Macro Express B (MEXB):
An Introduction to Macro Express Programming Principles

Table of Contents

MACRO EXPRESS B (MEXB): ................................................................................................................................................... 1

0. INTRODUCTION ......................................................................................................................................................... 3

0.1 WHY CREATE MACROS?.............................................................................................................................................. 3
0.2 WHAT KINDS OF MACROS CAN BE CREATED? ......................................................................................................... 4

1. MACRO PROGRAMMING – EXERCISE 1 ...................................................................................................................... 5

1.1 BASIC TEXT TYPE.................................................................................................................................................. 6
1.2 TEXT TYPE WITH ALEPH HOTKEY NAVIGATION ............................................................................................... 8
1.3 TEXT TYPE WITH ALEPH HOTKEY NAVIGATION AND TIMING ........................................................................... 8
1.4 WINDOW ACTIVATION........................................................................................................................................ 9
1.5 SIMPLE USER INPUT.......................................................................................................................................... 10
1.6 USER INPUT USING VARIABLES; PROMPTS ........................................................................................................ 11
1.7 USER INPUT USING VARIABLES: MULTIPLE-CHOICE MENUS ......................................................................... 13
1.8 MODIFYING TEXT VARIABLES............................................................................................................................. 17
1.9 LOGIC: IF STATEMENTS .................................................................................................................................. 19
1.10 LOGIC: IF STATEMENTS WITH “OTHER” MENU OPTION .................................................................................. 21
1.11 LOGIC: IF STATEMENTS: TROUBLE-SHOOTING END-USER ERRORS ............................................................... 24
1.12 LOGIC: IF STATEMENTS: FURTHER TROUBLE SHOOTING END-USER ERRORS .............................................. 26
1.13 LOGIC: SWITCH/CASE STATEMENTS ................................................................................................................... 27
1.14 MODIFYING NUMERIC VARIABLES.................................................................................................................... 29
1.15 LOGIC: IF MESSAGE ........................................................................................................................................ 30
1.16 SUBMACROS AND THE RUN MACRO COMMAND ............................................................................................... 35

2. MACRO PROGRAMMING – EXERCISE 2 ................................................................................................................... 37

2.1 BASIC ITEM RECORD EDITING ............................................................................................................................... 37
2.2 OPENING ITEM RECORD TO A SPECIFIC TAB .................................................................................................... 41
2.3 FURTHER ENHANCEMENTS ................................................................................................................................. 42

3. DOCUMENTING MACROS ......................................................................................................................................... 44

3.1 WHY DOCUMENT YOUR MACROS ..................................................................................................................... 44
3.2 WHERE AND HOW TO WRITE DOCUMENTATION ................................................................................................ 44
3.3 WHAT TO DOCUMENT .............................................................................................................................................. 45
3.3.1 Minimum Documentation .................................................................................................................................. 45
3.3.2 Remark lines ................................................................................................................................................... 45
3.3.3 Notes tab ....................................................................................................................................................... 46
3.3.4 Remark Lines Macro Tool ................................................................................................................................. 50

4. TROUBLE-SHOOTING.................................................................................................................................................. 51

5. MACRO CREATION VIA THE CAPTURE FUNCTION .................................................................................................. 51

6. GETTING HELP......................................................................................................................................................... 54

6.1 ALEPH SUPPORT CENTER AND DOCUMENTATION WEBSITE ........................................................................... 54
6.2 HUL MACRO EXPRESS ISITE RESOURCES PAGE ................................................................................................. 54
6.3 HUL MACRO EXPRESS USERS’ GROUP (MUG) .................................................................................................... 54
6.4 MACRO EXPRESS SOFTWARE BUILT-IN HELP ................................................................................................... 54
0. Introduction

The purpose of this class is to give attendees a foundation in Macro Express programming principles, including an introduction to some of the most commonly-used Macro Express commands in the HUL environment. Much of the course will be devoted to two hands-on exercises that demonstrate basic aspects of macro development and design: an extended exercise involving a bibliographic record and a shorter one that works with an Aleph item record form. The exercises will be followed by an overview of macro documentation and, time permitting, a brief discussion of the capture function. Several appendices provide information on topics outside the scope of the class—including an in-depth discussion of trouble-shooting—and can serve as a resource for independent study and exploration.

Beyond the mechanics of making macros, the class will address conceptual issues involved in macro design. The most important part of macro creation comes before programming begins: when analyzing the action/workflow that the macro is supposed to facilitate. What exactly do you want the macro to do? What steps must be reproduced? Is end-user input required, and how can that best be achieved? If these questions are addressed ahead of time, the macro will more effectively serve the purpose for which it is created.

0.1 Why Create Macros?

A macro is an end-user-created program (in this case, one created using Macro Express) that allows the reproduction of keystrokes, mouse-clicks, and other computer functions within a single application or across multiple applications. Macros can be used to automate repetitive or common keyboard tasks, benefiting workflow, ergonomics, productivity, and training. They can reduce the time needed to complete tasks—whether as part of a project or regular workflow—reduce the risk for repetitive stress injuries, and increase data input accuracy, and have proven to be very beneficial to a wide array of units across HUL.

Projects can often particularly benefit from macros; they are often characterized by large quantities of materials or records needing to be processed in a tight timeframe, sometimes by staff who do not have prior training in the system being used. Therefore, as part of project planning, a workflow analysis should be done to determine if macros would be useful.

Macros should also be considered in analyzing regular workflow. Tasks that incorporate repetitive or predictable elements can benefit from macros, freeing up staff time for more complex portions of the workflow.
Many macros are available in the shared HUL Shared Macro set. Before creating a macro of your own, verify that one does not already exist that would serve your purposes, either as is, or modified by you. You should also determine that the application for which you are programming a macro does not have native productivity tools that would serve your purposes (for example the macros in OCLC Connexion). Finally, you should also judge if the time that it will take you to create the macro will be offset by time savings (or other benefits) to yourself, or other staff in your units. In some cases creation of a macro may take minimal effort; in other cases it may require a significant time commitment.

### 0.2 What Kinds of Macros Can Be Created?

You can create macros within and across any computer applications used in your library work, In addition to Aleph, these include bibliographic utilities, databases, web browsers, and administrative software.

You may not create macros to supply passwords, or otherwise provide workarounds for computer security procedures in place in your unit. Units also may not implement macros that eliminate human intervention in multiple-repetition activities, particularly updates of MARC records in Aleph, except with permission of the Aleph Standing Committee. Units wishing to implement such a macro for a particular project must submit a request in writing to the Aleph Macro Working Group (AMWG). The AMWG will review the request to determine if it could provide benefit without negative impact and make a recommendation to the Aleph Standing Committee as to whether the request should be approved.

Limited-repetition macros are permitted as long as they have built-in human review and error-checking. (An example of a limited-repetition macro permissible without special approval is a macro that processes all payments on a particular invoice.) For the full HUL policy on macro creation and use, see the HUL Macro Express policy:

http://hul.harvard.edu/ois/systems/aleph/docs/macroexp_policy.pdf
1. Macro Programming – Exercise 1

The catalog record example used for this training session is a collection of 17th, 18th and 19th century opera librettos cataloged in Houghton. Our task is to design a series of increasingly complex macros that will enable a cataloger to create and add content to frequently-used fields with optimal ease and flexibility.

Sample Record: Hollis #1599177

<table>
<thead>
<tr>
<th>FMT</th>
<th>BK</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDR</td>
<td>01789cam 2200373 a 4500</td>
</tr>
<tr>
<td>001</td>
<td>001599177-6</td>
</tr>
<tr>
<td>005</td>
<td>20070430155831.0</td>
</tr>
<tr>
<td>008</td>
<td>880916s1695 it a 00000 ita d</td>
</tr>
<tr>
<td>0351</td>
<td>a MAHSISL000316</td>
</tr>
<tr>
<td>040</td>
<td>a MH-Mu</td>
</tr>
<tr>
<td>1001</td>
<td>a Perti, Giacomo Antonio,</td>
</tr>
<tr>
<td>24010</td>
<td>a Nerone fatto Cesare.</td>
</tr>
<tr>
<td>24510</td>
<td>a Nerone fatto Cesare : b drama per musica /</td>
</tr>
<tr>
<td>2600</td>
<td>a In Roma : b Per Gio. Francesco Buagni : b Si vendono in piazza Navona nella libraria di Carlo Giannini,</td>
</tr>
<tr>
<td>300</td>
<td>a 72 p.;</td>
</tr>
<tr>
<td>500</td>
<td>a &quot;Dedicato all'illustrissima, &amp; eccellentissima signora, la signora D. Felice Ventimiglia d'Aragona Pignattelli, e Barberini, prencipessa di Palestrina.&quot;</td>
</tr>
<tr>
<td>500</td>
<td>a Three acts.</td>
</tr>
<tr>
<td>500</td>
<td>a Dedication signed and dated: Carlo Giannini, 1 Feb. 1695.</td>
</tr>
<tr>
<td>500</td>
<td>a Music by Giacomo Antonio Perti.</td>
</tr>
<tr>
<td>500</td>
<td>a &quot;Rappresentanti : Agrippina ... ; Nerone ... ; Tigrane ... ; Osmano ... ; Pallante ... ; Ate ... ; Seneca ; Zelto&quot;--P. 6.</td>
</tr>
<tr>
<td>5104</td>
<td>a Sartori, C. Libretti,</td>
</tr>
<tr>
<td>650</td>
<td>a Operas</td>
</tr>
<tr>
<td>655</td>
<td>7 a Operas</td>
</tr>
<tr>
<td>655</td>
<td>7 a Librettos</td>
</tr>
<tr>
<td>7001</td>
<td>a Noris, Matteo,</td>
</tr>
<tr>
<td>7001</td>
<td>a Giannini, Carlo,</td>
</tr>
<tr>
<td>7003</td>
<td>a Buagni, Giovanni Francesco,</td>
</tr>
<tr>
<td>7002</td>
<td>a Barberini Pignatelli, Felice Ventimiglia d'Aragona,</td>
</tr>
<tr>
<td>7001</td>
<td>a Macnutt, Richard,</td>
</tr>
<tr>
<td>7102</td>
<td>a John Milton and Ruth Neils Ward Collection (Harvard Theatre Collection)</td>
</tr>
<tr>
<td>7102</td>
<td>a Teatro Capranica.</td>
</tr>
<tr>
<td>752</td>
<td>a Italy</td>
</tr>
<tr>
<td>H015</td>
<td>a AHM4041</td>
</tr>
<tr>
<td>H03</td>
<td>a MHAHM40415HU</td>
</tr>
<tr>
<td>SYS</td>
<td>001599177</td>
</tr>
</tbody>
</table>

As we learned in MEXA, we need to be in the Macro Express Editor to create a new macro.

- In the open macro file (MEXB.mex), create a new macro with the following settings:
  - Hotkey activation: WIN+Z
Nickname: Exercise 1
Scope: CATALOG.EXE (check off ‘Run If On Top’)

- Go to the Script Tab: this is where we will build our new macro

**Programming Tools Introduced:** Script Tab

### 1.1 Basic Text Type

Our initial macro is very simple: a **Text Type** command to add content to an existing field in the bibliographic record. In this example, we will program the macro to type ‘Italy $d Rome’ in a 752 (hierarchical place name) field that has already been inserted by the cataloger.

