Mac2E

COLLABORATORS

	TITLE : Mac2E	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		March 28, 2025	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Mac2E

1.1 Mac2E

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

                                  Mac2E (v4.0)
                        Macro preprocessor for the E language
                          Archive of September 1994, the 2nd
                        © 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.

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

Better and better to define a macro...

Using several lines for the body
Comments in the body

1.14 Using several lines for the body

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 !
    is a macro which has as it's body
[1 space -> ][2 spaces ->  ][3 spaces ->   ] and that's all !
```

1.15 Comments in the body

```
    Starting from Mac2E v4.0, you can put comments inside the body of a macro.
These comments are introduced by the two characters "->" and end with the
carriage return at the end of the line (or eventually with the end of the
file). The principle is so the same as the new Amiga E v3.0 comments (besides
the idea comes from there !). All the comments are of course ignored by the
preprocessor.

    1st example :
#define MacroName MacroBody -> Here is a comment
    is a macro which has as it's body MacroBody .

    In the same way that for a macro which body isn't composed of comments
(see  Using several lines for the body ), you can extend over several lines
the body of macro which body is composed of comments.

    2ème exemple :
#define NomMacro FirstPieceOfBody  -> First comment  \
                 SecondPieceOfBody -> Second comment \
                 LastPieceOfBody   ->  Last comment
    is a macro which has as it's body
FirstPieceOfBody  SecondPieceOfBody LastPieceOfBody   .
```

1.16 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.

1.17 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)
```

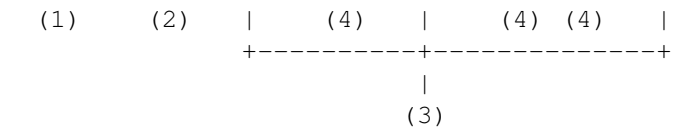
However, the preprocessor will not recognize it in the following instructions sequences:

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

1.18 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)
   |      |    |     |    |     |   |    |     |
```



where
(1) the name of the macro
(2) opening parenthesis immediately following the macro name, and which can be followed by any number of carriage returns (starting from Mac2E v4.0 only)
(3) 1 or more parameters (any combination of characters other than carriage returns)
(4) comma to separate each parameter, and which can be followed by any number of carriage returns (starting from Mac2E v4.0 only)
(5) closing parenthesis, which can be preceded by any number of carriage returns (starting from Mac2E v4.0 only)
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, except the carriage returns which follow the opening parenthesis, follow the commas and precede the closing parenthesis (starting from Mac2E v4.0 only).

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	

1.19 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.

1.20 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.

1.21 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.

1.22 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))).
```

1.23 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).
```

1.24 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.

Starting from Mac2E v4.0, the brackets "[" and "]" are processed in the same way as parenthesis. So you can give as macro parameter an E list. The coherency rules also applied to the brackets : for each opening bracket, a closing bracket must exists in the parameter.

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([1,(),'str'],character ",")	[1,(),'str']	character ","
toto(),4)	E R R O R	
toto(4,,)	E R R O R	

1.25 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 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 files separate from your source code, called the macro files.
After, you can use Mac2E to replace the macro calls in your source code files.

Macro files

1.26 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 outside the macro definitions. 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.

Starting Mac2E v4.0, it's also possible to put comments inside macro definitions, but these ones obey to a special syntax (see comments in the body).

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

1.27 Calling Mac2E

The calling Mac2E syntax is the following :
FROM/A,TO/A,WITH/M,PA=PREANALYZE/S,VER=VERBOSE/S,KS=KEEPSPACES/S,DEBUG/S.

In fact, the exact behaviour of Mac2E is determined by the PREANALYSE flag (shortcut PA). If this one is set, Mac2E is going to pre-analyze a macro file, and in the opposite case, it's going to behave like a preprocessor.

