Mac2E

COLLABORATORS			
	TITLE : Mac2E		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		January 8, 2025	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Mac2E

1.1 Mac2E

```
*****************************************************************************

                                Mac2E (v3.0)
                        Macro preprocessor for the E language
                               Archive of 10 March 1994
                        © Copyright 1993, 1994, Lionel Vintenat

*****************************************************************************

WARNING ! All the executables in this archive require Workbench 2.0 or higher
to run. Sorry to 1.3 users.
```


1.2 Introduction

```
    This paragraph answers the 3 essential questions :
    - Why Mac2E ?
    - What does Mac2E do ?
    - How does it do it ?
```

1.3 How it all started...

In the beginning, there was Amiga E and me. It was great, the two of us, we wrote wonderful programs in record time. As I didn't have (and still don't have) the RKM's, these programs were very ugly, without graphical interfaces, but no matter, they were good times...

And then, MUI arrived, and nothing has been the same since between Amiga E and me. Why? Well, Amiga E does not permit the use of macros, and programming MUI without macros is almost crazy! On the other hand, it was inconceivable to pass up something like MUI. So, I retreated for a while to the language C: it was the beginning of dark times for my Amiga...

Then I got access to the INTERNET. I spoke about my problem with Wouter who advised me to use a C preprocessor: there was a great idea! But after trying, it turned out to be very tedious to use: compilation times were increased by a factor of 100, and the compiler didn't give the correct line numbers for errors. It was then that I got the idea for Mac2E...

1.4 General Presentation

Mac2E is a preprocessor for the Amiga E compiler by Wouter van Oortmerssen, but it only knows how to do one thing: replace macros in an E source. In other words, the "conditional compilation" and "file inclusion" aspects, for example, are not handled by Mac2E, whereas they are with most C preprocessors.

Oh! I almost forgot: all the executables in this archive are of course written in Amiga E!

1.5 Spirit

I designed Mac2E with 3 ideas in mind:

- make something easy to use (in the spirit of Amiga E)
- solve the problems that I encountered in using a C preprocessor with Amiga E (see how it all started...)
- make a preprocessor whose use doesn't make E sources dependant on it; in other words, if another version of Amiga E which contains a preprocessor comes out, the conversion of your sources from Mac2E to the new preprocessor should require very few modifications

I think that this version 3.0 effectively implements these 3 ideas, in that:

- Mac2E remains very close to the level of use of a classical C preprocessor, so learning it will be very quick for most programmers
- using Mac2E on a file takes about the same amount of time as the compilation itself, which, taking into account the speed of Amiga E itself, aught to be acceptable even for slow Amigas
- Mac2E never introduces line feeds when it replaces a macro, so that the compiler always indicates the correct line number when reporting errors
- Macro definition is done in separate files from the source, and the macro files are passed directly to the command Mac2E without your source needing to be modified by a single character, so the passage from Mac2E to the future (maybe) Amiga E preprocessor will be very simple and will require a minimum of modifications to your source code.

1.6 What is a macro ?

In a simple manner, we define a macro by associating an identifier (the macro's name) with a chain of characters (the body of the macro). Then, instead of putting the body of the macro in your source, you simply put it's name and the preprocessor replaces the name of the macro with the body.

In my opinion, the macros are very useful in 3 cases :

- to avoid rewriting the same sequence several times
  -> see example 1
- to facilitate the use of abstract values
  -> see example 2
- to logically regroup a program sequence
  -> see example 3

Of course, this is a very superficial view of the notion of macros. Macros, as implemented in almost all preprocessors today, permit lots of things. The following paragraphs present the use of macros in detail.

1.7 Example 1

Consider the example of a program which reads memory sequentially. For this purpose, 2 variables are defined:

```
DEF memory_pointer:PTR TO CHAR, character
```

Accessing a byte is done in the following manner:

```
character:=Char(memory_pointer++)
```

Without macros, you need to type that line with each read. If you are performing reads in several procedures, this can quickly become tedious. The solution is to define a macro with the name ReadMemory and the body character:=Char(memory_pointer++). You then need only type ReadMemory each time.

