

# INMEMO

## Users Guide

Written, published and distributed by:

MicroSabio  
22704 Ventura Blvd #476  
Woodland Hills, CA 91364 USA  
voice: 818-710-8437  
fax. 818-704-4882  
email: [info@microsabio.com](mailto:info@microsabio.com)  
web site: [www.microsabio.com](http://www.microsabio.com)

Revision F00, Jun 2002

Copyright © 2002 Jack McGregor

Copyright © 1984 - 2002 Jack McGregor  
All rights reserved

Publisher

MicroSabio  
22704 Ventura Blvd #476  
Woodland Hills, CA 91364 USA  
voice: 818-710-8437  
fax. 818-704-4882  
email: info@microsabio.com  
web site: www.microsabio.com

Authors

Jack McGregor has written this document and its underlying software, and is responsible for most of the words, concepts and information contained herein. Jack has been working on this body of documentation, and using it to document the operations of and changes to INMEMO, since 1984. Ty Griffin has contributed some organizational and editing services beginning in 2002.

Related Information

This document contains the latest information available at the time of publication. It is also part of the greater MicroSabio documentation set, which describes all of the software products written, supported and distributed by MicroSabio. The most current information on all MicroSabio software products, as well as the latest versions of all documentation, may be found at [www.microsabio.com](http://www.microsabio.com).

Legal Notice

This documentation and the software it describes contain proprietary information belonging to MicroSabio, a California proprietorship owned entirely by Jack McGregor. The software and this related information is provided under a license agreement containing restrictions on use and disclosure, and is protected by copyright law. This information is confidential between MicroSabio and the client, and remains the exclusive property of MicroSabio.

Due to continued product development, the information contained herein may change without notice. It is believed to be accurate and reliable, at least at the time of its writing. However, no responsibility for the accuracy, completeness or use of this information is assumed by MicroSabio. If you find any problems in the documentation, please report them to the publisher in writing.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the expressed or licensed permission of MicroSabio.

Copyright © 1984 - 2002 Jack McGregor  
All rights reserved

#### Trademarks

Alpha Micro, AMOS, AlphaBASIC are registered trademarks of Alpha Micro Products, Inc.

AlphaLAN is a trademark of U.A. Systems, Inc.

Autolog is a trademark of Bob Rubendunst.

C-ISAM and Informix are registered trademarks of Informix Corporation.

Microsoft and Windows are registered trademarks of Microsoft Corporation.

Softworks Basic is a trademark of Bob Salita.

UNIX is a registered trademark of The Open Group.

ZTERM is a trademark of Rod Hewitt.

MicroSabio, A-Shell, INMEMO, INFLD and EZ-SPOOL are trademarks of Jack McGregor.

All other products, services, companies, events and publications are trademarks, registered trademarks or servicemarks of their respective owners.

#### Document History

<b>Revision</b>	<b>Date</b>
F00	1 June 2002
E01	10 July 1998
E00	7 October 1991
D01	5 January 1990
D00	10 October 1989
C00	March 1987
B01	August 1986
B00	November 1985
A02	November 1984
A01	May 1984
A00	February 1984

This page left blank

# Table of Contents

---

<b>Chapter 1: Overview .....</b>	<b>1</b>
Common Uses .....	1
On Screen Help.....	1
Selection Lists or Menus .....	2
Dialog Boxes with Radio Buttons .....	2
Table Lookup.....	2
Specifications and Features .....	2
Operating Systems.....	2
Language Interfaces.....	2
Terminal Requirements .....	3
Memory Requirements .....	3
File Locking.....	3
Memo File Format.....	4
Memo Screen Format .....	4
Memo Types.....	4
Memo and File Size Limitations .....	4
Paging.....	4
Save and Restore Screen Area.....	4
Editing Controls.....	5
Function Keys.....	5
Error Trapping .....	5
Exit Codes .....	5
Color.....	5
8-Bit Latin-1 Characters .....	5
Memory Based Memos.....	5
<b>Chapter 2: Example of Operation.....</b>	<b>7</b>
<b>Chapter 3: Description of Operation .....</b>	<b>9</b>
Primary Data File .....	9
Primary Data File Maintenance Program.....	9
Auxiliary Memo File.....	9
INMEMO.SBR .....	10
Locking Module(s).....	10
Scrap Buffers.....	10
Function Key Translation Table.....	10
<b>Chapter 4: Terminology.....</b>	<b>13</b>

<b>Chapter 5: Calling Parameters.....</b>	<b>15</b>
OPCODES .....	15
MMO'DSP: Display.....	15
MMO'EDT: Edit.....	15
MMO'BDR: Border.....	15
MMO'LID: Smart Line Insert/Delete .....	16
MMO'DEL: Delete .....	16
MMO'LIN: Return One Line.....	16
MMO'OPN: Open.....	16
MMO'CLS: Close.....	16
MMO'NBR: Don't Redraw Border .....	16
MMO'NMR: Don't Redraw Border or Text.....	17
MMO'SCH: Search.....	17
MMO'SIL: Silent Mode.....	18
MMO'DPG: Display Paging.....	18
MMO'MNU: Menu Mode .....	19
MMO'TBL: Table Lookup .....	20
MMO'FST: Fast Menu Mode .....	20
MMO'EWU: Edit Without Update.....	20
MMO'UWE: Update Without Edit.....	21
MMO'SVA: Save Screen Area .....	21
MMO'RSA: Restore Screen Area.....	21
MMO'CXV: Start at Specified XY position.....	21
MMO'NOA: No Arrows.....	22
MMO'AAH: Auto Adjust Height .....	22
MMO'IPG: Intelligent Paging Mode .....	22
MMO'DBM: Display Bottom of Memo .....	22
MMO'ISL: Insert Into Sorted List .....	22
MMO'APS: Alternate Prompt Style .....	23
MMO'NAF: No Auto Format.....	23
MMO'OTX: Output to Text.....	23
MMO'FFM: Free Form Menu .....	24
MMO'OPT: Optimized Disk I/O .....	24
MMO'RET: Return Key Exit.....	25
MMO'HDR: Memo Header Option.....	25
TEXT: General purpose text exchange parameter .....	25
CHANNEL: Memo file channel or specification.....	27
STROW: Starting row.....	29
STCOL: Starting column .....	29
ENDROW: Ending row .....	29
ENDCOL: Ending column.....	29
LINK: Pointer to location of memo within memo file.....	29
XPOS: Return status codes .....	29
VSPEC: Set maximum physical width and height.....	30
MMOCLR: Color Specifications .....	31
EXTCTL: Extended options .....	32
<b>Chapter 6: Examples .....</b>	<b>37</b>
Opening the Memo File .....	37
Display Memo Pad.....	37
Edit Memo Pad.....	38

File Full .....	39
Delete Memo Pad.....	39
Printing Memo Pads.....	39
Application Controlled Memo Editing.....	41
Memory-Only Pop-Up Pick List.....	42
Inserting an Invisible Memo Header.....	42
Closing the Memo File.....	43
File Errors .....	43
Paging .....	44
Using INMEMO for Online Help.....	46
Pop-Up Field Level Pick Lists.....	47
Dialog Boxes with Radio Buttons.....	47
<b>Chapter 7: Internal Format of Memo Storage.....</b>	<b>49</b>
4-Byte Link Format.....	50
Link Byte Arrangement .....	50
Space Compression.....	51
8-Bit Latin-1 Symbols.....	51
Carriage Returns .....	52
Invisible Headers.....	52
<b>Chapter 8: Operator Usage .....</b>	<b>53</b>
Exiting the Memo Pad.....	53
Aborting an Editing Session .....	53
Editing Controls .....	53
Wrap-Around .....	55
Scrolling.....	55
Vertical Menu Selections.....	55
Free Form Menu Selections .....	56
Invisible Headers.....	56
<b>Chapter 9: Installation .....</b>	<b>57</b>
Installing INMEMO for A-Shell.....	57
Making an Ersatz Account.....	57
Restoring the Files .....	58
Adjusting File Names and Locations .....	58
INMEMO.SBR/XBR.....	58
Compiling the Utilities .....	59
Moving CMD and LIT Files.....	59
INMEMO.HLV .....	59
INFLD/INFLDX.....	59
Files Included.....	60
Loading the Locking Module in System Memory .....	61
Scrap Buffers.....	61
Loading INMEMO.SBR in System Memory.....	62
Making and Loading Function Translation Tables .....	62
Verifying the Installation .....	63
Testing INMEMO .....	64



# Chapter 1

## Overview

---

The INMEMO subroutine provides an easy way for programmers to store and manipulate variable-length text structures. Although these structures may include light-bar menus, help screens, and lookup tables, the most common and traditional use would be to include free form, expandable memo pads within file maintenance programs. Typically, this would be used in a program like customer maintenance, in which you have many fixed length fields describing and categorizing the customer, but in which you also want to leave room for some additional comments. This can be done by simply creating a comments field, of say, 60 characters. The trouble with this approach is that 60 characters is a very significant increase in the size of the file record, yet it is not nearly long enough for much in the way of meaningful comments. What the customer seems to always want is unlimited comment space; the programmers are always telling him they can't do that because of storage limitations that he doesn't understand. He sees it as a programming limitation, which it is.

INMEMO was designed specifically to make both the customer and the programmer happy by making it easy to satisfy the wishes of the former within the time constraints of the latter. It solves the problem of having to allocate a huge amount of disk space by pooling the comment space among all the records in the file, and by compressing spaces. Records that have no comments use no space, leaving more for those records which have a huge number of comments. It solves the problem of manipulating a large amount of text by allowing you to define a rectangular area of any size (up to the size of the screen), by providing VUE-like editing controls, and by allowing memo text to be scrolled through a smaller window (if necessary). It solves the programming hassle by doing all of the file handling for the auxiliary file internally; all you have to do is open and close the file.

### Common Uses

---

Other than the obvious use of storing variable-length text attached to fixed-length records, there are several other common uses for INMEMO.

#### **On Screen Help**

INMEMO is a convenient vehicle for composing, storing, and displaying on-screen help information. By setting a variable switch in your program, you can actually key in the help text while running the program, in the exact same environment that the text will be displayed. This makes creating on-screen help much easier than having to visualize or write down notes about where and when help is needed and then enter the actual text in an external editor (or even worse, hard code it in the program.)

The storage convenience is obvious, since help text is notoriously variable in length. The display convenience comes from the fact that you can position and size the help window in a place appropriate for each screen, without having to worry about the format the text was originally entered in, and you can take

advantage of the Tracker or modern terminals to pop-up help windows, with the original display being automatically restored on exit. When the help text is larger than the window, INMEMO will take care of scrolling the text inside the window (both horizontally and vertically). The fact that the user retains total control over scrolling and when to exit allows you to write your on-screen help in a pyramid (or newspaper column) structure with the most important and brief information and the start, proceeding then to more and more detail. The user just quits when he's seen enough.

### **Selection Lists or Menus**

INMEMO has a mode in which the user moves a highlighted bar or arrow up and down through the memo, returning back to the program the text of the line which was selected when the user hit **Return**. A vertical menu memo consists of a series of one-line selections, such as this:

1. Prepaid
2. COD
3. On Invoice
4. 2%10, Net 30

Menus can also be horizontal or free form with the light bar jumping from one selection to another based on the horizontal and vertical arrow keys (as well as by typing one or more characters identifying the selection.)

### **Dialog Boxes with Radio Buttons**

Free form INMEMO menus can also contain text besides the choices. This makes it ideal for popping up warnings and other messages that require a multiple-choice response (such as [Proceed] [Cancel] [Help]).

### **Table Lookup**

INMEMO can also be used as a simple yet powerful table lookup and conversion utility. Using a variation of the pattern search option, you pass to INMEMO a pattern string, and it returns to you the remainder of the text on the line it found the pattern on. Using the example of the shipping methods above, you could pass it a string containing "2," and it would return "UPS BLUE." Again, using an appropriate opcode switch, you can provide advanced operators or programmers the ability to update the tables without ever leaving the program in which the table is queried.

## **Specifications and Features**

---

This section serves as a reference and also as a preview to some of the features that are discussed in the following sections.

### **Operating Systems**

INMEMO (version 2.3(416) and later) runs under all known versions of AMOS up through 2.3A at the time of this writing. A C version is available for Softworks Portable Basic and is compatible with all versions of Portable Basic supported by Softworks. Another C version is included in A-Shell and continuously kept up to date for all of the Windows and UNIX platforms supported by A-Shell.

### **Language Interfaces**

The AMOS version of INMEMO interfaces to AlphaBASIC and BasicPlus. The C version interfaces to Softworks Portable Basic and A-Shell. (The Softworks C version supports about 85% and the A-Shell

version supports about 95% of the features described in this manual, although it also supports a few additional features that are not in the AMOS version. The differences are noted where practical.)

### **Terminal Requirements**

INMEMO requires a CRT with XY cursor positioning for proper operation. In addition, the following features will enhance the operation:

- smart insert and delete line
- graphics character set
- smart box commands
- AM72-style color
- AM72-style save and restore area commands (or Tracker).

For best results, the terminal driver should conform to the AMOSL 1.3 standard and should support the **TRMCHR** Monitor Call. However, earlier terminal drivers will also work provided they contain the # of rows and columns in the terminal driver header. INMEMO will adjust itself to the current screen dimensions (up to 255 by 255), truncating the window size as necessary. Note that we also distribute the Tracker which emulates AM72-style save and restore area commands on nearly any terminal, allowing memos to be popped up and removed without losing the underlying screen display.

### **Memory Requirements**

Under AMOS, INMEMO is fully reentrant and may be loaded into either user memory or system memory (to be shared by several users). In addition to the memory required to load the routine itself, INMEMO requires a significant amount of workspace. The complete requirements are shown below:

Module Size:	23000 bytes (system or user memory)
Semaphore Module:	12 bytes (system memory)
Work Area:	500 bytes (user memory)
Buffer:	* based on window size (user memory)

\* The buffer size is at least equal to the size of the window, and may be much larger if the paging option is used (see Paging).

Note that the work area and buffer memory requirements are only applicable while INMEMO is actually executing. Between INMEMO calls, the memory is relinquished to BASIC and is available for use by other routines.

Under A-Shell and Softworks Basic, INMEMO is linked into the executable so it does not require extra runtime memory, other than for the work buffer.

### **File Locking**

Under AMOS, INMEMO uses an internal file-locking scheme based on semaphores which does not conflict with any known or imaginable locking schemes (such as FLOCK, XLOCK, and LOKSER). If your memo file is controlled by LOKSER, then you must open it with the random'forced option to have multi-user access to the memo pad.

Under the C versions, the standard OS locking mechanism for each platform is used.

## **Memo File Format**

The memos are stored in 64-byte linked records in a standard AMOS-style contiguous file. The links may be either 2 or 4 bytes, leaving 62 bytes of memo text per record if 2-byte links are used, or 60 bytes of memo text per record if 4-byte links are used.

A compression algorithm is used which stores up to 255 contiguous blanks as two bytes, yet allows the exact format to be reproduced for later display or printout. The actual number of characters used to store a memo is equal to the number of non-blank and enclosed single blank characters, plus 2 bytes for every enclosed block of 2-255 blanks (don't count trailing blanks), plus one byte for each logical line. The number of 64-byte memo records required is equal to the number of bytes divided by 60 (4 byte links) or 62 (2 byte links) and rounded up to the nearest integer.

## **Memo Screen Format**

The memo is displayed and edited in a rectangular portion of the screen which is user definable of any size from 1 row by 1 column to the size of the screen. At your option, INMEMO may draw a box around the window and/or display a prompt of your choice just above the memo window. A paging option (see below) allows more text to be edited than can be displayed at one time in the window.

## **Memo Types**

INMEMO supports at least 4 memo types, distinguished by their logical format and usage: 1) ordinary free-form text, 2) vertical light-bar menus, 3) free-form light bar menus (similar to dialog boxes with radio buttons and supported under AMOS only), and 4) lookup tables. These are described in more detail in the sections on Calling Parameters and Examples

## **Memo and File Size Limitations**

The maximum memo file size is limited by the size of the memo links and the operating system. You can use either 2-byte or 4-byte links, yielding a maximum record count (8 records per block) of 65535 or  $2^{32}$ , respectively. Note that pre-2.0 versions of AMOSxx have a limitation of 65535 blocks, which is about 524,000 records. Individual memos can occupy any number of records (from 1 to the maximum number of records in the file.)

The Softworks Basic version is limited to a maximum memo size of about 6K, while the A-Shell is limited to 32K under 16 bit platforms and available virtual memory under 32 bit platforms. In both C versions, the amount of working memory required is less than for AMOS, since stores the text as a series of variable length lines rather than as a rectangular grid.

## **Paging**

INMEMO supports window paging, which allows the screen memo window to be moved up, down, and sideways through a much larger memo. With this option, the maximum memo size is only limited by the amount of free memory, subject to a parameter you pass explicitly to INMEMO and the restrictions given above.

## **Save and Restore Screen Area**

INMEMO supports AM72-style save and restore area TCRT commands and provides opcodes to handle the saving and restoring of the screen used by the memo. It does not, by itself, have the ability to track or read your screen; if you don't have an AM72 or a suitable emulator (such as the Tracker), it will return a status code informing the program that it will have to handle its own restoration of the screen (assuming this is necessary or even desired.)

Note that Tracker is included within A-Shell, so save and restore is always available on that platform.

## **Editing Controls**

The controls and keystrokes used in editing of memos are essentially those used in VUE. A list of those codes can be found in the Operator Usage...Editing Controls section of this document.

## **Function Keys**

INMEMO supports the same three function key translation systems that INFLD (a companion MicroSABio product) does: a proprietary format, the FIXTRN format, and the SET PFK format.

