foremost, make a backup copy of *all* of the original dictionary files. Put the backups on tape or floppy disk or some other medium where they cannot be easily erased. This is extremely important — later you will need these files to keep your dictionary compatible with later releases of Caucus. See section 11.5 for details.

Next, make copies of the original dictionary files, with different names. For the sake of example we'll name the copies so that they end in "1" instead of "0". The copies would be called **diction1**, **america1**, **help1**, and so on.

Now edit the **diction1** file. Change all instances of "<local000" to "<local001". There should be three of these, one for **local000**, one for **local000.nam**, and one for **local000.ptr**.

Finally, create the file **local001**. Edit it, and insert the lines shown below:

```
:<help1
:<america1
:<sysmac1
:<menus1
:<exec1
:<macros1
```

All of your changes and translations must be made to the files ending in "1". When you compile your translated dictionary with cv2mkmd (see section 3.5) you will compile **diction1** instead of **diction0**. At that time, you will notice that the dictionary compiler processes all of the "1" files, *followed in turn* by the "0" files. This is intentional. It ensures that any strings accidentally deleted in the translation process, or any new strings added in a Caucus upgrade, will still become part of the compiled dictionary.

11.4 Translating Dictionary Strings

All of your changes and translations must be made to the copies of the original dictionary files (the "1" files in the example in the previous section). Before making *any* changes, you ***must*** read Chapter 3 of this manual.

The rest of this section describes what strings can be changed, how they can be changed, and why. The description of the strings is arranged by file, using the "1" filenames from the previous section.

11.4.1 Sysmac1 Strings

The file **sysmac1** contains the definitions of all system-dependent macros. For the most part you must leave the definitions alone, although you may change any text that the Caucus "echo" commands display to the user.

11.4.2 Help1 strings

The file **help1** contains much of the Caucus help text. Do not change the string "helpptrs" except to add new help text as described in sections 3.3.3 through 3.3.5. You may change the words in "helpnames", but do not change their order. The remaining strings are all plain help text and may be changed as you see fit — but make sure that the help text matches the actual behavior of your translated Caucus!

11.4.3 Macros1 strings

The file **macros1** contains the definitions of the regular macros, those that are the same for all operating systems. Like **sysmac1**, you should normally change only text that will be displayed to the user. Exception: if you change the names of ordinary Caucus commands (such as ADD, DELETE, and so on) in **america1**, then you must also change them wherever they are used — such as in the macros. This is discussed further in section 11.4.12.

Macros1 also contains the paired tables "editnames" and "editptrs" which control which text editors may be used with Caucus. For more information on how to use these tables, see chapter 5. For more information on paired tables, see section 3.3.6.

11.4.4 Menus1 Strings

The file **menus1** holds the long and short-form "teaching" menus. The text of these menus are contained in the strings whose names end in "Txt" and "Txt.S". Note that many of the menu selections may be chosen by typing a number *or* a keyword. If you change the name of the keyword(s) in a "Txt" or "Txt.S" string, you must also change the name of that keyword in the matching "CH" string. For more information about menu strings, see sections 3.3.7 and 7.10.

11.4.5 Exec1 Strings

The file **exec1** contains the "executive" menus. The text of these menus are in the strings whose names end in ".T". All of the selections in the executive menus are chosen by number, so you should only need to change the menu text.

11.4.6 Diction1 Strings

Diction1 contains only two strings: the paired tables "macronames" and "macroptrs". Each name of a macro in "macronames" corresponds to the name of a macro definition (or "pointer") in "macroptrs". The first entry in "macronames" corresponds with the first entry in "macroptrs", the second with the second, and so on.

You may change the name of a macro in "macronames", but do not change its pointer, or the one-to-one correspondence between macro names and macro pointers. If you do change the name of a macro, be careful to change its name everywhere it is used — such as in other macros and in the menus.

Instead of changing the name of an existing macro, another approach is to add a synonym for the macro. For example, suppose the strings in **diction1** look like this:

```
:macronames
new  dnb  lnb  ...
#
:macroptrs
M_new  M_dnb  M_lnb  ...
```

Suppose you want to translate the "new" macro into French and call it "nouveau". Rather than change "new" to "nouveau" (and then find and change all uses of the "new" macro), add "nouveau" as another macro with the same definition as "new". The strings now look like this:

```
:macronames
new  nouveau  dnb  lnb  ...
#
:macroptrs
M_new  M_new  M_dnb  M_lnb  ...
```

Now both "new" and "nouveau" may be used, and have exactly the same effect.

