Texx

Version 0.1

By

Jose Aguirre

June 19, 1992

Texx by Jose Aguirre

Table of Contents

Preface

Texx is written and copyrighted by Jose Aguirre. This is a shareware offering, meaning if you use this product, I would like to receive $25 from you. I think you will find that $25 is a very reasonable price for a product such as this. For $25 you will get the latest version, if you don't already have it, notification of future releases, and I'll cut you a deal on upgrade prices, assuming that I am still publishing this product. Since I plan to release significant upgrades, I also plan on increasing the shareware fee. So, for $25, you are getting a pretty good deal. Included with the Texx package will be a registration form that should be filled out and sent in with your shareware fee. One last note, if you do not compensate me for my work, it will be very difficult for me to continue supporting and upgrading this product, I will be forced to move on to something else. Please support your shareware authors.

I grant permission for this product to be distributed freely with the following restrictions:

1. The original compressed file that includes the Texx application, all the documentation, including the shareware registration form, and sample Texx execs must be the only thing distributed. None of the files may be altered. If you have written execs that you think would be useful to others, submit them to me and I will include them on the next release or I will release a collection of them. Any execs submitted to me for release will be considered public domain and the authors should not expect any royalties. Of course, exec authors are free to release execs themselves and charge a shareware fee for them. I will not be responsible for execs released by other authors.
2. This product may not be sold by anybody but myself, or any fees collected by anybody else except for costs incurred for the shareware distribution of this product. Such fees would include connect and download fees, disk duplication fees, and printing fees. Since I am the author and sole owner of this product, I am the only one authorized to profit from its distribution. Online services and bulletin boards are not authorized to charge any additional charges for downloading this product other than what they normally charge for connect and downloading. If the documentation is printed, the only allowable fee will be the cost of materials.
3. The documentation for this product may be printed and distributed, however, the shareware registration form and a diskette containing the original Texx package must be included with the printed documentation. See above for any fees that may be collected for the printing service.

The shareware fee of $25 dollars is for a single copy, i.e. one CPU. If you will be using it on more than one CPU, an additional shareware fee will be required. I will also sell site licenses for users of multiple CPUs. Please note the need for a site license on the registration form.

Disclaimer

I, Jose Aguirre, or any company or organization represented by me makes no warranty or representation, either expressed or implied, with respect to this software, its quality, performance or fitness for a specific purpose. This software is licensed "as is" and the entire risk as to the suitability, quality and performance is left to the buyer. In no event will Jose Aguirre, or any company or organization represented by Jose Aguirre, or any suppliers and distributors be liable for direct, indirect, incidental or consequential damages resulting from any defect in the software, documentation, or any part of the Texx application package. Additionally, the above mentioned parties shall have no liability for any program, data, software, or hardware which is lost or damaged. The above mentioned parties assume no responsibility for the physical, emotional, monetary, or any other damages as a result of the usage of the Texx application package or as result of defect with the Texx package. In other words, if something bad happens to you for using Texx or because of a defect in Texx, that is your responsibility and not my fault. Before you trust "any" software with your business or critical data, you should thoroughly test your environment and prepare disaster recovery plans should the need arise.

The above paragraph should not scare anybody from using Texx. Every function that this release of Texx supports has been tested and so far has been found bug free. However, since there are so many different flavors of Macs and different releases of System 7, not to mention all the different extensions and control panels that Mac users use, some bugs are bound to be found. This manual will later list what steps should be followed and what documentation to provide in order for the bug or error to be fixed. In the event an error is discovered, I may be reached at the following places:

1. Compuserve - 71407,441

2. Jose Aguirre
   3704 Edgar Park Ave.
   El Paso, Texas 79904

Chapter 1

Texx Language Description

Introduction to Texx

The Texx application adds "scripting" ability to the Apple Macintosh computer. The Texx language is based on an IBM scripting language mostly found as a component on IBM's VM operating system. The name has been changed so as to avoid any copyright infringement or any other legal problems. Every effort has been made to conform to the IBM language definition. However, this version, 0.1, is only a subset of the formal language. Also, since the Macintosh environment is like no other, changes were necessary to make it more useful in a Macintosh system. Those changes, up to now, have been minor and should pose very little problems for experienced "exec" authors.

The Texx language can be described as a cross between C and Pascal with the simplicity of HyperTalk. Although that is a very simplistic description, the Texx language has similarities to the three languages. It is not as strongly typed a language as Pascal while at the same time does not have the complex features of C. It is uncomplicated and forgiving like HyperTalk. Despite its simplicity, the Texx language is extremely powerful and will satisfy the experienced programmer, and forgiving enough that the non-programmer will be able to write very useful and productive "execs" without much difficulty.

Texx gives the user the ability to automate repetitive tasks quickly, efficiently and without errors. Texx uses the interapplication communication (IAC) facilities provided by System 7. At the present time, only Apple Events are used. Publish and Subscribe and the Edition Manager facilities are not used at the present. Future upgrades will make full use of the IAC capabilities. Texx will support the four core Apple Events. Future upgrades will provide support for Apple Events as stated in the Apple Event Registry and also provide a mechanism for supporting custom Apple Events. As long as other applications support the standard Apple Events, communication between them and Texx is possible.

System Requirements

Since Texx makes extensive use of IAC, System 7 is required. Texx will not operate with anything less than System 7.0. That also means a minimum of two megabytes of memory just run System 7. A more realistic memory requirement is a minimum of four of five megabytes of memory depending on the applications that are being run. That is because Texx and whatever application that it is communicating with must both be running. Of course, virtual memory would ease the real memory requirements.

Language Description

The Texx application is an interpreter much like HyperTalk. Texx "execs" are interpreted rather than compiled. In Mac circles the programs are called "Scripts", in IBM circles the programs are called "Execs". Both terms will be used interchangeably throughout the documentation, the meaning will be the same, regardless of which one is used. Execs are interpreted line by line and word for word from left to right. The advantage of using an interpreter is that when errors occur, the interpreter stops immediately and the error is usually clearly indicated. When corrected, the exec can be re-run immediately.

A Texx exec must start with a comment. This is an IBM implementation restriction, its purpose is to distinguish it from an older, obsolete interpreter that implements a different language. Following the opening comment, are clauses which are composed of sequences of tokens. Tokens are scanned from left to right and may be separated by zero or more blanks. Blanks are generally ignored, however, sometimes they are used to separate tokens. Clauses are terminated by a semicolon (;), a carriage return implies a semicolon, however, a carriage return does not necessarily terminate a clause. A semicolon does terminate a clause. That means that the semicolon is not necessary to terminate a clause, however, it is good practice to terminate Texx clauses with semicolons.

Tokens are composed of sequences of ASCII characters. Some characters have been changed from the standard IBM EBCDIC character set because an ASCII version does not exist. For those that don't know, IBM uses the EBCDIC character set for their mainframes while the rest of the world uses the ASCII character set. The tokens may be any length up to 254 characters. The individual tokens may be separated by either blanks or other tokens. Tokens are separated into several classes as follows:

Comments:

Any sequence of characters that are delimited by the C-like /* and */. Comments may be on multiple lines, however, this implementation restricts the length to 254 characters. Comments may appear anywhere a blank may appear and are ignored just like blanks. Comments do, however, act as token separators.

```
/* This is a valid comment */
```

Strings:

Any sequence of characters that are delimited by either the single quote (') or the double quote ("). There is no distinction between the two different quotes, however, they may not be mixed. If a string is started with a single quote, it must be terminated with a single quote and vice versa. Two consecutive single quotes ('') in a string delimited by single quotes represent the single quote character in the string. Same rules apply to the use of double quotes. A string is a literal constant and may not be altered. The implementation maximum size is 254 characters.

These are valid strings:

```
'John'
"This is a string"
'I can''t get this' /* same as "I can't get this"*/
```

Hexadecimal Strings:

Not supported in this release.

Symbols:

Sequence of ASCII characters from the alphabetic and numeric characters (A-Z, a-z, 0-9) and/or from the characters @#$!? and underscore. Any lowercase alphabetic characters are converted to uppercase. Symbols must begin with an alphabetic character. The maximum length for this implementation is 254 characters. A symbol may be a label, Texx keyword, or variable. If a variable is used before it has been assigned a value, its default value is the name of the variable itself.

These are valid symbols:

```
Address
Hi!
WHO?
```

Numbers:

Sequence of characters from the numeric characters. This implementation only supports integer numbers. Decimal points (.) and exponential notation are not allowed. Optionally, a number may be preceded by a plus (+) or minus (-) sign. This implementation uses a LONGINT size to represent numbers, that is a range from -2147483648 to +2147483647. A number, including sign, must be consecutive characters without intervening blanks or other characters.

These are valid numbers:

```
12
-17
+100
```

Operators:

These are special characters that are used in expressions. They are generally used in arithmetic and logical expressions. Operators that are represented by multiple characters must appear without intervening spaces or other characters. Each supported operator will be discussed in later sections.

Implied Semicolons and Statement Continuations:

A semicolon (;) marks the end of a clause or statement. A carriage return ends the current line and also implies the presence of a semicolon. For the most part, a carriage return terminates the clause or statement. However, in certain circumstances, if a clause or statement is continued to a second line, Texx may ignore the implied semicolon of a carriage return and continue interpreting on the next line. Continuations are strongly discouraged, all statements should be on only one line and terminated by a semicolon. In this release of this implementation, continuations are not guaranteed to work. If you must continue a statement or clause, the line being continued must end in a comma. The comma is discarded and the following line is treated as if it were part of the previous line.