## **Error Trapping**

INMEMO is potentially subject to any of the types of file errors that can occur under AMOSL (disk not mounted or not ready, illegal record #, protection violation, etc.). INMEMO returns all standard AMOSL file service system error codes in a parameter which the calling program should test after every call to INMEMO. In addition, it will report (both to the screen and by code to the calling program) various types of memo file structure errors, such as illegal links.

## **Exit Codes**

In addition to error trapping, INMEMO will report other status information via a block of exit parameters. These include information about how the memo was exited (via `Escape`, abort or function key), whether it was able to comply with a request to save or restore the screen area, and where the cursor was when the exit command given. A corresponding feature allows you to enter a memo for re-editing starting at a specific place. Together with function key command support, this provides the ability to pop in and out of memos.

## **Color**

INMEMO supports AM72-style color via `tab(-2,x)` and `tab(-3,x)` commands, allowing separate color palettes to be specified for the memo text, border, prompt, arrows, and error messages.

## **8-Bit Latin-1 Characters**

INMEMO supports the full 8-bit Latin-1 character set for compatible terminals, such as the AM65. An optional add-on, ISOTRAN, can be purchased to provide the ability to support the Latin-1 character set in a mixed environment of 7 and 8 bit terminals. This is accomplished by providing a compose function to input special characters with multiple keystrokes, and a configurable translation table for mapping display characters to the nearest suitable symbol (graphic or text) on the target terminal at display time.

## **Memory Based Memos**

Although memos are normally stored in a memo file, INMEMO can be used for editing of memos purely in memory. Memo text to be loaded and updated memo text (after editing) are passed between INMEMO and the calling program via a string parameter. This allows INMEMO's display and editing features to be used even when the actual text will be stored in some other file format. It is also very handy for temporary displays and selection lists which are generated in memory at runtime and then discarded after use.



## Chapter 2

# Example of Operation

---

This is an example of how a typical memo pad feature might be added to a typical file maintenance program. Imagine a customer maintenance program which has a screen like this:

CUSTOMER MASTER FILE MAINTENANCE

1. NAME	Lo-Tech Products	8. TYPE	Z
2. ADDRESS #1	999 Cybercrud Ave	9. MTD SALES	\$999.99
3. ADDRESS #2		10. YTD SALES	\$999.99
4. CITY	Manganese Valley	11. TAXABLE?	N
5. STATE	California	12. CATEGORY	1
6. ZIP	99999	13. TERMS	N
7. REGION:	7	14. TURKEY?	Y

Comments

Run by Valley Guys, they buy raw materials from us for their PC-Shoulder Holsters and MOEPROMS (Maxed-Out Eraseable Programmable Read-Only Memory). Contacts

Carlo di Lithium	x007	(Payables)
Maria Galleria	x199	(Purchasing)

The memo pad area can be treated almost just like a normal field. The main differences are:

- Instead of using INPUT and PRINT to enter and display the field, you will use an XCALL INMEMO... statement.
- When the customer record is deleted, you will use another XCALL INMEMO... statement to delete the memo pad.
- To print the memo pad, you will use a series of XCALL INMEMO... statements within a loop to retrieve the memo text one line at a time.

From the operator's standpoint, the memo pad works very much like VUE. You can use the arrow keys to move the cursor around anywhere within the defined area. You can also use the VUE control codes to do things like insert or delete (character or line), move cursor home, start of line or end of line, and next word or previous word. To finish editing, just press the ESC key (as if you were exiting from display mode in VUE).

# Description of Operation

---

The components involved in the INMEMO system are:

- Primary data file
- Primary data file maintenance program
- An auxiliary contiguous file
- INMEMO.SBR
- Locking module(s)
- Scrap buffers (optional)
- Function key translation table (optional)

## Primary Data File

---

The primary data file requires one or possibly two changes to implement the memo pad:

Addition of a 2-byte or 4-byte binary link field. This field is used by INMEMO.SBR to link the primary data record to the corresponding comments which are stored in the auxiliary file. This field does not have to be manipulated in any way by the primary file maintenance program, except to supply it as an argument in the XCALL INMEMO... statement.

Removal of existing comments field. This is optional of course, but there doesn't seem to be any need for other comments if you have INMEMO.

## Primary Data File Maintenance Program

---

The primary data file maintenance program needs to be modified to include a call to INMEMO in the appropriate places. You'll probably want to include the following types of calls:

- A display call to display the comments within your existing display record routine.
- A delete call to delete the comments within your delete record routine.

An edit call to allow comment editing in the existing add and change routines. This call automatically writes the updated comments to the auxiliary file.

## Auxiliary Memo File

---

You will have to allocate an auxiliary contiguous file to store the comments. A special program is provided (MAKMMO.RUN) to allocate the file and initialize it. You can make the file any size you like, since the INMEMO routine will inform the operator when it fills up. Another program (EXPMO.RUN) can then be

used to expand it. You will probably want to allocate a file that is large enough for the average amount of comments for each record.

## INMEMO.SBR

---

The fourth component in the memo pad system is the INMEMO subroutine file (INMEMO.SBR). The routine is reentrant, so it may be loaded into system memory or into user memory. If not previously loaded into either one, it will be fetched from the disk when needed. The routine has several parameters which tell it the position on the screen, the auxiliary file channel, and various options. This is described in greater detail in the following sections.

## Locking Module(s)

---

INMEMO uses a semaphore system (independent of LOKSER, XFLOCK, XLOCK, etc.) to control multi-user access to memo files. The system allows multiple user simultaneous editing within a given memo file; it only grabs exclusive use during the actual updating of the memo which occurs on exit from an editing session. (It is assumed that the primary data record to which the memo pads are linked will be locked to prevent multiple users from editing the same memo at the same time.)

As a second safety check, INMEMO keeps track of the first record of each memo currently being edited, and prevents multiple users from gaining simultaneous edit-level access to the same memo. Consequently, even if your primary record locking scheme has loopholes in it, INMEMO will protect itself (and your files) from conflicting updates. If an attempt is made to edit a memo currently being edited, INMEMO will display an appropriate message and allow the user the option of waiting or aborting the INMEMO request.

Although the single semaphore module in INMEMO.SYS will work for any number of memo files on the system, it will only allow file updates to a single file at one time. If you have a lot of users and multiple memo files, you can set up a semaphore module for each memo file to reduce the chance of an update bottleneck when many users exit from memos at the same time.

Before attempting to perform a memo file update, INMEMO will first look for a module in system memory with the same name as the memo file and an MLK extension. If that is not found, it will look for either INMEMO.SYS or INMEMO.MLK. See the **Error! Reference source not found.** section for more information about setting up the semaphore modules.

## Scrap Buffers

---

You may optionally create a scrap buffer which both INFLD.SBR (another MicroSabio product) and INMEMO.SBR can use to transfer, store, and recover text. Both routines will automatically transfer the last text deleted to the scrap buffer, and the user may recover it using the `^S^O` command. See the **Error! Reference source not found.** section.

## Function Key Translation Table

---

You may optionally create a function key translation table to allow the function keys on most terminals to be translated to either editing and text sequences or command exit codes. Three different mechanisms for creating translation tables are provided under AMOS. (Only the FIXTRN method is supported under A-Shell.) The first uses a template (example) translator source file, in which you code in the desired translations and assemble the file. (This format is compatible with INFLD's translation tables.) The second method uses Alpha's FIXTRN utility to create function key translation tables of the type used by AlphaWRITE, AlphaCALC, MULTI, AlphaBASE, etc. (This method allows for multi character output of

the translations. For example, you could make F1 print a string of 80 characters to be used as a format guide.)

The third method is identical to the second except it uses Alpha's SET PFK utility, which is functionally equivalent to FIXTRN but produces translation tables of a different format. Fortunately, INMEMO can determine the format at run time so you can any of the 3 types in any combination. (You will need to keep track of the type in order to re-edit them; use the CHFUNC.LIT utility provided to display the translation table format and contents.

Command exitcode translations cause the memo editing session to be terminated (with update) and an exitcode corresponding to the function key to be returned to the calling program in the EXTCTL parameter.



# Chapter 4

## Terminology

---

Before going any further, we ought to agree on a few terms to simplify the description of the INMEMO system. The terms to be defined are printed in boldface; further references to them will be in italics to remind you of the technical definition given earlier.

**Memo File** refers to the entire auxiliary data file containing all of the individual memo pads.

**Memo Pad Window** (or **Display Window**, or **Memo Window**) refers to the rectangular area on the screen in which the memo pad editing and display takes place. The Memo Window is defined by its 4 corners in parameters passed to INMEMO.SBR.

A **Memo Pad** is a single logical block of memo text associated with one primary data file record. This may include any number of **Physical Memo Records**, and may be smaller or larger than a Memo Window. In the picture of a screen in section 2.0, the text shown in the Memo Window makes up a single Memo Pad.

The terms **Logical Memo Record** and **Memo Pad** are equivalent.

**Scrolling** is the process of moving the Memo Window throughout a larger Memo Pad. The Memo Window may be scrolled both horizontally and vertically to allow editing of Memo Pads both wider and longer than the Memo Window.

**Paging** is equivalent to **Scrolling**, although it usually refers to moving the display forward or back by the size of the Memo Window (as in **Next Page** or **Previous Page**).

**Absolute Pad Dimensions** (or **Virtual Pad Dimensions**) refers to the maximum width and length allowed for a particular memo pad. This determines the limits of scrolling.

A **Physical Memo Record** is a single 64-byte record in the Memo File. Physical Memo Records are linked together to form Memo Pads in a fashion similar to the way physical disk blocks are linked together to form sequential files in AMOS. Note that a Physical Memo Record is actually a logical disk record in AMOS. The internal layout of a Physical Memo Record is described in Section 7.

The **Memo Control Record** is a special case of a Physical Memo Record, which is always the first record in a Memo File. The Memo Control Record is used to keep track of the allocated and free portions of the Memo File; it contains the master pointers to the start of the deleted record chain and the first eof-marked record.

The **Invisible Header** is an optional string of up to 60 characters which is stored at the beginning of a memo but that does not normally display. This header can be used to store information allowing a memo to be identified independent of any link from another file.



## Calling Parameters

---

Call the INMEMO routine from a BASIC program via:

```
XCALL INMEMO, OPCODE, TEXT, CHANNEL, STROW, STCOL, ENDROW, &  
ENDCOL, LINK, XPOS, VSPEC, MMOCLR, EXTCTL
```

Where:

(The term standard numeric parameter will be used below to indicate that a parameter may be either a string which evaluates to a number, a floating point, a 1 to 4 byte binary, a 1 to 4 byte integer (type I in BasicPlus), or an expression which evaluates to a number.)

### OPCODES

---

OPCODE is standard numeric parameter specifying the calling mode and options. This parameter must be supplied as the sum of the individual values for the desired mode and options. The name in italics to the left of the opcode numbers are the equivalent symbols as defined in MMOSYM.BSI. We strongly advise that you ++include MMOSYM.BSI in your programs and use the symbols rather than the hard coded numbers when specifying OPCODE. For example, to specify edit with a border and smart line insert/delete, but not redisplay the border, you should specify the opcode as MMO'EDT+ MMO'BDR+ MMO'LID+ MMO'NBR rather than 1+2+4+128. (Another example of this type of parameter is the SWITCHES variable in the SPOOL.SBR).

#### **MMO'DSP: Display**

Bit value: 0

Display memo and return. (Also see MMO'DPG: Display Paging).

#### **MMO'EDT: Edit**

Bit value: 1

Edit memo (vs. display only)

#### **MMO'BDR: Border**

Bit value: 2

Display graphics border around comment area (defined by STROW, STCOL, ENDROW and ENDCOL). Note that the border displays outside the area defined by the 4 corners, so the option can be switched on and

off without affecting the size of the window. Note that the border will only display when there is room on the screen for it. (For example, if STROW is 1, then there will be no top border. If STCOL is 1, there will be no left side border.) Important Note: You should include this opcode even when you have drawn your own border, since certain display operations require the border to be restored. See opcodes MMO'NBR: Don't Redraw Border and MMO'NMR: Don't Redraw Border or Text for information on using a border that has already been drawn on the screen.

### **MMO'LID: Smart Line Insert/Delete**

Bit value: 4

Use smart insert and delete (providing terminal has those features). This option should be selected whenever the comment area is more than about 5 lines high and there will be no text on the screen to either the left or right of the memo pad area. It is important that no screen display exists to either side of the memo because INMEMO will use the terminal's insert and delete line operations to perform certain display operations, which will shift or delete information outside of the memo window. You can prevent this by simply not including MMO'LID in the opcode.

### **MMO'DEL: Delete**

Bit value: 8

Delete memo linked to pointer LINK. This call should be used whenever the associated primary data file record is deleted. If you are not deleting the primary data record, then you must remember to update the data record with the zero LINK or else you will get a MEMO LINK ERROR the next time you try to display the memo using the old LINK.

### **MMO'LIN: Return One Line**

Bit value: 16

Return 1 logical memo line in the variable TEXT. This call is provided to allow easy printing of the comments on reports.

### **MMO'OPN: Open**

Bit value: 32

This opcode can be used to open the memo file instead of opening it with a Basic OPEN statement in the calling program. This is mainly an advantage when interfacing INMEMO to some language other than Basic, in which case there may be difficulties in passing the equivalent of a file channel. Note that this opcode should not be combined with any others (i.e. you need to make a separate XCALL to open the memo file.)

### **MMO'CLS: Close**

Bit value: 64

This opcode is used to close a memo file opened with opcode MMO'OPN, and should be used prior to exiting a program. Although no disk updating will be performed on the close operation (since disk buffers are always flushed after each update), this is still a good idea to cleanly exit from the LOKSER database. (Do not use this opcode unless you opened the file with the MMO'OPN opcode.)

### **MMO'NBR: Don't Redraw Border**

Bit value: 128

Don't redraw border on initial display of memo pad. This option is useful when you are drawing your own border or know that the border is already on the screen. Note that should also specify the MMO'BDR opcode in addition to MMO'NBR. Note that it doesn't hurt anyone to redraw the border, it simply wastes display time and may make your program appear a bit stupid.

### **MMO'NMR: Don't Redraw Border or Text**

Bit value: 256

Don't redraw the border or redisplay the text of the memo pad. This is like MMO'NBR except that it skips the redisplay of the memo text also. It is useful when you have already the memo correctly displayed on the screen and you want to reedit it. Adding MMO'NMR to an editing opcode only eliminates the initial redisplay; you must be careful to make sure that the existing display on the screen matches the memo to be edited. For your convenience in using this opcode, INMEMO will always reset the display window to the beginning of the memo pad in case scrolling moved it.

### **MMO'SCH: Search**

Bit value: 512

Search the current memo pad for a pattern contained in the TEXT parameter. If the pattern is found, POS is returned  $\diamond 0$ . The pattern in TEXT must start with 10 special control characters which define the actual symbols to be used for certain logical operators and wildcards. (The character shown in the second column below is the *suggested* character, but there are no defaults.)

Byte#	Char	Description
1	?	Matches exactly one character
2	*	Matches any string (including null).
3		Pattern delimiter; used at either or both ends of the pattern to prevent matches within a sub string. E.g., `an' will match `band', whereas ` an ' will only match `an' surrounded by spaces, end or beginning of memo, or punctuation.
4	&	Logical AND; used to search for memos containing more than one pattern.
5	!	Logical OR; used to search for memos containing one of a choice of patterns.
6	@	Matches any one alphabetic character.
7	#	Matches any one numeric character.
8		Reserved for later use
9		Specifies record terminator in MMO'TBL mode. If blank, INMEMO will return everything from the pattern match to the end of line, otherwise it will return up to the first occurrence of the terminator character (following the pattern match). Line-ends are marked by a single byte with ASCII value 13 decimal.
10		Either '^' which specifies SEARCHFOLD or blank, which indicates no upper/lower case conversion.

The remainder of the string in TEXT (starting in byte 11) specifies the logical combination of pattern(s) to be matched. Note that all spaces are significant, including leading and trailing spaces and those surrounding logical operators. Note, however, that leading spaces will also match the start of a memo, and a single space anywhere will match any number of contiguous spaces in a memo. The pattern string **MUST** contain an explicit trailing null. A null pattern is a guaranteed match.

Using the characters suggested above, a sample pattern to look for "DOG" or "CAT" might be:

```
TEXT = "?#|&!@# ^DOG!CAT"
```

For examples of using the search facility, see the source code of the DMPMMO utility program and/or the TSTMMO sample program.

### **MMO'SIL: Silent Mode**

Bit value: 1024

The addition of this opcode makes the editing session and all corresponding display updates invisible. This is only useful during application-controlled loading of memos when you don't want the user to have to watch the operation. Naturally, eliminating the display will also speed up the operation considerably. Make sure your program exits from INMEMO (by ending the TEXT string to be pre-loaded with an ESC).

### **MMO'DPG: Display Paging**

Bit value: 2048

Display with Paging controls. Performs a display operation similar to opcode 0 except that it allows the operator to use scrolling controls to display a memo pad larger than the memo window. Exit from the display operation with **Escape** or **Control-C**. Note the following points before implementing:

vs. Normal Display Mode: You may wonder why you wouldn't always use MMO'DPG instead of MMO'DSP when displaying memos. One reason is that MMO'DPG requires that the operator hit **Escape** to exit from the display mode - this is not always desirable. (See MMO'IPG: Intelligent Paging Mode to avoid having to hit **Escape** for when displaying memos that fit entirely in the display window.) It is probably more convenient for operators if your program first displays one page of the memo using the regular MMO'DSP display mode, and then allows the operator to go into paging mode by entering a command or function key.

vs. Edit Mode: MMO'DPG is a natural alternative to MMO'EDT (edit) mode in situations where the operator's security level does not permit them to edit.