**Assumptions:**
- Application: Aleph Cataloging
- Cataloger is editing a bibliographic record (focus is on the record pane)
- Cataloger has created a new 752 field, cursor is positioned after $$a

**New Macro Commands:**
- **Macro Control:** Remark
- **Text:** Text Type

```plaintext
// Adds text to a new 752 MARC field created in a bibliographic record

Text Type: Italy $$d Rome.
```

**New Programming Tools Introduced:** Insert Macro Command (Before, After, At End)

Before going further with this macro, it is a good idea if we add at the top a short description of what we intend this macro to do.

```plaintext
// This macro adds 650, 655, and 752 fields to libretto records
//
// Adds text to a new 752 MARC field created in a bibliographic record

Text Type: Italy $$d Rome.
```

**Programming Tip:**
- It is good to establish the habit of documenting your macro as you are writing the code. We recommend that you add a general macro description **Remark** and a few empty **Remark** lines before you begin to insert commands. Use the remark lines to describe the macro in plain English, then insert the actual code between
your comments. (For the purpose of this exercise, we will have you create the 
\textbf{Remark} line before each new command series that you enter in the macro.)
### 1.2 Text Type with Aleph Hotkey Navigation

Let’s make this macro more useful by having it create the 752 field in addition to filling in the text. To do this, we need to invoke an Aleph hotkey command—in this case the <F6> for inserting a new field. Any function that can be invoked by a keyboard command or shortcut within a computer application (such as the <F6> in Aleph) can be invoked with the same key combination by Macro Express.

```plaintext
// Add a MARC field to a bibliographic record: 752 $a Italy $d Rome.
Text Type: <F6>752<SPACE><SPACE>aItaly$$dRome.
```

**Programming Tips:**

- **Avoid hitting Enter at the end of the string that you type in the Text Type command; it will result in MEX sending the Enter key to Aleph, which may have undesirable consequences. Instead, mouse-click on OK or tab to the OK button (tab backwards by pressing Shift-Tab) before hitting Enter.**

- **Documentation on Aleph Windows Client Navigation, and Aleph Keyboard Equivalents (hotkeys), and Windows hotkeys can be found on the HUL Macro Express iSite Documentation Page:**
  [http://isites.harvard.edu/icb/icb.do?keyword=k1846&pageid=icb.page108403](http://isites.harvard.edu/icb/icb.do?keyword=k1846&pageid=icb.page108403)

- **Although the demonstrations in this class show macro commands such as Insert Macro Command Before using clickable screen icons, most Macro Express programming functions can also be accomplished via hotkeys, right-click menus, and program-level menus. See Appendix C for Macro Express hotkeys.**

### 1.3 Text Type with Aleph Hotkey Navigation and Timing

We will now learn about the timing command when using serial text commands within a macro.

**New Assumptions:**
Cataloger is editing a bibliographic record in single editor mode—if cataloger is working in split editor mode, the record being updated must be in the left pane.

**New Macro Commands:**

- **Text: Wait Text Playback** (also found in the Timing command group)
Programming Tip:

- Using event-related wait commands such as **Wait Text Playback** improves the reliability of macros significantly. However, in some cases this may unacceptably slow down the macro too much, and you may wish to use absolute time delays (i.e., the **Delay Command** with a setting of 300 milliseconds, 500 milliseconds, etc.). You will need to judge this by experience in your environment; it is best to start with **Wait Text Playback**.

**New Programming Tools Introduced:** Duplicate Command, Move Command Up (Down)

Troubleshooting Advice:

- Another way to improve accuracy (and intelligibility of the macro code) is to break up text lines into separate commands.
- `<CONTROL>2` may not always work reliably on some computers. If this happens, you can use the more explicit command: `<CTRLD>2<CTRLU>` in the Text Type box. This instructs the macro to hold Control key down until the number 2 has been entered.

1.4 Window Activation

Our macro is designed with the assumption that the cursor is active in the Records tab of Aleph Cataloging. However, if the user has clicked on a window in another application—the macro will fail. Moreover, if you try to invoke from a floating or popup menu, the macro will not run at all since the menu is, by definition, outside the Aleph Cataloging application.

To prevent this failure, we will change the macro’s scope by un-checking “Run If On Top” next to CATALOG.EXE in the Scope tab, and we will add the Activate Window command to the beginning of the macro. (The Activate Window command is also used in
macros that move between two or more application windows, for example copying a record from OCLC to Aleph.)

**New Assumptions:**
- Application: Aleph Cataloging is running, but not necessarily the active application

**New Macro Commands:**
- **Windows/Programs:** Window Activate
- **Timing:** Wait for Window Title

```plaintext
// Activate Aleph Cataloging Module window.
Activate Window: "ALEPH Cataloging"
Wait For Window Title: "ALEPH Cataloging"
Text Type: <CTRLD>2<CTRLU>
Wait Text Playback
Text Type: <F6>752<SPACE><SPACE>aItaly$$dRome.
```

Because the Aleph Cataloging Module window will not necessarily be on top when this macro runs, you need to adjust the scope; it will still be Program-Specific to CATALOG.EXE, but the ‘Run If On Top’ option must be unchecked.

**Programming Tips:**
- **When using the Activate Window command,** use a partial window title to avoid issues with version numbers and login names. A window title such as “ALEPH Cataloging – Version 18.01 Library: HVD01 – Bibliographic (HVD01) Server lms01.harvard.edu:6505 (18.01) User: WIDSMITH” should be truncated to the simplest unique identifier, i.e., “ALEPH Cataloging.”

- **When using one of the Wait commands (in this case Wait For Window Title),** one has the option to “Wait indefinitely” or “Wait a Maximum of X minutes X seconds.” It may be useful to select “Wait indefinitely” so that if something causes an unexpected delay (the network is slow to respond, the user has a phone call and does not hit the waited-for key as quickly as usual, etc.), the macro will not abort before completing its task.

### 1.5 Simple User Input

Exercises 1.5-1.8 will demonstrate different ways in which the user may vary the content entered by the macro. In this first example, the simplest form of user input is demonstrated: the macro is programmed to simply end at the point where user input is required.

**New Assumptions:**
- After the last line of the macro is completed, cataloger will complete the date field.
1.6 User Input Using Variables: Prompts

The previous macro would only be practical if all the librettos were from Rome. What if the librettos in question were from several different cities in Italy? In this situation, the macro might include a command that prompts the cataloger for a city name. The name entered will be stored in a text variable which the macro will use at one or more appropriate points to type out the city name.

Definition: A variable is a programming object which holds a value that can be set and modified during the course of the macro.

Variables may be included in a Text Type command where they begin and end with the % character (the % signs are not part of the variable name, simply a Macro Express convention for designating a variable in a command line). There are three kinds of variables that can be included in a Text Type command: String (prefix: T), Integer (prefix: N), and Decimal (prefix: D). (There are also Control variables, preceded by a C, but they are beyond the scope of this class.)

A single macro may use up to 99 of each kind of variable, and variable names are made up of the appropriate prefix followed by a number from 1 to 99—for example, T3 indicates String Variable #3. It is a good idea to include a list of Remark lines explaining the function of each variable used at the start of the script. You may assign variables in any order.

A given variable may be used in more than one place in a macro (in this case, the T1 City variable appears in the 752 d and 655 z subfields) and may be repurposed later in the macro, an asset in the rare instance that a highly complex macro drains the pool of available variable names. Finally, the same variable may be shared by more two or more macros if one is a submacro of the other(s), but, in general, the meaning assigned to a variable is specific to a particular macro—the same variable can be defined in as many ways as there are macros in a macro file.
New Macro Commands:
• Variables: Variable Set String

```
// This macro adds 650, 655, and 752 fields to libretto records
//
// T1 = City
// Prompt box for city name to be saved in variable T1
Variable Set String %T1% from Prompt
Activate Window: "ALEPH Cataloging"
Wait For Window Title: "ALEPH Cataloging"
Text Type: <CTRLD>2<CTRLU>
Wait Text Playback
Text Type: <F6>752<SPACE><SPACE>aItaly$$d%T1%.
Wait Text Playback
Text Type: <F6>650<SPACE>0aOperas$$y17th century$$vLibrettos.
Wait Text Playback
// Add a MARC field to a bibliographic record. / 655 Librettos $z Italy
$z Rome $y 16 [cataloger to complete year]
Text Type: <F6>655<SPACE>7aLibrettos$$zItaly$$z%T1%$$y16
```

Troubleshooting Advice:
• If the macro does not reliably return focus to the application (in this case, Aleph) after the prompt box is closed, adding the Window Activate command will ensure that the correct application is in focus before the macro continues. (In our example that command has already been added to the macro script.)

Prompt programming window looks like this:
Resulting prompt box looks like this:

![Prompt Box Image]

### 1.7 User Input Using Variables: Multiple-Choice Menus

If the city names are known, and limited in number, another option is a multiple-choice menu. The name selected from the menu is stored in a text variable which the macro will call upon at one or more appropriate points in order to type it out.

**New Macro Commands:**
- Variables: Multiple Choice Menu

**New Programming Tools Introduced:** Disable (Enable) Command

**Programming Tip:**
- When changing the programming in part of a macro, disable commands rather than deleting them, at least until you finalize the macro.

Prior to this we have added commands or edited existing commands. This is the first case in which we will be substituting one command for another: the prompt box will be replaced by a multiple-choice menu. We could simply delete the remark and command lines in question; however, when making a significant change to a macro, it is generally preferable to disable the lines being replaced instead of deleting them. This way, the earlier version remains intact in the script and can easily be restored by re-enabling the original commands. *Note: Macro Express does not have an "undo" command, so that once you delete a line, it is gone permanently.* (Eventually, when you are completely satisfied with the new programming, you may choose to delete disabled remarks and command lines, in order to keep the script uncluttered.)

To disable a command or remark line, select it and click on the disable/enable icon, a toggle button, pictured here in both its aspects. The disabled line remains in the script, crossed-out and non-functioning, and can be re-enabled by clicking the (changed) symbol a second time.

Multiple command lines may be selected using the Shift key (for consecutive lines) or the Ctrl key (for non-
consecutive lines) and disabled or enabled in the same way.

// This macro adds 650, 655, and 752 fields to libretto records
//
// T1 = City
// Prompt box for city name to be saved in variable T1
Variable Set String %T1% from Prompt
// Multiple choice menu to select city name - Milan, Rome, or Turin
Multiple Choice Menu: City
Activate Window: "ALEPH Cataloging"
Wait For Window Title: "ALEPH Cataloging"
Text Type: <CTRLD>2<CTRLU>
Wait Text Playback
Text Type: <F6>752<SPACE>aItaly$$d%T1%.
Wait Text Playback
Text Type: <F6>650<SPACE>0aOperas$$y17th century$$vLibrettos.
Wait Text Playback
// Add a MARC field to a bibliographic record. / 655 Librettos $z Italy
 $$z Rome $y 16 [cataloger to complete year]
Text Type: <F6>655<SPACE>7aLibrettos$$zItaly$$z%T1%$$y16
Multiple-choice menu programming window looks like this:
Programming Tip:

- You have the option of saving the item text or the item value in the variable. Choosing “Save Item Text” will store the actual text of the menu list item (in this case, Milan, Rome, Turin). Choosing “Save Item Value” will store the choice’s letter/number value (A, B, C).
  
  - If you are likely to change the order of the items on the list in the future, it is best to use Save Item Text; otherwise you may end up having a variable logic statement based on the Item Value that will no longer be valid.
  
  - On the other hand, using the “Save Item Value” option gives you very short variable values – where the menu values are long and are not needed to be typed out by the macro, but are the trigger for further logic (see sections 1.9 and following), the “Save Item Value” may be useful.

With the settings above, the resulting multiple-choice menu looks like this:

![Multiple Choice Menu Example]

Programming Tips:

- If one of the menu item choices is more likely than the others, the Multiple Choice Menu can be enhanced by setting one of the choices as the “default” option. This is done by setting the target variable value before the menu command:

```
Variable Set String %T1% "Milan"
Multiple Choice Menu: City
```

When the macro is run, the menu displays with that option already selected in the menu. This also places the focus in the section of the dialog box where the options are listed, so the user can scroll through the list with arrow keys and press <Enter> to select.

- When setting the "default" value of a Multiple Choice Menu using the "Item Text" option, it is a good idea to select one line from inside the menu, copy it, and paste it in the Variable Set String command. This helps prevent typos and ensures that the default value is identical to one of the actual choices in the menu.
1.8 Modifying Text Variables

A refinement of the basic multiple-choice menu can eliminate the need to use the mouse in selecting a variable value.

If you put an ampersand before an option, the first letter of the text will be underlined on the menu, and, as in other Windows applications, the user may select the item by simply typing that letter. (The underlined shortcut letter need not be the first letter in the option text; you can enter the ampersand before any character.) However, if the “Save Item Text” setting is used, as in this macro, then the ampersand will appear as part of the text typed into the 650 z and 752 d subfields. You can fix this by adding a command that modifies the T1 string variable, to strip out the "&" character.