11.4.7 America1 Strings

America1 contains many different kinds of strings, and figuring out which ones can be changed may at first seem a daunting task. Fortunately, the names of most strings follow a pattern that describes their purpose. See section 3.3 and the header comments in **america1** for more information on this pattern.

In particular, all of the strings whose names contain "_E", "_T", "_P", and "_H" are pure text strings, and may be changed in any way that you like. (*Exceptions*: sys_Tset0 and sys_Tset0cms. See section 11.3.9.)

11.4.8 America1 _F Format Strings

The format strings in **america1** all contain "_F" as part of their name. These strings control the precise layout of certain Caucus output. The format strings mix plain text with special format codes such as "%d" and "&x". You may change the text in these strings as you like, but do not

change the "%" codes or the order in which they appear. For information about the purpose and use of the "&" codes, see Chapter 6.

11.4.9 America1 _A Table Strings

Another class of strings contain "_A" as part of their names. These are "table" strings. Most of them contain tables of the allowed responses to various Caucus prompts. You may change or translate the words in those tables, but you must retain their order and meaning.

Some of these table strings have very specific entries that must not be translated. These include the strings ech_Aopts, gen_Acomlin, mai_A822flds, and mai_Ainex.

The entries in the table gen_Aopttab are the **master.opt** keywords. They may be translated, but only if the contents of **master.opt** are also translated. Otherwise the options in **master.opt** will not be recognized and will be ignored.

11.4.10 America1 sys_Acollate

The "sys_Acollate" string was removed in version 2.5. As of this writing, there is no replacement. 8 bit characters are stored in their natural collating order.

11.4.11 America1 _K Keyword Strings

The "_K" keyword strings are used for a variety of special purposes. This section describes each _K string and its purpose.

The most common use of an "_K" string is to help parse a Caucus command. Any "_K" strings that are used only for command parsing may be translated freely. These strings are listed below. Note that while you *may* translate any of these strings it does not follow that you *should* translate them. In particular, strings like mac_Knames, that are never seen or typed by the user, are best left alone.

chg_Kminus chg_Kplus	Used to parse long form of CHANGE SUBJECT, as in: CHANGE SUBJECT "IDEAS" + ITEM 5.
sch_Kfor	Used to parse "FOR" in: SEARCH ITEM 5 FOR "CAUCUS".
gen_Kprint1 gen_Kprint2	Used to parse "PRINT" in: SHOW ITEM 4 PRINT.
for_Knone	"None" response to prompt from FREEZE and FORGET commands.
mac_Knames	Contains the names of the predefined Caucus variables. (See section 7.5 for details.)

Another set of strings is used to expand the short forms of certain Caucus commands into their more precise long forms. Des_Kdescitem expands the potentially ambiguous "LIST 5" into "LIST ITEM 5". Sho_Kshowitem does the same thing for "SHOW 5". The first word of these two strings must match the equivalent command word in the "commands" string. The second word must be an object type from the "objectable" string.

Similarly, the string mai_Kjoinstop turns the STOP command into "JOIN STOP", which is how Caucus actually exits. The first word of this string must match the 14th word of "commands". The second word must match the first word of "mai_Ajoin".

The contents of string for_Knof must match the fifth word of "gen_Aparse". It is used in processing the FORGET command.

Ced_Kquest is the text input prompt for the Caucus editor that appears at the beginning of every line.

Sys_Kdel and sys_Kctrl contain plain text used in displaying the name of the <CANCEL> key.

Cst_Kinexdef contains the default text of a USERLIST for a conference that is completely open to anyone to join.

Peo_Kpunct contains a list of punctuation characters. These characters, and only these characters, are ignored when Caucus tries to match a word typed by the user against a person's name, an item title, or a subject category. For example, suppose that peo_Kpunct contains a "(", and there is a person registered as "Abraham (Honest Abe) Lincoln" Then either of the commands "SHOW PERSON (HONEST" or "SHOW PERSON HONEST" would display the information about that person.

11.4.12 America1 "short strings"

There is a special set of frequently used strings that Caucus keeps resident in its memory. They all begin with "ss_", which stands for "short string". These strings are very short, often only a few characters long. They are used for several purposes, including keyword and key-character matching, and output punctuation.