Display Idiosyncrasies: The memo display is left exactly where it was when the operator exits from the display-scrolling mode. This is because the operator may want to continue looking at the memo text after exiting (as with help information.) This, however, conflicts with the normal editing operation in which the memo display is restored to the home position after editing, to allow a subsequent editing operation to be performed without another repainting of the display (using opcode MMO'NMR.) Therefore you must make sure not to use MMO'NMR (with either editing or display) on a memo that was just displayed with MMO'DPG.

Variations: opcodes ÀMMO'BDR, MMO'LID, MMO'NBR and MMO'NMR affect the display-scrolling mode (MMO'DPG) in the same way they affect the normal editing mode (MMO'EDT). That is, MMO'BDR causes a border to be used, MMO'LID causes smart vertical scrolling, MMO'NBR avoids a redraw of the border, and MMO'NMR avoids redisplaying the memo text. Use caution with MMO'NMR, since the routine always starts the display (internally) from the home position, so if you allow repeated opcode MMO'DPG+MMO'NMR calls on the same memo and the operator exits from one call with the memo window not in the home position, the subsequent call will be out of synch with the display. .

Bottom Prompt: When a border is used with a MMO'DPG opcode display call, the bottom border is taken over to display a help prompt indicating how to scroll and exit. When a border is not used, the bottom line of the memo window is used for the help prompt. You may want to keep this in mind if you are meticulously designing help screens - set your window size one line shorter when you are editing the help screen than when displaying it (if no border is used.)

The help prompt has various versions depending on the width of the memo window. The prompt may be less than clear for some applications, and excessive for others (in which you may want to use your own

prompts.) The alternative is to supply your own bottom prompt in the TEXT parameter. (See TEXT: General purpose text exchange parameter )

### **MMO'MNU: Menu Mode**

Bit value: 4096

Vertical light-bar menu. Works very similar to the MMO'DPG mode except that instead of scrolling the entire contents of the window, a selection bar or arrow is moved up and down (selecting one line at a time). Of course, if the memo is too big to fit inside the window, moving the selection bar down to the bottom will cause the text to scroll as well.

The main purpose of this mode is to provide the user with a convenient means of choosing a single item from a list of possibilities. For example, to select the terms on an invoice you might have a memo looking like:

1. Prepaid
2. COD
3. On Invoice
4. 2%10, Net 30

Positioning the Light Bar: The light bar may be positioned with the **Up arrow** and **Down Arrow** keys and/or by typing one or more characters which identify the selection. (The bar moves to the first selection matching the characters entered so far, until no possible match can be made, at which point it beeps and starts the selection process over from the beginning.)

Making a Selection: Actual selections are made by pressing **Enter** or any enabled function key when the light bar is on the desired item. The entire text of that line is then returned in the TEXT parameter, along with the appropriate exit code in the EXTCOD parameter. It is up to the calling program to extract the appropriate control information from the string.

Aborting: If **Escape** is entered, nothing is returned in the TEXT parameter. **AC** will also return nothing in TEXT but it will set EXTCOD to 1 (MMX'QUI). You may support various other types of aborts by interpreting function key exitcodes as you like. Note that on most terminals, **Shift Up Arrow** and **Shift Down Arrow** act like function keys, so you may want to interpret them as meaning to preserve the prior menu selection and move up or down to the next field on the screen. See Making and Loading Function Translation Tables in the **Error! Reference source not found.** section for more information.

Hidden Control Information: You can append a "\ " (backslash) followed by any sort of text control information you like to individual menu items. The backslash and anything following it is not displayed, but it is returned as part of the menu selection. As an example, you might create a program-chaining menu which looked something like this:

1. Accounts Payable\APMENU.CMD
2. Accounts Receivable\LOG AR:\$RUN ARMENU
- etc.

The program could then extract the control information following the backslash and build the appropriate command file (or whatever.)

Variations: See opcodes MMO'FST: Fast Menu Mode and MMO'FFM: Free Form Menu for variations of this mode. Also note that for hard-coded program menus, you may want to use memory based (No-File) memos; see CHANNEL: Memo file channel or specification.

**MMO'TBL: Table Lookup**

Bit value: 8192

Works just like the MMO'SCH (search) mode, except that in addition to the code returned indicating whether the pattern was found, it returns the remainder of the line the pattern was found on in the TEXT parameter. Since you also use the TEXT parameter to specify the search pattern, you may want to make a copy of it for later use.

This mode is useful for implementing table lookup. For example, using the invoicing terms memo shown above, you might have the code "3" in the terms field of the invoice record and wish to print the actual terms on the invoice. You would pass "3." to INMEMO as the search pattern, and it would return "On Invoice" as the response.

Indexing: You will probably need some sort of indexing method for managing special purpose table and menu memos. (These would not normally be attached to standard data records.) If you are managing a relatively small number of these tables or menus (up to, say, 100), you may want to store the index itself as a memo and use another table lookup operation to locate the memo record number in the index. We provide a ready-to-interface on-line help system built on this architecture. See the description of the MMO'ISL: Insert Into Sorted List opcode and the discussion of the On-Line Help and Pop-Up Pick Lists in the Calling Parameters section for more details.

Of course, you are always free to build any other type of memo index externally - the only requirement is that you be able to quickly locate the index record for a particular menu or table name (e.g. "SHIPPING METHOD", "TERMS", etc.), and that the index record then contain the starting memo link.

Of course, you are always free to build any other type of memo index externally - the only requirement is that you be able to quickly locate the index record for a particular menu or table name (e.g. "SHIPPING METHOD", "TERMS", etc.), and that the index record then contain the starting memo link.

For tables in which the items are each one line and the search key is always at the beginning of each line, you may find that MMO'TBL + MMO'ISL is easier to use.

**MMO'FST: Fast Menu Mode**

Bit value: 16384

May be used in conjunction with MMO'MNU and MMO'FFM so that you can select the menu item by hitting the first character(s) of the corresponding line, rather than using the arrows and the **Enter** key. This works best when each menu line begins with a unique character, in which case the selection will be made (and the menu exited) immediately upon pressing the character. However, if one or more menu items share the same starting letter(s), INMEMO will move the bar to the first matching choice, but wait until you type sufficient characters to make a unique match. For example, if a menu contained the choices (AR, AP, GL, and PR), it would exit immediately with "G" or "P", but if "A" was entered, the bar would move to "AR", but wait for the next character. If the next character was not "R" or "P", it would beep, indicating an invalid selection, and the process would start over again at the beginning.

**MMO'EWU: Edit Without Update**

Bit value: 32768

When combined with the MMO'EDT opcode, causes the updated memo text to be returned to the calling program inside of the TEXT parameter, instead of being written to disk. The purpose of this option is to allow you to defer the actual disk update until a later time, such as after the user answers a question about updating that doesn't appear until the entire screen has been entered. Use the MMO'UWE opcode (below) to actually write the memo to disk (replacing any original copy of the memo on disk.) Note that this is not to be confused with NO-FILE mode, which gives you back the memo text in a very usable format. The format of

the memo text returned by opcode MMO'EWU is not documented since there is no reason for you to alter it, and if you did, undefined results may occur when you write it to disk. This operation is prone to confusion and is therefore not highly recommended.

### **MMO'UWE: Update Without Edit**

Bit value: 65536

Used in conjunction with a previous call using opcode MMO'EWU to write a previously edited memo (which has been buffered in the TEXT parameter) to disk. No display operation is performed, so the window size and position parameters are irrelevant. This operation is prone to confusion and is therefore not highly recommended.

### **MMO'SVA: Save Screen Area**

Bit value: 131072

This opcode tells INMEMO to save a copy of the screen area to be covered by the memo prior to putting the memo up on the screen. (As of release 2.0, INMEMO only supports the AM72 save and restore area TCRT's and any emulator, such as the Tracker or A-Shell.) If INMEMO cannot fulfill the save area request (because the appropriate support hardware or software is not present), it will continue to perform the operation specified by the remainder of opcode, and will return the exit status, MMX'SAF (Safe Area Failure) in the EXT COD variable (if the EXT CTL parameter is specified.) Your application should use this information to trigger some other method of restoring the screen display.

Note that opcode MMO'SVA can be combined with other opcodes (such as MMO'EDT or MMO'DSP) or used by itself. It must be coupled eventually, though, with a call specifying opcode MMO'RSA to restore the saved area. The memory used to store the saved areas (whether inside the AM72 or in system memory) is handled like a stack - the last area saved will be the first one restored. If you don't issue a restore request for every save area request, you will eventually fill up the memory allocated, thus disabling the feature. Also note that in the case of the AM72, no error is reported by the terminal to indicate that its save area buffer is full. (The terminal has enough memory to save the equivalent of 1 entire copy of the screen.)

Also note that if using Tracker on a field terminal (like an AM62A or Wyse 50), and extra blank column will be used on either side of the memo box in order to prevent field attributes from spilling over into the memo area.

### **MMO'RSA: Restore Screen Area**

Bit value: 262144

This opcode tells INMEMO to restore the copy of the screen area made earlier (using opcode MMO'SVA), prior to returning to Basic. If no save and restore facility is available, it will return the same MMX'SAF error code described above for the MMO'SVA opcode.

Please refer to the note above regarding the coupling of MMO'SVA and MMO'RSA opcodes with each other and with other opcodes. The most common usage will be to include both the MMO'SVA and MMO'RSA on a single call, together with display or edit opcodes. This will have the effect of a pop-up memo window, which appears and disappears without affecting the underlying screen display.

### **MMO'CX Y: Start at Specified XY position**

Bit value: 524288

This opcode tells INMEMO to start the editing session at the position specified by the EXT ROW and EXT COL fields (within the EXT CTL parameter). Since these same fields are set with the ending cursor

position on exit, and since you can exit from a memo with a function key whose value is returned, you can provide the appearance of popping in and out of a memo on certain function keys.

### **MMO'NOA: No Arrows**

Bit value: 1048576

Eliminates the navigation arrows which otherwise appear in the borders when the virtual pad size is greater than the display window. This is primarily used when there is no border, since in that case the arrows may overwrite memo text.

### **MMO'AAH: Auto Adjust Height**

Bit value: 2097152

Causes INMEMO to reduce the specified number of display rows (height of the window) to match the current memo size. This is aesthetically useful in conjunction with menus to eliminate blank lines from the display, particularly when the menus may be modified at runtime, making it very difficult to set your window size parameters in advance. Note that the ENDROW parameter specified originally is the maximum size; INMEMO will only reduce this - it will never increase it.


### **MMO'IPG: Intelligent Paging Mode**

Bit value: 4194304

May be used in conjunction with MMO'DPG to cause INMEMO to use the scrolling prompt only if the actual memo text size is larger than the display window. If not, it will act exactly like MMO'DSP (display only) mode, exiting without waiting for a response.

### **MMO'DBM: Display Bottom of Memo**

Bit value: 8388608

May be used in conjunction with any display or editing call, including menus, to cause the initial display window to be at the bottom of the current memo text, rather than the top. This is essentially equivalent to forcing a , except without the excess display thrashing. It is most convenient when used with chronologically oriented memos, where the most recent stuff is at the bottom.

Note that if you do include forced type ahead characters, the type ahead will commence at the beginning of the bottom used line of the memo. Also note that MMO'DBM will be ignored if used in conjunction with MMO'NMR (No Memo Redisplay).

### **MMO'ISL: Insert Into Sorted List**

Bit value: 16777216

May be used by itself or in conjunction with MMO'DEL or MMO'TBL to perform insert, delete, and lookup of items in a sorted list. Note that in all cases, there is no display, and thus the window coordinates don't matter, but the entire memo must fit in memory within the size specified by VSPEC.

When used by itself, it inserts the TEXT parameter (which is assumed to be one line) alphabetically into the memo specified by LINK, which is assumed to be a sorted list. For example, if TEXT contained "RUTABAGA\0.69" and the memo pointed to by LINK contained:

```
DIAKON\0.89
EGGPLANT\1.49
ENDIVE\2.19
JICAMA\0.59
SQUASH\0.39
```

then the insertion would be made between JICAMA\0.59 and SQUASH\0.39.

This is one way to use INMEMO to maintain an index which may be accessed with table lookup calls.

Looking Up Items in a List: Although you could use the normal MMO'TBL lookup mode, combining the MMO'ISL + MMO'TBL opcodes allows an easier-to-use alternative. The main advantage of the MMO'ISL + MMO'TBL method is that you don't have to specify the search control characters, and you don't have to worry about inadvertent search matches in the middle of line. MMO'ISL + MMO'TBL use the same sort of search as MMO'ISL + MMO'DEL does. That is, it matches the contents of TEXT (with no wildcards) against each line of the memo, starting at the beginning of the line. Another difference is that the entire line is returned, not just the remainder following the search pattern. The major disadvantage of using the MMO'ISL table lookup (as opposed to the normal one) is that like all MMO'ISL operations, it takes place entirely in memory and thus it is most useful on smaller lists.

Trapping Errors: EXTCOD will contain 1 (MMX'QUI) if there is no room to insert on a MMO'ISL insert operation, or if the search key is not found on a MMO'ISL delete or table lookup operation. It will contain 2 (MMX'VTS) if the VSPEC specified was not large enough to contain the entire memo. (Remember, MMO'ISL operations all require that the entire memo fit in memory in the confines specified by VSPEC). An EXTCOD value of 0 indicates success. In addition, for table lookup, TEXT will contain the line matching the search key if found, or null if not found.

### **MMO'APS: Alternate Prompt Style**

Bit value: 33554432

Turns off the reverse video for the memo title prompt. This may also be accomplished by specifying the MMOCLR parameter and setting the PBCLR field (Prompt Background Color) to 0 (for black).

### **MMO'NAF: No Auto Format**

Bit value: 67108864

This opcode disables the normal auto-formatting which occurs when a memory based (NO-FILE) memo is loaded for display or editing. This may be necessary when loading memos containing lines which barely fit in the allotted width, or when re-loading memos which have been edited in memory already to avoid unwanted line breaks. This problem can occur when the MMO'OTX opcode is used to output a memo direct to a memory array - the array width is equal to the memo width with no extra room for a line terminator. When it is reloaded from this format (without MMO'NAF), INMEMO triggers a word-wrap when the last character fitting on a line is entered, changing the line breaks if the last character was a non-space.

### **MMO'OTX: Output to Text**

Bit value: 134217728

Causes the updated memo to be output directly to the TEXT parameter in a VROWS by VCOLS array format, instead of being written to disk. This is more efficient than reading a memo one line at a time using MMO'LIN provided you have the memory space for the array. Each line is padded with trailing spaces to fill it out to the array width. As an example:

```

MAP1 VSPEC
  MAP2 VWIDTH,B,2,60
  MAP2 VROWS,B,2,100
MAP1 MEMO'IN'MEMORY
  MAP2 MEMO'ARRAY(100),S,60
MAP1 TEXT,S,6000,@MEMO'IN'MEMORY

```

The above example illustrates mappings for a physical memo size of 60 columns by 100 rows. The variable `TEXT`, mapped on top of an array of 100 60-byte lines, will be passed as the `TEXT` parameter. A single call to `INMEMO` with opcode set to `MMO'OTX` will return the memo in the `MEMO'ARRAY` format, allowing easy access to individual lines.

Note that if `MMO'OTX` is combined with the `MMO'EDT` opcode, the file-based memo will be loaded from disk (provided you specify a valid, non-zero `CHANNEL` parameter), normal editing will take place, and then the memo will be output (returned) to the `TEXT` parameter instead of writing it to disk. If `MMO'OTX` is used alone, then no display or editing takes place; the memo is essentially just copied from disk into the memory array.

### **MMO'FFM: Free Form Menu**

Bit value: 268435456

This works similarly to regular vertical menu mode but allows the menu layout to be free form. Instead of treating each line as a menu selection, in free form mode each item must be enclosed inside of [brackets]. (Actually, brackets are the default delimiters - you can change them by specifying some other characters in the `EXTSMI` and `EXTEMI` fields in the `EXTCTL` parameter. The free form menu can contain other text which is not inside of brackets and thus is not selectable. It can also contain any number of rows and columns, just like a regular memo. The only restriction is that individual menu items must be contained on a single line - they cannot wrap from one line to the next.

One use for this menu mode is to make horizontal menu bars, similar to those found on many PC software products. You can create your own pull-down menus by combining a top-level horizontal menu with a series of traditional vertical menus. (You will also probably need either a terminal which supports save and restore area, or `Tracker`, or `A-Shell`, which includes `Tracker`.)

Another way to use free form menus is to implement "Dialog Boxes". Typically these are pop-up boxes which appear with some sort of brief warning or message, and 2 or more selections (sometimes called "Radio Buttons") for you to respond. (Again, for this concept to work cleanly, you need the ability to save and restore screen areas, but note that `INMEMO` will take care of all the dirty work for you, provided you have `Tracker` or a capable terminal, or `A-Shell`.)

The `TSTNFL.BAS` test/demo program demonstrates two examples of free-form menus - we suggest you take a look at it. Note that the menu delimiters don't display. If you want to display brackets to make the choices more obvious in a menu containing other text, you will have to change the delimiters and then surround your selections with brackets as well as the new delimiters. `TSTNFL.BAS` gives an example of this also.

### **MMO'OPT: Optimized Disk I/O**

Bit value: 536870912

(Only applies under `AMOS` as `C` version always uses optimized mode.) In the normal `AMOS` mode, when a memo is updated, an entire new copy is written to the file first, then the old copy is deleted, and finally the first record of the memo is relocated to take over the original link position. If the file fills up or another error occurs during writing of the new copy, the original copy is preserved. However, the cost of this additional

safety is greater fragmentation and a lot more disk I/O, especially for large memos. In the optimized mode, the new memo is written directly on top of the old one, adding to it or truncating as needed. This is much more efficient but does mean that you may end up with a partially written memo if the file filled up while writing it out.