Expressions and Texx Operators

Expressions:

Texx clauses, also known as statements, consists of terms interspersed with operators and other special characters. Terms may be strings, symbols or numbers as defined above. Evaluation of an expression is left to right, modified by parentheses and the algebraic operator preferences. Expressions are evaluated in their entirety unless an error occurs during their evaluation. Expressions can be thought of as Texx instructions, Texx language instructions or assignment statements.

Operators:

Operators, except for prefix operators, act on two terms yielding a single result. Operators may be arithmetic or logical operators. This implementation defines three types of operators.

Concatenation:

The concatenation operator is used to combine two strings into one. Even though the previous sentence said strings were combined, the concatenation operator applies also to numbers. If numbers are used, they are converted into strings and then combined yielding a string result. Numbers may be used for one or both operands. The operands are combined without any blanks or other characters inbetween them. The concatenation operator is the double vertical bar (||).

Example:

```
"This is string one" || "This is string two" yields
"This is string oneThis is string two"

x = 1;
"This is string " || x yields "This is string 1"
```

Arithmetic:

The normal arithmetic operators common with most programming languages exist with Texx. The valid operators are listed below in order of operator precedence. In cases where like operands appear, they are evaluated in left to right order. Parenthesis may be used to alter the order of the evaluation.

****** Raise a number to a whole number power. This operand has the highest priority of any other operand.

`5 ** 2` yields `25`

***** Multiply numbers by each other.

`5 * 2` yields `10`

/ Divide numbers. Same operator precedence as the Multiply operator.

`10 / 2` yields `5`

% Divide and return the integer (whole number) part of the result. The remainder or decimal part is lost. Same operator precedence as the Multiply operator.

`11 % 2` yields `5`

// Divide and return the remainder or modulo result in this implementation. Same operator precedence as the Multiply operator.

`11 // 2` yields `1`

+ Addition operator. Lower precedence than the operators listed above.

`5 + 2` yields `7`

- Subtraction operator. Same precedence as the Addition operator.

`5 - 2` yields `3`

Additionally, the plus (+) and minus (-) operators may also be used as prefix operators. They are used to set the sign of numeric entries.

Comparative:

The comparative operators return boolean results of TRUE or FALSE. These operators are typically use with IF statements.

= True if the terms are equal.

== True if the terms are exactly equal. In this implementation, the '=' and '==' operators are treated identically.

!=, /=, ><, <> True if the terms are not equal. All four operators have the same definition and are treated identically.

!==, /== True if the terms are exactly not equal. Both operators have the same definition and are treated the same. In this implementation, the exactly not equal operators are treated the same as the not equal operators above.

>	True if the first term is greater than the second term.
<	True if the first term is less than the second term.
>=	True if the first term is greater than or equal to the second term.

!<	True if the first term is not less than the second term.
<=	True if the first term is less than or equal to the second term.
!>	True if the first term is not greater than the second term.

Clauses

Clauses are what a Texx exec is composed of. A clause is the basic interpreted instruction of the Texx language. A clause may be divided into five different types.

Null clauses:

A null clause consists of only blanks and/or comments. A null clause may be a blank line used to improve the exec's readability or a comment describing the exec. In either case, null clauses, are ignored by the interpreter. A null clause is not a statement. It may not be substituted for a null statement, the NOP instruction is provided for that purpose.

Labels:

A label consists of a single symbol followed by a colon without an intervening space. A label must be the first token on a line and does not belong to any statement that may follow it. Labels are used for branching. The branch instruction referencing the label should not include the colon. A label may be used as a statement by itself. It is not an executable instruction, it simple creates a symbol table entry.

Assignments:

Assignments are single clauses of the form **symbol = expression**. An assignment statement causes the symbol to be assigned a value. Assignment statements may be used to assign numeric or string values. If a symbol is assigned a string value, it may later be assigned a numeric value and vice versa. The Texx language is not a "typed" language like Pascal or C. Therefore, variables can take on any value, string or numeric. Obviously, the assignment must not have mixed operands, i.e. string and numeric. The only exception to the rule is the concatenation operand as explained above.

Instructions:

An instruction is a Texx language clause. An instruction may be a single clause or multiple clauses grouped into one by another clause. Such a grouping-like clause would be a **DO** loop. A **DO** loop groups a series of instructions that are treated as one. These grouping-like instructions may also be nested within themselves or others.

Commands:

Commands are single clauses consisting of just an expression. The expression is evaluated and passed on as a command to some external environment.

Return Codes

Certain Texx instructions and commands will return a status code indicating the success of the previous command or instruction. The return code is returned in the pre-declared variable **RC**. **RC** is a numeric variable that may be checked after the instruction or command has been executed. A return code of zero (0) is considered a successful execution of an instruction or command. Anything other than zero is considered to be an error or a warning. The return code should be checked anytime an instruction or command is used that sets it. Please note that the return code is only set by routines that set it. If checked, the numeric value returned will be the last one set. Some return codes may indicate that something needs to be done before the command or instruction will execute without error.

There are some additional return codes that had to be defined due to Apple's implementation of Interapplication Communication. Not that Apple's implementation is bad, it provides additional information that is useful in debugging not only the Texx application, but also Texx execs. Two additional return codes have been implemented for this reason. **AERC** is the Apple Event version of **RC** described above. It contains the return code of the Apple Event functions used to assemble and send an Apple Event. To send an Apple Event to another application, there are several steps, any of which could fail. If the Apple Event was successfully sent, then **AERC** will contain the return code of the target application's handler. A value of zero (0) is considered a successful handling of the Apple Event. Anything else is considered an error or warning.

The second additional return code is **AERCSTRING**. The target application may send a string value that gives a more descriptive error message. Texx will put this string in the pre-declared variable **AERCSTRING**.

Note: Even though these are pre-declared variables, at the present time they may not be assigned values. These variables are always cleared before any routine that may set them.

This page was intentionally left blank.

Chapter 2 Texx Instructions

Texx Commands

ADDEXECMENU

ADDEXECMENU *pathname ;*

Where *pathname* is a single symbol or string denoting a pathname of a TEXX exec. The pathname must be a full pathname to the exec. If used with the **PATH** command, described later, the *pathname* may be a partial pathname.

This command is used to add entries to the EXEC menu. These entries are TEXX execs whose names are appended to the EXEC menu. Once added to the menu, selecting an entry will cause the execution of the exec. Even though a full pathname is provided, only the name of the exec will appear in the EXEC menu. Please note that this command is not part of the formal language description, it was added to ease the execution of execs.

When **ADDEXECMENU** is used, there are verification steps that must be passed before the exec is added to the menu. First, a full pathname must be supplied. A pathname is considered full if it contains a volume name. What is supplied as a volume name is checked against the currently mounted volumes. Next, the pathname is checked for valid characters and valid lengths of folders and file names. Then the existance of the file is checked and finally, the file is checked for a file type of 'TEXT'. If all of the verification tests pass, then the name of the exec is appended to the EXEC menu. If any test fails, a return code is returned in the pre-declared variable **RC**. **RC** should be checked after each **ADDEXECMENU** instruction is executed.

Return Codes:

5005 - Out of memory. The memory needed to execute the command was not available.

5020 - Error pop'ing the expression stack. There was an error retrieving the value of the expression, i.e. pathname.

5021 - Error in expression. An error was detected in evaluating the expression, i.e. pathname.

5025 - Expression stack overflow. The expression stack overflowed during evaluation of an expression. The expression stack is large enough where this error should not occur.

5044 - Invalid path. The pathname supplied was found to be invalid. A number or errors will cause this error. A partial pathname, invalid characters, invalid lengths of folders and volumes are the most common errors. If this error is reported, carefully check the pathname for errors. If the **PATH** instruction is used, it may be in error.

5050 - Invalid path name. This error is identical to **5044**. Please refer to that error.

5052 - FSSpec error. An FSSpec is generated during the processing of the command. If this error is returned, it probably means that the exec does not exist.

5053 - Alias error. This error is similar to **5052**. The same thing applies to this one.

Examples:

```
/* This will add the Resedit.exec to the menu */
PATH 'External HD 105:MPW:Applications';
ADDEXECMENU Resedit.exec;

/* This will add HyperCard.exec to the menu */
HyperCardPath = 'External HD 105:HyperCard Folder';
PATH HyperCardPath;
ADDEXECMENU HyperCard.exec;

/* This will add an exec to the menu and check */
/* the return codes for completion */
PATH 'External HD 105:MPW';
ADDEXECMENU MPW.exec;
IF RC <> 0 THEN
DO
     SAY 'ADDEXECMENU - MPW.exec failed.';
     SAY 'RC = ' || rc;
END;
```

ADDRESS

ADDRESS [*environment*]

Where *environment* is a single symbol or string, which is treated as a constant.

This instruction is used to effect a change to the destination of command(s). The command destination is in effect until another **ADDRESS** instruction is executed. An **ADDRESS** instruction must be executed before any commands can be sent to any environment or application.

An **ADDRESS** instruction with an *environment* will target that environment or application. Texx will verify that the name is valid and that the application is active. If either of the two tests fail, a return code will be returned in the variable **RC** and the **ADDRESS** instruction will have no affect on the previous valid **ADDRESS** instruction.

An **ADDRESS** without an *environment* operand will simply swap the current setting with the previous setting. It may be used to toggle between two different environments or applications. If executed and no previous **ADDRESS** was executed, an error code will be returned in **RC** and no swapping will occur.