Most of these strings should not be changed or translated. The few that may be changed are:

ss_caucus This is the name of the built-in Caucus line editor, as used in SET EDITOR. If the default editor is the Caucus editor, the fifth word in sys_Tset0 must match this word. Otherwise, the Caucus editor can not be used.

ss_inprom When the user is entering the text of an item, response, or message, Caucus displays this prompt at the beginning of each line.

ss_all The keyword "all" is used to detect references to ALL PERSONS, ALL SUBJECTS, or ALL ITEMS. It must match the equivalent word in gen_Aparse.

ss_pause This text is displayed when Caucus pauses after a screenful of text.

ss_abort The user may type this keyword at the ss_pause prompt to abort the further display of text.

ss_cmiesc This is the prefix character that identifies a userid to CV2KILL, and a remote mail address at "ENTER NAME OF RECIPIENT". It must be exactly one character long.

11.4.13 America1: Other Tables

The eight tables that remain are so important that they have unique names. They may all be translated or changed, but great care must be taken in doing so as such changes may affect macros, help text, and other strings. These tables are named and described below.

The "stopcoms" table contains all of the words that may be used to exit Caucus (such as STOP, EXIT, QUIT, and so on). Note that, unlike most commands, these words must be typed in full to exit Caucus.

The "commands" table contains all of the Caucus commands, or verbs. Do not change the order of the words in this string! As in the example in section 11.4.6, rather than replacing the words in this table with their translations, you may decide to add the translations as synonyms. However, since this is not a paired table (as macronames and macroptrs are) you must use a different method. To provide a synonym for a command, replace the command name with a list of the command name and its synonyms separated by slashes. For example, to make READ a synonym for the SHOW command, replace "show" in "commands" with "show/read".

The "objectable" table is the list of objects or nouns that Caucus understands. It follows the same format and rules as the "commands" table . Each object may also have synonyms in the same format as in the "commands" table.

The "ced_table" table contains the commands for the built-in Caucus line editor. It also follows the format of the "commands"table, including synonyms.

The "yesnotable" table contains the list of valid responses to any Caucus prompt that requires a yes or no answer (such as "DELETE THIS ITEM?"). The first word in this table must mean "no", and the second and third words must mean "yes". As distributed, "yesnotable" contains "no yes ok".

The "onofftable" table contains the words for "on" and "off" as used by the SET command. The "optonoff" table contains the words for "on" and "off" as used by the Caucus options files (see chapter 2). The contents of optonoff must agree with the words you use in your option files, or the options diles will not be processed correctly.

The "set_Aorder" table controls the order in which the SET options are displayed when the user types SET, with no option specified. You can change set_Aorder to make the display appear in alphabetical order.

11.5 Updating Your Translation

As Caucus evolves, new strings are added to the dictionary, and the precise meaning of old strings may also change. In order to keep your translated dictionaries up to date and compatible with new releases of Caucus, there are two important steps you should follow:

1. *Follow the directions in section 11.3.* This is very important. Otherwise your translated dictionary may not work with new releases of Caucus. This applies to any significant rewrite of the Caucus dictionary, as well as to translations to other languages.

2. *Read the history of changes*. The comments in each dictionary file contain a complete history of the changes made to that file. The change history lists new strings added and old strings changed or deleted.

Compare the history of changes in the new Caucus dictionary files with the history in the backup copies you made before beginning your translation. The differences between them will tell you which are the *new* changes and consequently which changes or additions you need to make to your translations.

Chapter 12

The File Attachment Map

12.1 Introduction

Caucus version 2.5 adds the concept of a file "attachment" to an item or response. This allows Caucus users to store files of any size and type as part of a conference, and to retreive them when needed.

But the concept of the Caucus "attachment" goes beyond simply storing and retrieving files. Users also may display files that are attached to an item or response. Caucus automatically attempts to figure out *how* to display a file, based on what *kind* of file it is.

For example, displaying an ordinary text file might mean just showing it on the user's screen, one page at a time. But "displaying" a spreadsheet might mean running the appropriate spreadsheet application program and then loading the spreadsheet data.

Deciding what to do with each kind of file is the purpose of the attachment map.

12.2 SET TERMINAL

In order to display a file attachment, Caucus needs to know three things:

1. The user's terminal type
2. The file type
3. What should be done with a given file type for each terminal type.

Items 2 and 3 are handled by the attachment map. Item 1 is the user's terminal type as set by the command SET TERMINAL.

Note that this is not necessarily the same as the user's terminal type as known by Unix or Vax/VMS. The default value for SET TERMINAL is "ascii_unix", which means a generic ASCII terminal.

The Caucus manager may define additional terminal types in the attachment map.