New Macro Commands:
  • Variables: Variable Modify String

```plaintext
// T1 = City
// Multiple choice menu to select city name - Milan, Rome, or Turin
Multiple Choice Menu: City
// Modification of %T1% city name variable to strip out the ampersands
Replace "&" with "" in %T1%
Activate Window: "ALEPH Cataloging"
Wait For Window Title: "ALEPH Cataloging"
Text Type: <CTRLD>2<CTRLU>
Wait Text Playback
Text Type: <F6>752<SPACE><SPACE>aItaly$$d%T1%.
Wait Text Playback
Text Type: <F6>650<SPACE>0aOperas$$y17th century$$vLibrettos.
Wait Text Playback
// Add a MARC field to a bibliographic record. / 655 Librettos $z Italy
  $$z Rome $y 16 [cataloger to complete year]
Text Type: <F6>655<SPACE>7aLibrettos$$zItaly$$z%T1%$$y16
```
The Multiple Choice Menu programming window looks like this:

Multiple-choice menu looks like this:
The **Variable Modify String** programming window looks like this:

![Variable Modify String Window](image)

### 1.9 Logic: If Statements

You can construct more robust and complex macros utilizing variables with **If / Else / End If** statements. These logical elements enable a macro to make decisions based on a variety of conditions, including user input. In our example, **If / Else** statements will instruct the macro to assign a country name based on the city selected by the user (e.g., Paris, France or Rome, Italy).

**New Macro Commands:**
- **Logic: If Variable, Else, End If**
// This macro adds 650, 655, and 752 fields to libretto records
//
// T1 = City
// T2 = Country
// Multiple choice menu to select city name - Paris or Rome
// Multiple Choice Menu: City
// Modification of %T1% city name variable to strip out the ampersands
// Replace "&" with "" in %T1%
// Set value of variable T2 (Country) based on the city selected using
// IF/ELSE statements.
If Variable %T1% = "Paris"
    Variable Set String %T2% "France"
Else
    If Variable %T1% = "Rome"
        Variable Set String %T2% "Italy"
    End If
End If
Activate Window: "ALEPH Cataloging"
Wait For Window Title: "ALEPH Cataloging"
Text Type: <CTRLD>2<CTRLU>
Wait Text Playback
Text Type: <F6>650<SPACE>0aOperas$$y17th century$$vLibrettos.
Wait Text Playback
// Add a MARC field to a bibliographic record. / 655 Librettos $z
[Country] $$z [City] $y 16 [cataloger to complete year]
Text Type: <F6>655<SPACE>7aLibrettos$$z%T2%$$z%T1%$$y16

Every **If** clause requires an **End If** to close the logical statement. The **End If** may be thought of as a period without which the sentence is incomplete. By the same token, the **Else** command has the syntactical function of a comma or semicolon, setting one **If** clause off from another.

**Programming Tip:**
- Always use matching pairs of **If** and **End If** commands when nesting **If** statements. Note the indentation of nested **If** statements; each one should have a similarly indented **End If**. Macro Express will not allow you to save a macro that contains a dangling **If** command.

- The “Save Item Value” option may be particularly useful in a macro using **If** statements, where the menu choices trigger a set of coding options, but where the menu item text is not itself needed to be typed out by the macro. An example of this is a menu a menu giving the options A – French or B – Spanish, with **If** statements supplying language and country codes based on these choices, but with the words “French” or “Spanish” not themselves needed in a **Text Type** command.
1.10 Logic: If Statements With “Other” Menu Option

There may sometimes be a need for the user to deviate from the supplied options in a multiple choice menu. In this case, an "Other" option may be added to the menu.

New Macro Commands:
- **Logic**: If Variable, Else, End If
- **Dialogs**: Variable Set String from Prompt
This macro adds 650, 655, and 752 fields to libretto records

T1 = City
T2 = Country

Multiple choice menu to select city name – Paris, Rome Or Other

Modification of %T1% city name variable to strip out the ampersands
Replace "&" with "" in %T1%

Set value of variable T2 (Country) based on the city selected using IF/ELSE statements.
If Variable %T1% = "Paris"
    Variable Set String %T2% "France"
Else
    If Variable %T1% = "Rome"
        Variable Set String %T2% "Italy"
    Else
        // Set value of variables T1 and T2 from prompts when "Other" is selected.
        If Variable %T1% = "Other"
            Variable Set String %T1% from Prompt
            Variable Set String %T2% from Prompt
        End if
    End If
End If

Activate Window: "ALEPH Cataloging"
Wait For Window Title: "ALEPH Cataloging"
Text Type: <CTRLD>2<CTRLU>
Wait Text Playback
Text Type: <F6>752<SPACE><SPACE>a%T2%$$d %T1%.
Wait Text Playback
Text Type: <F6>650<SPACE>0aOperas$$y17th century$$vLibrettos.
Wait Text Playback

Add a MARC field to a bibliographic record. / 655 Librettos $z
[Country] $$z [City] $y 16 [cataloger to complete year]
Text Type: <F6>655<SPACE>7aLibrettos$$z%T2%$$z%T1%$$y16
When Other is chosen, the macro must execute an additional If query—“If Variable T1 = ‘Other’”—and prompt the user to enter the city name. Choose the “Prompt for Value” option in the Variable Set String command:

The user is presented with a question:

Whatever the answer, there must be further input from the user regarding the country. An additional Variable Set String from Prompt command is needed to assign a country name to variable T2.
1.11 Logic: If Statements: Trouble-Shooting End-User Errors

The logic of the macro as it stands in 1.9 and 1.10 assumes that the user will select one of the displayed menu options (e.g., Paris, Rome, or Other in 1.10). However, a good macro should also anticipate what will happen if a user does not behave as expected. In the current macro, if the user simply hits <Enter> without selecting a city name, variable T1 will remain blank; the logic for setting T2 will fail; and every occurrence of a city or country name in the bibliographic record will come up blank.

How can we strengthen our macro to account for this eventuality and bring the problem to the user’s attention? One possibility is to program the macro to stop if no city name is given, using the Macro Stop command. Whenever using the Macro Stop command in this way, it is considerate to alert the user to the fact that the macro has purposely been stopped and the reason why it is happening; this is accomplished using the Text Box Display command.

New Macro Commands:
- **Dialogs:** Text Box Display
- **Macro Control:** Macro Stop

```plaintext
// This macro adds 650, 655, and 752 fields to libretto records
// T1 = City
// T2 = Country
// Multiple choice menu to select city name – Paris, Rome or Other
Multiple Choice Menu: City
// Modification of %T1% city name variable to strip out the ampersands
Replace "&" with "" in %T1%
// Set value of variable T2 (Country) based on the city selected using IF/ELSE statements.
If Variable %T1% = "Paris"
    Variable Set String %T2% "France"
Else
    If Variable %T1% = "Rome"
        Variable Set String %T2% "Italy"
    Else
        If Variable %T1% = "Other"
            Variable Set String %T1% from Prompt
            Variable Set String %T2% from Prompt
        End if
    End If
End If
// Troubleshooting - If user doesn't make a menu selection, macro alerts user and stops.
If Variable %T1% = ""
    Text Box Display Error
    Macro Stop
End If
Activate Window: "ALEPH Cataloging"
Wait For Window Title: "ALEPH Cataloging"
Text Type: <CTRLD>2<CTRLU>
Wait Text Playback
Text Type: <F6>752<SPACE><SPACE>a%T2% $$d %T1%.
Wait Text Playback
Text Type: <F6>650<SPACE>0aOperas$$y17th century$$vLibrettos.
```
Dialog box programming window looks like this:

![Dialog box programming window](image)

The resulting error dialog box will look like this:

![Error dialog box](image)
1.12 Logic: If Statements: Further Trouble Shooting End-User Errors

A more sophisticated way of handling the “no-choice” situation would be to offer a second chance to enter the city name before terminating the macro. From the point of view of the end-user, this portion of the macro would do the following:

1. Give a warning that the macro will stop unless a city name is selected;
2. Present the user once again with the multiple-choice menu of city names, and
   a) assign a name based on user input, or
   b) stop the macro if no input is supplied;
3. Go on to type the desired fields into the catalog record with the user-supplied and logically-derived values.

One last potential glitch can be avoided with an additional set of If Variable statements that will terminate the macro if no city or country name has been entered (i.e., T1 or T2 is null) as a result of the user not inputting a city or country in the prompts that result from the Other menu option.

Programming Tip:
- When using more than one error Text Box in a macro, assign unique box headers to differentiate one box from another; it will make reviewing the macro script easier (also, if you use the “Floating Box until ‘Close Box’ command” option, unique headers may be necessary to avoid closing of an incorrect box).

```
// This macro adds 650, 655, and 752 fields to libretto records
//
// T1 = City
// T2 = Country
// Multiple choice menu to select city name – Paris, Rome, or Other
// Multiple Choice Menu: City
// Troubleshooting - If user doesn't make a menu selection, macro alerts user, gives user another chance, rechecks and stops if no answer is supplied.
If Variable %T1% = ""
   Text Box Display Warning
   Multiple Choice Menu: City
End If
// Set value of variable T2 (Country) based on the city selected using IF/ELSE statements.
If Variable %T1% = "Paris"
   Variable Set String %T2% "France"
Else
   If Variable %T1% = "Rome"
      Variable Set String %T2% "Italy"
   Else
      If Variable %T1% = "Other"
         Variable Set String %T2% from Prompt
      Else
         Variable Set String %T2% from Prompt
      End if
   End If
End If
```
// Modification of %T1% city name variable to strip out the ampersands
Replace "&" with "" in %T1%
If Variable %T1% = ""
    Text Box Display Error-City
    Macro Stop
End If
If Variable %T2% = ""
    Text Box Display Error-Country
    Macro Stop
End If
Activate Window: "ALEPH Cataloging"
Wait For Window Title: "ALEPH Cataloging"
Text Type: <CTRLD>2<CTRLU>
Wait Text Playback
Text Type: <F6>752<SPACE><SPACE>a%T2% $$d %T1%.
Wait Text Playback
Text Type: <F6>650<SPACE>0aOperas$$y17th century$$vLibrettos.
Wait Text Playback
// Add a MARC field to a bibliographic record. / 655 Librettos $z
[Country] $$z [City] $y 16 [cataloger to complete year]
Text Type: <F6>655<SPACE>7aLibrettos$$z%T2%$$z%T1%$$y16

Programming Tip:
- In some cases a sequence of logical commands may need to be repeated several times, or even indefinitely, until the variable being tested comes up with a positive value. A Repeat loop may be placed around the desired sequence for this purpose. See Macro Express help for information on Repeat commands.

1.13 Logic: Switch/Case Statements

If / Else / End If logic can get quite complicated as options multiply, making it difficult to keep track of the different layers of nested statements. Macro Express offers an alternative structure for determining actions based on different values of a given variable: Switch / Case / End Case / End Switch statements.

The Switch command allows you to choose the variable you wish to test. Using a series of Case commands, you may specify a series of values for that variable, then define outcomes for each of those values. In our example, we will switch values of T1 (city name) which will assign, on a case-by-case basis, the resulting values of T2 (country name). As with If statements, Switch and Case statements are incomplete without their corresponding End commands.

For simplicity, in this programming scenario, we will only have a fixed set of cities in the Multiple Choice Menu – there won’t be an Other option.

New Macro Commands:
- Logic: Switch / End Switch, Case / End Case

// This macro adds 650, 655, and 752 fields to libretto records
//
// T1 = City
// T2 = Country
// Multiple choice menu to select city name - Paris or Rome
Multiple Choice Menu: City
// Modification of %T1% city name variable to strip out the ampersands
Replace "&" with "" in %T1%
// Multiple choice menu to select city name - London, Milan, Paris, or Rome
Multiple Choice Menu: City
// Set value of variable T2 (Country) based on the city selected using IF/ELSE statements.
If Variable %T1% = "Paris"
    Variable Set String %T2% "France"
Else
    If Variable %T1% = "Rome"
        Variable Set String %T2% "Italy"
    Else
        If Variable %T1% = "Other"
            Variable Set String %T1% from Prompt
            Variable Set String %T2% from Prompt
        End if
    End If
End If
// Troubleshooting - If user doesn't make a menu selection, macro alerts user, gives user another chance, rechecks and stops if no answer is supplied.
If Variable %T1% = ""
    Text Box Display Warning
    Multiple Choice Menu: City
    If Variable %T1% = "Paris"
        Variable Set String %T2% "France"
    Else
        If Variable %T1% = "Rome"
            Variable Set String %T2% "Italy"
        Else
            If Variable %T1% = "Other"
                Variable Set String %T1% from Prompt
                Variable Set String %T2% from Prompt
            End if
        End If
    End If
End If
If Variable %T1% = ""
    Text Box Display Error-City
    Macro Stop
End If
If Variable %T2% = ""
    Text Box Display Error-Country
    Macro Stop
End If
// Set value of variable T2 (Country) based on city name using CASE.
Switch (T1)
    Case: London
        Variable Set String %T2% "England"
    End Case
    Case: Milan
        Variable Set String %T2% "Italy"
    End Case
    Case: Paris
        Variable Set String %T2% "France"
    End Case
    Case: Rome
        Variable Set String %T2% "Italy"
    End Case
    Case: Other
End Case
End Switch

Activate Window: "ALEPH Cataloging"
Wait For Window Title: "ALEPH Cataloging"
Text Type: <CTRLD>2<CTRLU>
Wait Text Playback
Text Type: <F6>752<SPACE><SPACE>a%T2% $$d %T1%.
Wait Text Playback
Text Type: <F6>650<SPACE>0aOperas$$y17th century$$vLibrettos.
Wait Text Playback
// Add a MARC field to a bibliographic record. / 655 Librettos $z
[Country] $$z [City] $y 16 [cataloger to complete year]
Text Type: <F6>655<SPACE>7aLibrettos$$z%T2%$$z%T1%$$y16

Programming Tip

- In many situations, the **If / Else / End If** and **Switch / Case / End Switch** options are interchangeable. If there are more than two or three possibilities, the **Switch / Case / End Switch** function may be preferable, because it avoids complex series of nested If statements. On the other hand, **If / Else / End If** statements allow for more robust logic and are, in some situations, the only option for building the desired functionality.