Note The use of this command is a little different from the formal language definition. The TEXX application will verify the *environment* by checking for a valid name, i.e. valid length and characters. If a valid name is supplied, then a check is made to see if the targeted application is active. The target application must be active in order to send it any commands. If a non-zero return code is returned, then TEXX is not able to send any commands to the target. The **ADDRESS** command may also be used to check for an active application.

Return Codes:

5017 - Invalid process name. The application or environment name used was invalid.

5018 - Process not active. The application or environment is not currently active. Since it is not active, it is not possible to send it any commands.

5034 - No previous **ADDRESS** executed. A swap of command destinations could not be performed because a second value not present.

Examples:

```
ADDRESS 'APP1' /* Will route commands to APP1 */

App = 'APP1'
ADDRESS App1   /* Will route commands to APP1 */

ADDRESS 'APP1' /* Will route commands to APP1 */
ADDRESS 'APP2' /* Will route commands to APP2 */
ADDRESS        /* Will toggle to APP1        */
ADDRESS        /* Will toggle to APP2        */

/* This will route external commands to the   */
/* Finder.                                    */
ADDRESS Finder;
```

DO

```
DO    [ repetitor ] ;
      [ instruction(s) ]
END ;
```

Where:

repetitor is one of the following:

name = expri [**TO** *exprt*] [**BY** *exprb*] [**FOR** *exprf*]
FOREVER
exprr

DO is used to group statements together and optionally repeat them a specified number of times. For repetitive execution, a control variable may be stepped through a range of values by a specified step value. **DO** loops are very powerful and very flexible. The following syntax rules apply to **DO** loops:

1) *exprr, expri, exprb, exprt* and *exprf*, if any are present, may be any expression that evaluates to a number.

2) *exprr* and *exprf* must evaluate to positive integer (whole number) numbers.

3) *exprb* defaults to 1, if relevant.

4) If **TO, BY,** and **FOR** are used, they may be in any order.

DO Loop Components

A **DO** loop or **DO** group may contain some or none of the above listed components. A **DO** without any of the above components, is called a **simple DO group**. It is called a **group** instead of a **loop** because the instruction does not repeat. It is used strictly for grouping instructions. These grouped instructions are treated as one. The **repetitor** is considered to be 1.

Simple repetitive loops contain the **FOREVER** keyword or an expression, *exprr*. **FOREVER** repeats the loop indefinitely, i.e. forever. The only way to exit the loop is to use the Texx instruction **LEAVE**. Care must be exercised to make sure that the **LEAVE** is executed to exit the loop. The other form of a simple repetitive loop is with the use of an expression, *exprr*. The expression, *exprr,* must evaluate to a positive integer number. *Exprr* is decremented by 1 each time through the loop until it reaches the value of zero, at which time the loop is exited.

Controlled repetitive loops use the other **DO** loop constructs, **FOR, BY,** and **TO**. The controlled form uses a **control variable** to control the execution of the loop. When *name = epri* is present, *name* is the **control variable**. The value of the **control variable** determines whether the loop will be executed or exited. The **control variable** may be altered within the loop or by the other loop constructs.

The limits of the **control variable** are set by the **FOR, BY,** and **TO** loop constructs. The **TO** expression, *exprt,* sets the upper bounds of the loop. The loop will execute until the **control variable** reaches the *exprt* value. The **control variable** is further bounded by the **FOR** construct. The **FOR** expression, *exprf,* must evaluate to a positive number. The *exprf* value acts as a loop counter, it is decremented each through the loop. If the *exprf* value reaches zero before the other limits of the loop are reached, the loop will be terminated. In other words, whichever loop limit is reached first will terminate the loop. The **BY** expression, *exprb,* sets the increment value of the loop limits, i.e. the increment value of the *expri* expression. The default value is one (1).

The loop limits are tested at the beginning of each loop. Therefore, it is possible that the loop will never get executed. The **TO, BY,** and **FOR** expressions are evaluated only once, when the **DO** instruction is first executed and before the **control variable** is assigned its initial value.

Return Codes:

none.

Examples:

```
/* The instructions between DO and END will  */
/* be executed if a = 3.                     */
IF a = 3 THEN DO
               a = a + 1;
               say "A = " || a;
              END;

/* This displays "Hello" three times         */
DO 3
  say 'Hello';
END;

/* This loop will be executed forever        */
DO FOREVER
  say 'This is executing forever';
END;

/* This will display 1, 0, -1                */
DO I = 1 TO -1 BY -1
  say I;
END;

/* This will display 1, 2, 3                 */
DO I = 1 TO 10 BY 1 FOR 3
  say I;
```

```
                END;
```

DROP

DROP *name* [*name*] [*name*] ...;

Where:

Name is a variable separated from others by one or more blanks.

DROP is used to unassign values to variables. Variables will be set to their original unassigned value. Each variable specified will be dropped from left to right. It is not an error to list variables more than once.

NOTE: One or more blanks separate the variable names if more than one is listed. Most programming languages would use commas (,). However, the Texx language uses spaces instead of commas. This applies to the entire Texx language.

Return Codes:

None.

Example:

```
/* This will drop the value of a and b  */
a = 1;
b = 'This is b';
DROP a b;
```

GOTO

GOTO *labelname* ;

Where:

labelname is a constant symbol that names a label defined elsewhere in the exec.

The **GOTO** causes a change in the flow of control. Control passes to the instruction immediately following the label referenced. The label must match the *labelname* less the colon that follows a label definition. Case is not important since Texx converts everything to uppercase. In this implementation of Texx, **GOTO** and **SIGNAL** are identical. Please note that the label does not cause any special processing for the statement that follows. The label is there only to be used by the **GOTO** and **SIGNAL** statements.

Return Codes:

none.

Example:

```
/* This will jump to label "TEXXLABEL" */
GOTO TexxLabel;
     Other Instructions;
TEXXLABEL: Say 'Hi';

/* This will jump backwards in the exec */
TEXXLABEL: Say 'Hi';
     Other Instructions;
GOTO TexxLabel;
```

EXIT

EXIT;

The **EXIT** is used to unconditionally leave a Texx exec. The exec is terminated immediately and control is returned to the Texx application for further use.

Return Codes:

none.

Example:

```
/* This will terminate before any message is printed */
x = 'this is a message';
EXIT;
SAY x;
```

IF

IF *expression* [;] **THEN** [;] *instruction*
[**ELSE** [;] *instruction*]

The **IF** statement is used to conditionally execute an instruction or group of instructions depending on the evaluation of the *expression.*

The instruction after the **THEN** is executed only if the result of the *expression* is 1. If an **ELSE** is present, the instruction after it is executed only if the result of the *expression* is 0. An instruction could be multiple instructions as grouped by a **DO** statement. Instructions are treated as one, even if multiple instructions are grouped together. If an **ELSE** is present, it binds to the nearest **IF** at the same level.

Return Codes:

none.

Examples:

```
/* This will display "X is greater than 1"   */
x = 2;
IF x > 1 THEN SAY 'X is greater than 1';

/* This will display 'X is equal to 2'  */
x = 2;
IF x <> 2 THEN SAY 'X is not equal to 2';
          ELSE SAY 'X is equal to 2';

/* This will do nothing  */
x = 2;
IF x <> 2 THEN
               DO 5
                    x = x + 1;
                    SAY x;
               END;
```

LEAVE

LEAVE ;

The **LEAVE** causes immediate exit and termination of **DO** loops and **DO** groups. Execution of the grouped instructions is terminated and control is passed to the instruction immediately following the **END** statement. On exit, the **control variable** (if any), will retain the value it had when the **LEAVE** statement was encountered. The **LEAVE** statement will terminate the innermost loop or group.

The **LEAVE** is only useful inside a **DO** loop or group. An error is not generated if called outside of a loop or group. In this case, the instruction has no effect.

Return Codes:

none.

Example:

```
/* This will execute the loop only twice     */
DO i = 1 TO 5
     IF i > 2 THEN LEAVE;
END;
```

NOP

NOP ;

The **NOP** statement is a dummy instruction that has no effect. It can be useful as a target of an **THEN** or **ELSE** clause.

Return Codes:

none.

Example:

```
/* This IF statement has no THEN clause, use NOP  */
IF x = 2 THEN NOP;
         ELSE x = x + 1;
```

PATH

PATH [*pathname*];

Where *pathname* is a single symbol or string, which is treated as a constant.

This instruction is used to set a path for filename specifications. Using a **PATH** prior to specifying a filename will eliminate the need for the filename to contain the full path name. Instead, only the filename will be necessary. The **PATH** specification will remain in effect until another **PATH** instruction is executed.

The *pathname* supplied is checked for several things. First, it is checked for valid file name characters. Next, the first section of the *pathname*, before the first ':', is checked for a length of 28. Thereafter, the sections are checked for a length of 31. If either of these tests fail, the **PATH** instruction is invalid. An error message will be posted and the exec will terminate.

If the **PATH** instruction is supplied without a *pathname*, the current setting will be swapped with the previous setting. It may be used to toggle between two different *pathnames*. If not previous **PATH** had been executed, the instruction is considered invalid. An error message is posted and the exec is terminated.

Return Codes:

none.

Examples:

```
/* This will set the path to the Preferences folder */
PATH 'External HD:System Folder:Preferences';

/* This will do the same as above */
PathName = 'External HD:System Folder:Preferences';
PATH PathName;

/* If the current pathname is 'External HD' */
/* And the previous pathname was 'Internal HD' */
/* The following will set the current pathname to */
/* 'Internal HD'    */

PATH;     /* this will toggle to previous setting */
```

SAY

SAY [*expression*] ;

SAY is used to display the result of the *expression* to the user. The result of the *expression* may be any length up to 254 characters or any type. The output of this command is displayed in the Texx System Window. It will be necessary to select the Texx System Window from the Window Menu to bring the system window to the front.