### **MMO'RET: Return Key Exit**

Bit value: 1073741824

Causes the **Enter** key to exit from the memo (same as **Escape**) if the cursor is already on the last line (as defined by VROWS.)

### **MMO'HDR: Memo Header Option**

Bit value: 2147483648

This may be added to MMO'LIN to make the return-one-line operation work like the example in a previous version of this document with respect to the handling of invisible headers. The manual used to show the invisible header being returned bracketed by chr\$(26) when in fact, MMO'LIN mode without MMO'HDR returns invisible headers bracketed by [brackets]. This is a case where we were forced to add a frivolous opcode just to stand behind an error in a previous edition of this manual. (AMOS version only.)

## **TEXT: General purpose text exchange parameter**

TEXT is a string whose use depends on OPCODE. For all of the display and edit calls, TEXT should contain the memo area prompt. If not specified, there will be no prompt (or title). (In the picture in section 2.0, the memo prompt is "Comments"). You may also specify a bottom prompt or message by appending a chr\$(13) followed by the bottom prompt text. If you include the chr\$(13) but nothing after it, you will have no bottom prompt, even in menus and other situations which normally contain an automatic bottom prompt. For example:

```
TEXT = "Top Title" + chr$(13) + "Bottom Title"
```

TEXT will be displayed in reverse video (see MMO'APS: Alternate Prompt Style for exceptions), left justified along the top of the memo. If you want to center the TEXT string, add sufficient leading spaces to move it over the desired amount; the spaces will not display in reverse video but they will shift the text string over to allow centering.

The other uses of TEXT are discussed below along with the corresponding memo operation:

**Return 1 Logical Line:** For this operation, the line is returned in TEXT. Note that you must check POS (XPOS) to determine if the returned TEXT is valid.

**Application Controlled Memo Editing:** For edit operations on file-based memos, INMEMO will process characters from the TEXT parameter prior to turning the keyboard over to the operator, allowing you to pre-load initial memo values, time and date stamp, insert a blank line at the top of the memo, etc. This is accomplished by appending a tilde (~) to the prompt (if any), followed by any string of characters to be interpreted exactly as if they had been typed, and calling INMEMO in an editing mode.

A few pointers on application-controlled memo editing are in order here. First, you can mix editing and text characters together, and you may want to terminate the string with an ESCAPE character (chr\$(27)) or else the user will be left in editing mode after all of the TEXT characters have been processed. You can rely on INMEMO to word-wrap the lines for you, although this gets visually messy if the absolute line length is longer than the window (see VSPEC: Set maximum physical width and height below.) If adding a lot of text at once, you might want to turn off the display by adding MMO'SIL to the OPCODE parameter. If you

put your own line breaks in, use only a carriage return (`chr$(13)`) and no line feed between lines. If you want to time and date stamp each new entry in a memo pad, use a control-E (`chr$(5)`) followed by a carriage return to move the cursor to the end of the memo and start a new line. Follow with the time and date string and then end the TEXT string there; the user will be left in editing mode next to the time and date which you entered automatically. (You can set up reverse chronological order in a memo consisting of one-line entries by inserting a line with control-B at the start of the memo for each edit operation.)

Note that if you don't want the time-stamped entry to be saved unless the operator actually adds something to it, then end the TEXT string with a tilde. This resets the internal status flag which keeps track of whether the memo has been modified which in turn determines whether it needs to be re-written to disk on exit.

If using the application-controlled editing feature for some sort of demonstration purpose, you might want to slow the processing of the text. This can be accomplished by inserting additional tilde characters in the text to be loaded. Each tilde (after the initial one) will cause a 1/10 second delay. Try the TSTMMO program supplied with INMEMO for an example of this - you can examine its source code for more details.

No-File Mode: In no-file mode (specified by setting the CHANNEL parameter to zero), all memo text is read from and written to the TEXT parameter (instead of the disk file). The format for TEXT is the same as for application controlled memo editing, except that the characters are treated as if they were loaded from a file instead of entered from the keyboard, and thus they do not update the cursor position and editing commands are not recognized. If you want to preload some text and then follow with some editing commands (for example, to position the cursor at the end of the last line with a control-E followed by control-N), then append a `chr$(3)` to the end of TEXT, followed by your editing commands. For example,

```
TEXT$ = TITLE$ + "~" + MEMO$ + chr$(3) + chr$(5) + chr$(14)
```

See the discussion on CHANNEL: Memo file channel or specification below for more information.

Specifying Defaults for Selection Menus: For opcode MMO'MNU (menu mode), you can use TEXT to specify the starting position of the selection bar. This can be quite important for selection menus within a maintenance screen to prevent the selection from accidentally being changed back to the first option during editing. For example, in a menu of 10 choices, if you selected the third choice by moving the selection bar down and hitting **Enter**, you would expect the selection bar to start on that same choice if you re-enter this selection menu. To accomplish this, append a `chr$(3)` to the end of the TEXT parameter, followed by enough characters of the item description to make a unique match. For example, consider this selection menu:

1. UPS Red
2. UPS Blue
3. UPS Ground
4. US Mail

To specify the 3rd option (UPS Ground) as the default (initial) position of the selection bar, you need to specify at least "UPS G" to make a unique selection:

```
TEXT$ = TITLE$ + "~" + MENU$ + chr$(3) + "UPS G"
```

Note that this example assumes you were in no-file mode, where MENU\$ was a string containing the list of menu options. The tilde (~) is always needed to terminate the prompt (or title) unless that is the only thing contained in TEXT. The `chr$(3)` is always needed to flag the end of the no-file memo text and the beginning of the default selection specification. Here are some examples of other valid strings for TEXT with menu defaults:

```
TEXT$ = TITLE$ + "~" + chr$(3) + DEFLT$
TEXT$ = "~" + chr$(3) + DEFLT$
```

The first example illustrates a file-based menu; note that both the tilde and the `chr$(3)` character are needed - one to terminate the `TITLE$` and the other to terminate the null pre-load string. The second example shows the same situation without a title. Again, note the need to specify both the tilde and the `chr$(3)`.

If the default menu item is not found, the menu bar will start at the top.

**Specifying Invisible Menu Headers:** You may pass an invisible header string to `INMEMO` when editing or adding a new memo. This string is stored at the beginning of the memo and does not display, but it can be retrieved using `MMO'LIN`, edited with `^S^H` (see Operator usage) and dumped with `DMPMMO.BAS`. To specify an invisible header, enclose it in the preload area of `TEXT` as follows:

```
TEXT = "Title" + "~" + MMO'INVHDR + "Invisible Header" &
+chr$(19) + "other preload text"
```

`MMO'INVHDR` is defined in `MMOSYM.BSI` and actually equals `chr$(19) + chr$(8)`.

**Edit without Update and Update without Edit:** For these opcodes (`MMO'EWU` and `MMO'UWE`), the `TEXT` parameter is used to temporarily store the updated memo (after the `MMO'EWU` operation) until it is written to disk (with the `MMO'UWE` operation.) Note that you can still pass a title and preloaded text into the `MMO'EWU` operation, but that it will be replaced by the updated memo text. Naturally the size of `TEXT` must be large enough to accommodate the memo.

**Opening the Memo File:** When using the `MMO'OPN` opcode to open the memo file, `TEXT` must contain the file specification of the memo file.

**Output memo to print file: (A-Shell only.)** As a convenience, you may output an entire memo directly to an open print file by replacing the `TEXT` parameter with a numeric variable (of type B or F) containing the channel number of a file open for sequential output. In order to prevent the output from going beyond the end of the page, you may limit the number of lines output by setting the maximum in `EXTLIN`. (If `EXTLIN` is 0, there is no maximum.) If the operation is interrupted by the maximum, the next call will continue where it left off, allowing you to use your own page header logic. Also, to position the memo lines on the page in the proper column position for your report, you may specify a margin in `EXTMGN`. (See `EXTCTL`: Extended options for mapping of `EXTLIN` and `EXTMGN`.)

## CHANNEL: Memo file channel or specification

`CHANNEL` specifies the memo file channel number (0 for memory based memos.) It may also specify the memo file `ddb/buffer` area or, under `AMOS`, point to the `ddb/buffer` area.

If the memo file is opened with a Basic `OPEN` statement, then `CHANNEL` must contain the open channel number. (`INMEMO` will use a variation of the `SYSLIB` routine `$FLSET` to locate the file `ddb`.)

If you wish to have `INMEMO` open and close the file for you (using opcode `MMO'OPN` and `MMO'CLS`) then `CHANNEL` must either be mapped as a file buffer or it must be a pointer to a file `ddb/buffer` area. The easiest method is to map `CHANNEL` as follows:

```
MAP1 CHANNEL, X, 800
```

(The size must be large enough to contain a file `ddb` and buffer; we recommend 800 bytes to allow for some minor expansion.) Also note that you cannot use this mapping for `NO-FILE` mode (described below), since there is no clean way to give it a numeric value.

(AMOS only.) As an alternative, you can allocate an 800-byte area somewhere and set CHANNEL to point to it. This is an advanced technique and is probably only useful to programmers using other host languages besides Basic. If used, the format of CHANNEL must be a standard 4-byte long word (high word, low word).

Under A-Shell, as a convenience, you may map CHANNEL as a string and load it with the file specification of the memo file and INMEMO will automatically take care of opening and closing the file for you. If the file does not exist, INMEMO will automatically create it (32K.) INMEMO will also automatically expand it as needed. (This shortcut simplifies casual use of INMEMO considerably, since it eliminates the need to open and close the file, as well as the need to use the MAKMMO utility to create the file and the EXPMMO utility to expand it.)

INMEMO will determine whether CHANNEL is a file channel, a ddb/buffer, or a pointer to a ddb/buffer by examining its size and value.

If CHANNEL is 0, INMEMO operates in NO-FILE mode. In this mode, it does all of its I/O through the TEXT parameter. For input, the TEXT should be set up as for preloading characters - place a tilde (~) after the prompt and follow with the text to be loaded. The only difference is that these characters will be loaded directly into the memo-editing buffer, rather than processed as if you were typing them. Consequently, the text should not include any control characters except carriage returns (optional) to terminate each line. INMEMO will do its own word wrapping if necessary to make the lines fit. We decided to do it this way because the display comes up much faster if all the formatting is done in advance of the display, and it also eliminates having to do any scrolling if the memo text extends beyond the window boundaries. Just as with a normal disk-based memo, the first 'page' will display initially, and you can always use the scrolling commands to reposition the window.

On output, (if you set the MMO'EDT bit), it writes the updated memo text back to the TEXT parameter. The format of the returned memo is identical to the format during editing, which is defined by the VSPEC parameter. It is simply a linear array of VWIDTH times VROWS characters, with all lines padded with spaces and no line terminating characters. The only terminator is a null byte that is placed after the last non-blank character in the memo. (E.g., if the VSPEC format was 20 rows of 80 characters, and the returned memo ended in the middle of the 5th row, then 4 rows of 80 characters would be output.) Also note that the maximum number of characters to be output in this mode is limited to the smaller of the TEXT parameter size, the memo size, and 64K characters.

Note that NO-FILE mode can be used in conjunction with any of the other modes, except the search mode and table modes. For example, you can use it with display-only, display-with-scrolling, editing, or menu mode.

Why use this NO-FILE mode? Obviously, if you are supplying the text directly, the display-only modes are really no more than glorified PRINT statements. But that may be very handy in some situations, particularly when taking advantage of the scrolling and automatic formatting capabilities. The editing mode may be useful as an extended INPUT (or INFLD) call, allowing you to directly input a 'whole bunch' of text at a time, (possibly storing it in some other file system.) Menu mode may be the most useful, since it is perfect for pop-up menus. Although there may be an argument for storing pop-up menus on disk, creating them on the fly with NO-FILE mode is much faster. This is particularly handy for selecting among choices that are stored in another type of data file. For example, you could implement a customer lookup routine which used a normal input field to input a partial name, then scanned a key file locating all the customers that matched the partial key and built a list of them in memory, then used INMEMO to display and select the desired customer from the list.

A very simple program to demonstrate no-file mode is provided with the INMEMO release called TSTNFL.BAS. Just compile it, load INMEMO.SBR, and RUN it (and then look at the source code.) It demonstrates many of the possibilities for no-file memos.

## **STROW: Starting row**

---

A standard numeric parameter specifying the starting row. (See note concerning border under OPCODE.)

## **STCOL: Starting column**

---

A standard numeric parameter specifying the starting column. (See note concerning border under OPCODE.)

## **ENDROW: Ending row**

---

A standard numeric parameter specifying the ending row. (See note concerning border under OPCODE.)

## **ENDCOL: Ending column**

---

A standard numeric parameter specifying the ending column. Note that INMEMO will automatically reduce ENDCOL to the maximum width of the terminal if necessary. This allows you to define 132 column wide windows for use when the terminal supports it without sacrificing compatibility with 80 column terminals. (See note concerning border under OPCODE.)

## **LINK: Pointer to location of memo within memo file**

---

A 2 byte or 4 byte binary variable containing the link to the first record of the comments for the current primary data record. Note that this field will have to be included in the primary data file record, and that it must be the same size as the links in the memo file. (Also see XPOS: Return status codes.)

On return from any editing call, LINK will contain the updated pointer to the memo pad, and should be saved in the primary data file record. Note that although INMEMO will preserve the same LINK when editing an existing memo, it may change from 0 to some other number (when adding a memo) or from some other number back to zero (when all of the text within a memo pad is deleted.)

## **XPOS: Return status codes**

---

A 4 byte or 6 byte unformatted variable. If LINK size is 2 bytes then XPOS must be at least 4 bytes; if LINK size is 4 bytes then XPOS must be 6 bytes.

XPOS has three separate uses. For opcode 16 it is used internally by INMEMO to keep track of its position between return 1 logical line (opcode MMO'LIN) calls; for search calls it comes back as zero to indicate pattern not found, else non-zero; for all other calls it is used to return error status (0 if ok, 1 if the memo file filled up, -1 if an illegal link was encountered, and all other codes refer to the corresponding AMOSL file service system error numbers.) Refer to the standard INMEMO include file, MMOSYM.BSI for symbol names for these errors; also see the EXTCTL: Extended options parameter for more information about error reporting.

You should map this variable in your program on an even boundary, preferably as follows:

```
MAP1 XPOS,X,4 ! (2 byte LINK)
MAP1 POS,B,4,@XPOS
```

or

```
MAP1 XPOS,X,6 ! (2 or 4 byte LINK)
MAP1 POS,B,4,@XPOS
```

The variable POS is used with 4 byte LINKs to provide numeric access to XPOS (maximum binary size is 5 bytes). The XCALL INMEMO statement will specify XPOS, but other references to XPOS in the BASIC program will be via POS. For the 2 byte LINKs, the secondary variable POS serves no purpose other than to make the variable usage for the two versions identical (thus simplifying the examples). A suggested mapping that will work for both link sizes is provided in the standard INMEMO include file, MMOPAR.BSI.

POS (first 4 bytes of XPOS) must be all null (zero) for the initial return 1 logical line call for each memo pad. (BASIC will do this for you when your program starts and INMEMO will take care of it from there so you don't need to ever actually set it to zero.)

For search calls, the calling program has to check POS to determine the outcome of the search.

For return-one-line calls you have to check the value of POS on return to determine if the TEXT returned is valid. If the TEXT is valid, POS will be set non-zero. If there were no more comments to retrieve, POS will be returned as zero (leaving it ready to retrieve the next memo pad.)

For all opcodes involving editing, POS is returned as zero if all was ok, -1 (MMO'LKE) if an illegal link was encountered, 1 (MMO'FUL) if the file filled up before all of the memo could be written, and any other AMOSL file service system error code to indicate other errors. You must follow all edit calls with a test similar to the following:

```
XCALL INMEMO, . . .
IF POS <> 0 CALL INMEMO'ERROR
```

The error routine should distinguish among the various classes of errors:

```
INMEMO'ERROR:
  IF POS = MMO'LKE CALL ILLEGAL'LINK'ERROR
  IF POS = MMO'FUL CALL MEMO'FILE'FULL
  PRINT "ERROR #";POS;"IN INMEMO"
```

The symbols MMO'LKE and MMO'FUL are defined in the MMOSYM.BSI ++include file and should be used instead of the numbers.

## VSPEC: Set maximum physical width and height

(Optional) A composite (unformatted) variable used to configure paging options. It should be mapped as:

```
MAP1 VSPEC
  MAP2 VWIDTH,B,2
  MAP2 VROWS,B,2
```

VWIDTH is the maximum width of a memo pad and VROWS is the maximum length (or height) of the memo pad. VWIDTH and VROWS must be at least as large as the memo window specified by ÅSTROW, STCOL, ENDROW, ENDCOL.

If they are equal to the window size specified by STROW, STCOL, ENDROW and ENDCOL, then paging is effectively disabled.

If they are larger, they define the limits of scrolling in both the horizontal and vertical directions.

If VSPEC is not specified, the default for VWIDTH is 132 minus STCOL and the default for VROWS is arrived at by dividing the smaller of 6000 bytes and the amount of free memory by VWIDTH. (6000 bytes was chosen as a likely upper limit for memo pad size.)

When establishing suitable values for VWIDTH and VROWS, consider that the entire buffer must be cleared each time INMEMO is called, so for very large buffers there may be a noticeable delay when calling INMEMO. Also consider the range of uses of the memo text. For example, if it is to show on reports, then you may be limited by the report format to a certain number of columns or lines. A very important consideration is the amount of memory available at run time on each of the user jobs.