1.14 Modifying Numeric Variables

Up to this point, our macro has been applicable to 17th century librettos only and has stopped with the cursor active at the end of the 655 field for the cataloger to complete the year of publication. A more elegant approach to working with dates uses numeric variables to calculate the century in which the opera appeared based on the publication year, which the user will supply at a prompt. This will extend the macro’s functionality to works from any century.

Numeric variables differ from the string variables we have been using in that they can be manipulated by computation. In our example, the first two digits of the year of publication will be converted to an Integer Variable (the other numeric option is the Decimal Variable which will not be covered in this class), then incremented by a unit of one to yield the numerical value of the corresponding century.

New Macro Commands:

- **Variables**: Variable Modify String, Variable Set Integer, Variable Modify Integer

// This macro adds 650, 655, and 752 fields to libretto records
//
// T1 = City
// T2 = Country
// T3 = Year
// T4 = Temporary Variable for Editing Year
// N1 = Century
// Multiple choice menu to select city name - London, Milan, Paris, or Rome and put it in variable %T1%
Multiple Choice Menu: City
// Cataloger prompted for year of score
Variable Set String %T3% from Prompt
// Copy the first two characters of the date to a temporary variable
Variable Modify String: Copy Part of %T3% to %T4%
// Convert the two copied characters from text to a number
Variable Modify String: Convert %T4% to integer %N1%
// Increment the copied characters by one, so that something published in 18-- generates a century of "19"
Variable Modify Integer: %N1% = %N1% + 1
// Set value of variable T2 (Country) based on city name using CASE.
Switch (T1)
  Case: London
      Variable Set String %T2% "England"
  End Case
  Case: Milan
      Variable Set String %T2% "Italy"
  End Case
  Case: Paris
      Variable Set String %T2% "France"
  End Case
  Case: Rome
      Variable Set String %T2% "Italy"
  End Case
  Case: Other
      Variable Set String %T1% from Prompt
      Variable Set String %T2% from Prompt
  End Case
End Switch
Activate Window: "ALEPH Cataloging"
Wait For Window Title: "ALEPH Cataloging"
Text Type: <CTRLD>2<CTRLU>
Wait Text Playback
Text Type: <F6>752<SPACE><SPACE>a%T2% $$d %T1%.
Wait Text Playback
Text Type: <F6>650<SPACE>0aOperas$$y
[Nth century]$vLibrettos.
Wait Text Playback
// Add a MARC field to a bibliographic record. / 655 Librettos $z
[Country] $z [City] $y [Year]
Text Type: <F6>655<SPACE>7aLibrettos$$z%T2%$$z%T1%$$y%T3%

Programming Tip

• In this case, since the number of centuries is limited, an “If” or “Switch” statement could have been used to set the century. The Variable Modify String and Variable Modify Integer commands were used to demonstrate these programming options.

1.15 Logic: If Message

As a macro designer, you may sometimes want to offer the user a choice as to how the macro should proceed at a certain point—for example, whether to save a record to the server after the macro has updated it, or to stop, enabling the user to continue editing manually. With an If Message command, you can pose a question to the user and program the macro to execute one of two actions based on the reply.
New Macro Command:
- Logic: If Message

Below is an example of a simple If Message construct:

```
// Ask user if macro should save the record
If Message: "Save the record?"
  // Open File menu, select "Save on Server and Local Drive"
  Text Type: <ALTD>ce<ALTU>
  Wait Text Playback
  // Wait for the Save on Server window and then press Enter for
  Continue
  Wait For Window Title: "Save on Server and Local Drive"
  Text Type: <ENTER>
  Wait Text Playback
End If
```

Below is an If Message construct with an Else clause:

```
// Ask user if macro should save the record -- Else option to do
// something else if user says No
If Message: "Save the record?"
  // Open File menu, select "Save on Server and Local Drive"
  Text Type: <ALTD>ce<ALTU>
  Wait Text Playback
  // Wait for the Save on Server window and then press Enter for
  Continue
  Wait For Window Title: "Save on Server and Local Drive"
  Text Type: <ENTER>
  Wait Text Playback
Else
  Text Box Display: Record not saved
End If
```
If Message programming box with standard button options looks like this:

The resulting dialog box looks like this:
If Message button options may be customized, as in the example below where "Yes" and "No" have been replaced by "Save" and "Do Not Save."

The resulting dialog box looks like this:

![Image of customized message box with options]

Do you wish to save the record?

Save  Do Not Save
The **Text Box Display** command programming box looks like this:

![Text Box Display Diagram]

And the resulting Text Box looks like this:

![Resulting Text Box]

- You did not save the record.
- Please do so after the macro has finished running.
1.16 Submacros and the Run Macro Command

As your macro grows more complex, you may consider moving part of the code into a separate macro which the first macro can call upon using the **Macro Run** command. This simplifies the original script, making it easier to review and trouble-shoot. Subroutine macros (also called *submacros*) are particularly useful for sequences of commands that are frequently reproduced, either within a single macro or shared by several macros. The code stored in a submacro need only be written once and eliminates duplication of effort when revisions are required.

Saving an Aleph catalog record is a task that may be performed in many macros (or multiple times within the same macro). Therefore, it makes a good candidate for a subroutine macro.

New Macro Commands:

- **Macro Control: Macro Run**

The subroutine macro looks like this:

```
// This macro saves the currently open Aleph record.
// This macro can be used as a subroutine or run independently.
//
// Open File menu, select "Save on Server and Local Drive"
Text Type: <ALTD>c<ALTU>
Wait Text Playback
// Wait for the Save on Server window and then press Enter for Continue
Wait For Window Title: "Save on Server and Local Drive"
Text Type: <ENTER>
Wait Text Playback
```

In the script in the main (or parent) macro looks like this (the submacro name is “1.16b Save Record Submacro”):

```
// Ask user if macro should save the record
If Message: "Save the record?"
  Macro Run: 1.16b Save Record Submacro
Else
  Text Box Display: Record Not Saved
End If
```

**Programming Tips:**

- Beware that variable values set in one macro are passed on to the subroutine macros, and variable values set in subroutine macros are passed on to the remainder of the parent macro. Be sure to document input and output variables in both macros. You can copy the **Remark** lines from one macro to the other.
- Different uses of subroutine macros include:
  - Running macros in sequence by having the first macro call the second and so on.
  - Segregating a frequently-used segment of code into a subroutine to save time and effort in coding, testing, debugging, and revising.
- Dividing long or complex macros into subroutines to make them easier to work with.
- Using a parent macro, or a wrapper, to allow running of one macro with different default values or slightly different functionality. For example, a parent macro might provide a choice of barcoding options (circulating book, non-circulating book, circulating DVD, etc.) and then call on a series of nearly identical submacros with different values for the Material Type and Item Status. Although this could be accomplished with Switch / Case statements, the resulting macro might get rather complex, for what are minor differences in function. And at the same time, the submacros will also be available for free-standing use as well.
2. Macro Programming – Exercise 2

While the bibliographic record is an obvious place to use macros, other parts of Aleph are equally macro-friendly—you just have to know the keyboard equivalents that allow you to move among panes, bars, and boxes without using the mouse. You can find these keyboard equivalents (or hotkeys) at the Aleph documentation website:

- Introduction to Aleph & Staff Searching Options, Appendix 1 (OPAC/Search tab) http://hul.harvard.edu/ois/systems/aleph/docs/train_stfs.pdf
- Aleph 18 keyboard equivalents (Client/Print/MacroExpress tab) http://hul.harvard.edu/ois/systems/aleph/docs/keyboard-equivalents.pdf

Our second exercise will demonstrate how to use hotkeys to program a macro to edit an item record; the techniques used here can be applied to any mouse or tab-heavy part of Aleph such as order records, arrival forms, etc.

The example used in this exercise is a Lamont compact disc item record that was created at the point of order using generic defaults; these default codes will be updated during the cataloging process.

2.1 Basic Item Record Editing

In this example we will program a macro to edit an existing Lamont item record as follows:

- Delete the system-supplied barcode
- Change the Material Type to RCD (for Sound recordings – Compact disc)
- Change the Item Status to 29 (the Lamont loan period for compact discs)
- Delete the Item Process Status
- Leave cursor in the Barcode field ready for user to enter new barcode

Assumptions:

- Application: Aleph Cataloging, Item Tab
- An item record has been created earlier in the ordering or cataloging process and is currently open
- Item Record Tab 2 opens automatically

Macro Commands:

- Timing: Wait Text Playback
- Timing: Wait for Key Press
Before beginning, we will take a minute to chart the macro’s route through the Item form; this will spare us from constant toggling between Macro Express and Aleph as we write the script.

- Bring item record editing pane into focus (cursor will default to barcode field)
- Tab 4 times to arrive at Material Type field
- Tab 9 times to arrive at Item status field
- Tab 1 times to arrive at Item Process Status field
- Tab 10 times to arrive back at Barcode field

Upon opening an item record, focus will default to the Items List pane; from there, the macro will need to move to the editing pane at the lower right. You can navigate the window panes and bars of the Items module using Ctrl+Tab, but the most direct path to the lower right pane of any Aleph record is Ctrl+3, which will take you there from any starting point in the window.

As we learned in Exercise 1, the Macro Express Text Type command can be used to invoke Aleph hotkeys.

```// Assumes the Items List pane is in focus when macro starts and that tab 2 is open.  
// Fills out the appropriate Material Type and Item Status codes for music CDs and deletes the 
// Item Process Status and the system supplied barcode, leaving the 
// cursor ready to scan in the barcode.  
// Moves to the item record editing pane. 
Text Type: <CTRLD>3<CTRLU>
```

We could move immediately to our next action; however, with Text Type commands, a macro may send keystrokes to the computer’s buffer faster than the computer can process them, overflowing the buffer and causing the macro to fail. To prevent this, we will add a Wait Text Playback command.

```// Moves to the item record editing pane.  
Text Type: <CTRLD>3<CTRLU>  
Wait Text Playback 
```

**Programming Tips:**

- Event-related delays such as Wait Text Playback and Wait for Window are useful when the program (Aleph) will perform an action that the macro can recognize and use to kickstart itself back into action.

- Absolute delays, in seconds or milliseconds, are useful in situations where the Aleph display refreshes after the macro sends a command (e.g., when pushing records or moving between panes or tabs). A good general guideline is to use a Wait Text Playback command after every Text Type command and to add a Delay command (Delay in Seconds or Delay in Milliseconds) whenever you notice that Aleph takes time to respond to the user. Note that it may take some trial and error to come up with an optimal delay. Occasionally you might find that the Wait Text Playback command in your environment makes the macro
respond slowly and play back choppyly; in such cases you may choose to use a Delay command with a setting of 300-500 milliseconds instead. Again, experimentation will be required to determine the best practice for your working environment. Also, make sure to test your macros on other computers where they will be used, as delays that are sufficient on your computer may not be sufficient on all others.

- If the delay commands don’t work well, you could also reduce the macro’s playback speed or keystroke speed using the Macro Playback Speed or Keystroke Speed commands either for the entire macro or for a portion of it.

- The Wait for Key Press command provides an opportunity for the user to control macro timing. It is often used to resume operation of the macro after a pause for user input and/or review and can be combined with a Text Box instruction to the user to strike a certain key when ready to proceed. When the macro runs, there will be no visible sign that the macro is waiting for a key press; so, if you do not use a dialog box for instruction, make sure to document the points where the macro will wait for a key press. That way others trying out your macro will know what to do when the macro pauses for no apparent reason.