Return Codes:

none.

Examples:

```
/* This will display the number 10 */
SAY 10;

/* This will display 'This is a string' */
SAY 'This is a string';

/* This will display 'This is a string' */
String = 'This is a string';
SAY String;

/* This will display the value of 5 * 2 */
SAY 5 * 2;
```

SIGNAL

SIGNAL *labelname* ;

Where:

labelname is a constant symbol that names a label defined elsewhere in the exec.

The **SIGNAL** causes a change in the flow of control. Control passes to the instruction immediately following the label referenced. The label must match the *labelname* less the colon that follows a label definition. Case is not important since Texx converts everything to uppercase. In this implementation of Texx, **GOTO** and **SIGNAL** are identical.

Return Codes:

none.

Example:

```
/* This will jump to label "TEXXLABEL" */
SIGNAL TexxLabel;
     Other Instructions;
TEXXLABEL: Say 'Hi';

/* This will branch backwards          */
TEXXLABEL: Say 'Hi';
     Other Instructions;
SIGNAL TexxLabel;
```

This page intentionally left blank

Chapter 3

External Commands

Introduction to External Commands

External commands are what makes Texx so appealing and powerful. The ability to pass a command to another application to carry out a task is the main function of Texx. This ability allows you to automate repetitive tasks quickly, efficiently and without errors. It also allows you to define the desktop or application environment anyway you want. Texx is also the perfect compliment to other applications that take advantage of the IAC capabilities of System 7.

External commands are English-like commands that are translated into Apple Events as defined in the Apple Event Registry, version 1.0. The Apple Event Registry is divided into several "suites". These "suites" include the four required Apple Events as defined in Inside Macintosh VI, Finder only events, Core events, Text events, Quickdraw events, Table events, and Miscellaneous events. This version of Texx, 0.1, only supports the four required events and the Finder events. Future versions will support the other events. Under consideration is the ability to support other custom Apple Events defined by users and developers that are not defined by the Apple Event Registry. For more information on Apple Events consult Inside Macintosh VI and the Apple Event Registry, both are available from APDA.

As stated previously, Texx makes extensive use of Interapplication communication (IAC). This capability is only available with System 7. Also, the bits in the SIZE resource that indicate that the application supports high-level events must be set. For more information, consult Inside Macintosh VI. Applications that support high-level events should not only state that in the documentation, but also list the high-level events supported.

Description of External Commands

Texx external commands are one or two word commands followed by zero or more parameters. Some external commands have no parameters, while others have up to three different parameters. These parameters may consist of file names, path names, vertical and horizontal positions, window types, and other miscellaneous parameters. The parameters are separated from the command and from each other by one or more spaces. Previously assigned variables or quoted strings will be accepted as parameters. Texx will interpret the supplied parameters and assemble the Apple Event and send it to the desired application. Upon completion of the Apple Event, control returns to the exec for further instructions.

The external commands as accepted by Texx are simple and very straightforward. However, the implementation is far from simple. Depending on the command and the number of parameters required, there may be a significant number of steps required to implement the command. An error could occur at any of those steps. Return codes will provide information on whether the command was successful or not. There are three different possible return codes. One is returned by the Texx routines that are called to interpret and assemble the Apple Event. This return code indicates the reason that a step failed. These errors are documented in the Appendix. These return codes are meant to be short but descriptive and also contain errors discovered by the parsing routines. These return codes are returned in the pre-declared variable **RC**. **RC** is discussed in chapter 1. The second type of return code is generated by the Apple Event routines called to assemble the Apple Event. These routines and their result codes are documented in the Apple Event chapter of Inside Macintosh VI. The **AERC** pre-declared variable will have the return codes from this second type of Apple Event errors. **AERC** will also contain the return code from the application that received the Apple Event from the Texx exec. The third return code is one returned from the target application. It is a descriptive string message that may or may not be returned by the target application. That message, if returned can be accessed by the pre-declared variable **AERCSTRING**.

When an external command is executed, the return codes mentioned above should be checked. The formal description of the language does not include the **AERC** or **AERCSTRING** return code. It was implemented in Texx due to the way IAC is implemented by Apple. Because of the IAC implementation, **RC** and **AERC** codes should be checked to ensure that the command was successfully executed. **AERCSTRING** is optional and is used to give a written error message as opposed to an error number. The three error codes were implemented in Texx because of the complexity of IAC and because the additional information will aid in debugging. **RC** should be checked first. An error may occur before any of the Apple Event routines are called. These errors will be returned in **RC**. If **RC** contains a non-zero value, the external command failed. It is possible that **RC** returns an error but **AERC** does not. That is because the parsing routines may have recognized an error before any Apple Event routines were called. It is also possible that **RC** may be zero and **AERC** may be non-zero. If **RC** returns an error, the Apple Event failed and **AERC** may or may not contain additional information. If **RC** returns a zero, then **AERC** needs to be checked, if it contains a zero, then the external command was successfully executed, if not, check the error in Inside Macintosh VI or in the target application's documentation. **AERCSTRING** may be checked, but since it is optional, it is possible that even though an error occurred, it may still be blank. **RC** and **AERC** are reliable and should always be checked after an external command is executed.

NOTE: Apple Events belonging to the Finder suite cannot be used for other applications. Also, events used for other applications should not be sent to the Finder. The Apple Events Registry states that the Finder suite was developed before the Apple Event Manager was, obviously the Apple Event Manager was changed during its development.

About Finder Events

Apple feels that the Finder is not meant to be scriptable, in fact, Apple does not support the Finder events listed later in this manual. The Finder doesn't even use the Apple Event Manager routines to process Apple Events. I totally disagree, the Finder is the ideal scriptable application. It is my feeling that any operation that can be done with a keyboard and mouse should be available to a scriptable application, even responding to dialog boxes. While the Finder events that are currently supported useful, there are many more Finder operations that should be supported. Events giving information such as open finder windows, window locations, and icon locations are just some of the Finder events that would greatly enhance the Finder's useability. Apple intends to change the Finder events to conform to the Core events, hopefully, they will.

Because of the lack of support, the Finder presents some possible problems for the Texx application. If a Finder event is passed to the Finder that is incorrect, i.e. a file or folder does not exist, the Finder does not return an error or acknowledgement. Therefore, Texx is not aware that the command did not work. Texx will just sit there waiting until it times out. If Texx is brought to the foreground, the EXEC menu is highlighted and Texx looks like it died. Be patient, Texx is waiting for a response from the Finder or a timeout, whichever comes first. **RC** and **AERC** will reflect that a timeout occurred.

External Commands

There are several parameters that are used by many of the external commands. They are listed and described below.

Pathname :

The pathname is either a file name, a folder name or a window name. The name specified must be a full pathname, i.e. the entire hierarchial path must be specified. Partial pathnames may only be specified if a **PATH** instruction had been previously executed. Window specifications are also considered to be pathnames because an open Finder window is simply an open folder.

Pathname List :

Some external commands take a list of pathnames as a parameter. The pathnames are the same as above, the only difference is that a list of names is required. The list may consist of one or more pathnames.

Vertical Position, Horizontal Position :

These are two integer values that specify the top and left offsets of the content region for the placement of objects such as icons. These offsets apply to the destination location.

Window Type :

There are three types of window types. The window type is normally specified with a pathname. The window types and keywords are as follows:

MAINWINDOW - This is a normal Finder window. An example would be an open folder window.

INFOWINDOW - This a window from a Get Info selection from the File Menu of the Finder.

SHARINGWINDOW - This is a window from a Sharing selection from the File Menu of the Finder.

View Type :

There are nine view types that may be used. These are the same as could be selected from the Finder's View menu. The keywords are self explanatory and are as follows :

BYCOMMENT
BYDATE
BYICON
BYKIND
BYLABEL
BYNAME

BYSIZE
BYSMALLICON
BY VERSION

Height, Width :

These are two integer values that represent the size of a window. These values are in pixels. These values are used to size a window.

FINDER SUITE

ABOUT

ABOUT ;

Event Suite : Finder

This external command asks the Finder to display the About This Macintosh window. This command has the same effect as selecting About This Macintosh from the Apple Menu of the Finder. It is up to the user or another external command to close this window. Although this command is included with the Finder events, it is a core event and may be sent to other applications that support it.

Return Codes:

RC -	See the Appendix in this manual.
AERC-	See Inside Macintosh Volume VI.

Example:

```
/* This will display the About This Macintosh */
Address 'Finder';
About;

/* This will instruct TEXX to display its About */
Address Texx;
About;
```

CLOSE WINDOW

CLOSE WINDOW *pathname windowtype ;*

Event Suite : Finder

This command asks the Finder to close one of its windows. This command accepts only a single pathname and the window type must be supplied.

Return Codes:

RC -	See the Appendix in this manual.
AERC-	See Inside Macintosh Volume VI.