1.8 Example 2

To open a library, you need to pass it's name in lowercase to the function OpenLibrary(). If you write OpenLibrary('Dos.library',0), there will be no error during compilation, but the library will not be found during execution. The solution is to define a macro with the name DosLibraryName and the body 'dos.library'. This way, you need only type OpenLibrary(DosLibraryName,0) to open the dos library, without worrying about making a typing mistake.

1.9 Example 3

If you want to be sure that stdout is non null, the Amiga E documentation advises placing a WriteF('') at the beginning of your program. A more elegant solution is to define a macro with the name OpenStdout and the body WriteF(''). You then simply use OpenStdout in your source, which is much more eloquent.

In this simple example, the difference between this case and the 2 previous ones is not very clear, but what is important to understand is that the OpenStdout macro is not local to one program (as in example 1), but can be used in all programs where it is necessary for stdout to be non null. In addition, OpenStdout behaves like a mini-procedure (as opposed to example 2) which accomplishes a task.

1.10 Defining a macro

The following paragraphs explain the different syntaxes for defining a macro, from the simplest to the most complex.

Definition of a macro without parameters
Definition of a macro with parameter(s)
Advanced definition of a macro

1.11 Definition of a macro without parameters

A simple macro definition has the following syntax:

```
#define macro_name macro_body
   |   |     |    |     |    |
  (1) (2)   (3)  (2)   (4)  (5)
```

where

(1) #define marks the beginning of the definition and can be found anywhere on the line (not necessarily at the beginning)

(2) 1 or more spaces and tabs

(3) the name of the macro (any combination of numbers, uppercase and lowercase letters, and "_" characters)

(4) the body of the macro (any combination of characters other than carriage returns)

(5) carriage return which marks the end of the macro definition

Examples :

```
#define ReadMemory      character:=Char(memory_pointer++)
#define DosLibraryName  'dos.library'
#define OpenStdout      WriteF('')
```

1.12 Definition of a macro with parameter(s)

Like a procedure, a macro can have parameters. The definition syntax is then:

```
#define macro_name(parameter1,parameter2,...,parameterN) macro_body
   |   |   |       |    |     |    |     |   |    |     |       |    |
  (1) (2) (3)     (4)   |    (5)   |    (5) (5)   |    (7)     (8)   (9)
                        +----------+--------------+
                                   |
                                  (6)
```

where

(1) #define marks the beginning of the definition and can be placed anywhere on the line (not necessarily the beginning)

(2) 1 or more spaces and tabs

(3) the name of the macro (any combination of numbers, uppercase and lowercase letters, and "_" characters)

(4) opening parenthesis immediately following the macro's name

(5) comma separating each parameter

(6) 1 or more parameters (any combination of numbers, uppercase and lowercase letters, and "_" characters); each parameter can be preceded and followed by any number of spaces and tabs

(7) closing parenthesis which can be followed by any number of spaces and tabs

(8) the body of the macro (any combination of characters other than carriage returns)

(9) carriage return marking the end of the macro definition

Examples :

```
#define Power2( x )                 ((x)*(x))
#define SwapVariablesXY(X,Y,TEMP)   TEMP:=X; X:=Y; Y:=TEMP
#define Max( x , y )                (IF (x)>(y) THEN (x) ELSE (y))
```

Parameters specified in the macro definition are called formal parameters.

1.13 Advanced macro definition

It can happen that the body of a macro is too long to fit in one line on the screen. It is possible to break the body of the macro into several pieces. To tell the preprocessor that the body is continued on the next line, place a "\" character before the carriage return ending the line. The preprocessor will skip the "\" character and the carriage return and will interpret the next line, starting with the first character, as part of the body of the macro. A macro can extend over several lines in this manner. The definition syntax in this case is:

```
#define macro_name(parameters)  body_piece1 \
                body_piece2 \
                ...
                body_pieceN
```

Warning: the "\" character must be immediately followed by a carriage return for the preprocessor to interpret it correctly.