Next, we will delete the system-supplied barcode. When we switched focus to the bottom pane, the text in the barcode box was highlighted by default—all we have to do is instruct the macro to “hit” the Delete key.

```
// Moves to the item record editing pane.
Text Type: <CTRLD>3<CTRLU>
Wait Text Playback
// Deletes the system supplied barcode.
Text Type: <DELETE>
Wait Text Playback
```

Our next action is to edit the Material Type field. We are going to need to tab to the appropriate edit box and enter the code for a compact disc. Since Aleph automatically capitalizes anything typed into this box, we don’t have to worry about capitalization here.

```
// Deletes the system supplied barcode.
Text Type: <DELETE>
Wait Text Playback
// Moves forward four boxes to Material Type and enters the correct code for a compact disc (RCD)
// Aleph automatically formats text entered here as all caps.
Text Type: <TAB><TAB><TAB><TAB>rccd
Wait Text Playback
```
Now we will tab to the Item Status box where we can enter the correct loan code for this item.

```plaintext
// Aleph automatically formats text entered here as all caps.
Text Type: <TAB><TAB><TAB><TAB>rcd
Wait Text Playback
// Moves forward nine boxes to Item Status and enters the correct Lamont loan code for a compact disc (29)
Text Type: <TAB><TAB><TAB><TAB><TAB><TAB><TAB><TAB>29
Wait Text Playback
```

At this point we are finished inputting data, but we still need to delete the Ordered-Received code from the Item Process Status box to leave that field blank.

```plaintext
// Moves forward nine boxes to Item Status and enters the correct loan code for a compact disc (29)
Text Type: <TAB><TAB><TAB><TAB><TAB><TAB><TAB><TAB>29
Wait Text Playback
// Moves forward one box to Item Process Status and deletes the Ordered Received codes.
Text Type: <TAB><DELETE>
Wait Text Playback
```

Finally, we will tab all the way back to the barcode input box where we can scan in the permanent barcode when the macro is finished.

```plaintext
// Assumes the Items List pane is in focus when macro starts and that tab 2 is open.
// Fills out the appropriate Material Type and Item Status codes for music CDs and deletes the
// Item Process Status and the system supplied barcode, leaving the cursor ready to scan in the barcode.
//
// Moves to the item record editing pane.
Text Type: <CTRLD>3<CTRLU>
Wait Text Playback
// Deletes the system supplied barcode.
Text Type: <DELETE>
Wait Text Playback
// Moves forward four boxes to Material Type and enters the correct code for a compact disc (RCD)
// Aleph automatically formats text entered here as all caps.
Text Type: <TAB><TAB><TAB><TAB>rcd
Wait Text Playback
// Moves forward nine boxes to Item Status and enters the correct Lamont loan code for a compact disc (29)
Text Type: <TAB><TAB><TAB><TAB><TAB><TAB><TAB><TAB>29
Wait Text Playback
// Moves forward one box to Item Process Status and deletes the Ordered Received codes.
Text Type: <TAB><DELETE>
Wait Text Playback
// Moves forward ten boxes to Barcode, user can now scan in a barcode.
Text Type: <TAB><TAB><TAB><TAB><TAB><TAB><TAB><TAB><TAB><TAB><DELETE>
```
**Programming Tips:**

- *It is a good idea to break up long Text Type commands; a string that is too long or too complicated can cause unexpected and unwelcome results.*

- *To shift backwards in a form, use Text Type command* 
  
  `<SHIFTDOWN><TAB><SHIFTUP>` *with as many TAB’s as needed.*

### 2.2 Opening Item Record to a Specific Tab

Our macro assumes that the item record will open in Tab 2, the default setting. Within a given session, however, Aleph will return to the last tab opened. To account for this possibility and enhance reliability, we will program the macro to open Tab 2 using the Aleph hotkey, Alt+2.

```plaintext
// Assumes the Items List pane is in focus when macro starts; any tab can be open.
// Fills out the appropriate Material Type and Item Status codes for music CDs and deletes the Item Process Status and the system supplied barcode, leaving the cursor ready to scan in the barcode.

// Moves to the item record editing pane.
Text Type: <CTRLD>3<CTRLU>  
Wait Text Playback
// Moves to Tab 2
Text Type: <ALTD>2<ALTU>  
Wait Text Playback
// Deletes the system supplied barcode.
Text Type: <DELETE>  
Wait Text Playback
// Moves forward four boxes to Material Type and enters the correct code for a compact disc (RCD)  
Aleph automatically formats text entered here as all caps.
Text Type: <TAB><TAB><TAB><TAB>rcd  
Wait Text Playback
// Moves forward nine boxes to Item Status and enters the correct Lamont loan code for a compact disc (29)
Text Type: <TAB><TAB><TAB><TAB><TAB><TAB><TAB><TAB><TAB>29  
Wait Text Playback
// Moves forward one box to Item Process Status and deletes the Ordered Received codes.
Text Type: <TAB><DELETE>  
Wait Text Playback
// Moves forward ten boxes to Barcode, user can now scan in a barcode.
Text Type: <TAB><TAB><TAB><TAB><TAB><TAB><TAB><TAB><TAB><TAB><DELETE>
```
2.3 Further Enhancements

Using the principles and commands learned in Exercise 1, we could devise a number of enhancements that would increase the macro’s scope and versatility. We don’t have time to consider them all here, but here are a few suggestions.

A. Start the macro in the bibliographic record using **Text Type** to "push" a record from the Records tab to the Items tab and open the Items List (see sample script below). For this to work properly, the macro should pause, allowing the user to verify that the correct item record is highlighted before the macro starts to fill in any values.

```plaintext
// Assumes the bibliographic record is open when the macro starts (any pane can be in focus).
// Macro begins in the bibliographic record and opens the item list;
// pauses to allow the user to highlight the correct item from the item list;
// then fills out the appropriate Material Type and Item Status codes for music CDs and deletes the
// Item Process Status and the system supplied barcode, leaving the cursor ready to scan in the barcode.
//
// Go to the navigation pane (lower left pane) in the bibliographic record.
Text Type: <CTRLD>4<CTRLU>
Wait Text Playback
// Go to the "items" node of the pane
Text Type: i
Wait Text Playback
// From the Record Manager menu choose Load/Create Record
Text Type: <ALTD>ml<ALTU>
Wait Text Playback
Delay 200 Milliseconds
// Pauses to let user highlight the correct item record from the Item List
// Text Box Display with the “Floating Box” option and with Text Box Close command after the user has verified the correct item record
Text Box Display: Choose item record to edit
Activate Window: "ALEPH Cataloging"
// Wait For Key Press options: Key to Wait For = Shift ; Choose Wait indefinitely.
Wait for Key Press
Text Box Close: Choose item record to edit
// Moves to the item record editing pane.
Text Type: <CTRLD>3<CTRLU>
Wait Text Playback
// The rest of the macro is the same as Exercise 2.2
```

B. Create several similar item record update macros and bundle them into a Popup menu; this is especially useful if you are working with multiple collections and/or...
material types, each with its own Item Status value, etc. (Popup menu activation was covered in MEXA.)

C. Use multiple choice windows or prompts for gathering variable values to fill in.

D. Save bibliographic or holdings record text as a variable to enter into the appropriate item field (the description field, for example).
3. Documenting macros

3.1 Why Document your Macros

Even if you only write macros for your own use, it is important to document the purpose and functions of the macro and save the documentation as part of the macro. For a handful of short macros that you use regularly, the notes need not be detailed or extensive, but as your macros grow in length, complexity, and number—and particularly if you preserve alternative approaches to certain operations within the macro, as in our examples of If / Else versus Switch / Case logic—clear documentation becomes essential. Documentation should include a statement of the macro’s purpose, explanations of how the various macro commands will function in the host application, and, as a guide to future macro development, even the rationale for choosing one approach over another.

When sharing macros with others, it becomes critical to provide explanatory notes that you can pass along with the macro itself. Documentation takes time up front but saves trouble later on, and those savings multiply each time the macro is shared with another person or unit. Once a macro leaves your hands, its utility will likely depend on how well the macros can be interpreted by others. Deciphering an undocumented macro script can be far more time-consuming, and more frustrating, than writing one from scratch.

Documentation is mandatory for all macros submitted to the HUL Shared Macro Set. Every shared macro must be documented in a way that enables staff in other units to understand its purpose, functions, and requirements; to determine its suitability for local workflow; and to gauge the nature and degree of modification needed for local use.

Good documentation will not make your macro accessible to others but will aid you in macro debugging and maintenance—tasks that can often take more time than the initial design and writing of the macro.

3.2 Where and How to Write Documentation

Macro Express allows for two kinds of documentation to be stored with the macro:
Remark lines within the script and a freeform text area under the Notes tab:

- Remark lines are explanatory comments inserted between command lines to facilitate reading and comprehension of the script. Remarks work best when kept short enough to view without horizontal scrolling.

- The freeform text area in the Notes tab can be used record any information that is relevant to the macro. There is no practical limit to the length of text entered here.
You have already used Remark lines in earlier exercises in this class. In the following section we will discuss both Remarks and Notes in more detail and introduce two tools created to expedite macro documentation.

### 3.3 What to Document

#### 3.3.1 Minimum Documentation

At the very least, you should write out the **purpose** of the macro: what the macro is supposed to accomplish. This will be clear to you as you are writing the macro, but may not be so obvious to anyone else, or even to you a month or a year later. You should also state the **requirements** of the macro: the application (and version) for which it was written and the conditions under which it can be run.

Include the purpose and requirements of the macro in the very first Remark lines of the macro script as well as in the Notes tab. Use the ample space in the Notes tab for any important details that don’t fit the Remark lines.

#### 3.3.2 Remark lines

Here are some examples of Remark line comments that may be helpful; some of these you have already used in earlier exercises in this class:

- **Functions**
  - Give a brief description of what the macro does. When you are trying out commands for the first time or using them in new or unusual ways, you may wish to document each command line with a remark. In longer macros, it is helpful to describe the functions of each segment.

- **Variables**
  - Add a Remark line at the first occurrence of each variable, giving a brief description of what it will be used for. Longer, more complex macros may require certain variable definitions to be repeated when those variables appear later in the script. In macros that run submacros, it is useful to document all input and output variables in both the submacros and the “parent macro.”

- **Hidden information**
  - You may want to add a comment revealing any variable, filename, or other information that is hidden inside a command line in the script. This will eliminate the need to open these commands when reading through the script.

- **Revisions**
  - When you make revisions to a macro that is already in use in your local unit or shared with other units, add a comment with the date and your name or initials giving a brief description of the revisions and, if relevant, the reason for them.
• **Visual effects**
  Longer macros can be subdivided visually into segments by adding Remark lines with horizontal lines. Don’t be afraid of leaving blank comment lines if it helps you to read a long macro.

### 3.3.3 Notes tab
In this section, we will demonstrate documentation for the Notes Tab using as an example one of the macro we created in this class in Exercise 1, making use of the template (Macro_Notes_Template.dot) available in the Macro Documentation area of the HUL Macro Express iSite:
http://isites.harvard.edu/icb/icb.do?keyword=k1846&pageid=icb.page108403
The template will guide you through some basic categories of information used in documenting HUL shared macros. Keep in mind, however, that the Notes tab is a free text area: you need only add such information as you find helpful. The template is simply a tool for analyzing and organizing notes about your macros; not every category will be relevant to a particular macro.
Explanations to the macro documentation template:

1. **Name:** Here you can enter the name of the macro in descriptive, easy to understand terminology. You are not limited to 50 characters, as you are with the Macro Express nickname. If you wish, you can enter the nickname here instead.
2. **Purpose:** This is a one- or two-sentence paragraph that describes the purpose of the macro clearly but concisely.
3. **Hotkey:** This area can be used for documenting the activation method. If you choose a hotkey activation, you can record it here. If you choose another activation method, you can take this part out or edit it as needed. Shared macros are distributed without activation.
4. **Assumptions:** Here you can write any conditions under which the macro is designed to run.
5. **Starting point:** This gives the user the exact location where the cursor must be when the macro is invoked, including the application, the window, the pane, the tab, and the field, as applicable.
6. **Functions:** This section describes the steps that the macro goes through. It should be detailed enough to allow the reader to understand which manual tasks the macro replaces and clear enough for the user to follow the macro steps on screen and to identify when something goes wrong.
7. **What if:** Here you can give the user instructions on how to handle predictable errors that the macro does not account for. Ideally, if you know of a real possibility that something might go wrong, or just differently, when running the macro (for example, an Aleph message telling the budget has no money to pay), you would want to write the macro so that it will detect and deal with the error. However, sometimes this is not feasible, and in that case, it is a good idea to let the user know what to do if this happens.
8. **Subroutine macros:** This area is for listing all subroutine macros (also called submacros) that the macro runs, either directly or through other submacros.
9. **Files accessed**
   and
10. **Files created:**
    Whenever a macro uses a helper file – whether it creates this file or looks for an existing file – write down the name and full path of the file. The area 9. is for listing all files that the macro requires in order to run as designed; these files must exist in the specified location before running the macro. The area 10. is for listing all files that the macro creates, edits, overwrites, or deletes.
11. **Shortcuts or key commands:** This area is for listing of shortcuts or key commands that the macro relies on, to alert the user about any shortcuts that are known to fail from time to time on some computers. The best-known examples of these are the Aleph shortcuts for copying and pasting.
12. **Additional information:** This area is for any other information considered useful to someone using the macro for the first time, considering implementing the macro, or trouble-shooting the macro.
13. **Please note:** Any additional information to which you wish to draw the macro users’ attention can be included here. A suggestion to empty the Windows
Recycle bin periodically is included in the template; this should be deleted if the macro does not delete any files.