If you specify VSPEC and the job doesn't have enough memory, you will get an *insufficient memory* error. However, if you use the defaults (by not specifying VSPEC), INMEMO will automatically adjust the limits to fit the available memory (although this may cause longer memos to be truncated when they are edited).

## MMOCLR: Color Specifications

(Optional) A composite (unformatted) variable used to specify the color palette. It should be mapped as:

```
MAP1 MMOCLR
    MAP2 BFCLR,B,1  ! border foreground
    MAP2 BBCLR,B,1  ! border background
    MAP2 TFCLR,B,1  ! text foreground
    MAP2 TBCLR,B,1  ! text background
    MAP2 AFCLR,B,1  ! arrows foreground
    MAP2 ABCLR,B,1  ! arrows background
    MAP2 PFCLR,B,1  ! prompt (title) fg
    MAP2 PBCLR,B,1  ! prompt (title) bg
    MAP2 WFCLR,B,1  ! warning mssg fg
    MAP2 WBCLR,B,1  ! warning mssg bg
    MAP2 SFCLR,B,1  ! status line fg
    MAP2 SBCLR,B,1  ! status line bg
    MAP2 RFCLR,B,1  ! ruler/reserved fg
    MAP2 RBCLR,B,1  ! ruler/reserved bg
```

The effect of any of these color selections can be disabled by setting it to -1, which causes INMEMO to use the color setting which was in effect when INMEMO was called. The standard AM72 color selections range from 0-15, where 0 is usually black, so if your memo seems to disappear, you should check if you accidentally left any of the color palettes 0 on 0 (black on black).

The border colors (BFCLR and BBCLR) are used to display the border (box) around the memo. Although you can pick any combination you want, you will probably like to leave the border background the same as the screen background, which you can do easily by setting BBCLR to -1.

The text colors (TFCLR and TBCLR) are used for the text contents of the memo. We recommend using a text background color to stand out from the background of the rest of the screen.

The arrow colors (AFCLR and ABCLR) are used only when navigational arrows are displayed in the memo border to indicate text that is outside of the window. (You may want to make these the same color as the border, or at least the same background color.)

The prompt colors (PFCLR and PBCLR) are used to display the optional prompt (or title) of the memo which appears embedded in the top line of the border.

The warning/message colors (WFCLR and WBCLR) are used for displaying warnings and other messages (e.g. `MEMO LOCK', `LINK ERROR', etc.) If you are using INFLD, these colors should normally be

the same as INFLD's MFCLR and MBCLR. We only named them differently to avoid duplicate symbols when using the standard mapping.

The status line colors (SFCLR and SBCLR) are used for restoring the status line area after displaying any warnings or messages which appear on the status line (normally the bottom line of the terminal.) If you are using INFLD, these colors should normally be the same as INFLD's OFCLR and OBCLR. We only named them differently to avoid duplicate symbols when using the standard mapping. Note that as long as the current ambient color scheme when you call INMEMO is the same as you use for the status line, then you can safely set SFCLR and SBCLR to -1.

The ruler/reserved colors (RFCLR and RBCLR) are not used in release 2.0. We added them now because we intend to use them in the next release and figured you might as well map them now while you're at it. The first projected use will be for a ruler, but it may also be used for protected text, background text, etc.

Note that we did not provide different sets of colors to be used in display vs. editing situations like we did with INFLD, because most people prefer to omit redisplaying the text when switching from display to edit mode to save time. If you want to change the colors, since you use different xcalls to display and edit the memo, just use different colors for each xcall. (If you use different colors for editing but don't redisplay the text first, you will have the interesting effect of highlighting the text that has actually changed in the editing colors while the unchanged text remains in the display colors.)

INMEMO will automatically return to the pre-existing color scheme on exiting from a field, so you don't need to worry about it changing your ambient colors.

INMEMO does not check whether your terminal supports color, relying instead on your terminal driver to discard unsupported commands. You should program as if color was supported, since in most cases, color will have no effect on monochrome screens (unlike the PC software where color commands may be interpreted as other video effects on monochrome monitors). However, by passing all color commands to the terminal driver, we leave you the option of modifying the driver to substitute color commands for some other attribute. (Should you attempt this, note that the color commands should not occupy a space, and should operate as mode attributes.)

All of the Basic utility programs (ANAMMO, MAKMMO, FIXMMO, etc.) have been modified to use the color parameters, which are loaded by means of an include file (GETCLR.BSI) and a parameter file (COLOR.DEF).

## **EXTCTL: Extended options**

---

(Optional) A composite (unformatted) variable used for additional return status and control options. It should be mapped as:

```

MAP1 EXTCTL
  MAP2 EXTCOD,B,2 ! exit code
  MAP2 EXTERR,B,2 ! exit error status
  MAP2 EXTMAP,B,4 ! exit code enable bitmap
  MAP2 EXTCOL,B,2 ! cursor column
  MAP2 EXTROW,B,2 ! cursor row
  MAP2 EXTBYT,B,4 ! # bytes in memo
  MAP2 EXTHIT,B,2 ! # rows
  MAP2 EXTWID,B,2 ! longest row length
  MAP2 EXTPRW,B,2 ! # protected rows
  MAP2 EXTSMI,S,1 ! start menu item char
  MAP2 EXTEMI,S,1 ! end menu item char
  MAP2 EXTMRW,B,2 ! error message row
  MAP2 EXTTOP,B,2 ! top window offset
  MAP2 EXTLFT,B,2 ! left window offset
  MAP2 EXTPCL,B,2 ! # protected columns
  MAP2 EXTTIM,B,2 ! max inter-char timeout
  MAP2 EXTLIN,B,2 ! MMO'LIN max; see TEXT
  MAP2 EXTMGN,B,2 ! MMO'LIN margin; see TEXT
  MAP2 EXTOTH,B,2 ! (A-Shell only) other menu exit keys:
    ! 02 = left arrow (EXITCODE -40)
    ! 04 = right arrow (EXITCODE -41)
    ! 08 = up arrow (EXITCODE -42)
    ! 16 = down arrow (EXITCODE -43)
    ! 32 = tab key (EXITCODE -44)
  MAP2 EXTJNK,X,14P ! for expansion

```

The EXTCTL parameter provides access to a number of extended memo control capabilities, and leaves room for future expansion.

EXTCOD is similar to the EXITCODE parameter in INFLD; it receives a code indicating how the memo was exited. The codes are defined as symbols in the MMOSYM.BSI include file:

MMX'OK	0	Normal Updated exit
MMX'QUI	1	Non-Updated exit (Control-C)
MMX'VTS	2	Non-Updated exit (VSPEC too small)
MMX'TIM	4	Updated exit (time out)
MMX'TAB	7	TAB used to select menu item
MMX'SPC	13	Spacebar used to select menu item
	#	Command function exit *

\* Command function exits are returned as negative numbers F1=-1, etc.

EXTERR contains a value indicating various errors. These are also defined in the MMOSYM.BSI include file:

MME'ERR	2	abnormal exit (see POS)
MME'SAF	1	failure to save or restore area
MME'OK	0	normal exit

You may wonder why we had to add two variables to return errors and status instead of just one. The reason is some of the MME ' errors can be independent of the MMX ' exit status, particularly MME ' SAF.

Note that for convenience you can check EXTERR prior to checking the POS parameter for errors; if EXTERR is not equal to MMX'ERR then you don't need to check for POS errors.

When checking for function key exit codes in EXTCOD, keep in mind that since Basic binary variables (less than 5 bytes) are unsigned, negative numbers actually appear as very large positives. If you want to transfer them to signed floating-point variables, use this formula:

$$FLT = EXTCOD - 65536$$

If you are using exclusively Alpha BasicPlus, which supports signed integers (data type I), then you won't have this problem if you map the binary parameters as I, 2 instead of B, 2. (The internal format of these two-byte types is identical, only the interpretation of negatives is different.) Be careful, though, that you do not assume signed behavior if you are using a mixture of Basic and BasicPlus; it may be better to live with the unsigned B format since it behaves the same under both Basics.

EXTMAP is a bitmap used to specify which command key exit codes you want to enable by setting the corresponding bit. Bit 0 = F1, Bit 5 = F6, Bit 31 = F32. If the EXTCTL parameter is omitted or EXTMAP is zero, no command key exits will be allowed. Command key exits work just like the **Escape** key (memo is updated) except that they return a code in the EXTCOD parameter indicating which function key was used.

EXTROW and EXTCOL will return the position of the cursor on exit, relative to the virtual space of the memo. 0,0 is the upper left corner of the memo. These can also be used with the MMO'CX Y opcode to force the cursor to start in a particular position.

EXTBYT returns the physical size of the memo written to the disk, in bytes, not counting the links or the bytes unused in the last physical memo record.

EXTHIT returns the height (max row) and EXTWID returns the width (max col) of the updated memo.

EXTPRW contains the number of rows, starting from the top, to be designated as protected. Protected rows may be displayed, but the cursor is not allowed to move up to those rows. Although the protected area may only consist of a whole number of lines, starting from the top, it may be specified relative to the top (using positive numbers) or bottom (using negative numbers). For example, EXTPRW = 5 would protect the top 5 rows, whereas EXTPRW = -2 would protect all but the bottom row. EXTPRW = -1 protects the entire (existing) memo, which in effect puts the user in append-only mode.

EXTSMI and EXTEMI may be used optionally to specify alternate free form menu item delimiter characters. If not specified, the defaults are " [ " and " ] ".

EXTMRW may be used optionally to specify where unusual messages (e.g. File full, Memo link error, etc.) are displayed. Otherwise they display on line 24 or within the memo box.

EXTTOP and EXTLFT may be used optionally to specify the scrolled position of the editing window within a larger memo. For example, if you set them to 5 and 10, respectively, the memo window would start out with the upper left corner of the window over row 5, column 10 of the memo.

EXTPCL contains the number of columns, starting from column 1, to be designated as protected. Protected columns are displayed, but the cursor is not allowed to move into those columns, and some editing operations which would damage those columns (such as delete line and concatenate lines) are disallowed. Under A-Shell, when you are preloading text (such as a date stamp) into a memo, the protection is turned off during the preload, allowing you to put the date stamp in the protected area.

EXTTIM may be used to specify a maximum number of seconds of no keyboard activity (to avoid letting a memo be kept in use indefinitely by someone forgetting to exit.) The timer is reset after each character

input. If the timer expires, INMEMO saves and exits, as if the user had exited normally with the **Escape** key, except that the EXTCOD parameter is returned as MMX'TIM (4) instead of MMX'OK (0).

EXTLIN and EXTMGN are used with an A-Shell-only feature in which you can use the MMO'LIN opcode to output an entire memo to a sequential output (print) file in one step rather than one line at a time. (See the description of the TEXT parameter for more information on invoking that feature.) In this mode, EXTLIN may be used to set a maximum limit on the number of lines output (presumably to avoid going beyond the end of the current page.) The next call (assuming the parameters are not disrupted) will continue from where it left off (presumably after the program did whatever was appropriate to start a new page of output.)

Leaving EXTLIN set to 0 removes the maximum limit. EXTMGN may be used in this mode to specify a left margin for the lines output, allowing you to align them in a particular column of the report.

EXTJNK was defined to allow for additional expansion without requiring you to reallocate your parameter space



# Chapter 6

## Examples

---

This section discusses in greater detail many of the types of INMEMO calls.

### Opening the Memo File

---

You must open the memo file before doing any other operations with it. There are two ways to open the memo file. The old method was to open it in Basic as follows:

```
CHANNEL = 100
OPEN #CHANNEL, "CUSMAS.MMO", RANDOM' FORCED, 64, FILE100
```

The variable CHANNEL must be used in all of the INMEMO calls. The record size is always 64. The record number variable (in this case FILE100) is not used by INMEMO, but must be specified to comply with BASIC syntax. We recommend using RANDOM' FORCED regardless of whether you are operating with LOKSER or not.

The alternate method of opening the memo file is with a call to INMEMO. In this variation, CHANNEL is a ddb/buffer area rather than the file channel number. Map it as:

```
MAP1 CHANNEL, X, 800
```

The other parameters would be set as follows:

OPCODE should be MMO'OPN.

TEXT should contain the memo file specification.

POS will be set to zero if ok, else it will contain the file error #.

EXTERR will be set to MME'OK if ok, else it will be set to MME'ERR.

All other parameters are irrelevant.

### Display Memo Pad

---

To display a single memo pad, the parameters should be set as follows:

OPCODE should be MMO'DSP (display) plus any of the following: MMO'BDR (draw border) MMO'NBR (eliminate redundant redisplay of border). If you want to allow the display to be scrolled through multiple pages within the memo pad, add MMO'DPG (display paging) and possibly MMO'LID (smart line insert/delete).

TEXT should contain the prompt to be displayed above the upper left corner of the memo pad. In the example shown in section 2.0, TEXT was set to "Comments". If no prompt is desired, just set TEXT to null. Note that if the border option is used, the prompt will be embedded in the border, since it too displays above the comment window. Also note that this makes it impossible to use a prompt if STROW=1, since there is no place for it to display.

CHANNEL is the file channel number that the memo file is open on. If the file is not opened, an error message will display and the program will abort to the monitor.

STROW, STCOL, ENDROW and ENDCOL must specify the 4 corners of the memo pad window. If the actual memo pad is too big to fit within the window defined (and the MMO'DPG option is not specified), the remainder will not display. This will not cause the remaining text to be deleted.

LINK must contain the starting physical memo record number of the memo pad to display.

POS will be set to zero if no errors occurred, otherwise it will contain the error code.

EXTERR will be set to zero if no errors occurred, otherwise it will contain MME'ERR.

VSPEC is not needed for the 1 page display (opcode MMO'DSP). It must reflect the absolute pad dimensions for the scrolling display (opcode MMO'DPG), just like it does for editing calls.

## Edit Memo Pad

The Edit Memo Pad call is used to either add a new memo pad or to modify an existing one. The parameters are as follows:

OPCODE should be MMO'EDT (edit vs. display) plus the values of the border and smart insert/delete options desired. For example, if you want to draw a border around the window and use smart insert and delete, set OPCODE to MMO'EDT+MMO'BDR+MMO'LID. If you don't want the border but do want the smart insert/delete, set OPCODE to MMO'EDT+MMO'LID. Note that INMEMO will still check in the terminal driver to make sure smart insert/delete functions are available before using them.

TEXT, CHANNEL, STROW, STCOL, ENDROW and ENDCOL are set the same as in the display memo pad call (see above).

LINK depends on whether you want to start a brand new memo pad or modify an old one. If you are adding a new one, then LINK should be set to zero. On return, INMEMO will set LINK to point to the first physical memo record in the memo pad.

If you are modifying an existing memo pad, LINK must contain the starting physical memo record number of the existing memo pad. This will allow INMEMO to display the memo pad before editing, and also to allow it to delete the old memo pad before adding the new one to the file. Again, on return, INMEMO will set LINK to point to the first physical memo record in the memo pad. This should then be stored in the primary file record to which the memo pad is attached. Note that when editing an existing memo pad, the LINK variable will not change, unless you delete all of the text in the memo pad, in which case the LINK will come back as zero. (Removing all of the text from a memo is functionally equivalent to deleting it with OPCODE = MMO'DEL.) Since you can't control whether the operator deletes all of the text, you must either **always** rewrite the updated LINK back to the primary data record or at least check if changed, and then write it back. Failure to do this will result in memo link errors.

VSPEC should be set to control the limits of horizontal and vertical scrolling of the memo window.

POS will be set to zero if no error occurred, otherwise it will contain the error code.

EXTERR will be set to zero if no error occurred; it will be set to MME'SAF if you requested to save and/or restore the screen area and this could not be done, and it will be set to MME'ERR if any other error occurred.

EXTCOD will be set to 0 if the memo was exited normally (with Escape), MMX'QUI if the user aborted with ^C, or it will equal the negative of the function key number used to exit the field. Note that you have to convert the unsigned binary EXTCOD to a signed format in order to check for negative numbers.

### **File Full**

If the memo file is too full to add the new memo pad to, INMEMO will display the appropriate error message on the bottom of the screen and require the operator to hit RETURN to proceed. INMEMO will then return to the calling program in the normal fashion, except that it will set POS to MMO'FUL (1). The program should have some sort of error recovery to process this and all other INMEMO errors:

```
XCALL INMEMO,MMO'EDT, . . .
IF POS <> 0 CALL INMEMO'ERROR'RECOVERY
```

Note that you can use EXPMMO to expand the memo pad. Also note the use of XPOS and POS - see the parameter definition of XPOS: Return status codes for more details.

## **Delete Memo Pad**

The Delete Memo Pad call is used to delete a memo pad when the corresponding primary data file record is deleted. The parameters are:

OPCODE should be MMO'DEL.

CHANNEL is set the same as for the display memo pad and edit memo pad call.

TEXT, STROW, STCOL, ENDROW and ENDCOL are not used.

LINK must contain the starting physical memo record number (as in the display memo pad and edit memo pad calls). It will be returned as 0.

POS will return as zero if the delete was successful, else it will contain the error code.

EXTERR will be set to 0 if no error occurred, else MME'ERR.

VSPEC is not used.

## **Printing Memo Pads**

There are three methods of printing a memo pad. One is to use the MMO'LIN opcode to retrieve one logical print line at a time. The second is retrieve the entire memo into an array in one step using the MMO'OTX opcode, and then print the individual lines of the memo. To use the second method, just set the parameters the same as if you were going to display the memo, except that OPCODE should be set to MMO'OTX, and TEXT should be double mapped as described under MMO'OTX in Calling Parameters section of this document. Then just print the individual lines of the array as you would normally.

The third method is available under A-Shell only, and is described after the more traditional one-line-at-a-time method below.