1st example :

```
#define SwapVariablesXY(X,Y,TEMP) TEMP:=X; \
X:=Y; \
Y:=TEMP
```

is a macro which has as it's body TEMP:=X; X:=Y; Y:=TEMP
(note that the ";" characters are necessary preprocessor skips the carriage

```
returns after the "\" characters)

    2nd example :
#define SayHello WriteF('Hello, I'm the one who wrote \
the great program Mac2E (pub) !\n')
    is a macro which has as it's body
WriteF('Hello, I'm the one who wrote the great program Mac2E (pub) !\n')
(note that the single "\" character followed by a carriage return was
interpreted as a signal that the body of the macro is continued)

    3rd example :
#define UselessMacro [1 space ->\
 ][2 spaces ->\
  ][3 spaces ->\
   ] and that's all !
    it a macro which has as it's body
[1 space -> ][2 spaces ->  ][3 spaces ->   ] and that's all !
```

1.14 Using a macro

Defining macros is not everything, we also want to use them! To do that, you simply have to place the names of the macros you have defined where you need them in your source code the same way you would use normal instructions. But be careful, a macro is not an instruction recognized by the compiler. Before compiling a program containing macros, you must use the preprocessor. The purpose of this program is to find all the macro names in a source code file and replace them with the body of the associated macro. The body of a macro should contain instructions recognized by the compiler! Once the preprocessor is finished, the file can be compiled.

The following paragraphs explain in detail how the preprocessor proceeds to find and replace a macro name.

1.15 Identifying a macro name

For a macro name to be recognized by the preprocessor, the name which you put in the source code file must be:

- exactly the same as the one specified in the definition; the preprocessor distinguishes between uppercase and lowercase
- preceded and followed by a character other than a letter, a number or a "_" character

If these 2 conditions are met, the preprocessor will recognize the macro name.

Examples :

Suppose that you have defined a macro name toto (the body's contents don't matter). It will be recognized in the following instruction sequences:

```
a:=toto+1
WriteF('Silly string to introduce \d !\n',toto)
```

However, the preprocessor will not recognize it in the following instructions sequences:

```
a:=different_than_toto+1
WriteF('Silly string to introduce \d !\n',toto1)
```

1.16 Handling of argument passing

You have seen in a previous section that we can give parameters (called formal) to a macro in it's definition, as you would for a procedure. Then, as for a procedure, when you use a macro you must provide it with arguments (called real parameters). The calling syntax for a macro (I use the word call as an analogy to procedures) is the following:

```
macro_name(parameter1,parameter2,...,parameterN)
   |      |    |     |    |     |   |    |     |
  (1)    (2)   |    (4)   |    (4) (4)   |    (5)
               +----------+--------------+
                          |
                         (3)
```

where

(1) the name of the macro

(2) opening parenthesis immediately following the macro name

(3) 1 or more parameters (any combination of characters other than carriage returns)

(4) comma to separate each parameter

(5) closing parenthesis

Warning: the parameters are bounded by commas and parentheses, and between these 2 consecutive symbols, all the characters are taken into account and interpreted as being part of a parameter.

If a macro was defined without parameters, it's calling syntax is simply macro_name.

When the preprocessor analyses a macro call, it naturally expects to find as many real parameters as formal parameters! In particular, a macro defined without arguments should not be followed by a "(" character, otherwise the preprocessor will think that the macro is being called with arguments.

If the calling syntax for a macro is correct, the preprocessor associates each real parameter with the corresponding formal parameter, as the compiler does for a procedure.