14. **Macro Express nickname:** This is the name by which the macro is listed in the macro file. It can be copied from the Properties tab inside the macro.

15. **Development history:** Here you should state the application(s), including the version, the macro is written for. This information will be extremely important when applications are upgraded and macros are reviewed and tested on the new version. A first version of any macro should also include a date when the macro was created and the name of the person or unit responsible for its creation.

16. **Revision history:** When you edit a macro later, either your own or a shared macro, it is good to document your edits here. A running list of all revisions will be helpful to others who may have used an earlier version of the macro.

**Exercise 3: Example of Macro Notes, Using Macro from Exercise 1**

**Name:** Add 650/655/752 to Libretto Record

**Purpose:** Write a one- or two-sentence paragraph that describes the purpose of the macro clearly but concisely.

*This macro facilitates adding specific subject headings with varying geographical subdivisions and hierarchical place names to bibliographical records for 17th, 18th and 19th century opera librettos.*

**Hotkey:** Write the hotkey combination using angle brackets around command key names.

<Win>z

**Assumptions:** List the conditions under which the macro can be run.

*This macro assumes that a bibliographic record is open in the Record Editing pane and that, if the Split Edit mode is on, the record is in the left editing pane.*

**Starting point:** Specify the place where the cursor must be when running this macro.

*Start this macro with the bibliographic record open and in focus.*

**Functions:** Describe the steps that the macro goes through.

*This macro*
- prompts user for a city name and, if necessary, for a country name
- prompts user for the year
- adds a 752 field with the city and country
- adds a 650 field for "Operas" with the century in subfield y
- adds a 655 field for "Librettos" with the country and city in subfields z and the first two digits in subfield y for the year

**What if:** Describe what happens if the user does not select a city name.

*If the user does not select a city from the multiple-choice menu, the macro will prompt for user to type in the city. If the user will still not enter a city name, the macro will stop.*
Submacros: This macro does not run any subroutine macros, so delete this section.

Helper files: This macro does not use any helper files, so delete this section.

Shortcuts or key commands: This macro relies on the Aleph shortcut F6, and this is one of those shortcuts that sometimes (although not often) fails if there is a problem that causes Aleph shortcuts to fail; therefore, state this fact here.

This macro relies on the following Aleph shortcut or key command, among others: F6 for adding a new field.

Additional information: All the necessary information fits in other sections, so delete this section.

Please note: This note does not apply to this particular macro; delete the note. You can leave this blank.

Macro Express nickname: Copy this from the Properties tab inside the macro.

Exercise 1.

Development history: Write these statements using today’s date and your name, Macro Express version 3.0d, and Aleph version 18.01.

Created 2008-07-24 by Marian Librarian
Created in and for Macro Express 3.0d
Created for Aleph 18.01
Last revised 2008-07-24

Revision history: This is a new macro, so nothing needs to be entered here. Delete the text and leave this area blank.

3.3.4 Remark Lines Macro Tool

As we have tried to demonstrate in the exercises today, it is a good practice to use Remark lines to document each step or section of the macro you are writing. With more complex macros, it may be useful to extract these Remark lines, in order to include them wholly or in part on the Notes tab or for other external documentation purposes. A Remarks Lines tool (a macro itself) is available on the HUL Macro Express iSite on the Macro Downloads page in the MacroTools.mex file.

http://isites.harvard.edu/icb/icb.do?keyword=k1846&pageid=icb.page71143

The Remark Lines Macro Tool (mex_CollectRemarkLinesFromScriptToFile); for a target macro, this macro tool creates and opens a text file called “macro-remark-lines.txt” in the Macro Express program directory (if macro-remark-lines.txt already exists, the macro overwrites it) and pastes all of the remark lines extracted from the target macro.
4. Trouble-Shooting

Trouble-shooting is a critical component of macro development. Debugging and maintenance can take more time than the initial coding of a macro. Good coding techniques such as built-in error checking and verification should be brought to bear from the outset of macro design and will facilitate trouble-shooting and maintenance later on.

When creating a macro it is important to think through as many scenarios as possible by asking various "What if..." questions and accounting for those scenarios in the macro code. Thoroughly testing a macro with various variables and under diverse conditions will help ensure macro accuracy and reliability.

When, in testing, a macro does not work as intended and trouble-shooting is necessary, debugging commands and tools may be helpful. If you use logic (If statements) in your macros, you can insert Pause commands at critical points to see that the logic works as intended. Also helpful can be the Step Through Macro function, which allows you to run the macro one command at a time by pressing F8. If you use variables, you may wish to learn about the View Variable Values function.

See Appendix B: Macro Express debugging tools, for a more complete list and explanation of useful debugging commands and tools.

5. Macro Creation via the Capture Function

There are several ways to create and edit Macro Express macros. This class focuses on writing macros line by line in the Scripting Editor. Another method is to “record” a macro by capturing keystrokes and mouse clicks—the Capture function. Some staff have found Capture useful, because it creates macros quickly and requires little knowledge of programming commands. In general, however, this method is discouraged, because it creates macros that will only run if the machine conditions are identical to those at the time of capture: a slight change in window sizing, for example, can cause a macro created by capture to fail. Moreover, captured macros tend to be extremely large, comprising hundreds of lines of barely comprehensible code, and are difficult to edit.

The macro that we created in Exercise 2 offers a good example of the inefficiency and unwieldiness of the capture method. As written in the Scripting Editor, the macro contains 25 command lines and an additional 16 remarks lines; the same macro created by Capture (see below), is 905 lines long and contains neither explanatory comments nor any of the programming tools (prompts, menus, delays, variable manipulation, etc.) that enhance the power and versatility of macros.
Exercise 2 Created Via Script Editor:

// Assumes the Items List pane is in focus when macro starts and that tab 2 is open.
// Fills out the appropriate Material Type and Item Status codes for music CDs and deletes the
// Item Process Status and the system supplied barcode, leaving the cursor ready to scan in the barcode.
// 
// Moves to the item record editing pane.
Text Type: <CTRLD>3<CTRLU>
Wait Text Playback
// Deletes the system supplied barcode.
Text Type: <DELETE>
Wait Text Playback
// Moves forward four boxes to Material Type and enters the correct code for a compact disc (RCD)
// Aleph automatically formats text entered here as all caps.
Text Type: <TAB><TAB><TAB><TAB><TAB><TAB><TAB><TAB><TAB><TAB><DELETE>
// Moves forward one box to Item Process Status and deletes the Ordered Received codes.
Text Type: <TAB><DELETE>
Wait Text Playback
// Moves forward ten boxes to Barcode, user can now scan in a barcode.
Text Type: <TAB><TAB><TAB><TAB><TAB><TAB><TAB><TAB><TAB><TAB><DELETE>

Exercise 2 Re-Created Via Capture:

Macro Playback Speed: Normal Speed
Mouse Move Screen 530, 425
Delay 77 Milliseconds
Mouse Move Screen 528, 425
Delay 15 Milliseconds
Mouse Move Screen 524, 428
Delay 0 Milliseconds
Mouse Move Screen 514, 434
Delay 15 Milliseconds
Mouse Move Screen 506, 438
Delay 0 Milliseconds
Mouse Move Screen 494, 443
Delay 15 Milliseconds
Mouse Move Screen 482, 448
Delay 0 Milliseconds
Mouse Move Screen 470, 452
Delay 15 Milliseconds
Mouse Move Screen 454, 458
Delay 0 Milliseconds
Mouse Move Screen 443, 463
Delay 14 Milliseconds
Mouse Move Screen 425, 468
Delay 15 Milliseconds
Mouse Move Screen 420, 469
Delay 0 Milliseconds
Mouse Move Screen 417, 469
Delay 15 Milliseconds
Mouse Move Screen 414, 469
Delay 0 Milliseconds
Mouse Move Screen 413, 470
Delay 14 Milliseconds
Mouse Move Screen 412, 470
Delay 328 Milliseconds
Mouse Move Screen 412, 469
Delay 0 Milliseconds
Mouse Move Screen 412, 468
Delay 31 Milliseconds
Mouse Move Screen 412, 467
Delay 30 Milliseconds
Mouse Move Screen 412, 466
Delay 156 Milliseconds
Mouse Move Screen 412, 465
Delay 15 Milliseconds
Mouse Move Screen 412, 464
Mouse Left Button Down
Delay 93 Milliseconds
Mouse Move Screen 389, 464
Delay 14 Milliseconds
Mouse Move Screen 387, 464
Delay 0 Milliseconds
Mouse Move Screen 382, 464
Delay 16 Milliseconds
Mouse Move Screen 378, 464
Delay 0 Milliseconds
Mouse Move Screen 372, 464
Delay 14 Milliseconds
Mouse Move Screen 363, 464
Delay 0 Milliseconds
Mouse Move Screen 355, 464
Delay 15 Milliseconds
Mouse Move Screen 350, 464
Delay 0 Milliseconds
Mouse Move Screen 349, 464
Delay 15 Milliseconds
Mouse Move Screen 340, 464
Delay 0 Milliseconds
Mouse Move Screen 304, 464
Delay 15 Milliseconds
Mouse Move Screen 303, 464
Delay 0 Milliseconds
Mouse Move Screen 302, 464
Delay 0 Milliseconds
Mouse Move Screen 301, 464
Delay 14 Milliseconds
Mouse Move Screen 300, 464
Delay 202 Milliseconds
Mouse Left Button Up
Delay 720 Milliseconds
Mouse Move Screen 300, 463
Delay 78 Milliseconds
Text Type: <BACKSPACE>
Delay 218 Milliseconds
Mouse Move Screen 301, 463
Delay 15 Milliseconds
Mouse Move Screen 302, 464
Delay 0 Milliseconds
Mouse Move Screen 304, 466
Delay 15 Milliseconds
Mouse Move Screen 309, 469
Delay 0 Milliseconds
Mouse Move Screen 409, 548
Delay 16 Milliseconds
Mouse Move Screen 409, 549
Delay 0 Milliseconds
Mouse Move Screen 408, 550
Delay 14 Milliseconds
Mouse Move Screen 407, 552
Delay 0 Milliseconds
Mouse Move Screen 406, 553
Delay 15 Milliseconds
Mouse Move Screen 405, 555
Delay 0 Milliseconds
Mouse Move Screen 404, 556
Delay 15 Milliseconds
Mouse Move Screen 403, 557
Delay 0 Milliseconds
Mouse Move Screen 402, 559
Delay 14 Milliseconds
Mouse Move Screen 401, 559
Delay 0 Milliseconds
Mouse Move Screen 401, 560
Delay 15 Milliseconds
Mouse Move Screen 400, 560
Delay 187 Milliseconds
Mouse Move Screen 399, 560
Delay 15 Milliseconds
Mouse Move Screen 398, 560
Delay 0 Milliseconds
Mouse Move Screen 396, 560
Delay 0 Milliseconds
Mouse Left Button Down
Delay 15 Milliseconds
Mouse Move Screen 394, 560
Delay 0 Milliseconds
Mouse Move Screen 390, 560
Delay 0 Milliseconds
Mouse Move Screen 377, 561
Delay 15 Milliseconds
Mouse Move Screen 375, 562
Delay 0 Milliseconds
Mouse Move Screen 364, 562
Delay 15 Milliseconds
Mouse Move Screen 353, 562
Delay 0 Milliseconds
Mouse Move Screen 345, 563
Delay 14 Milliseconds
Mouse Move Screen 336, 564

Page 52 of 69
In spite of these caveats, some macro designers may still choose Capture on occasion as a quick and easy way of creating small macros for personal use. Users seeking more information on the topic should consult the Macro Express Help files, but bear in mind that Capture is not recommended for more complex functions or for macros that will be shared with multiple users.
6. Getting help

6.1 Aleph Support Center and Documentation Website

A copy of this document can be found on the OIS Aleph Documentation Website in the Macros Section at http://hul.harvard.edu/ois/systems/aleph/docs/train_mxeb.pdf

You can also contact the Aleph Support Center for:

- Questions about the HUL Shared Macro Set
- Suggestions for additional macro training or documentation.
- Submit macros that you have created for inclusion in the HUL Shared Macro Set

The Aleph Support Center is located at: http://hul.harvard.edu/ois/systems/aleph/support.html

Select the Macro Express category to send the Aleph Macro Working Group your questions, feedback and other requests. Please note, however, that support for macro programming problems is not available from the Aleph Support Center.