Examples:

```
/* This will close the Texx Folder */
PATH 'External HD:';
ADDRESS 'Finder';
CLOSE WINDOW 'Texx Folder' MAINWINDOW;

/* This will close the disk window */
ADDRESS 'Finder';
CLOSE WINDOW 'External HD' MAINWINDOW;

/* This will close the HyperCard Folder */
ADDRESS Finder;
HyperCardPath = 'External HD:Applications Folder';
HyperCardFolder = 'HyperCard Folder';
PATH HyperCardPath;
CLOSE WINDOW HyperCardFolder MAINWINDOW;

/* this will close an info window opened by a */
/* get info on a file */
Address finder;
Path 'Internal HD:Test Folder';
CLOSE WINDOW 'Sample File' INFOWINDOW;
```

DRAG SELECTION

DRAG SELECTION *pathname list vertical position horizontal position destination pathname ;*

Event Suite : Finder

This external command asks the Finder to move copies of one or more icons in the same folder to a new folder. Please note that the icons selected must be in the same folder and that the destination folder must be the same for all selected icons. Also note that the icons are copied, the originals stay intact in their original place. There is a move command, described later, to move the originals without coping them. This is the same as selecting icons and dragging them to a destination with the mouse.

Return Codes:

RC -	See the Appendix in this manual.
AERC-	See Inside Macintosh Volume VI.

Examples:

```
/* This example will move the file 'Test File' to */
/* vertical location 100, horizontal location 100 */
/* in folder 'Test Folder' */
ADDRESS finder;
PATH 'External HD:Utilities Folder'; /* loc of file */
DRAG SELECTION 'Test File' 100 100 'Test Folder';

/* This will move 3 files to 'Test Folder' */
ADDRESS 'Finder';
PATH 'External HD:Utilities Folder';
DRAG SELECTION file1 file2 file3 100 100 'Test Folder';
```

DUPLICATE SELECTION

DUPLICATE SELECTION *pathname list ;*

Event Suite : Finder

This external command instructs the Finder to duplicate one or more icons in the same folder. This is the same as selecting one or more icons and selecting DUPLICATE from the Finder's File menu.

Return Codes:

RC -	See the Appendix in this manual.
AERC-	See Inside Macintosh Volume VI.

Examples:

```
/* This will duplicate the file 'test file' */
ADDRESS finder;
PATH 'Internal HD:Test Folder';
DUPLICATE SELECTION 'test file';

/* This will duplicate 3 files */
ADDRESS finder;
PATH 'Internal HD:Test Folder';
DUPLICATE SELECTION file1 file2 file3;
```

EMPTY TRASH

EMPTY TRASH ;

Event Suite : Finder

The **EMPTY TRASH** external command asks the Finder to empty the trash can. This command has the same effect as selecting Empty Trash from the Finder's Special Menu and selecting OK in the dialog box that appears. Use of this command bypasses the verification dialog box that may appear.

Return Codes:

RC -	See the Appendix in this manual.
AERC-	See Inside Macintosh Volume VI.

Example:

```
/* This will empty the trash */
ADDRESS 'Finder';
EMPTY TRASH;
```

GET INFO

GET INFO *pathname list ;*

Event Suite : Finder

This command instructs the Finder to display Info windows for one or more icons in the same folder. This command creates the INFOWINDOW window type used by other commands.

Return Codes:

RC -	See the Appendix in this manual.
AERC-	See Inside Macintosh Volume VI.

Examples:

```
/* This will display the get info window for the file */
/* 'Test file' */
ADDRESS finder;
PATH 'External HD:HyperCard Folder';
GET INFO 'Test file';

/* This will display the get info windows for 3 files */
ADDRESS finder;
PATH 'External HD:HyperCard Folder';
GET INFO 'file 1' file2 'file 3';
```

MAKE ALIAS

MAKE ALIAS *pathname list ;*

Event Suite : Finder

This command instructs the Finder to create aliases for one or more icons in the same folder. This command has the same effect as selecting one or more icons in the same folder and choosing Make Alias from the Finder's File Menu.

Return Codes:

RC -	See the Appendix in this manual.
AERC-	See Inside Macintosh Volume VI.

Examples:

```
/* This example will make an alias for the file */
/* 'Archive file' */
ADDRESS finder;
MAKE ALIAS 'Internal HD 40:Archive file';

/* This example will make alias for 3 files */
ADDRESS finder;
PATH 'Internal HD 40';
MAKE ALIAS file1 'file 2' file3;
```

MOVE SELECTION

MOVE SELECTION *pathname list vertical position horizontal position destination pathname ;*

Event Suite : Finder

This external command asks the Finder to move one or more icons to a new folder. This is the same as selecting one or more icons and dragging them to a new folder. Please note that if the icons are moved to a folder on another disk drive, the originals stay intact and copies are placed in the new folder.

Return Codes:

RC -	See the Appendix in this manual.
AERC-	See Inside Macintosh Volume VI.

Examples:

```
/* This example will move a file to another folder */
/* at the specified locations */
ADDRESS finder;
PATH 'External HD:HyperCard Folder';
MOVE SELECTION 'Texx Stack' 10 10
     'External HD:Texx Folder';

/* This example will move 3 files to another folder */
/* at the specified locations */
ADDRESS finder;
PATH 'External HD:HyperCard Folder';
MOVE SELECTION 'stack 1' stack2 'stack3' 10 10
     'Internal HD:Misc Folder';

/* This is an example of throwing a file in the trash */
Address finder;
Path 'External HD';
MOVE SELECTION 'Appl Folder:Temporary File' 0 0
               Trash;
```

MOVE WINDOW

MOVE WINDOW *pathname window type vertical position horizontal position*

Event Suite : Finder

This command will ask the Finder to move a window to the specified location. This command is the same as moving the mouse to a window title bar, pressing the mouse and dragging it to a new position.

Return Codes:

RC -	See the Appendix in this manual.
AERC-	See Inside Macintosh Volume VI.

Examples:

```
/* This will move the window to vertical      */
/* position 40 and horizontal position 10 */
Address finder;
MOVE WINDOW 'External HD' MAINWINDOW 40 10;

/* This will move the window to a new location */
Address finder;
VerticalPosition = 50;
HorizontalPosition = 20;
MOVE WINDOW 'External HD' MAINWINDOW
            VerticalPosition HorizontalPosition;
```

OPEN SELECTION

OPEN SELECTION *pathname list ;*

Event Suite : Finder

This command will ask the Finder to open one or more icons in the same folder. This command is used to launch other applications. Please note that the Finder will launch the application and not Texx. Also note that this command will work with applications that do not support Apple Events. The reason being is that the Finder checks the application's SIZE resource and if it supports Apple Events, the Finder will use Apple Events. If the application does not support Apple Events, the Finder will use the old pre-System 7 method of launching and passing files to open.

Return Codes:

RC -	See the Appendix in this manual.
AERC-	See Inside Macintosh Volume VI.

Examples:

```
/* This example will startup HyperCard */
ADDRESS finder;
PATH 'External HD:HyperCard Folder';
OPEN SELECTION 'Home Stack';

/* This example will startup a word processor */
/* and open 3 files */
ADDRESS finder;
PATH 'Internal HD:Word Processing Folder';
OPEN SELECTION 'Doc 1' 'Doc 2' Doc3;
```

PAGE SETUP

PAGE SETUP *pathname window type;*

Event Suite : Finder

This command asks the Finder to display the Page Setup window for a specified Finder window. It is up to the user to change page setup options and to close the Page Setup window. There are no Apple Events that allow the change of page setup options or to close the Page Setup window.

Return Codes:

RC -	See the Appendix in this manual.
AERC-	See Inside Macintosh Volume VI.

Examples:

```
/* This will display the page setup for the disk */
Address finder;
PAGE SETUP 'External HD' MAINWINDOW;

/* This will display the page setup for a folder */
Address finder;
Path 'External HD:Applications Folder';
PAGE SETUP 'SpreadSheet Folder' MAINWINDOW;
```

PRINT SELECTION

PRINT SELECTION *pathname list ;*

Event Suite : Finder

This command asks the Finder to print the specified documents. The Finder does not display any dialog boxes to handle this command. However, the application that prints the document may display dialog boxes. It is up to the user to respond to any dialog boxes displayed. If the application involved in printing a document is already active, it simply prints the document and remains active. If the application is not active, the Finder will launch it. After printing the document, the Finder will send a Quit Application event to the application.

Return Codes:

RC -	See the Appendix in this manual.
AERC-	See Inside Macintosh Volume VI.

Examples:

```
/* This example will instruct the word processor */
/* to print the file 'Document 1' */
ADDRESS finder;
PATH 'External HD:Word Processing Folder';
PRINT SELECTION 'Document 1';

/* This example will instruct an application to */
/* print 3 files */
ADDRESS finder;
PATH 'External HD:Misc documents folder';
PRINT SELECTION 'Document 1' document2 document3;
```

PRINT WINDOW

PRINT WINDOW *pathname window type ;*

Event Suite : Finder

This command asks the Finder to print the contents of one of its windows. When executed, the Finder displays a Print dialog box. It is up to the user to respond and close this dialog box. There are no Apple Events that allow response to these dialog boxes.

Return Codes:

RC -	See the Appendix in this manual.
AERC-	See Inside Macintosh Volume VI.

Examples:

```
/* This will print the disk window */
Address finder;
PRINT WINDOW 'External HD' MAINWINDOW;

/* This will print a folder window */
Address finder;
Path 'External HD';
PRINT WINDOW 'Applications Folder' MAINWINDOW;
```

PUT AWAY

PUT AWAY *pathname list ;*

Event Suite : Finder

This external command asks the Finder to put one or more icons on the desktop or in the Trash back into the folders from which they were last moved. This is the same as selecting one or more icons, either on the desktop or in the Trash, and choosing Put Away from the File Menu of the Finder.

Return Codes:

RC -	See the Appendix in this manual.
AERC-	See Inside Macintosh Volume VI.