Examples :

Suppose that you have defined a macro toto like this:

```
#define toto(param1, param2) any_body
```

Here's a table of what will happen for several calling sequences:

calling sequence	assoc'd to param1	assoc'd to param2
toto(a,1)	a	1
toto(a , 1)	a	1
toto((3+2)*5 ,WriteF('Ah !\n'))	(3+2)*5	WriteF('Ah !\n')

```
| toto (a,1)                          |             E R R O R             |
| toto(1,2,3)                         |             E R R O R             |
+-------------------------------------+-----------------+-----------------+
```

1.17 Replacing a macro

If a macro to be replaced was defined without parameters, the preprocessor simply substitutes the macro's name with it's body.

If, on the other hand, the macro to be replaced was defined with parameters, the preprocessor still replaces the macro's name with it's body, but also substitutes all the formal parameters in the body with the corresponding real parameters.

1st example:
Consider the following macro definition:

```
#define DosLibraryName 'dos.library'
```

We will then have, for example, the call OpenLibrary(DosLibraryName which will be replaced by OpenLibrary('dos.library').

2nd example:
Consider the following macro definition:

```
#define Square(x) ((x)*(x))
#define Max(x,y) (IF (x)>(y) THEN (x) ELSE (y))
```

We will then have, for example, the call

```
a:=Square(4+3) * Max(7,2*(8-2)) which will be replaced by
a:=((4+3)*(4+3)) * (IF (7)>(2*(8-2)) THEN (7) ELSE (2*(8-2)))
```

Note how the many parentheses present in the bodies of these 2 macros control the evaluation priority of the expression. Without them, the result will not be what was expected. Generally, you must be very careful in creating a macro. In effect, even if a macro resembles a procedure or a function, it's not exactly the same! The body of a macro is never evaluated during a call, it is simply substituted for the macro name. It can therefore find itself stuck right next to another expression. The example macro Square is a good example of this kind of problem.

1.18 Advanced use

By now you should have mastered the definition and the usage of macros. If this is not the case, return to the preceding sections.
The following paragraphs explain more technical aspects of macro use, but they are nonetheless still important to be aware of.

Macros, comments and strings
Macros in a macro body
Macro calls as macro arguments
Special characters

1.19 Macros, comments and character strings

It has been previously stated that a macro call can be placed anywhere in a source code file. Well, that's not true! In reality, the preprocessor does not look for macro calls in comments (including nested comments) or in strings. In effect, macros are there to regroup under one name a section of code. There is therefore no reason to put macro calls within comments and strings.

In practice, this signifies that you can put whatever you want in comments and strings, the preprocessor will not touch it.

1.20 Macros in a macro body

When you define a macro, you can put whatever you want in it's body, even calls to other macros. The preprocessor will handle this kind of call during the substitution of the surrounding macro's name. In effect, the preprocessor makes as many substitutions as possible, and when it is finished, there will not be a single macro call left in the source code file. Of course, the calling arguments for a macro within the body of another macro can be the formal parameters of the outer macro. There is no limit to the depth of these imbrications.

Warning: the body of a macro cannot contain calls to itself, otherwise the preprocessor would make the same substitution infinitely, until all free memory is used up... or the user's patience is!

1st example :

Suppose you define 2 macros as follows:

```
#define InfiniteValue $FFFFFFFF
#define FinitePositiveNumber(x) (((x)>0) AND ((x)<>InfiniteValue))
```

Then you can, for example, make the call:

```
IF FinitePositiveNumber(A*B)=FALSE THEN WriteF('Error !\n') which will
be replaced by
IF (((A*B)>0) AND ((A*B)<>$FFFFFFFF))=FALSE THEN WriteF('Error !\n').
```

2nd example :

Suppose you define two macros as follows:

```
#define AbsoluteValue(x) (IF (x)>0 THEN (x) ELSE -(x))
#define MaxOfAbsoluteValues(x,y) (IF AbsoluteValue(x)>AbsoluteValue(y) THEN
(x) ELSE (y))
```

You can then, for example, make the call

```
a:=MaxOfAbsoluteValues(5,-(A*B)) which will be replaced by
(IF (IF (5)>0 THEN (5) ELSE -(5))>(IF (-(A*B))>0 THEN (-(A*B)) ELSE -(-(A*B)))
THEN (5) ELSE (-(A*B))).
```

1.21 Macro calls as macro arguments

In the preceding examples, you probably noticed that the arguments passed to a macro can be anything, as long as they are coherent, obviously (carriage returns are not, for example, allowed in an argument). You can even use a macro call as the argument for another macro. Again, the preprocessor will handle first the macro call included in the argument, and then afterwards the surrounding macro call with the substituted argument. There is no limit to the depth of these imbrications.