6.2 HUL Macro Express iSite resources page

The HUL Macro Express iSite resources page has a variety of documentation to assist in macro use and development. See also the Tips section on the documentation page for information on best practices and specific problems. You are encouraged to submit tips of your own. The HUL Macro Express iSite is located at http://isites.harvard.edu/icb/icb.do?keyword=k1846

6.3 HUL Macro Express Users’ Group (MUG)

The HUL Macro Express Users’ Group list provides an online peer community where you can ask questions and solicit macro programming tips (mug@hulmail.harvard.edu). This group also meets every two to three months, providing regular opportunities for discussion. You can subscribe to the MUG discussion list by going to: http://hul.harvard.edu/resources/hul_lists.html

6.4 Macro Express Software Built-In Help

The help file in Macro Express software is extensive, and topics can be searched by keyword or browsed through an alphabetical list. The Help menu also gives access to a built-in or online tutorial and a keyword-searchable knowledgebase on the company’s Website. Last but not least, every command dialog box also contains a context-sensitive Help button that opens the relevant page in the Help file.

6.5 Macro Express Website

The website of the makers of Macro Express, Insight Solutions, has a number of useful resources:

- Homepage: http://www.macros.com/
- Support: http://www.macros.com/support.htm
- Tutorial: http://www.macros.com/tutorial/ME3Tutorial.htm
6.6 Macro Express Book

Those seeking to do significant macro programming to support the workflow needs of their unit may find it worthwhile to obtain the book Macro Express Explained, by Joseph Weinpert (Victoria, Canada: Trafford, 2004), which provides detailed and clear explanations of all of the Macro Express commands.

6.7 Macro Express Online Communities

You may also wish to consider joining one of the Macro Express related online communities, such as the Macro Express Forums hosted by PGM or the Aleph Macro Express discussion list.
Appendix A: Macro Express Commands

Macro Express offers a wealth of commands beyond those covered here, and it is worth exploring the lists below for any that may be of use to you. Information about all of these commands is available in Macro Express online help and in the book: Macro Express Explained, by Joseph Weinpert (Victoria, Canada: Trafford, 2004).

The Clipboard, Files/Folders, and Repeat Categories are particularly worth exploring and may become the focus of future meetings of HUL Macro Express Users Group. Some commands, involving ftp, deletion of files, registry, and repeat functions, should be used with care; when in doubt, the Aleph Macro User Group should be consulted before implementing. Repeat functions to be used in Aleph should be implemented with appropriate safeguards and user input; large-scale batch macros are not permitted.

A. 1. Macro Express Commands By Name

(Commands are followed by the command category or categories where they can be found.)

Activate or Launch > Windows/Programs Category
Alt Key > Keyboard Category [Also can be done via Text Type.]
And > Logic Category
ASCII File Begin Process > Files/Folders Category
ASCII File End Process > Files/Folders Category
Audio Balance > Multimedia Category
Audio Bass > Multimedia Category
Audio Mute > Multimedia Category
Audio Mute Toggle > Multimedia Category
Audio Treble > Multimedia Category
Audio Unmute > Multimedia Category
Audio Volume > Multimedia Category
CAPS Lock > Keyboard Category [Also can be done via Text Type.]
Capture Control > Window Controls Category
Case > Logic Category
CDROM Back > CD-ROM Category
CDROM Close Tray > CD-ROM Category
CDROM Eject > CD-ROM Category
CDROM Go to Track > CD-ROM Category
CDROM Next Track > CD-ROM Category
CDROM Pause > CD-ROM Category
CDROM Play > CD-ROM Category
CDROM Previous Track > CD-ROM Category
CDROM Step > CD-ROM Category
CDROM Stop > CD-ROM Category
Change Directory/Folder > Files/Folders Category
Clipboard Append Text > Clipboard Category
Clipboard Copy > Clipboard Category
Clipboard Cut > Clipboard Category
Clipboard Empty > Clipboard Category
Clipboard End Copy > Clipboard Category
Clipboard File Copy > Clipboard Category
Clipboard Graphic Copy > Clipboard Category
Clipboard Paste > Clipboard Category
Clipboard Save Graphic > Clipboard Category
Clipboard Save Text > Clipboard Category
Clipboard Start Copy > Clipboard Category
Clipboard Type Text > Clipboard Category
Control Key > Keyboard Category [Also can be done via Text Type.]
Control Panel Open > System Category
Control Panel Run > System Category
Copy File or Files > Files/Folders Category
Create Folder > Files/Folders Category
Create Registry Key > Registry Category [Note: Not activated in Harvard environment for security reasons.]
Date > Text Category
Date/Time > Text Category
Default Case > Logic Category
Default Display Size > System Category
Default Printer > System Category
Delay > Timing Category
Delete File or Files > Files/Folders Category
Delete Folder > Files/Folders Category
Delete Registry Key > Registry Category [Note: Not activated in Harvard environment for security reasons.]
Desktop Cascade > Desktop Category
Desktop Minimize All > Desktop Category
Desktop Restore All > Desktop Category
Desktop Tile Horizontally > Desktop Category
Desktop Tile Vertically > Desktop Category
Dial-Up Networking > Internet Category
Dial-Up Networking > Network Category
Else > Logic Category
E-Mail Send > Internet Category [Note: Not activated in Harvard environment for security reasons.]
Empty Macro Recycle Bin > Macro Express Category
Empty Recycle Bin > System Category
Encrypted Text > Text Category
End Case > Logic Category
End If > Logic Category [Also Clipboard, Files/Folders, Macro Control, Network, Variables, and Window Controls Categories]
End Switch > Logic Category
File Attributes > Files/Folders Category
FTP Change Directory > Internet Category
FTP ChMod > Internet Category
FTP Delete Directory > Internet Category
FTP Delete File > Internet Category
FTP Get Current Directory > Internet Category
FTP Get File > Internet Category
FTP Get Filesize > Internet Category
FTP Keep Alive > Internet Category
FTP List Directory > Internet Category
FTP Make Directory > Internet Category
FTP Rename File > Internet Category
FTP Send File > Internet Category
FTP Site Command > Internet Category
FTP Site Connect > Internet Category
FTP Site Disconnect > Internet Category
FTP Site File > Internet Category
Get Control > Window Controls Category
Get Mouse Position > Mouse Category
Get Pixel Color > System Category
If Clipboard > Clipboard Category
If Clipboard > Logic Category
If Control > Logic Category / Window Controls Category
If Dial-Up Successful > Logic Category / Network Category
If File Exists > Files/Folders Category / Logic Category
If File Ready > Files/Folders Category / Logic Category
If Folder Exists > Files/Folders Category / Logic Category
If Macro [Enabled/Disabled] > Logic Category / Macro Control Category
If Message > Logic Category
If Not File Exists If Online > Logic Category / Network Category
If OS Version > Logic Category
If Ping Successful > Logic Category / Network Category
If Program [On Top/Running] > Logic Category
If Variable > Logic Category / Variables Category
If Window [On Top/Running] > Logic Category
Keyboard Repeat Delay > Keyboard Category / Timing Category
Keyboard Repeat Speed > Keyboard Category / Timing Category
Keystroke Speed > Keyboard Category / Timing Category
Launch and Activate > Windows/Programs Category
> Files/Folders Category / Logic Category
If Not File Ready > Files/Folders Category / Logic Category
If Not Folder Exists > Files/Folders Category / Logic Category
If Not Program [On Top/Running] > Logic Category
If Not Window [On Top/Running] > Logic Category
Load Macro Text File > Macro Control Category
Load New Macro > Macro Control Category
Log Errors > Debug Category
Log Messages > Debug Category
Logoff > Network Category
Logoff > System Category
Macro Delete > Macro Control Category
Macro Disable > Macro Control Category
Macro Enable > Macro Control Category
Macro Playback Speed > Timing Category
Macro Playback Speed > Macro Control Category
Macro Return > Macro Control Category
Macro Run > Macro Control Category
Macro Stop > Macro Control Category
Mouse Click on Control > Mouse Category / Window Controls Category
Mouse Left Button > Mouse Category
Mouse Middle Button > Mouse Category
Mouse Move > Mouse Category
Mouse Right Button > Mouse Category
Mouse Speed > Mouse Category / Timing Category
Mouse Wheel > Mouse Category
Move File or Files > Files/Folders Category
Multiple Choice Menu > Variables Category
Multiple Choice Menu > Dialogs Category
Network Connect > Network Category
Network Disconnect > Network Category
Network Toggle > Network Category
Num Lock > Keyboard Category [Also can be done via Text Type.]
Open Explorer To > Explorer Category
Open Folder in Explorer > Explorer Category
Open Folder To > Explorer Category
Or > Logic Category
Password Protection > Macro Control Category
Pause > Debug Category / Dialogs Category / Timing Category
PostMessage > System Category / Window Controls Category
Power Off > System Category
Program Launch > Windows/Programs Category
Program Shut Down > Windows/Programs Category
Read Registry Decimal > Registry Category
Read Registry Integer > Registry Category
Read Registry String > Registry Category
Reboot > System Category
Remark > Macro Control Category
Rename File or Files > Files/Folders Category
Rename Folder > Files/Folders Category
Repeat Counter > Repeat Category
Repeat End > Repeat Category [Also Files/Folders Category and Variables Category]
Repeat Exit > Repeat Category [Also Files/Folders Category and Variables Category]
Repeat Prompt Start > Repeat Category
Repeat Start > Repeat Category
Repeat Until > Repeat Category
Repeat With Folder > Repeat Category [Also Files/Folders Category.]
Repeat With Variable > Repeat Category [Also Files/Folders Category.]
Reset Hooks > Macro Express Category
Run Dialog Window > System Category
Screen Saver > System Category
Scroll Lock > Keyboard Category [Also can be done via Text Type.]
Set Focus > Window Controls Category
Shift Key > Keyboard Category [Also can be done via Text Type.]
Shutdown > System Category
Sound Beep > Multimedia Category
Sound Wave File > Multimedia Category
Suspend Computer > System Category
Switch > Logic Category
Terminate Macro Express > Macro Express Category
Terminate Process > Windows/Programs Category
Text Box Close > Dialogs Category / Text Category
Text Box Display > Dialogs Category / Text Category
Text File Begin Process > Files/Folders Category
Text File End Process > Files/Folders Category
Text Type > Keyboard Category / Text Category
Time > Text Category
Undock Computer > System Category
Variable Get Control Text > Variables Category / Window Controls Category
Variable Modify Control > Variables Category / Window Controls Category
Variable Modify Decimal > Variables Category
Variable Modify Integer > Variables Category
Variable Modify String > Variables Category
Variable Restore > Variables Category
Variable Save > Variables Category
Variable Set Control Text > Variables Category / Window Controls Category
Variable Set Decimal > Variables Category
Variable Set From File > Files/Folders Category / Variables Category
Variable Set Integer > Variables Category
Variable Set String > Variables Category
Video Clip Play > Multimedia Category
Wait for Control > Timing Category / Window Controls Category
Wait For File Exist > Files/Folders Category / Timing Category
Wait For File Ready > Files/Folders Category / Timing Category
Wait for Key Press > Keyboard Category / Timing Category
Wait for Text > Text Category / Timing Category
Wait for Web Page > Internet Explorer Category / Timing Category
Wait for Window Title > Timing Category
Wait Left Mouse Click > Mouse Category / Timing Category
Wait Middle Mouse Click > Mouse Category / Timing Category
Wait Program Terminate > Timing Category
Wait Right Mouse Click > Mouse Category / Timing Category
Wait Text Playback > Text Category / Timing Category
Wait Time Delay > Timing Category
Wait Time Elapse > Timing Category
Wait Window Lose Focus > Timing Category
Wallpaper > System Category
Web FTP Site > Internet Category
Web Site > Internet Category
Win Key > Keyboard Category [Also can be done via Text Type.]
Window Activate > Windows/Programs Category
Window Close > Windows/Programs Category
Window Hide > Windows/Programs Category
Window Maximize > Windows/Programs Category
Window Minimize > Windows/Programs Category
Window Reposition > Windows/Programs Category
Window Resize > Windows/Programs Category
Window Restore > Windows/Programs Category
Window Show > Windows/Programs Category
Window Sizing Border > System Category
Windows Sizing Border > Windows/Programs Category
Write Registry Decimal > Registry Category [Note: Not activated in Harvard environment for security reasons.]
Write Registry Integer > Registry Category [Note: Not activated in Harvard environment for security reasons.]
Write Registry String > Registry Category [Note: Not activated in Harvard environment for security reasons.]
XOR > Logic Category
A. 2. Macro Express Commands by Category