12.3 attach_map file

The attachment map is a single text file, called **attach_map**. It is located in the Caucus home directory, under the **MISC** sub-directory.

While a default attach_map is distributed with Caucus, it is intended that the Caucus manager will edit this file to best suit the needs of the users on that host. The rest of this chapter describes the format of the attach_map file.

The attachment map is a plain ASCII text file, and may be edited with an ordinary text editor or word processor. (If you use a word processor, be sure to save the file as "ascii" or "text", without any formatting codes.)

Lines that begin with "#" are comments and are ignored by Caucus.

All other lines define mappings. There are three kinds of mappings:

FORMAT	define a file type
DISPLAY	describe actions needed to display a file
CHANGE	describe actions needed to alter or change a file

(CHANGE mappings currently are ignored. They are provided as "stubs" for future enhancements to Caucus.)

12.4 FORMAT mappings

A FORMAT mapping defines a rule that Caucus uses to determine the type of a file. In the **attach_map** file, a FORMAT mapping looks like this:

```
FORMAT   name        magic     kind   ext
```

where:

name — is an arbitrary name that is given to a file type. (The name is case sensitive, i.e. "GIF" and "Gif" are two different names.) Required.

magic — is a hexadecimal number. If the file begins with the byte values represented by that number, then the file is recognized to be type *name*. Optional (may be just "-", which means there is no magic number).

kind — "A" or "B" for ASCII or binary. An ascii file may contain only printable characters, plus the characters for carriage return, line feed, or tab (hexadecimal values 0x0D, 0x0A, and 0x09, respectively). Files that contain any other characters are considered to be binary. Required.

ext — The typical file name extension for files of this type. It currently is ignored, but provided for future expansion. Optional (may be just "-").

There always should be exactly one definition of type "A" that has no magic number; in the default **attach_map**, this has the FORMAT name "ASCII". There also should always be exactly one definition of type "B", with no magic number; in the default **attach_map**. This has the FORMAT name "BINARY".

12.5 DISPLAY mappings

A DISPLAY mapping defines a rule that tells Caucus how to display a particular type of file on a particular type of terminal. A DISPLAY mapping looks like this:

```
DISPLAY     name     terminal   commands
```

where:

name — is a file type name (from a FORMAT mapping). The name is case sensitive.

terminal — is a particular terminal type name, such as "ASCII_UNIX". The terminal type is case-*in*sensitive, i.e. "ASCII_UNIX" and "ascii_unix" are identical.

commands — is a list of operating system commands, separated by semicolons (";"). The symbol "$1" refers to the file name, as described below.

When Caucus displays a file attachment, it performs the following steps:

1. Determines if there is a DISPLAY mapping that matches the attachment's file type and the user's terminal type. (If not, complains to the user and stops.)

2. Copies the file from the Caucus database to a temporary file (in /tmp).

3. Substitutes the name of the temporary file for "$1" in the list of commands for this DISPLAY mapping.

4. Executes those commands.

Caucus does not automatically delete the temporary file. The list of commands in the DISPLAY mapping must take care of this.

12.6 Conclusion

The Caucus file attachment map provides a powerful and flexible way to perform actions on attached files. While the sections above have described these actions as a way of "displaying" a file, *any* action can be performed.

For example, consider the popular "RIP" ("remote imaging protocol") file type. RIP files can describe fairly complicated graphical images with a small amount of data. RIP interpreters are available for many machines; some PC terminal emulators have a RIP interpreter built-in.

Now imagine a Caucus discussion that has RIP files attached to several responses. Some users may be able to display these files; some may not. Suppose the Caucus manager defines the following terminal types:

Terminal	Capabilities
vt100	Plain old "vt100" style character terminal
pc	Personal computer with downloading capability, but no RIP
pcrip	Personal computer with RIP interpreter
work	Workstation with full-fledged graphics and windows
remote	Remote user on another system,unknown terminal type

Then the manager can define DISPLAY mappings that will perform an appropriate action when the users try to display the RIP files. Typical actions might be:

Terminal	Actions
vt100	Tell user RIP file cannot be displayed on this terminal.
pc	Tell user file cannot be displayed, but offer to download it
pcrip	Send contents of file directly to terminal
work	Open a new window, run RIP interpreter in window with file
remote	e-mail a copy of the file to the user's home system

This scheme requires that each user know their terminal type, and how to use SET TERMINAL. This process could be made even easier by adding a menu or macro that helps users determine and set their own terminal types.

Index

E

F

H

I

J

K

L

M

W