Remember: the general rule is that the preprocessor handles absolutely all the macro calls in a source code file, where it finds them, except in comments and strings.

1st example :
Consider the following examples:

```
#define SillyValue 12
#define Double(x) (2*(x))
```

The macro call Double(SillyValue) would be replaced by (2*(12)).

2nd example :
Consider the following examples:

```
#define MaskWeightStrong(x) ((x) AND $FFFF)
#define Average(x,y) (((x)+(y))/2)
```

The macro call Average(100,MaskWeightStrong(100000)) would be replaced by (((100)+((100000) AND $FFFF))/2).

1.22 Special Characters

You have no doubt wondered what would happen if a macro's argument contained the characters "(", ")" or ",". As they are used to delimit the arguments, their presence can cause confusion with their recognition. Well, it doesn't matter; the preprocessor is intelligent enough to distinguish which of these characters are there to delimit the arguments and which are part of the arguments.

Warning: the arguments must nonetheless remain coherent! For example, every opening parenthesis must have a corresponding closing parenthesis. Also, a comma must be enclosed within quotes or in a string enclosed within quotes.

Examples :
Consider a macro toto defined as having 2 formal parameters.
Here is a table of what will happen for different calling sequences:

calling sequence	1st parameter	2nd parameter
toto((3+4)*(5-6),'1, 2 et 3')	(3+4)*(5-6)	'1, 2 et 3'
toto((((())())()),character ",")	(((())())())	character ","
toto(),4)	E R R O R	
toto(4,,)	E R R O R	

1.23 Using Mac2E

To understand what follows, you should know what a macro is, and particularly how to define and use a macro as it is done in the language C. If this is not the case, return to the section What is a macro? .
If you already know all about C macros before getting this program, reading this section is not necessary. However, you should refer to it to verify syntax. Basically, the following paragraphs discuss the use of PreMac2E and Mac2E only, without recalling anything about macros.

The goal of these programs is, let me remind you, to permit the use of macros in your source code.

The first thing to do is therefore to define some macros. This is done in a file separate from your source code, called the macro file. Next, you should pre-analyze this file with PreMac2E. Finally, you can use Mac2E to replace the macro calls in your source code files. The handling of a source code file thus requires 3 steps, which are described in detail by the following paragraphs.

1.24 Macro files

A macro file is an ASCII file containing only macro definitions. I remind you that you cannot define macros in your source code, you must do it separately in a macro file.

These files may also contain comments. These can be placed anywhere except within a macro definition. In other words, the comments are placed between the macro definitions. Note that the comments can be placed in the file as they are, with neither beginning nor ending delimiters.

Normally, the macro files are placed in the sub-directory MacroFiles/ which is in the directory where you installed Amiga E.

See Defining a macro

1.25 Calling PreMac2E

When you have defined a macro file, it is not directly usable by Mac2E, the preprocessor in this archive. In effect, you must first pre-analyze it. This is done by PreMac2E, which is called as follows:
PreMac2E macro_file pre_analyzed_macro_file
where
- macro_file designates the macro file (eventually with path)
- pre_analyzed_macro_file designates the file (eventually with path) resulting from the pre-analysis

For example, to pre-analyze the macro file mui.e provided with this archive, I typed PreMac2E MacroFiles/mui.e PreAnalysedMacroFiles/mui.e.