The first and more traditional method is logically similar to reading a sequential file, one line at a time, and outputting it to a print file as you go. The parameters and procedure for this method are as follows:

OPCODE should be MMO'LIN

TEXT will receive 1 logical line of memo text.

CHANNEL is the memo pad channel number, as in the other memo pad calls.

STROW, STCOL, ENDROW and ENDCOL are not used.

LINK must contain the pointer to the FIRST physical memo record of the memo pad to be fetched. (This is just like in the other memo pad calls.)

POS must be zero for the first call. Normally, it will be zero naturally (unless there was an error on the previous operation, or unless you didn't finish retrieving the previous memo pad.) Note that POS must be mapped as a 4 byte variable starting at the same memory location as XPOS, which is either a 4 or 6 byte unformatted variable.

The only error condition that you should check for is POS = MMO'LKE (link error) in which case you should either set TEXT = "[LINK ERROR]" and print it or skip printing TEXT. But in either case, you should return to the loop as you would normally and wait for POS to come back as zero (which it will on the next call). This will reset POS properly for the following call.

XPOS will be set on return to a value that will enable INMEMO to start from where it left off on the next call. When there is no more text to retrieve in the current memo pad, POS will be set back to zero.

The following section of code shows an example of printing one logical memo pad. It is assumed that LINK points to the start of the memo pad, POS is zero initially (and mapped as in Section 5.0), the memo file channel is 5, and that TEXT is a string. Note that if the memo contains an invisible header, it is returned as the first line, surrounded by chr\$(26) characters. Since these don't print very well, we translate them below into brackets; you may choose to just discard them:

```
PRINT'MEMO'LOOP:
  XCALL INMEMO,MMO'LIN,TEXT,CHAN,0,0,0,0,LINK,XPOS
  IF POS=MMO'LKE TEXT = "[ILLEGAL LINK]"
  IF POS=0 GOTO DONE
  PLINE=TEXT ! (PLINE used in PRINT'LINE)
  CALL PRINT'LINE ! print 1 line on report
  GOTO PRINT'MEMO'LOOP
DONE: ...)
```

Note that the variable XPOS is specified in the XCALL statement, but references are made to POS.

Under A-Shell, you can transfer a logical memo directly to an open print file in one step. The trick is to replace the normal string TEXT parameter with a numeric (B or F) parameter specifying the channel number of a file open for sequential output, as shown below...

```
OPEN #14, "REPORT.PRT", OUTPUT
```

<for each record you want to print, CALL PRINT'RECORD...>

```

PRINT'RECORD:
IF LINCNT > MAXLIN-5 CALL PRINT'HEADER
<print data fields of record on report>
LINK = <link from from data rec>
EXTLIN = MAXLIN-LINCNT      ! (max memo lines to output)
EXTMGN = 30                ! (left margin on memo)

PRINT'MEMO:
  XCALL INMEMO,MMO'LIN,14,CHAN,0,0,0,0,LINK,XPOS
  IF POS=MMO'LKE PRINT #14,"[ILLEGAL LINK]"
  IF POS=0 RETURN
  ! memo too long for page...
  CALL PRINT'HEADER
  GOTO PRINT'MEMO      ! (print rest of memo)

```

## Application Controlled Memo Editing

Any characters following the first tilde (~) in the TEXT parameter will be processed exactly as if they were entered from the keyboard (provided this is not a NO-FILE memo. See next example for that.) This allows the application to transfer text from another source into memo pads. All other parameters should be set up as for an Edit Memo Pad call:

OPCODE same as for the Edit Memo Pad call described above, except that you might want to add the MMO'SIL option to turn off the display. This is useful for speed and visual cleanliness when you are transferring a lot of text and you aren't going to allow editing from the keyboard after the text is complete. (Keyboard editing will be difficult without any display)

TEXT may contain an optional prompt (as with the display and edit operations), followed by a tilde (~) and then the string of characters. There is no particular limit to the length of the string, other than that you have to map the string in your program (and thus provide memory for it). Obviously, you have to define a large enough memo pad (via the VSPEC parameter) to hold the text characters.

CHANNEL, STROW, STCOL, ENDROW and ENDCOL are set the same as in the display or edit memo pad call (see above).

LINK depends on whether you want to start a brand new memo pad or modify an old one, just as with the edit memo pad call. If you are adding to an existing memo, you will probably want to use a Control-E and a carriage return in your TEXT string to move the cursor to the end of the memo.

VSPEC should be set to control the limits of horizontal and vertical scrolling of the memo window. POS is set to zero if no error occurred, otherwise it will contain the error code.

EXTCOD and EXTERR will be set the same as for a normal editing session.

Note that if you are transferring text from one memo pad to another, you may wish to use the XFRMMO.SBR utility, which is faster and easier than retrieving the text from one memo a line at a time and loading it into another. The following example illustrates an approach to time and date stamping memo entries:

```

OPCODE = MMO'EDT + MMO'BDR + MMO'LID
TEXT = PROMPT$ + "~"
IF LINK<>0 TEXT = TEXT + chr$(5) + chr$(13)
TEXT = TEXT + TODAY$ + " " + HOUR$ + " "
XCALL INMEMO,OPCODE,TEXT,CHAN,STROW,STCOL,ENDROW, &
    ENDCOL,LINK,XPOS,VSPEC,MMOCLR,EXTCTL
IF POS <> 0 CALL INMEMO'ERROR'RECOVERY

```

Note that if this is a new memo, we can start at the beginning. Otherwise, we use a Control-E and carriage return (`chr$(5)+chr$(13)`) to move to the end and start a new line. The cursor will be left just past the `HOUR$` for the user to enter the text.

## Memory-Only Pop-Up Pick List

This example demonstrates the use of `INMEMO` to display a pop-up pick-list of choices that is built on the fly. A situation where this would be useful is where a partial lookup key has been entered and now you want to retrieve a list of all the values that match the partial key, display them in a pick-list, and allow the operator to choose from the list.

```

KEY$ = <partial key>
CALL PICK'LIST ! (returns TEXT of choice)
IF TEXT="" GOTO <back to input a new partial key>
...

PICK'LIST:
TEXT = TITLE$ + chr$(13) + "Return to select,&
    ESC to abort" + "~" VROWS = 1
FOR <each record matching partial key...>
    TEXT = TEXT + <next key> + chr$(13)
    VROWS = VROWS + 1
NEXT <matching record>

SROW = 10 : EROW = (SROW+VROWS-1) MIN 23
! menu mode, border, save and restore window...
OPCODE = MMO'MNU + MMO'BDR + MMO'SVA + MMO'RSA
XCALL INMEMO,OPCODE,TEXT,0,SROW,SCOL,EROW,ECOL, &
    LINK,XPOS,VSPEC,MMOCLR,EXTCTL
RETURN

```

## Inserting an Invisible Memo Header

Invisible memo headers can be edited and inserted using the `AS AH` command. However, in most cases, if you are using invisible headers, you will want to insert them automatically under program control. The procedure is identical to that given in the previous example, except that we need to add the invisible header command keystrokes and text in the preload string. To insert the variable contents `CUSNUM$` in the example given above, we would have:

```

OPCODE = MMO'EDT + MMO'BDR + MMO'LID
TEXT = PROMPT$ + "~"
TEXT = TEXT + chr$(19) + chr$(8) + CUSNUM$ + chr$(19)
IF LINK<>0 TEXT = TEXT + chr$(5) + chr$(13)
TEXT = TEXT + TODAY$ + " " + HOUR$ + " "
XCALL INMEMO,OPCODE,TEXT,CHAN,STROW,STCOL,ENDROW, &
      ENDCOL,LINK,XPOS,VSPEC,MMOCLR,EXTCTL
IF POS <> 0 CALL INMEMO'ERROR'RECOVERY

```

Note that you may want to only insert the header when adding a new memo, although inserting one in a memo with an existing header which just replace the existing header.

## Closing the Memo File

You should always close the memo file before terminating the program to insure that the file buffer contents are written to the disk. There are two ways to close the file; the choice depends on how you opened it. If you opened the file in Basic, then you have to close it in Basic:

```
CLOSE #CHANNEL
```

where CHANNEL is still set to the same number as it was for the OPEN statement above.

If you opened the file with the MMO'OPN opcode, then you need to make a separate XCALL to close the file, using opcode MMO'CLS:

```

OPCODE = MMO'CLS
XCALL INMEMO,OPCODE,TEXT,CHANNEL,...
IF POS <> 0 CALL INMEMO'ERROR'RECOVERY

```

The only parameters of interest for this call are opcode, CHANNEL and POS (for error reporting); all others are ignored.

## File Errors

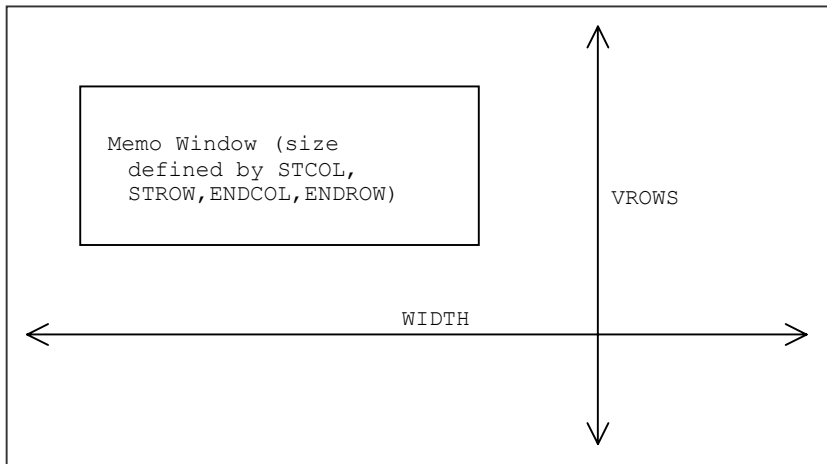
Other than link error (MMO'LKE) and file full (MMO'FUL), all other errors reported by INMEMO in the POS variable will be standard AMOSL file service system errors. The relevant error codes are listed below. (These are defined in SYS.M68.)

2	insufficient free memory	16	file not open
3	file not found	18	bitmap kaput
5	device not ready	19	device not mounted
7	device error	20	invalid filename
8	device in use	24	insufficient queue blocks
9	illegal user code	27	remote is not responding
10	protection violation	28	file in use
11	write protected	29	record in use
12	file type mismatch	33	record not locked
13	device does not exist	34	record not locked for output
14	illegal block number	35	LOKSER queue is full
15	buffer not INITed	37	illegal record size

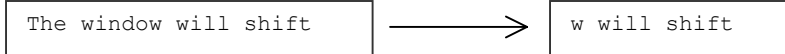
Most of these errors indicate some sort of system malfunction that may require a system reset or action by the system operator. Obviously, file full (MMO'FUL) requires that the memo file be expanded. You should first make sure that no other user is in the memo file and then expand it using the EXPMMO utility (described later). For illegal link errors (MMO'LKE), you will probably want to zero out LINK and write the primary data record back to the file to avoid having the error repeated over and over again. You may also want to run the FIXMMO, or better yet, the ANAMMO utility to determine the extent of the linkage damage and clean up the file.

## Paging

This section gives some examples and additional information about the VSPEC parameter. In general, the memo window within a larger memo pad area looks like this:

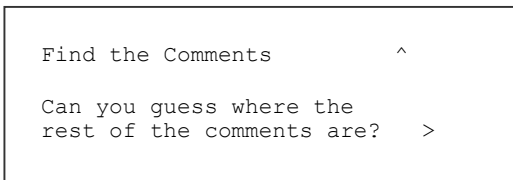


The window always starts in the upper left corner, but shifts automatically to keep the cursor within the window. For example, if you just start typing and don't hit RETURN when you get to the right side of the window, the window will shift to the left, leaving the cursor horizontally in the middle of the shifted window. This is what the window might look like immediately before and after the shift:

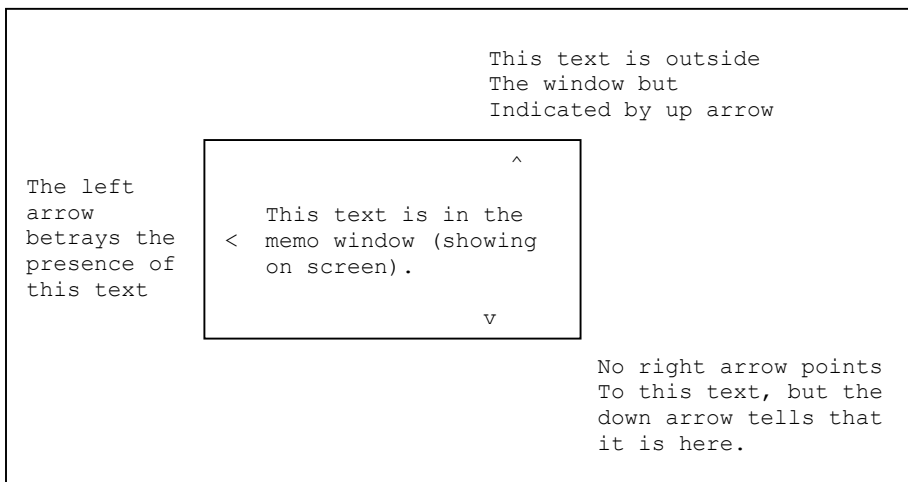


The same type of shift will take place when you move the cursor vertically off one end of the window. However, the window will never move outside of the absolute pad boundaries which are defined by VSPEC. You may disable scrolling in either of the vertical or sideways directions by setting VWIDTH and/or VROWS to equal the size of the window.

If you attempt to edit memo pads which are much larger than the window, you may find it easy to lose your bearings. INMEMO will help you by indicating in which directions outside of the window additional text may be found. If you are using the window border option, it will display arrows in the middle of the borders separating you from text. Without the border option, it will display the appropriate arrows in the upper right corner of the window. For example, if the window looks like this:



The arrow on the right indicates that there is more text to the right. Similarly, the up-arrow in the upper right corner (placed there to avoid the prompt) indicates that more text is above. The lack of arrows on the left and bottom indicate that no more text is in those directions. Note that the side arrows are only set when the text to the side is vertically aligned with the window, while the up and down arrows are set without regard to horizontal alignment. Consider the following example:



## Using INMEMO for Online Help

---

INMEMO is an ideal vehicle for building online help systems, because it automatically handles most of the difficult aspects of help systems: window oriented display, storage and retrieval of hundreds or thousands of text chunks varying widely in size, and easy updating at run time. As of Release 2.1, it also provides an indexing mechanism built into the memo file itself, so you don't need to maintain a separate index file. The only problem left is just sitting down to design how it will all fit together and writing the code. To make this vastly easier for you, we have provided a working prototype help subsystem designed to be `++include'd` in your existing programs with minor revisions. The only requirements are that you provide the trap the help key in your input routine and pass the help keyword to the help subsystem, and that you have screen tracking (via the Tracker or an AM7x/AM65 terminal.)

The subsystem consists of the following modules:

```
HLPMMO.MAP
HLPMMO.BSI
HLPMMO.INI
MSBOXX.BSI
MSBOXX.SBR
JOBPAR.SBR
<A memo file to store the help screens>)
```

A sample program TSTHLP.BAS is also provided which uses the subsystem. Note that it references the help subsystem modules in the ersatz account `HLPMMO:` - you will have to set this up, although you can make it the same as your `INMEMO:` account.

Specific instructions for building the subsystem into your programs can be found by printing out the `HLPMMO.BSI` module. As a rough overview, you have to do the following things:

Use `MAKHLP` to create and initialize a help memo file. This is nearly identical to the regular `MAKMMO` program, except it sets up a special index memo at link 2, and a "help screen not found" memo at link 3

Use `++include` to include the `MAP` and `BSI` files.

Customize the `HLPMMO.INI` file and call it within your program. This defines a number of options, such as the help file name, help screen coordinates, etc.

Specify a help keyword along with your field input parameters.

If you are using `INFLD`, you will want to put the keyword in the `HLPIDX` parameter.

Sense the designated help key in your input field routine and call the help subsystem, passing it the current keyword.

Once you have done this, you can run your program to start building the help screens. When you hit the designated help key, the help subsystem will look in its index to see if the specified help keyword is known. If not, it will ask you if you want to create a help screen for that keyword. If you say yes, it will then open up a memo window and allow you to type. When you exit, it will save the help memo and update the index, so that the next time you hit the help key with the same keyword, it will then bring up the existing memo.

If you want to see what other help screens are available when you are looking at one, hit the help key again. This time it will display the help index menu and put you in menu mode, allowing you to use the light bar to select another help screen. Since the index is sorted, systematic naming of your help screens will result in a system that is very easy to navigate.

Security: The ability to add or edit help screens as opposed to just displaying them is determined by comparing your current experience level (defined by MUSER.LIT in the USER.SYS file and returned by JOBP.SBR) against the MH ' SCRTY variable which is initialized in HLPMMO.INI. If you just get the message that the help screen does not exist, without an option to add one, then try bumping your experience level up using MUSER.LIT or lowering the MH ' SCRTY value.

## Pop-Up Field Level Pick Lists

---

A concept parallel to field-level help screens is that of pick lists which pop-up when you enter a particular field. For example, an invoicing program may have a field for shipping method which takes an integer. When the cursor enters that field, rather than forcing the user to enter the number corresponding to the method desired, you may want to instead pop-up a pick list or menu of the choices, allowing the selection to be made with the light bar. This type of thing can make programs easier to use for beginners, but they are a pain to program, and they can become a pain to use for experienced operators. One solution to the problem of annoying experienced users is to implement a menu key which may be hit when in a field to pop-up the field menu.