Examples:

```
/* This example will return the file 'test file' */
/* back to its previous folder */
ADDRESS finder;
PATH 'External HD:Misc Documents Folder';
PUT AWAY 'test file';

/* This example will retrieve the file 'test file' */
/* from the trash and return it to its previous */
/* location */
ADDRESS finder;
PATH 'Internal HD:Trash';
PUT AWAY 'test file';
```

RESIZE WINDOW

RESIZE WINDOW *pathname window type height width ;*

Event Suite : Finder

This command asks the Finder to set the size of a Finder window. This is the same as dragging the window's size box (lower right corner of the window). The upper right corner of the window does not move.

Return Codes:

RC -	See the Appendix in this manual.
AERC-	See Inside Macintosh Volume VI.

Examples:

```
/* This example will resize a window to a height */
/* 200 pixels and a width of 300 pixels */
Address Finder;
Path 'External HD';
RESIZE WINDOW 'Appl Folder' MAINWINDOW 200 300;

/* This example will resize the specified window */
Address finder;
Height = 500;
Width = 400;
Path 'External HD';
RESIZE WINDOW 'Word Processing Folder' MAINWINDOW
               Height Width;
```

RESTART

RESTART ;

Event Suite : Finder

This command asks the Finder to terminate all open applications and restart the computer. This has the same effect as selecting Restart form the Finder's Special Menu. Please note that terminating other applications may cause them to display dialog boxes that require user intervention. The user is responsible for responding to and closing all dialog boxes that are displayed.

Return Codes:

RC -	See the Appendix in this manual.
AERC-	See Inside Macintosh Volume VI.

Example:

```
/* This example will restart the macintosh */
ADDRESS FINDER;
RESTART;
```

REVEAL

REVEAL *pathname list ;*

Event Suite : Finder

This command asks the Finder to display the window for the folder that contains the specified icons and select those icons.

Return Codes:

RC -	See the Appendix in this manual.
AERC-	See Inside Macintosh Volume VI.

Examples:

```
/* This example will display the folder containing */
/* the application HyperCard */
ADDRESS finder;
PATH 'External HD:HyperCard Folder';
REVEAL HyperCard;

/* This example will display the folder containing */
/* the file 'test file' */
ADDRESS finder;
PATH 'External HD:Misc Documents Folder';
REVEAL 'test file';
```

SET VIEW

SET VIEW *pathname window type view type ;*

Event Suite : Finder

This command specifies what view of a folder window's contents the Finder should display. These views are what may be selected form the Finder's View Menu.

Return Codes:

RC -	See the Appendix in this manual.
AERC-	See Inside Macintosh Volume VI.

Examples:

```
/* This example will set the view of the given */
/* window to BY NAME */
Address Finder;
Path 'External HD';
SET VIEW 'Sample Folder' MAINWINDOW BYNAME;

/* This example will set the view of the given */
/* window to BY SMALL ICON */
Address finder;
SET VIEW 'Externah HD' MAINWINDOW BYSMALLICON;
```

SHARING

SHARING *pathname list ;*

Event Suite : Finder

This command asks the Finder to display Sharing windows for one or more icons in the same folder. This has the same effect as selecting one or more icons and choosing Sharing from the File menu of the Finder. Please note the it is up to the user to change sharing settings. The window may be closed with the **CLOSE WINDOW** external command.

Return Codes:

RC -	See the Appendix in this manual.
AERC-	See Inside Macintosh Volume VI.

Examples:

```
/* This example will display the sharing window */
/* for the folder 'HyperCard Stacks folder' */
ADDRESS finder;
PATH 'External HD:HyperCard Folder';
SHARING 'HyperCard Stacks folder';

/* This example will display the sharing windows */
/* the given folder names */
Address finder;
Path 'External HD:Companies Folder';
SHARING 'Company 1 Folder' 'Company 2 Folder'
          'Company 3 Folder';
```

SHOW CLIPBOARD

SHOW CLIPBOARD ;

Event Suite : Finder

This external command instructs the Finder to display the Clipboard window. It is up to the user to close the Clipboard window.

Return Codes:

RC -	See the Appendix in this manual.
AERC-	See Inside Macintosh Volume VI.

Example:

```
/* This example will instruct the finder to open */
/* the clipboard */
ADDRESS finder;
SHOW CLIPBOARD;
```

SHUTDOWN

SHUTDOWN ;

Event Suite : Finder

This command instructs the Finder to terminate all open applications in preparation for turning off the power. This is the same as selecting Shut Down from the Finder's Special Menu. Please note that applications that terminate may display dialog boxes requiring user intervention. It is up to the user to respond and close those windows.

Return Codes:

RC -	See the Appendix in this manual.
AERC-	See Inside Macintosh Volume VI.

Example:

```
/* This example will instruct the finder to shutdown */
ADDRESS finder;
SHUTDOWN;
```

SLEEP

SLEEP ;

Event Suite : Finder

This command puts a portable Macintosh or PowerBook in power conserving state also known as system sleep. This command has no effect on non-portable Macintosh machines. This has the same effect as choosing Sleep from the Special Menu of the Finder.

Return Codes:

RC -	See the Appendix in this manual.
AERC-	See Inside Macintosh Volume VI.

Example:

```
/* This command will instruct the finder to put the */
/* Macintosh into 'sleep' mode */
ADDRESS finder;
SLEEP;
```

ZOOM WINDOW

ZOOM WINDOW *pathname window type* IN | OUT ;

Event Suite : Finder

This command will either Zoom in or Zoom out a specified window. Under System 7, the largest window is the smallest window that displays the entire contents of the window. A Zoom out command may actually make the window smaller.

Return Codes:

RC -	See the Appendix in this manual.
AERC-	See Inside Macintosh Volume VI.

Examples:

```
/* This example will zoom in a given window */
Address finder;
ZOOM WINDOW 'External HD' MAINWINDOW IN;

/* This example will zoom out a given window */
Address finder;
ZOOM WINDOW 'External HD' MAINWINDOW OUT;
```

REQUIRED SUITE

OPEN APPLICATION

OPEN APPLICATION ;

Event Suite : Required

This command instructs an application to perform any tasks that it would when it first starts up. In other words, perform it initialization code. The Apple Events Registry strongly recommends against using this command. Use it with care.

Return Codes:

-10000 - The Apple Event handler faild when attempting to handle the Apple Event.

-10011 - Could not handle this Apple Event because it is not part of the current transaction.

-10012 - The specified transaction is not a valid transaction; the transaction may never have begun, or it may have been terminated.

-10016 - The server application only handles this Apple Event when it is sent from an application running on the same computer.

Other Result Codes -Other result codes may be produced that may have caused the Apple Event to fail. One example would be the Memory Manager. The server application handler may also return other return codes.

Example:

```
/* This example will instruct the targeted */
/* application to perform actions required */
/* when starting up */
ADDRESS texx;
OPEN APPLICATION;
```

OPEN DOCUMENTS

OPEN DOCUMENTS *pathname list ;*

Event Suite : Required

This external command instructs the application to open the supplied document(s).
This is like choosing Open from the application's File Menu. The only difference being that you may send a list of documents instead of one at a time.

Return Codes:

-10000 - The Apple Event handler faild when attempting to handle the Apple Event.

-10011 - Could not handle this Apple Event because it is not part of the current transaction.

-10012 - The specified transaction is not a valid transaction; the transaction may never have begun, or it may have been terminated.

-10016 - The server application only handles this Apple Event when it is sent from an application running on the same computer.

Other Result Codes -Other result codes may be produced that may have caused the Apple Event to fail. One example would be the Memory Manager. The server application handler may also return other return codes.

Examples:

```
/* This example will instruct the targeted */
/* application to open a file */
ADDRESS texx;
PATH 'Hard Disk:System Folder:Preferences';
OPEN DOCUMENTS 'Profile.exec';

/* This example will instruct the targeted */
/* application to open 2 files */
ADDRESS texx;
PATH 'Hard Disk:System Folder:Preferences';
OPEN DOCUMENTS Profile.exec Shutdown.exec;
```

PRINT DOCUMENTS

PRINT DOCUMENTS *pathname list ;*

Event Suite : Required

This external command instructs the application to print the supplied document(s).
This is like choosing Print from the application's File Menu. The only difference being that you may send a list of documents instead of one at a time.

Return Codes:

-10000 - The Apple Event handler faild when attempting to handle the Apple Event.

-10011 - Could not handle this Apple Event because it is not part of the current transaction.

-10012 - The specified transaction is not a valid transaction; the transaction may never have begun, or it may have been terminated.

-10016 - The server application only handles this Apple Event when it is sent from an application running on the same computer.

Other Result Codes -Other result codes may be produced that may have caused the Apple Event to fail. One example would be the Memory Manager. The server application handler may also return other return codes.

Examples:

```
/* This example will instruct the targeted */
/* application to print a file */
ADDRESS 'Navigator 3.1.1';
PATH 'Hard Disk:Navigator Folder';
PRINT DOCUMENTS 'Log File';

/* This example will instruct the targeted */
/* application to print 2 files */
ADDRESS 'MPW Shell';
PRINT DOCUMENTS 'Source File 1' 'Source File 2';
```

QUIT APPLICATION

QUIT APPLICATION ;

Event Suite : Required

This command instructs an application to terminate. This is the same as choosing Quit from the application's File Menu. This command may cause the application to display dialog boxes requiring user intervention. It is up to the user to respond and close the dialog windows.

Return Codes:

-10000 - The Apple Event handler faild when attempting to handle the Apple Event.

-10011 - Could not handle this Apple Event because it is not part of the current transaction.