Normally, the pre-analyzed macro files are placed in the sub-directory PreAnalysedMacroFiles/ which is in the directory where you installed Amiga E.

During a pre-analysis, PreMac2E tries to replace all macro calls within the bodies of other macros. In addition, it sorts the macros and classes them in a hash-coded table. Finally, it writes the contents of this table to the output file. Thus, when Mac2E processes a source code file, it will not use the original macro file but rather the pre-analyzed macro file. The speed increase is enormous.

The pre-analysis of a macro file is thus done only once (assuming there are no errors in the file, of course). Afterwards, you use only the pre-analyzed file. If you modify the original macro file, you must obviously redo the pre-analysis with PreMac2E for the changes to take effect.

The exact usage for PreMac2E is "FROM/A,TO/A,VER=VERBOSE/S,KS=KEEPSPACES/S". You can specify 2 supplementary parameters when calling it.

VERBOSE is explained in the section on error messages Messages_erreur}.

KEEPSPACES forces PreMac2E to conserve within the body of a macro the spaces and tabs at the start of the line, when the body of the macro spans several lines. By default, PreMac2E eliminates them, reducing the size of the pre-analyzed file and accelerating handling by Mac2E.

1.26 Calling Mac2E

Mac2E is the actual preprocessor in this archive. It is, in reality, the one that looks after replacing the macro calls in your source code file with the appropriate body. It has the following syntax:
Mac2E e_source_file e_destination_file pre_analyzed_file_list
where

- e_source_file is a source file (eventually with path) which contains the macro calls to be processed
- e_destination_file is the file (eventually with path) which will contain the e_source_file code but with the processed macro calls
- pre_analyzed_file_list is a list of 1 or more files of pre-analyzed macros (eventually with path)

The exact usage for Mac2E is "FROM/A,TO/A,WITH/A/M".

Called like this, Mac2E loads all the pre-analyzed macro files specified on the command line and, according to the macros defined within them, processes the source file accordingly.

See using a macro and advanced use

1.27 Error messages

All the error messages returned by PreMac2E and Mac2E are sufficiently self-explanatory. The line number where the error was found is also given.

The only exception to this is when PreMac2E processes macro calls within the bodies of other macros in a macro file. This phase of the pre-analysis is done after all the macro definitions are recorded. Therefore, PreMac2E is no longer working with line numbers. To find the location of an error reported during this phase, you must rerun PreMac2E with the VERBOSE option. This will force PreMac2E to show the name of every macro who's body it is processing. Thus, when it gives an error, you simply look at the name of the macro which was being processed when the error occurred, and find that macro in the file. The error is in the actual body of that macro.

1.28 Mac2E and MUI

If you read how it all started... , you know that Mac2E owes it's existence to the fact that MUI is infinitely easier to use with macros than without. That is why the first example (and the only one for the time being) of using Mac2E concerns MUI. You will find in this archive everything you need to use MUI with Amiga E, practically in the same manner that you would do it in C. To do this, you need 6 things:

- PreMac2E
- Mac2E
- mui.m
- muimaster.m
- mui.e
- OptiMUI2E

The general idea which governed the conception of mui.m and mui.e was, I recall, to make a "MUI-Amiga E interface" very close to that of C, so that during programming, it is quite useful to refer to the include file mui.h intended for C. It contains an impressive number of comments which are not all present in "the E version".

All of "this interface" is based on MUI 2.0. In the MUI archive, there are already some files for use in E, but neither as complete nor as practical as those supplied here, so forget about them and use these ones!

```
Mac2E
PreMac2E
mui.m
muimaster.m
mui.e
OptiMUI2E
```

1.29 mui.m

mui.m is, as it's name implies, a classical Amiga E include file. It contains all the structures defined in mui.h with the difference that all the names (of structures and their fields) are in lowercase. This limitation is due to Iconvert.

To use MUI structures in your programs, you need to put MODULE 'libraries/mui.m' at the beginning of your source code file.