We have developed a solution to the other problem of programming difficulty by creating a ready-to-use help menu subsystem which is nearly identical to the help screen subsystem described in the previous section, except that it is used to create and pick from field level menus rather than displaying help screens. Note that not only are these pop-up menus slick, they also take the place of definition files which you would need anyway for defining tables such as shipping method, customer type, terms codes, etc.

The help menu subsystem consists of the following modules:

- HLPMNU.MAP
- HLPMNU.BSI
- HLPMNU.INI
- MSBOXX.BSI
- MSBOXX.SBR
- JOBPAR.SBR
- <A memo file to store the help menus screens>

Setup for the help menu subsystem is nearly identical to that described above for the help screen subsystem with the main difference being that pop-up field menus only make sense for certain fields, and therefore you would only want to activate the designated menu key for those fields. As for the keyword, it can be the same as that used for the help screen, since the menus and the screens are maintained in separate files. (A field could have an associated help screen and/or an associated pick-list menu.)

You may print out the HLPMNU.BSI file for more programming details, and try out the TSTHLP program to get a feeling for how these work.

## Dialog Boxes with Radio Buttons

---

The free form menu mode (opcode MMO'FFM) can be used to create Dialog Boxes with Radio Buttons. A Dialog Box is a box or window which pops up on the screen in response to some sort of condition requiring the operator's immediate attention. A typical example would be to pop up a message warning the operator that you are about to print a report requiring a special form. In order to be considered a Dialog Box, some sort of operator response is generally required. In this example, possible responses might be to proceed, abort, or print an alignment. When the response can be expressed in terms of a multiple-choice question, a common interface technique is to use Radio Buttons, which allow the operator to select one of the choices

graphically. (The term Radio Buttons derives from the old-fashioned car radio station pre-select buttons, where there are a number of choices but only one can be picked at a time.)

To create the type of Dialog Box with Radio Buttons described above, we might set the parameters as follows:

```
DIALOG'BOX:
  OPCODE = MMO'BDR + MMO'SVA + MMO'RSA + MMO'FFM
  CHANNEL = 0
  STROW = 10 : STCOL = 15 : ENDROW = 15 : ENDCOL = 65
  TEXT = "~" + chr$(13) + &
    " Please mount form CHECKS in printer LPT0 "
  TEXT = TEXT + chr$(13) + chr$(13) + &
    "[Ready] [Abort] [Print Alignment Pattern]"
  xcall INMEMO, OPCODE, TEXT, CHANNEL, STROW, STCOL, &
    ENDROW, ENDCOL, LINK, XPOS, VSPEC, MMOCLR, EXTCTL

  if EXTCOD = MME'SAF call REPAINT'SCREEN

  if TEXT = "Ready" goto PROCEED &
    else if TEXT = "Abort" goto ABORT &
    else if TEXT = "Print Alignment Pattern" call ALIGN

  goto DIALOG'BOX
```

Notes: We used the MMO'FFM opcode to activate the free form menu mode, and we used MMO'SVA and MMO'RSA to save and restore the screen area used. (After the xcall, we tested EXTCOD for the error flag indicating if the save/restore worked. If not, we called a routine to repaint the screen manually. This may or may not be necessary or desirable depending on the situation.) Since we were building the Dialog Box directly in the TEXT parameter, we set CHANNEL to zero. The TEXT parameter started with a tilde (~) since we didn't see any need for a top title. This was followed by a blank line, a message, a blank line, and the three choices. Note that the brackets [ ] are the default menu item delimiters - we could have changed them using the EXTSMI and EXTEMI fields with EXTCTL.

In this case, we did our own formatting by including carriage returns as necessary. If the text message was longer than the memo width, INMEMO would have done the necessary word wrapping itself. Note however, that you cannot rely on INMEMO to properly wrap lines containing actual menu selections, unless none of the selections contains any spaces. Free form menus can scroll horizontally, but each item must begin and end on the same line. Items with spaces in them (like the third choice above) might be split over two lines if you left the formatting up to INMEMO's word wrap, causing the menu to malfunction.

On exit, we simply check for the possible responses. Note that the menu item delimiters (brackets in this case) are not returned as part of the item. Also note that the user can abort from these boxes by hitting **Escape** or **Alt-C**, which would cause nothing to be returned in TEXT. In the example above, this would cause the program to fall through the "if" statement and then go back to the top, displaying the Dialog Box over again. Depending on your requirements, you may want to treat this type of abort the same as selecting the [Abort] button.





swap the low and high words before using them in BASIC computations, and then swap them back again before writing them to the memo file. The following routine demonstrates this:

```
MAP1 MEMOLINK, B, 4
MAP1 BSWAP, B, 4
MAP1 BSWPX
    MAP2 W1, B, 2
    MAP2 W2, B, 2
MAP1 WX, B, 2

MEMOLINK = <link within memo file>
BSWAP = MEMOLINK
CALL BSWAP      ! convert to BASIC format
!              ! (low word, high word)
MEMOLINK = BSWAP ! MEMOLINK now in BASIC format

...

BSWAP:
    WX = W1      ! swap low and high words of BSWAP
    W1 = W2
    W2 = WX
    return
```

## Space Compression

INMEMO uses two different space compression algorithms to minimize the storage requirements. In memos created by INMEMO version 1.x, instead of storing  $N$  contiguous spaces in  $N$  bytes, they were compressed into one byte with the value  $128+N$ . Up to 127 contiguous spaces could be compressed into one byte this way.

As of INMEMO version 2.0, we can no longer use byte values 128-255 for space compression because we now support the full 8-bit Latin-1 character symbol set. Instead, another algorithm is used in which blocks of contiguous spaces are represented with two bytes; the first is always value 2 (`chr$(2)`) and the 2nd is the number of spaces to represent.

In order to allow INMEMO 2.0 to be able to read memos written in the earlier format, a special flag (`chr$(1)`) is written to the memo in advance of any true characters which require the 8th bit. When a memo is read from disk and a byte whose value is greater than 127 is encountered, INMEMO 2.0 can tell whether it is a Latin-1 character or an old-style compressed space by checking if the 8-bit flag byte has been read. All new memos are written with the new space algorithm. Note that the new DMPMMO utility displays an indication of which version of INMEMO was used to write the memo being dumped.

Space compression is transparent to both the operator and the programmer. The spaces are automatically expanded whenever they are displayed or returned. Old-style memos are automatically converted to the new format whenever they are updated by INMEMO 2.0.

## 8-Bit Latin-1 Symbols

INMEMO 2.0 supports the full 8-bit Latin-1 international character set, allowing special characters such as `×, Ò, É, Ð, Õ, Î, Ì, ¼, Í, 'a, Ñ, Á, Ç, 'z, Ë`, etc., to be entered and displayed on a Latin-1 compatible terminal such as the AM65. If you wish to take advantage of this feature but must operate in an mixed environment of 7 and 8 bit terminals, we have an add-on product, ISOTRAN, which allows the use of

compose sequences for inputting 8-bit values on 7-bit terminals, and configurable translation tables for mapping 8-bit values to the most suitable display symbol on a 7-bit terminal.

Note that under A-Shell/Windows, you must set `OPTIONS=LATIN1` (in `MIAME.INI`) or choose a font supporting the ANSI character set to permit the use of 8 bit characters. Under A-Shell/UNIX, support for 8 bit characters is dependent on the proper operating system settings. (Consult your UNIX system documentation or guru for further details.)

## Carriage Returns

---

In order to preserve the format of the comments on a logical line-by-line basis, carriage returns are inserted following the last non-blank on each line. This reduces the efficiency of the space compression slightly, since leading spaces cannot be merged together with trailing spaces from the previous line. However, this is made up for partly by the fact that we don't have to store trailing spaces anymore.

More importantly, the embedded carriage returns preserve the format even if the memo window area is expanded. It also allows the retrieval on a logical line-by-line basis for inclusion in reports where the size of the original window area is unknown.

## Invisible Headers

---

Invisible headers may be up to 60 bytes in length, and are always stored at the beginning of the memo, preceded and terminated by `chr$(26)` (Control-Z) characters. Since invisible headers are optional, you need to test the first logical byte (after the link bytes) to see if it is a `chr$(26)` to determine if the invisible header is present.

# Chapter 8

## Operator Usage

---

This section describes the control codes that are used to exit, abort, and edit the memo pad.

### Exiting the Memo Pad

---

To exit from a memo pad editing session, just press **Escape** (just as if you were exiting from display mode in AlphaVUE or SuperVUE). This will update the memo file with any changes made. The programmer may also provide a method of using one or more of the function keys on the terminal to both exit from the memo and send some additional order to the program. (See EXTCTL in the Parameters section for programming information.)

### Aborting an Editing Session

---

If you accidentally destroy some valuable comments during an editing session, hit **Control** - **C** to restore the memo pad to its original condition. You must do this before you hit **Escape**, since that updates the memo file. When you hit **Control** - **C**, INMEMO will return to the calling program without making any modifications to the memo pad.

### Editing Controls

---

In order to make editing large memo pads more efficient, several convenience editing control codes are provided. Note that you could duplicate the action of any of these codes just by using the space bar and typing, but these will save you a lot of time. For example, to insert a line, you could just retype all of the subsequent lines and space out the current line, but this could also be accomplished in one keystroke with **^B** or **Ins Line**.

The control keys chosen act nearly identically to the way they do in VUE or SuperVUE.

In the list below, the symbol **^** is used to indicate that the control key is held down while the following character is typed. For example, **^L** (Control-L) is accomplished by holding down the **Control** (sometimes **Ctrl**) key while you hit the **L** key.

Key	Usage
<b>^K</b> or <b>Up Arrow</b>	Move cursor up
<b>^J</b> or <b>Down Arrow</b>	Move cursor down
<b>^L</b> or <b>Right Arrow</b>	Move cursor right
<b>^H</b> or <b>Left Arrow</b>	Move cursor left
<b>^U</b>	Move cursor to start of current line
<b>^N</b>	Moves cursor just past the last printing character on the current line. Cursor will stop at the end of the current line if that is a printing character.
<b>^^</b> or <b>Home</b>	Moves cursor to 1st row and column. (May shift window.)
<b>^E</b>	Moves cursor to 1st column on last line of memo pad, shifting window if necessary.
<b>^A</b> or <b>Prev Word</b>	Moves cursor to start of previous word.
<b>^W</b> or <b>Next Word</b>	Shifts cursor to start of next word.
<b>^T</b> or <b>Next Page</b>	Shifts memo window to next page.
<b>^R</b> or <b>Prev Page</b>	Shifts memo window to previous page.
<b>^D</b> or <b>Del Char</b>	Deletes the character at the cursor, and slides all the characters past the cursor on the current line left one column.
<b>^F</b> or <b>Ins Char</b>	Inserts a space at the cursor, and moves all the characters past the cursor on the current line to the right. Cannot be done if the current line is full.
<b>^V</b> or <b>Del Word</b>	Deletes the word the cursor is on. Deleted word is moved to the scrap buffer (if defined).
<b>^B</b> or <b>Ins Line</b>	Inserts a new line and moves all the subsequent lines (including the remainder of the current one) down by one. Cannot be done if there is any text on the bottom line (this is to prevent accidental destruction of final line.) Note that this may proceed in either smart or dumb mode, depending on OPCODE and various parameters in the terminal driver.
<b>^O</b>	Appends the next line to the end of the current line, and moves all subsequent lines up one. This is the opposite of the ^B function. Terminal will beep and no action will take place if there is not enough room on the current line to accommodate the following line.
<b>^Z</b> or <b>Del Line</b>	Delete the line that the cursor is on. All subsequent lines move up by one. Deleted line is moved to the scrap buffer (if defined).
<b>^Y</b>	Erase from the cursor to the end of the current line. Deleted text is moved to the scrap buffer (if defined).
<b>^_</b> or <b>Del</b>	Erase the character immediately to the left of the cursor, and move the cursor back one. Usually used to correct the last character typed.
<b>^Q</b>	Toggle character insert mode on and off. In character insert mode, newly typed text is inserted at the location of the cursor rather than overwriting the existing text.
<b>^_</b>	Toggle line insert mode on and off. In line insert mode, a new line is inserted each time you hit ENTER.
<b>Escape</b>	Exit from edit mode (writes the comments out to the auxiliary file).
<b>Enter</b>	In editing mode, moves the cursor to the next line. In menu mode, selects the highlighted item. (See TAB and Space.)
<b>Tab</b>	If in normal (non-insert) editing mode, moves cursor to next modulo-8 position relative to the starting column of the memo pad, without moving any text. If in insert mode (see <b>^Q</b> ), moves the text to the left along with the cursor (like in VUE). In menu mode, provides another way of selecting an item (some applications may respond differently to menu selections made with ENTER, TAB, or Space.)
<b>Space</b>	In menu mode, provides another way of selecting an item (some applications may respond differently to menu selections made with ENTER, TAB, or Space.)

<b>^S^F</b>	(AMOS only.) Reformat paragraph by filling each line with as many whole words as will fit in the width defined by VSPEC. Reformatting starts from the line the cursor is on.
<b>^S^E</b>	(AMOS only.) Reformat entire memo, starting from the line the cursor is on. Note that if you do this accidentally, you can always 'undo' any editing session with ^C.
<b>^S^B</b>	(AMOS only.) Copy the current line to the scrap buffer if available. (This is done automatically prior to deleting a line with the ^Z (Del Line) or ^Y commands.
<b>^S^O</b>	(AMOS only.) Copy the scrap buffer to the current position in the memo. (This effectively undoes the previous delete line operation, or can be used to make copies of a line which has been previously copied to the scrap buffer.
<b>^S^H</b>	(AMOS only.) When entered while the cursor is in the first position of the memo, displays and allows editing of the invisible header. The existing header is display in dim. Whatever you type up to the next control character (including ENTER and Escape) becomes the new invisible header. The only way to preserve the existing header without retyping it is to hit Escape without entering any other keystrokes.
<b>^G</b>	The lead-in character for the alternate keyboard sequence for entering a function key. This is useful only when your terminal either does not support function keys or they don't work properly because of modems or other communications difficulties. To simulate F1, you would enter the sequence ^G 1; F9 would be ^G 9; F10 would be ^G A and F35 would be ^G Z.

## Wrap-Around

When you attempt to type beyond the boundaries of the memo window, one of two things will happen. If the programmer has defined a larger maximum memo pad size than the window and the window is not yet at the boundary, the window will scroll to reposition the cursor closer to the center of the window. If the window boundary coincides with the absolute memo pad size boundary, then the cursor will wrap to the first column on the next row, provided you are not also at the bottom limit of the memo pad. If so, instead of wrapping, the cursor will just stop.

## Scrolling

In situations where the memo window is smaller than the maximum memo pad size, the window will scroll to keep the cursor within it. To help you keep track of where text outside of the current window is, small arrows will display in either the window border (if there is one), or in the bottom right corner of the window (if there are no borders). You can cause scrolling by using any of the cursor movement keys.

## Vertical Menu Selections

When INMEMO is used to select an item from a vertical light-bar menu or list, the choices will appear in a single column with one of them highlighted with a reverse video bar or an arrow. If there are more options than appear at once, there will be a small indicator in the lower right hand corner of the border. To make the selection, move the highlighted bar to the desired item and hit **Enter**, **Tab** or **Space**. (Some applications may distinguish between these ways of selecting an item.) You can move the highlighted marker either with the up and down arrows or by typing the first character(s) of the item description. In some menus, (called fast menus), you don't have to hit any of the explicit selection keys. Instead, the selection is made as soon as you type enough characters to unambiguously select an item. (Typically these kind of menus have unique letters for all the choices, allowing the selection to be made with a single keystroke.) In normal menus, typing

characters will position the highlighted bar but you always have to hit **Enter**, **Tab** or **Enter** to make the selection. Note that when there are several items starting with the same letter(s), you may have to type many characters to position the highlight on the desired item. As soon as you type a letter that does not match, or you use an arrow key, the matching logic starts over from the first character. A little experimentation with the TSTNFL program (supplied with INMEMO as a demonstration) will illustrate the concept.

## Free Form Menu Selections

---

Free form menus work more or less just like vertical menus, except instead of the selections being aligned vertical, with one per row, the selections can be aligned in any format, with anywhere from zero to several selections on each line. Typing characters which match the starting characters of any of the items moves the highlight bar just like it does for vertical menus, and fast menu mode also works the same. The only difference is that the **Left Arrow** and **Right Arrow** keys may be used to move the highlight bar in addition to the **Up Arrow** and **Down Arrow** keys.

## Invisible Headers

---

Although invisible headers are normally used within a program as a way of identifying individual memos for internal use (such as file reconstruction), you can create and edit them as an operator. To do this, move the cursor to the first position in the memo and hit the **^S^H** keys. The terminal will beep, and if there is an existing header, it will display in dim, overwriting the existing display. To exit without changing the header, hit **Escape**. To remove the header, hit **Enter**. Otherwise type in a new header and hit any control key to exit from header mode. When you exit from header mode, the original first line of the memo will be redisplayed.

# Chapter 9

## Installation

---

Installation of INMEMO consists of making an ersatz account to store the files, restoring the tape into that account, and making a few minor changes to the system initialization file. You may optionally want to create some function key translation files at the same time. Each of these steps is discussed in more detail below.

### Installing INMEMO for A-Shell

---

Although a runtime version of INMEMO.SBR and XFRMMO.SBR are included within A-Shell, the rest of the utilities are not included. If you are just running an application that uses INMEMO, you don't need to do anything special to install INMEMO. However, if you want to develop with INMEMO under A-Shell, you need to purchase a separate INMEMO development license, just as you would for AMOS, and you can follow the instructions in this section for installing it, making the necessary adjustments for the different directory structure and operating system environment. Note, however, that the \*.SBR and \*.LIT files are irrelevant under A-Shell (they are embedded within the A-Shell executable) and there is nothing to load in system memory. Only Basic utilities and include files (which are released in source form) are of interest.