**CD-ROM Category**
- CDROM Back
- CDROM Close Tray
- CDROM Eject
- CDROM Go to Track
- CDROM Next Track
- CDROM Pause
- CDROM Play
- CDROM Previous Track
- CDROM Step
- CDROM Stop

**Explorer Category**
- Open Explorer To
- Open Folder To
- Open Folder in Explorer

**Files/Folders Category**
- ASCII File Begin Process
- ASCII File End Process
- Change Directory/Folder
- Copy File or Files
- Create Folder
- Delete File or Files
- Delete Folder
- End If
- File Attributes
- If File Exists
- If File Ready
- If Folder Exists
- If Not File Exists
- If Not File Ready
- If Not Folder Exists
- Move File or Files
- Rename File or Files
- Rename Folder
- Repeat End
- Repeat Exit
- Repeat With Folder
- Text File Begin Process
- Text File End Process
- Variable Set From File
- Wait For File Exist
- Wait For File Ready

**Clipboard Category**
- Clipboard Append Text
- Clipboard Copy
- Clipboard Cut
- Clipboard Empty
- Clipboard End Copy
- Clipboard File Copy
- Clipboard Graphic Copy
- Clipboard Paste
- Clipboard Save Graphic
- Clipboard Save Text
- Clipboard Start Copy
- Clipboard Type Text
- End If
- If Clipboard

**Debug Category**
- Log Errors
- Log Messages
- Pause

**Desktop Category**
- Desktop Cascade
- Desktop Minimize All
- Desktop Restore All
- Desktop Tile Horizontally
- Desktop Tile Vertically

**Dialogs Category**
- Multiple Choice Menu
- Pause
- Text Box Close
- Text Box Display
**Internet Category**
Dial-Up Networking
E-Mail Send [Note: Not activated in Harvard environment for security reasons.]
FTP Change Directory
FTP ChMod
FTP Delete Directory
FTP Delete File
FTP Get Current Directory
FTP Get File
FTP Get Filesize
FTP Keep Alive
FTP List Directory
FTP Make Directory
FTP Rename File
FTP Send File
FTP Site File
FTP Site Command
FTP Site Connect
FTP Site Disconnect
Web FTP Site
Web Site

**Logic Category**
And
Case
Default Case
Else
End Case
End If
End Switch
If Clipboard
If Control
If Dial-Up Successful
If File Exists
If File Ready
If Folder Exists
If Macro [Enabled/Disabled]
If Message
If Not File Exists
If Not File Ready
If Not Folder Exists
If Not Program [On Top/Running]
If Not Window [On Top/Running]
If Online
If OS Version
If Ping Successful
If Program [On Top/Running]
If Variable
If Window [On Top/Running]
Or
Switch
XOR

**Keyboard Category**
Alt Key
CAPS Lock
Control Key
Keyboard Repeat Delay
Keyboard Repeat Speed
Keystroke Speed
Num Lock
Scroll Lock
Shift Key
Text Type
Wait for Key Press
Win Key

**Macro Control Category**
End If
If Macro [Enabled/Disabled]
Load Macro Text File
Load New Macro
Macro Delete
Macro Disable
Macro Enable
Macro Playback Speed
Macro Return
Macro Run
Macro Stop
Password Protection
Remark

**Macro Express Category**
Empty Macro Recycle Bin
Reset Hooks
Terminate Macro Express
Mouse Category
Get Mouse Position
Mouse Click on Control
Mouse Left Button
Mouse Middle Button
Mouse Move
Mouse Right Button
Mouse Speed
Mouse Wheel
Wait Left Mouse Click
Wait Middle Mouse Click
Wait Right Mouse Click

Multimedia Category
Audio Balance
Audio Bass
Audio Mute
Audio Mute Toggle
Audio Treble
Audio Unmute
Audio Volume
Sound Beep
Sound Wave File
Video Clip Play

Network Category
Dial-Up Networking
End If
If Dial-Up Successful
If Online
If Ping Successful
Logoff
Network Connect
Network Disconnect
Network Toggle

Registry Category
Create Registry Key
Delete Registry Key
Read Registry Decimal
Read Registry Integer
Read Registry String
Write Registry Decimal
Write Registry Integer
Write Registry String

Repeat Category
Repeat Counter
Repeat End
Repeat Exit
Repeat Prompt Start
Repeat Start
Repeat Until
Repeat With Folder
Repeat With Variable

System Category
Control Panel Open
Control Panel Run
Default Display Size
Default Printer
Empty Recycle Bin
Get Pixel Color
Logoff
PostMessage
Power Off
Reboot
Run Dialog Window
Screen Saver
Shutdown
Suspend Computer
Undock Computer
Wallpaper
Window Sizing Border

Text Category
Date
Date/Time
Encrypted Text
Text Box Close
Text Box Display
Text Type
Time
Wait for Text
Wait Text Playback
**Timing Category**
- Delay
- Keyboard Repeat Delay
- Keyboard Repeat Speed
- Keystroke Speed
- Macro Playback Speed
- Mouse Speed
- Pause
- Wait for Control
- Wait for File Exist
- Wait for File Ready
- Wait for Key Press
- Wait for Text
- Wait for Web Page
- Wait for Window Title
- Wait Left Mouse Click
- Wait Middle Mouse Click
- Wait Program Terminate
- Wait Right Mouse Click
- Wait Text Playback
- Wait Time Delay
- Wait Time Elapse
- Wait Window Lose Focus

**Variables Category**
- End If
- If Variable
- Multiple Choice Menu
- Repeat End
- Repeat Exit
- Repeat With Variable
- Variable Get Control Text
- Variable Modify Control
- Variable Modify Decimal
- Variable Modify Integer
- Variable Modify String
- Variable Restore
- Variable Save
- Variable Set Control Text
- Variable Set Decimal
- Variable Set From File
- Variable Set Integer
- Variable Set String

**Window Controls Category**
- Capture Control
- End If
- Get Control
- If Control
- Mouse Click on Control
- PostMessage
- Set Focus
- Variable Get Control Text
- Variable Modify Control
- Variable Set Control Text
- Wait For Control

**Windows/Programs Category**
- Activate or Launch
- Launch and Activate
- Program Launch
- Program Shut Down
- Terminate Process
- Window Activate
- Window Close
- Window Hide
- Window Maximize
- Window Minimize
- Window Reposition
- Window Resize
- Window Restore
- Window Show
- Windows Sizing Border
Appendix B: Macro Express Debugging Tools:

Macro Express provides an array of tools to assist in macro debugging and troubleshooting, both within the macro command list itself, and in the Macro Express Debug menu.

Script commands found in the Debug command category:

- **Log errors** – place this command once at the beginning of a macro script to record any errors that occur during execution to a text file log with a date/time stamp. If there are no errors, a macro “Complete” message will be recorded at the end of the macro's run. This command is especially useful for macros that run on a schedule or while unattended.

- **Log messages** – this command may be inserted at any point in a macro script to record whether or not a specific condition has been met or to track a variable value. Each resulting message will be recorded to its own text file log with optional date/time stamp.

- **Pause** – temporarily halts the execution of a macro at runtime and displays a Pause dialog. The complex pause option allows for the display of a customized text message. The macro remains paused with a dialog box until the user clicks the “Resume” button to continue. The Pause dialog also gives the option to stop the macro by clicking an “Abort” button.

Commands found in other command categories useful for debugging:

- **Text box display** – can be used at strategic points in a macro to display messages noting whether certain conditions have been met or to track and display macro variable values.

Commands found in the macro file Debug menu:

- **Test run macro** (F9) – runs the macro from the scripting editor.

- **Step through macro** (Scripting Editor Only) (F8) – runs the macro from the scripting editor one command at a time. The macro performs each command and then pauses until prompted for the next command by pressing F8. This is a good way to track, step-by-step, how the macro is functioning. To step through a macro in Aleph, you must either include an Activate Aleph command at the top of the macro or check the “Change Focus to Previous Window” setting in the MEX Debug file menu.

- **Check syntax now** – checks the syntax of the macro for errors and reports any found (e.g., If statements missing End If commands, Shift Key Down commands without a corresponding Shift Key Up, etc.).

- **Toggle breakpoint** (Scripting Editor Only) (CTRL+B) – Breakpoints are junctures in a script where the macro will pause during testing. Breakpoints are set by highlighting a command or using the Toggle Breakpoint function in the Debug file menu. When the macro is executed, it will run commands until a breakpoint is reached. During the pause, you may review the status of certain elements in the macro—e.g., Variable Values under Debug Windows can be used to verify whether variables are being populated correctly. The macro can then be resumed with the Test Run Macro command (invoked with the F9 key or by clicking the green arrow on the tool bar) and will continue to execute commands until it reaches another breakpoint or is finished running.
• **Debug windows**
  - *Breakpoints* – displays any commands that have been assigned as a breakpoint in the macro.
  - *Variable Values* – displays a listing of any values assigned to Text, Integer or Decimal variables used in the macro. Best used in conjunction with the Step Through Macro and/or Toggle Breakpoint commands. Can also be useful after the macro has run to the end.

• **Use test window** – With this option checked, a Macro Express Test Window will be displayed when invoking the Test Run Macro command. The window consists of a memo type entry space and can be used to view the playback of text commands. Leave this box unchecked when testing more complicated commands.

• **Check syntax on save** – default option in Macro Express that you can toggle on or off. Upon saving, this setting checks the syntax of the macro for errors and reports any found (e.g., If statements missing End If commands, Shift Key Down commands without a corresponding Shift Key Up, etc.).

• **Change focus to previous window** – activate this option when using the Step Through Macro command to make the macro's focus alternate between the host application and Macro Express at each step. You may also need an Activate Window command at the beginning of the script for the Step Through Macro function to work.
Appendix C: Macro Express Hotkeys

For those of you who want to minimize the use of the mouse, Macro Express provides keyboard shortcuts for many commands:

**In Macro Explorer Window**

<Ctrl>F  Search Within Macro Names

**With Macro Script Open**

*Searching, Navigation, Printing*

<Ctrl>F  Search Within Macro Commands

*(does not find text in menus/dialog boxes)*

<Ctrl>G  Go to Macro Line #

<Ctrl>P  Print Macro Script

**Macro Editing**

<Ctrl>A  Select All

<Ctrl>N  Disable/Enable Command

<Ctrl>D  Duplicate Command

<Ctrl>E  Add Command to End

<Ctrl>I  Insert Command After

<Ctrl>K  Insert Command Before

<Ctrl>C  Copy Command

<Ctrl>V  Paste Command

<Ctrl><UpArrow>  Move Command Up

<Ctrl><DownArrow>  Move Command Down

<Shift><UpArrow>  Highlight Multiple Lines Up

<Shift><DownArrow>  Highlight Multiple Lines Down

<Ctrl>S  Save Macro

**Debugging**

<F8>  Step Through Macro

<F9>  Test Run Macro

<Ctrl>B  Toggle Breakpoint

Appendix D: Definitions

*Submacro* – A macro called by another macro; in some cases it may be a macro that can also run independently, in others it is a macro fragment that only runs as an element of a parent macro.

*Parent macro* – A macro that calls another macro.
*Host application* – The program in which the macro is designed to run (e.g., Aleph)