-10012 - The specified transaction is not a valid transaction; the transaction may never have begun, or it may have been terminated.

-10016 - The server application only handles this Apple Event when it is sent from an application running on the same computer.

Other Result Codes -Other result codes may be produced that may have caused the Apple Event to fail. One example would be the Memory Manager. The server application handler may also return other return codes.

Examples:

```
/* This example will instruct the targeted */
/* application to 'QUIT' */
ADDRESS Texx;
QUIT APPLICATION;
```

Miscellaneous Standards

DO SCRIPT

DO SCRIPT *Pathname;*

Event Suite : Micellaneous Standards

This command instructs an application to execute the given script. *Pathname* contains the name of the script. During the execution of this script, the targeted application may require user response. Depending on how the targeted application handles the **DO SCRIPT** command, the user may be required to manually respond to any requests by the application.

Return Codes:

-10000 - The Apple Event handler faild when attempting to handle the Apple Event.

-10011 - Could not handle this Apple Event because it is not part of the current transaction.

-10012 - The specified transaction is not a valid transaction; the transaction may never have begun, or it may have been terminated.

-10016 - The server application only handles this Apple Event when it is sent from an application running on the same computer.

Other Result Codes -Other result codes may be produced that may have caused the Apple Event to fail. One example would be the Memory Manager. The server application handler may also return other return codes.

Example:

```
/* This example will instruct the application execute */
/* the script 'DoMyWork' */
ADDRESS 'application name';
DO SCRIPT DoMyWork;
```

Notes:

The scripts passed to the targeted application are not Texx execs, unless the targeted application is Texx. The difference between the **DO SCRIPT** and the **OPEN DOCUMENTS** command is that the **DO SCRIPT** opens and executes the supplied file name. The **OPEN DOCUMENTS** command only opens the supplied file name.

Because of the automatic execution of execs feature of Texx, the **DO SCRIPT** and **OPEN DOCUMENTS** commands have the same effect. Either command will cause Texx to open and execute the supplied file name.

Chapter 4

Getting Started

Installation Requirements

Texx requires the Interapplication Communication capabilities of System 7. Texx will not operate on anything less than System 7. If an attempt is made to run Texx under an operating system less than System 7, Texx will post an error message and quit. The System 7 requirement also means a requirement of at least 2 megabytes of memory. However, a more useable figure is more like four or five megabytes of memory. Keep in mind that when Texx is communicating with another application, they must both be active. With today's growing applications, the memory needed is greater. Depending on the type of application that is running, five megabytes might not be enough for one application. Of course, virtual memory would ease the memory requirements.

Operating Modes

Texx has several operational features that may or may not be used, that is up to the user. There are two basic ways of running Texx, both have advantages and disadvantages. The preferred way is to place Texx, or an alias of Texx, in the Startup Items folder which is located in the System Folder. Doing this will startup Texx every time the Macintosh is started or restarted. Texx will just sit running in background until it has some work to do. At which time it may switch to the front and process an exec. The advantage to this method of operation is that it is always available when needed. The disadvantage is that it occupies memory, might be a problem on Macs with limited memory.

Another way to run Texx is to start it up manually when needed and shut it down when done. The advantage to this method is that the memory Texx uses is not occupied and therefore free for other applications. The disadvantage is that it is more time consuming and more cumbersome to use. While there is nothing wrong with running Texx this way, it is certainly not the preferred way.

Special Features

There are several features of Texx that might help is deciding which method of operation to use. When Texx starts up, it looks for an exec named "Profile.exec" in the Preferences folder located in the System Folder. If found, Texx will load and automatically execute it. The "Profile.exec" may be used to set up the user environment, i.e. startup other applications, pass commands to the Finder, and setup the Texx environment. If the exec is not found, Texx simply ignores it and continues with its operation. Upon shutdown, Texx looks for an exec named "Shutdown.exec". It will look for it in the Preferences folder located in the System Folder. If found, Texx will load and automatically execute it. The "Shutdown.exec" can be used for passing commands to other applications, and passing commands to the Finder. With System 7, one of the more useful commands to pass to the Finder is "Empty Trash". As with the "Profile.exec", if "Shutdown.exec" is not found, Texx simply ignores it and quits.

Texx is setup to do as much as possible automatically. If an icon for a Texx exec is double clicked, Texx assumes that the user meant to execute the exec. Texx will load and execute the exec. If Texx receives an OPEN DOCUMENTS apple event from another application, Texx will load and execute it. This is different from other Macintosh applications. If an icon representing an application document is double clicked, the application simply opens the document and displays it. Texx is different, Texx will not only open the document, but also execute it. To open an exec without executing, select Open from the File menu.

One of the most useful features of Texx is the AddExecMenu command. What this command does is add existing execs to the Exec menu. These execs may reside on any mounted disk and may contain any valid Texx commands. Once placed in the Exec menu, the user can execute the exec by selecting it from the menu. Upon selecting an exec from the Exec menu, Texx loads and executes it automatically. The best place to use this command is in the "Profile.exec" that is executed at startup. Using an exec to startup applications has several advantages. First, an exec is a more flexible way of initiating an application, it allows you to open multiple documents without performing finger gymnastics. It allows you to pass commands to the application in addition to just opening documents. It far exceeds the capabilities of the Apple Menu. And, possibly the most helpful advantage, eliminating the need for aliases. Texx will work with aliases, but Texx renders aliases unnecessary. The only need for aliases are those that applications may require.

Installing Texx

Prepare for Installing Texx

1. Decide which method of operation is desired, automatic startup or manual startup.

2. Locate the file named Texx.sea.

3. Move the Texx.sea icon to the desired install location.

4. Double click the Texx.sea icon to expand the Texx Folder.

5. If installing Texx for manual operation, skip the section Installing for Automatic Startup.

Installing Texx for Automatic Startup

1. Locate the icon for the Texx application in the Texx Folder.

2. Locate the Startup Items folder in your System Folder.

3. Drag the Texx icon into the Startup Items folder. You may create an alias for the Texx icon and drag the alias into the Startup Items folder.

4. Locate the files "Profile.exec" and "Shutdown.exec" in the Texx Install Folder.

5. Locate the Preferences folder in your System Folder.

6. Drag the icons for the "Profile.exec" and "Shutdown.exec" files into the Preferences folder. You may create an alias for both files and drag the aliases into the Preferences folder.

7. Restart your Macintosh. On restart, Texx will startup and execute the "Profile.exec".

8. Texx is now ready for further work.

9. Skip the following section Installing Texx for Manual Startup.

Installing Texx for Manual Startup

1. Drag the Texx Folder to the desired location.

2. Locate the files "Profile.exec" and "Shutdown.exec" in the Texx Folder.

3. Locate the Preferences folder in your System Folder.

4. Drag the icons for the "Profile.exec" and "Shutdown.exec" files into the Preferences folder. You may create an alias for both file and drag the aliases into the Preferences folder.

5. Locate the Texx icon in the Texx Folder and double click it. This will start Texx. On startup, Texx will execute the "Profile.exec" that was just put into the Preferences folder.

6. Texx is now ready for further work.

Operating Texx

Regardless of the mode of operation, once Texx is up and running, it behaves identically. Texx uses a window called "Texx System Window" for communications with the user. The Texx System Window displays all error messages and output of the SAY instruction. The Texx System Window must always be active, if it is closed, Texx will terminate. Texx will allow only four windows to be open at one time, that includes the Texx System Window. When an exec is executing, it requires a window, if one is not available, the exec will not execute.

Texx Menus

This version of Texx has <u>minimal</u> text editing facilities. Therefore, the menus are very basic and simplistic.

File Menu

NEW Or **Command-N** will open a new empty window. The user will be prompted for a new file name and location. After replying, a new, empty window will be opened.

OPEN Or **Command-O** will open an existing file. The user will be prompted for a file name with the standard file open dialog box. If the file selected is a Stationary Pad, the user will be further asked to name the new copy of the Stationary Pad. Once the new name is entered, a new file is created and the contents of the Stationary Pad are put into the new file.

CLOSE Or **Command-W** will close the frontmost window and the file associated with the window. Please note that if the frontmost window is the Texx System Window, Texx will terminate.

SAVE Or **Command-S** will save the contents of the frontmost window to its associated file. Neither the window or the file will be closed, both will remain open.

SAVE AS Will save the contents of the frontmost window to a new file. The user will be prompted for a new file name. Once the new name is entered, the contents of the window are saved to the new file and the old file is closed.

PAGE SETUP Not implemented in this release of Texx.

PRINT Not implemented in this release of Texx.

QUIT Or **Command-Q** will terminate Texx. If there are other open windows for which the contents have not been saved, the user is asked if the contents should be saved. After all the windows have been closed, Texx will terminate.

Edit Menu

CUT Or **Command-X** will cut the current selection from the frontmost window. The contents cut may be undone by Pasting them back into the window.

COPY Or **Command-C** will copy the current selection from the frontmost window. The contents may then be pasted elsewhere. The difference between this option and CUT is that the COPY leaves the source intact and CUT removes the selection from the souce window.

PASTE Or **Command-V** will paste the contents of the scrap into the cursor position in the frontmost window. The scrap contains data from other applications, or the CUT or COPY options.

CLEAR Will clear the selected source from the frontmost window. This option works similar to the CUT option.

Exec Menu

RUN EXEC Or **Command-R** will execute the exec in the frontmost window.

Additional entries These additional entries are appended to the Exec menu by the ADDEXECMENU command. These entries are names of execs. Selecting one of them will execute that exec.