1.30 muimaster.m

muimaster is, as it's name implies, a classical Amiga E file. It contains all the function definitions for the library muimaster.library. The function names are the same as in C except they all start with Mui instead of MUI (example: Mui_NewObjectA). This limitation is imposed by Amiga E as function names must have the first letter in uppercase and the second in lowercase.

To use the functions of the library muimaster.library in your programs

(and chances are you do use them!), you must put MODULE 'muimaster.m'
at the beginning of your source code file.

1.31 mui.e

mui.e is the key to the gateway to "this MUI-Amiga E interface" since it contains all the macros (constants and more complex things like the object definitions) of the file mui.h, but adapted for the E language. The syntax of the mui.e macros, as well as the syntax of their bodies, is exactly the same as in mui.h.

As well as it's advantage for using MUI, this file also constitutes a large library of examples of macro definitions.

The file mui.e is supplied in 2 copies: one in the MacroFiles/ directory and the other in the PreAnalysedMacroFiles/ directory. The first is a readable version, as opposed to the second which has been pre-analyzed with PreMac2E.

To use all these MUI macros in your E programs, you must run Mac2E on your source code file before compiling it:
Mac2E source.e destination.e PreAnalyzedMacroFiles/mui.e

1.32 OptiMUI2E

If you take a look at mui.e, you will see in the body of the macros defining new objects "TAG_IGNORE, 0", for example
"#define WindowObject Mui_NewObjectA('Window.mui', TAG_IGNORE, 0)"
This tag does, as it's name implies, absolutely nothing during execution. However, I was obliged to introduce them to keep the same usage syntax as in C. It's at this level that OptiMUI2E intervenes. It's job is to remove these useless "TAG_IGNORE, 0"'s from E source code. It's calling syntax is as follows:
OptiMUI2E e_source_file e_destination_file
where

- e_source_file designates the name of the source file (eventually with path) where there are "TAG_IGNORE, 0"'s to remove
- e_destination_file designates the name of the file (eventually with path) which will contain e_source_file with the "TAG_IGNORE, 0"'s suppressed.

The usage for OptiMUI2E is "FROM/A,TO/A".

Warning: OptiMUI2E sometimes removes carriage returns from your source code to respect line breaks on a comma, obligatory in E. Thus the file produced will not necessarily contain the same number of lines as the original file, which can cause possible problems with the line numbers reported by the compiler. It is therefore strongly advised not to use OptiMUI2E except for the final compilation when the finished program is tested. In any case, OptiMUI2E is absolutely not necessary to use MUI with Amiga E. It reduces the size of source code files and executables using the MUI macros, but only by a small amount.

1.33 Bugs

```
    Don't panic, none of the following points are bugs, but rather
limitations.

    * PreMac2E does not verify if the macro declarations are recursive, if
they are, PreMac2E will fail
    * PreMac2E and Mac2E do not verify if a macro is defined more than
once. In this case, Mac2E will use one of them, but the question is, which
one?!?!
    * The length of a macro name is limited to 255 characters.
    * The number of arguments is limited to 32.
    * The length of a macro body before pre-analysis is limited to 4Kb. If
this is not sufficient, you don't need a macro, you need a procedure!
    * The length of a macro body after pre-analysis is limited to 64Kb, and
that should be sufficient, should it not?
    * PreMac2E does not verify that the 4 preceding limitations are
respected.
```

1.34 History

```
Version 1.0 :  - 1st functional version (VERY VERY SLOW...)
Version 2.0 :  - modified version of v1.0 with lots of assembly
                   optimizations in the E source (10 times faster!)
               - addition of OptiMUI2E v1.0
               - 1st distributed version
Version 3.0 :  - addition of PreMac2E v1.0 to pre-analyze macro
                   files
               - use of an encoded hash-table (14 times faster!)
               - PreMac2E and Mac2E now give explicit error messages
               - verification of all memory allocations
               - a few minor bugs fixed
               - OptiMUI2E v1.1 works with 68000
               - mui.e is now commented
               - source for the function doMethod() supplied
               - source to all executables supplied
               - better documentation
               - update of mui.e according to MUI v2.0
Version 3.1 :  - a few minor bugs fixed
               - update of mui.e according to MUI v2.1
```

1.35 Future

```
    I'm awaiting (as you are) the next version of Amiga E, which shouldn't
take much longer according to Wouter... Of course, I'm also awaiting your
suggestions!
```

1.36 Distribution

This archive can be freely distributed, as long as no person gains any benefit from this distribution. No other type of sale can be made without the author's authorization.