### Making an Ersatz Account

---

All of the INMEMO files may be stored in one account, which can be anything, but must be defined as an ersatz account called `INMEMO:`. In addition, if you are planning to use the help screen subsystem or the field menu subsystem, two additional ersatz accounts must be created: `HLPMMO:` and `HLPMNU:`. These can all be the same account. The only reason you may want to keep them separate is if you decide to customize either of the subsystems without changing the file names, putting them in separate accounts will prevent accidental overwriting of those files when restoring an update.

To define the ersatz accounts, use VUE to edit the `SYS:ERSATZ.INI` file and add the following lines:

```
INMEMO: = DSKx: [p, pn]
HLPMMO: = DSKx: [p, pn]
HLPMNU: = DSKx: [p, pn]
```

where `DSKx: [p, pn]` should be replaced by the actual disk and account numbers you decide to use.

## Restoring the Files

---

If the INMEMO: ersatz account has been created, then log to that account:

```
.LOG INMEMO or .LOG DSKx: [p, pn]
```

where DSKx: [p, pn] is the account you plan to use. (Note that the ersatz definition does not become active until you reboot the system.)

Then restore all of the files into that account. For example, using a tape, the command would be:

```
.MTURES/T = ALL: []
```

If the INMEMO: account (or DSKx: [p, pn] does not exist, then you can log to the OPR: account and create it while you restore the tape:

```
.LOG OPR:
.MTURES/T DSKx: [p, pn] = ALL: []
```

where again, DSKx: [p, pn] should be replaced by the account you wish to use.

Restoring from a PC: If you are restoring the files to AMOS from a PC (having received the software via email, web download, or DOS diskette) you must use a binary-type transfer method (pcVision, ZTERM, AlphaLAN, ZMODEM, AutoLog, etc.) to get the files over to AMOS. However, note that all of the files with extensions ?MO, DAT, and MNU are meant to be contiguous files. Since this concept doesn't exist on the PC, you must use the appropriate file transfer switches available with your PC-to-Alpha transfer package to cause prevent these files from being converted to sequential files. If this is not possible, don't despair: all of the contiguous files are just samples used in some of the sample programs, and are not needed for normal operation.

## Adjusting File Names and Locations

---

After restoring the tape, there are a number of minor changes or relocations you will probably need to make to the files.

### **INMEMO.SBR/XBR**

INMEMO.SBR is released on tape with a numeric extension which represents your serial number. To use the file as a subroutine, you must first copy or rename it to have an SBR extension:

```
.COPY INMEMO.SBR=INMEMO.xxx ('xxx' is your serial number)
```

If you plan to use BasicPlus, you may also want to make a copy with an XBR extension, although there is now a compile switch (/S) which tells BasicPlus to expect SBR extensions on subroutines instead of XBR extensions. (Consult your BasicPlus documentation for more details.)

Once you have renamed INMEMO, you may want to copy it to the BAS: and/or BP: accounts, so it can be loaded dynamically at run time. However, we strongly advise against this for reasons of performance. INMEMO.SBR is over 40 blocks long, which will cause a noticeable delay if fetched from disk every time you need it. It is much more efficient to load it in system or user memory, in which case it can be loaded directly from the INMEMO: account.

## **Compiling the Utilities**

Once the files are installed on your system, you will want to COMPIL (or COMPLP if you are using BasicPlus) all of BASIC programs in the INMEMO: account. For example:

```
.LOG INMEMO:
.COMPIL MAKMMO
.COMPIL FIXMMO  etc.
```

or

```
.COMPLP MAKMMO.BAS/S
.COMPLP FIXMMO.BAS/S
```

In the example above, we used /S so that BasicPlus will expect subroutines to have .SBR extensions instead of .XBR.

Note that if you are using BasicPlus and are not on AMOSxx 2.2 or higher, you will need to change the references to RUN in any of the command (.CMD files in the INMEMO: account to RUNP.

Also note that there is no need to consider moving the \*.RUN or \*.RP files from the INMEMO: account to anywhere else (such as BAS:) because you will almost certainly always execute them via command files, which can specify the INMEMO: location.

## **Moving CMD and LIT Files**

Optionally, you may want to transfer the .LIT files to the SYS: account and the .CMD files to the CMD: account for easier access. However, if you don't want to clutter up your SYS: and CMD: accounts, it is not required. You will have to specify the INMEMO: ersatz device whenever you access any of the INMEMO files though.

For example, to execute the INFREE utility, you would have to type:

```
. INMEMO:INFREE
```

Or to execute the MAKMMO command file, you would have to type:

```
. INMEMO:MAKMMO
```

Actually, this is a small price to pay for keeping your disk well organized, especially since you would probably set up a menu system to access the desired utilities anyway.

## **INMEMO.HLV**

INMEMO is shipped with a VUE help file, INMEMO.HLV. Again, you can access this file from with VUE by entering:

```
>HELP INMEMO:INMEMO
```

although you may find it more convenient to just copy the file to the default VUE help account: HLP:

```
.LOG HLP:
.COPY = INMEMO:INMEMO.HLV
```

## **INFLD/INFLDX**

Many of the INMEMO utilities use INFLD, another popular subroutine produced by MicroSabio. If for some inexplicable reason you do not have a real licensed copy of INFLD, a stripped down version called INFLDX.SBR is supplied with INMEMO. To use it in place of INFLD, just rename it:

```
.LOG BAS:
.COPY INFLD.SBR = INMEMO:INFLDX.SBR
```

If you are using BasicPlus, you may prefer to do it this way, instead or in addition:

```
.LOG BP:
.COPY INFLD.SBR = INMEMO:INFLDX.SBR
```

Make sure to perform the above step ONLY if you don't already have INFLD.SBR since the version supplied on the INMEMO release only works for the INMEMO utilities. Also note that you may have INFLD.SBR on your system called INPUT.SBR - if so, check the version number using DIR/V INPUT.SBR. If the version is 4.0 or higher and the size is more than about 8 blocks, then it is almost certainly INFLD.SBR in disguise and you should make a copy of it called INFLD.SBR and use it instead of the INFLDX routine supplied on the tape.

If you have any doubt about whether your copy of INFLD is up to date or licensed properly, please contact MicroSabio.

## Files Included

Here is a list of the files in the system:

INMEMO.HLV	Help file for VUE (move to HLP:)
INMEMO.xxx	'xxx' is your serial number (INMEMO.SBR)
TSTMMO.BAS	A sample program using INMEMO.SBR
TSTMMO.CMD	Command file to execute the sample program
TSTNFL.BAS	A sample program demonstrating NO-FILE and MENU modes.
INMEMO.DAT	A sample primary data file (used by TSTMMO.BAS)
INMEMO.MMO	A sample memo file (used by INMEMO.BAS)
INMEMO.SYS	Locking module (must load in system memory)
EXPMMO.BAS	Program to expand memo files
EXPMMO.CMD	Command file to load INFLDX and run EXPMMO
MAKMMO.BAS	Program to create and initialize memo files
MAKMMO.CMD	Command file to load INFLDX and run MAKMMO
OPRMMO.BAS	Program to display memo file usage info
OPRMMO.CMD	Command file to load INFLDX and run OPRMMO
ANAMMO.BAS	Program to do a DSKANA-like analysis and cleanup of a primary file-memo file combination
ANAMMO.CMD	Command file to execute ANAMMO
CVTMMO.BAS	Program to convert memo files from 2 byte to 4 byte links
CVTMMO.CMD	Command file to load INFLDX and run CVTMMO
DMPMMO.BAS	Program to dump memo files
DMPMMO.CMD	Command file to execute DMPMMO
HLPMMO.BAS	A sample help system maintenance utility used to edit the help screens used by the other utilities
HLPMMO.CMD	Command file to execute HLPMMO
HLPMMO.MMO	The help memo pad
INFLDX.SBR	Special version of MicroSabio's INFLD.SBR used by the utility programs. If you don't already have INFLD.SBR (shame!), then contact MicroSabio to order it. In the meantime, rename the INFLDX.SBR to INFLD.SBR and copy it to the BAS: account to allow you to run the utility programs. Note that this copy of INFLD will only work with these utilities

JOBNAM.SBR	Routine used within EXPMMO.BAS
RENAM.SBR	Routine used within EXPMMO.BAS
NOEKO.SBR	Exactly like NOECHO.SBR; used by some utilities
LOGRIO.SBR	Routine used by ANAMMO for IsamPlus
XFRMMO.SBR	Utility to copy or move entire memo pads
MMOLOK.SBR	Utility to access the locking system from basic
INFREE.LIT	Utility to show status of INMEMO file locks, and release any locks manually
LOKMMO.LIT	A more sophisticated lock monitoring program useful for monitoring multiple semaphores.
INFBUF.LIT	Used to initialize one form of the scrap buffers
MAKBUF.LIT	Used to create and load the old form of scrap buffer
FUNKEY.M68	A handy subroutine for using the function key translation tables from BASIC
WYSE5X.M68	Sample source for a translation table (for WYSE50)
INMEMO.DIR	Directory listing with hash codes
MMOSYM.BSI	++Include file containing INMEMO symbols
MMOPAR.BSI	++Include file with INMEMO parameter mappings
INFPAR.BSI	++Include file with INFLD parameter mappings
GETCLR.BSI	++Include file to read in color definitions
COLOR.DEF	VUE-able color definition file (used with GETCLR)
CHFUNG.LIT	Handy utility to check function key translation tables
AM62A.IMX	Sample SET PFK style function key translation table.

## Loading the Locking Module in System Memory

You must load at least one semaphore-locking module (either INMEMO.SYS or INMEMO.MLK) into system memory if you are going to allow concurrent editing of the same memo file. If you have many users and multiple memo files, you may want to define additional locking modules to reduce the chance of a significant delay if several users exit from memo editing in different files at the same instant. The modules are only 8 bytes long, so you shouldn't have much difficulty squeezing them in.

To make additional locking modules, simply copy the INMEMO.SYS file provided. The names of the additional modules must equal the names of the memo files with .MLK extensions. For example, to create a unique lock module for the memo file TSTMMO, execute this command:

```
.COPY TSTMMO.MLK = INMEMO.SYS
```

To load it/them into system memory, you must modify your AMOSxx.INI file as shown below. Note that you should always make a backup copy of the INI file and modify the backup copy, then use MONTST to test it before finally updating your live copy of AMOSL.INI.

Use VUE to edit the copy of the AMOSL.INI file, and insert the following line somewhere between the first and last SYSTEM commands:

```
SYSTEM INMEMO:INMEMO.SYS
```

Note that we specified the ersatz account where the lock module is. If you have additional lock modules, use additional SYSTEM commands to load them.

## Scrap Buffers

(AMOS Only) INMEMO allows two methods of setting up scrap buffers. The first method uses a single module for each job named <job>.INF which may be loaded in either system or user memory. The modules should be as long as the widest memo (80 or 132 bytes will usually do). You can create the modules with

VUE (just type in the desired number of spaces and save the file). If you are only going to load them in user memory, you can use the utility MAKBUF to create and load a module on the fly as needed:

```
.MAKBUF
```

This creates a module of 132 bytes.

The second method of creating scrap buffers is probably preferable, particularly if you have a large number of users. It involves inserting a command in the AMOSL.INI file which creates a single module containing an array of buffers which can be used by jobs on a demand basis. Insert the following command somewhere in the middle of the SYSTEM commands in the INI file:

```
SYSTEM INMEMO:INFBUF/N <# jobs>,<size>
```

For example:

```
SYSTEM INMEMO:INFBUF/N 6,60
```

will allocate space for 6 buffers of 60 bytes each. If the size is omitted, 40 bytes will be assumed. If the # jobs is omitted, 10 jobs will be assumed.

These buffers will be assigned to specific jobs on a first come, first served basis. Once assigned, they can only be de-assigned using the INFBUF command from AMOS command mode:

```
.INFBUF
Buffer cleared (clear buffer of current job only)
```

## Loading INMEMO.SBR in System Memory

It is most convenient if you also load INMEMO.SBR in system memory. Just insert the following command somewhere in the SYSTEM commands in the AMOSL.INI file:

```
SYSTEM INMEMO:INMEMO.SBR
```

## Making and Loading Function Translation Tables

INMEMO provides three methods of translating function keys. One uses the same scheme as INFLD, which consists of an M68 file which you customize for your particular terminal. A sample file is provided on the release tape, WYSE5X.M68. If you print the file, you should be able to understand how to modify the program for other terminals and translation sequences. Basically, you just have to change the OBJNAM command near the top so that the assembled module has the same name as the terminal driver it applies to, and an extension of IMX. The other change is to the table at the label XLTTAB:. This method is a bit cumbersome since you have to deal with source code and assembly; its main advantage is that it provides a structure for loading and clearing the function key labeling lines that is somewhat more flexible than what the standard TAB(-1,X) commands allow. The FUNKEY.SBR subroutine is provided (see source file FUNKEY.M68 for documentation) to allow access to the load label line, clear label line, and translate function key sequence from within BASIC.

The second method of translating function keys is based on the standard used by MULTI, AlphaCALC, AlphaWRITE, AlphaBASE, etc. This method requires a utility program which builds a table for you by allowing you to hit the various function keys and then key in the translation. Alpha Micro makes available such a program called FIXTRN.LIT, which you may already have on your system. (It comes on the later 2.x release tapes and is readily available from other sources.) The advantages of this method over the previously described method are that the output of the translation can be several characters in length (not just one), and it is much easier to create the tables.

The third method is very similar to the second method but uses the `SET PFK` system command to build and maintain the tables. Internally, the format of the `SET PFK` and `FIXTRN` tables are different, but for all other practical purposes, they behave the same.

Note that the use of translation tables is optional. All functions are accessible through simple control-key sequences patterned after `VUE`. If you do decide to use function keys, whichever method you use, two rules apply:

The function key translation file must have the same name as your terminal driver, with an extension of `IMX`.

The function key translation file must be in system or user memory when `INMEMO` is called in order to be active. The reason for this is that unlike most programs which use translation tables (`VUE`, `CALC`, `WRITE`, etc.), `INMEMO` is running under `BASIC`'s `xcall` memory management which does not allow for the standard operation of fetching a module from disk into memory. Besides, if your application uses `INMEMO`, it probably calls it a lot, and therefore you wouldn't want to fetch the translation table from disk each time. The best way to handle this is to construct tables for all your terminal drivers and then load them into system memory with the `SYSTEM` command in your system initialization file. Since you are loading them in memory, it doesn't make any difference where they are on disk - we recommend that you leave them in your `INMEMO : ersatz` account.

Note that when `INMEMO` finds an `xxxxxx.IMX` file in memory that matches your terminal driver name, it determines which of the two types of translation tables it is based on the internal format, so there is no need to tell `INMEMO` which type you are using.

Besides the three translation table formats, there are two fundamentally different things that can be accomplished with a single key translation. The first is called an editing translation, and simply replaces the function key with a one or more characters which are processed as if they were typed individually. For example, you could translate `F1` to insert 3 lines and type the firm's name and address. The second method is called a command exitcode translation, and is used to exit from memo editing, update the memo, and return an exitcode to the calling program (as a signal to perform some other processing.) To set up command translations, just define the function key to send `Control-G` followed by a control character whose ASCII value is equal to the absolute value of the exitcode you wish to send. (Exit codes come back as negative values.) For example, to program `F4` to send exitcode 4, translate it to `Control-C` followed by `Control-D` (since `Control-D` is the 4th character in the ASCII collating sequence.)

The comments in the preceding paragraph only apply to function key translation table made with `FIXTRN` or `SET PFK`; the proprietary format translation tables do not allow multiple character translations but automatically allow for command exitcode translations. See the discussion of the `EXTCTL`: Extended options parameter for more information on command exit codes, and the notes on the `CHFUNG` utility for information on checking the format and contents of translation tables.

## Verifying the Installation

You will need to reboot to test the installation. If you followed the recommended procedure of working with a backup of `AMOSxx.INI`, use the `MONTST` command to reboot using the modified version of the `INI` file.

Verify that the system boots properly, and that the `INMEMO.SYS` (and any other lock modules) were loaded. You can use the `SYSTEM` command after system is booted to see the modules that were loaded into system memory. Also check for any scrap buffer modules (`<job>.INF` or `INFBUF.BUF`) and any function key translation tables.

If it didn't work, retrace your steps and try again. If still unsuccessful, contact your dealer or MicroSABIO for assistance. If it works fine, then proceed to copy the new `INI` file back to `AMOSL.INI`.

## **Testing INMEMO**

You will probably want to test the operation of INMEMO before taking time to include it in existing programs. We supply two sample programs for this purpose. The first is an example of a simple file maintenance program which uses a typical memo pad, plus a selection menu memo and memo help screens. To run it:

```
.LOG INMEMO :  
.TSTMMO
```

The program performs file maintenance for a sample file. Several of the product types have already been entered and you can also try adding a few (enter a product type number of 1-20). Try out the print and search options too.

Once you have entered/edited some comment records, run the OPRMMO program to see the file usage statistics. Then try the EXPMMO program. Expand or contract the file and then verify that the comments entered in the TSTMMO program are still there. Also run the OPRMMO program again to verify that the number of free records has increased or decreased by the amount of the expansion or contraction.

The second program is called TSTNFL and contains examples of using INMEMO in no-file mode, particularly for menus, selection lists, and displays. To run it:

```
.LOG INMEMO :  
.TSTNFL
```

Neither of these programs is meant to be a model of how a program should be written, rather they are easy-to-understand examples of how you can use various INMEMO options. By comparing the source code to the behavior of the programs, you should be able to resolve any questions you may have had from reading this manual.

This is the last page