Window Menu

TEXX SYSTEM WINDOW This option will move the Texx System Window to the front and make it the frontmost window. If the entry is check marked, the Texx System Window is already the frontmost. There is no action taken if the entry is checked and selected.

Additional entries	These additional entries are appended to the Window menu everytime a new window is opened and deleted everytime a window is closed. The frontmost window will be check marked. Selection one of these entries will bring that window to the front and make it the frontmost window.

This page intentionally left blank

Appendix

Texx Error Messages

Types of Error Messages

Texx has several errors that may be generated from startup to shutdown. In addition to the errors generated by Texx, other errors may be generated by the Macintosh operating system. This appendix will list only the Texx generated errors. For the Macintosh errors, please consult Inside Macintosh Volumes I - VI or whatever publication that may list them.

Texx errors are given to the user in one of three ways. First, some messages will display a dialog box with some text describing the error. For example, if Texx is started up on a pre-System 7 system, a dialog box will be displayed with the message that Texx must be run on a System 7 system. Second, the Texx System Window will display the error with a short text message describing the error. These errors are usually errors generated by the parser. In all cases, the execution of the exec is halted and the Texx System Window is brought to the front with the error message displayed. The third type of error will be returned in the pre-defined variable RC. When using instructions that affect RC, RC should always be checked after the instruction has been executed. In most cases, the instruction was not properly completed. It is up to the exec author to check the variable and take whatever action necessary. An example of this error is when sending an external command to another application. RC may have any number of different errors.

There are two other possible error messages. They are both generated when sending external commands. These error messages are returned in the pre-defined variables AERC and AERCSTRING. AERC is a numeric value returned by the Apple Event routines. These errors are documented in Inside Macintosh Volume VI and in the Apple Event Registry. AERCSTRING is a string variable that may be returned by the application receiving an external command. The application may return a string describing an error. This return string is optional and may not be present. AERC and AERCSTRING are meant to provide more information about the error than what RC can provide. IF RC is zero, than the command was successful. If, however, RC is not zero, then AERC and AERCSTRING can provide additional information.

Texx error messages are of the following format:

TEXXnnnn - descriptive message.

Every message has the prefix of TEXX followed by a number and a short text message describing the error. Use the number given to lookup the error message. In most cases, the text describing the error message is descriptive enough to determine the problem.

Texx Error Messages

5000 - Gestalt - Gestalt function not available.
The Gestalt function is not available. Texx uses the Gestalt function to determine its operating environment. If Gestalt is not available, Texx cannot get the required information. This error will probably mean that Texx is not running under System 7.

5001 - Too many windows, only 4 allowed.
Texx only allows four windows to be open. This message means that the four windows are already open. To open the window desired, close an open window.

5002 - Error reading resource file.
Error reading a resource from the Texx application file. Probably means that the Texx application is corrupted and should be reinstalled.

5003 - Error opening file.
Error opening a file. This error will probably not happen. If it does, it probably means that something is wrong with the disk directory.

5004 - File too large.
The file must be 32k in size. This is a restriction of TextEdit. It is doubtful that an exec will be larger than 32k.

5005 - Out of memory.
Texx is constantly allocating and deallocating memory. This error should rarely happen. Try terminating and starting Texx again.

5006 - Error reading the file.
If this error happens, it probably means that the file is corrupted.

5007 - Error closing the file.
Tried to close a file and couldn't. This error should be rare. Could be a problem with the disk directory.

5008 - Error writing the file.
This error could mean that the disk is full. Possibly an error with the disk.

5009 - Error converting the Clipboard.
When switching from an application to Texx, converting a clipboard could generate an error. Seems to happen when switching from a graphics program. Can be ignored unless trying to paste text from another application. Try it again.

5010 - Error installing Apple Event handler.
There was an error installing an Apple Event handler. Probably means that

Texx will have other problems with external commands. Check that System 7 is running and try starting Texx again.

5011 - Error processing Apple Event.
Texx received an Apple Event for which there is not handler. Try to find out what event it is and report it to me.

5012 - Print not supported.
Printing is not yet supported.

5013 - Gestalt error.
The Gestalt function returned an error. Try starting Texx again.

5014 - Not running under System 7.0.
Texx must run under System 7.

5015 - Texx command invalid.
An error was detected with a command. Depending on where the error occurred, it could be on the statement preceding the one displayed.

5016 - Invalid Texx exec, must start with comment.
A Texx exec must start with a comment.

5017 - Invalid process name.
The target name given in an ADDRESS instruction was invalid. Probably an invalid character or invalid length. This error should be returned in the variable RC.

5018 - Process not active.
The target name of an ADDRESS instruction was not valid. When an ADDRESS instruction is processed and the process name is verified, Texx then checks to make sure that the process (application) is active. If not active, Texx cannot pass external commands. This error is returned in the variable RC.

5019 - Missing '=' in expression.
An '=' was missing in an instruction.

5020 - Error Pop'ing expression stack.
There was an error getting an expression value from the expression stack. The expression stack is quite large and this error should not occur. If it does, break the instruction into smallar parts.

5021 - Error in expression.
An error was found in an expression. Depending on where the error occurred, the error may be in the statement previous to the one shown.

5022 - Incompatible operands.
An expression was evaluated with operands that are incompatible. For example, adding a string value to a numeric value would make them incompatible.

5023 - Invalid operands.
An expression was evaluated with operands not consistant with the expression.

5024 - Divide by zero.
An expression that would result in a division by zero was found.

5025 - Expression stack overflow.
The expression stack overflowed. This should be rare because the stack is quite large. If it does occur, split the expression into smaller pieces.

5026 - Missing right parenthesis in expression.
A matching right parenthesis in an expression is missing.

5027 - Error in BY expression of DO statement.
An error in the BY expression of a DO statement was found. Check the BY portion of the DO statement.

5028 - Error in DO statement.
An error in a DO statement was found.

5029 - Error in FOR expression of DO statement.
An error in the FOR expression of a DO statement was found. Check the FOR portion of the DO statement.

5030 - Error in TO expression of DO statement.
An error in the TO expression of a DO statement was found. Check the TO portion of the DO statement.

5031 - Error in IF statement expression.
An error in an IF expression was found.

5032 - Missing THEN statement in IF statement.
The THEN of an IF statement is missing.

5033 - Error or instruction not yet implemented.
An undetermined error was found. Probably an instruction that is not yet implemented.

5034 - No previous ADDRESS instruction executed.
An ADDRESS instruction with no parameters that is meant to swap with the previous ADDRESS executed was processed. The previous ADDRESS with which to swap with has not yet been executed. In other words, there is no process to switch.

5035 - Invalid label.
A label in invalid. Could be a variable of a different type.

5036 - Invalid target of SIGNAL or GOTO.
The target of the GOTO or SIGNAL was invalid. Could be a variable of a different type.

5037 - Invalid address of SIGNAL or GOTO.
The address (label) of a SIGNAL or GOTO is invalid. Could be missing

or a label of a different type.

5038 - Label not found.
A label was not found.

5039 - Invalid Apple Event target address.
The target address was invalid. This error is returned in the variable RC.

5040 - Error creating Apple Event.
An error occurred creating an Apple Event. This error is returned in the variable RC. The variable AERC will contain more information.

5041 - Error sending Apple Event.
An error occurred sending an Apple Event. This error is returned in the variable RC. The variable AERC will contain more information.

5042 - Invalid Apple Event.
The Apple Event is invalid. This error is returned in the variable RC.

5043 - Apple Event not found.
The Apple Event was not found. The Apple Event either is invalid or not yet supported.

5044 - Invalid path name.
The path name supplied was invalid. Could be invalid characters, invalid lengths, or a partial path name.

5045 - No previous path.
A PATH instruction without any parameters was processed. This PATH instruction is meant to swap the current path with the previous path. This error means that a previous was never executed.

5046 - Error creating Apple Event description record.
An error creating an Apple Event descriptor record was generated. This error is returned in the variable RC. The variable AERC will contain more information.

5047 - Error creating Apple Event Description list record.
An error creating a descriptor list was generated. This error is returned in the variable RC. The variable AERC will contain more information.

5048 - Error creating Alias for file.
An error occurred creating an Alias record. This error should not occur. However, if it does occur, the external command will not be processed.

5049 - Missing Apple Event parameters.
Apple Event parameters are missing. This error is returned in the variable RC. The variable AERC might contain more information.

5050 - Invalid path name.
The supplied path name is invalid. Probably the path name was a partial path name and the full path name could not be determined.

5051 - Find folder error.
There was an error with the FindFolder function. This error should not

happen. If it does, there is possibly an error with the System Folder.

5052 - **Error creating FSSpec.**
There was an error creating an FSSpec. This error should not occur.

5053 - **Error creating Alias.**
There was an error creating an Alias record. This error should not occur.

5054 - Invalid Finder window.
The finder window supplied was invalid.

5055 - Error in window specification.
An error in window type supplied. Should be MAINWINDOW, INFOWINDOW, or SHARINGWINDOW.

5056 - Invalid window view type.
Invalid window view type. Should be BYNAME, BYTYPE, etc.

5057 - Apple Event Long Integer error.
An error supplying numeric values such as icon locations or window sizes was found.

5058 - Invalid Zoom Window value, must be IN or OUT.
The Zoom Window IN or OUT parameters were missing.

5059 - Could Not Display ABOUT Dialog Box.
An error occurred trying to display the ABOUT dialog box.

5060 - Text file larger than 32k.
The current file has exceeded the TextEdit maximum of 32k. No more additions to the file can be made.