This archive can be included in public domain program collections, as long as the above conditions are satisfied.

The archive must be distributed in it's entirety and must contain the following file structure:

```
Mac2E/Bin/Mac2E
Mac2E/Bin/OptiMUI2E
Mac2E/Bin/PreMac2E
Mac2E/Docs/Mac2E.guideD
Mac2E/Docs/Mac2E.guideD.info
Mac2E/Docs/Mac2E.guideE
Mac2E/Docs/Mac2E.guideE.info
Mac2E/Docs/Mac2E.guideF
Mac2E/Docs/Mac2E.guideF.info
Mac2E/MacroFiles/mui.e
Mac2E/MacroFiles/mui.e.info
Mac2E/Modules/libraries/mui.m
Mac2E/Modules/muimaster.m
Mac2E/PreAnalyzedMacroFiles/mui.e
Mac2E/Sources/doMethod.e
Mac2E/Sources/doMethod.e.info
Mac2E/Sources/Mac2E.e
Mac2E/Sources/Mac2E.e.info
Mac2E/Sources/OptiMUI2E.e
Mac2E/Sources/OptiMUI2E.e.info
Mac2E/Sources/PreMac2E.e
Mac2E/Sources/PreMac2E.e.info
Mac2E/Liesmich.zuerst
Mac2E/Liesmich.zuerst.info
Mac2E/LisezMoi.d_abord
Mac2E/LisezMoi.d_abord.info
Mac2E/ReadMe.first
Mac2E/ReadMe.first.info
Mac2E/ReadMe.mui
Mac2E.info
```

All of these files (except mui.m, muimaster.m, mui.e, ReadMe.mui and all the icons) are copyrighted by the author. None of them can be modified without my permission.

I cannot be held responsible for the use of this program and any damages that it may cause: use it at your own risk!

This program is distributed according to the Freeware concept, that is to say, you are not obligated to send me anything! However, I would be happy to receive something, from an Amiga 4000/40 to a simple postal card from your holidays, 20FF or a simple letter! (see The author)

1.37 The author

You can reach me by mail at:

- my student address, up to and including July 1994

Lionel Vintenat
appartement 21
11 rue François Oulié
31500 TOULOUSE
FRANCE

- my family address:

Lionel Vintenat
3 impasse Boileau
Lotissement Les Termes
87270 COUZEIX
FRANCE

Write to me at my student address until July 1994 as I'm there much more often than at my family address.

You can also reach me on the INTERNET. My e-mail address is vintenat@irit.fr. I would prefer that you contact me by e-mail rather than mail. I will reply to all questions that are sent to me by e-mail, but don't expect a reply by mail (I am very lazy when it comes to picking up a pen...).

1.38 Acknowledgments

A big thank you:
- to the Amiga for being the best personal computer
- to Wouter van Oortmerssen for his work in the field of compilers (try his FALSE, guaranteed surprise!) in general and for Amiga E in particular
- to Brian Mury for the English translation of the documentation
- to Marc Schröer for the German translation of the documentation
- to Xavier Billault for his help in the conception of this documentation
- to all those on the French Amiga mailing list who have helped me
- to all those who write public domain programs in general

Finally, thank you to all those who alert me to bugs or send me suggestions, or who send me corrections or translations of this document (see The author)

Happy E programming and...

NEVER FORGET, ONLY AMIGA MAKES IT POSSIBLE!