```
@{ " Pre-analyze mode  " Link Mode_préanalyse      }
@{ " Preprocessor mode " Link Mode_préprocesseur   }
@{ " VERBOSE           " Link VERBOSE              }
@{ " KEEPSPACES        " Link KEEPSPACES           }
@{ " DEBUG             " Link DEBUG                }
```

It's possible to interrupt Mac2E at any moment and whatever its current mode by pressing Ctrl-C.

1.28 Pre-analyzis mode

In this mode, Mac2E is going to pre-analyze the macro file precised in the FROM field and to save the result of the pre-analysis in the file precised in the TO field. The field WITH is so useless and so must be empty during the Mac2E call.

The interest of pre-analyzing a macro file is of course the speed-up. Indeed, during a pre-analysis, Mac2E mainly does 3 things :

- to process in the body of all the macros the calls from other macros

- to classify the macros in a hashed-coded table
- to locate precisely the position of all the arguments in the body of the macros

Thus, when Mac2E will perform in preprocessor mode, if it uses a pre-analyzed macro file instead of the original macro file, it will have the body of all the macros ready to be inserted in the source file without any extra analysis to do.

However this method presents two disadvantages :
- after each modification to a macro file, you must update the corresponding pre-analyzed macro file with a new pre-analysis
- a pre-analyzed macro file is used by Mac2E in preprocessor mode as is without any extra analysis : thus if you call Mac2E in preprocessor mode with two macro files which one is pre-analyzed, Mac2E won't check that this last one contains some macros defined in the othe one or that some macros of this last one depends on macros defined in the other one

In a general manner, it's very interesting to pre-analyze the big macro files of general using (i.e. which contents is steady). For the little macro files, it's up to you : on a fast Amiga, the speed-up isn't really impressive.

Normally, the pre-analyzed macro files are put inside the subdirectory PreAnalysedMacroFiles/ of the place where you put Amiga E.

1.29 Preprocessor mode

In this mode, Mac2E acts like a real preprocessor : it processes in the file of the FROM field all the macro calls and writes the result in the file of the TO field.

With the WITH field, you precise as many macro files as you want. Starting from Mac2E v4.0, these files can be either pre-analyzed ones or "raw" ones. Mac2E will automagically make the difference.

If a macro file of the WITH field is "raw", it will be added to the set of already known macros of Mac2E, and Mac2E will check that it doesn't contain any macros already defined in an other macro file loaded before. Then, the macros it contained will be pre-analyzed only when needed, thus this limits speed loss relatively to a pre-analyzed macro file. During such a pre-analysis "on the fly", the internal macro calls are searched for the very all macros known at the moment of the analysis of the file of the TO field. In other words, a "raw" macro file can depend on an other macro file (if this one is of course precised in the WITH field).

If a macro file of the WITH field is pre-analyzed, it will be added as is to the set of the already known macros without checking if it contains any macros already defined in an other macro file loaded before, or any macros which depends on macros defined in an other macro file precised or not in the WITH field. In other words, a pre-analyzed macro must depend on nothing.

THe VERBOSE, KEEPSPACES and DEBUG flags also apply to the "on the fly" pre-analysis that Mac2E may have to perform in preprocessor mode.

1.30 VERBOSE

VERBOSE forces Mac2E during a macro body pre-analysis to display the name of this macro before this pre-analysis.

See error messages for a using example of this flag.

VERBOSE can't be used with DEBUG.

1.31 KEEPSPACES

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

1.32 DEBUG

DEBUG forces Mac2E during a macro body pre-analysis to display a full description of this macro :
- name of the macro
- number of arguments
- body of the macro before the pre-analysis
- body of the macro after the pre-analysis

This flag is there above all for debugging purposes (so its name !) when it's interesting to see exactly how Mac2E interpret a macro definition.

DEBUG can't be used with VERBOSE.

1.33 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 Mac2E processes macro calls within the bodies of other macros during a pre-analysis. Indeed, this analysis is done out of any file, so without line number to indicate precisely an error.
So, in order to locate where an error signaled without line number takes place, you have to run again Mac2E with VERBOSE flag. This will force Mac2E to display all the names of the macros which it pre-analyzes the body. Thus, when it prompts an error, you just have to look at the macro names it has just pre-analyzed, starting from the most recent until the oldest. For each one (following the previous order), you check if it was it which contained the error until you find the faulty one. It is likely that the last one was the faulty one, but it isn't sure. Indeed, consider a macro wich the body contains a correct macro call and after an other macro call, but incorrect this time. During the pre-analysis of this body, the first call will make Mac2E pre-analyze the corresponding macro, then once done, the body

pre-analysis will go on until the incorrect macro call. There, an error will be signaled, but the faulty macro will be the next to last displayed by VERBOSE because the last one will have been the macro corresponding to the first correct macro call.

Exactly in the same way, the VERBOSE flag is very usefull to determine the macro dependency cycle when Mac2E detects such a cycle.

1.34 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 5 things:

- Mac2E
- mui.m
- muimaster.m
- mui.ma
- OptiMUI2E

All of "this interface" is based on MUI v2.2. 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
mui.m
muimaster.m
mui.ma
OptiMUI2E
```

1.35 mui.m

mui.m is, as it's name implies, a classical Amiga E module 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.

Starting from Mac2E v4.0, all the constant definitions were put in this file instead of remained as macros in mui.ma. However the syntax is remained the same, in particular the lowercase parts of these constants are still lowercase. So your sources needn't to be adapted with one exception to this rule : you may have to include this mui.m module in some of your modules (in the Amiga E v3.0 case) now, whereas before preprocessing these last ones with Mac2E and mui.ma was enough.

Starting from Mac2E v4.0, mui.m also contains the new MUI_TRUE constant which has one for value. In fact, this corresponds to the TRUE constant of C language which was used to write MUI. So you must use MUI_TRUE instead of TRUE (E language constant) with MUI.

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

1.36 muimaster.m

muimaster is, as it's name implies, a classical Amiga E module 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.

1.37 mui.ma

mui.ma is the key to the gateway to "this MUI-Amiga E interface" since it contains all the macros different from constants (starting from Mac2E v4.0) of the file mui.h, but adapted for the E language. The syntax of the mui.ma 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.ma 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 Mac2E.
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.ma

Among all the macros of mui.ma, two of them need more precisions : StringMUI() and set().
The String() macro was renamed to StringMUI() because String() already correponds to an E language function.
If you already used the set() macro with a Mac2E anterior to the 4.0 version, you probably noticed that it didn't always work with MUI. One solution was to write instead domethod(object , [MUIM_Set , attribute , new_value]). But this wasn't very smart ! Thanks to Jan Hendrik Schulz who discovered the problem origin, I could change the set() macro definition in order it works all the time. All I have said before also applies to the nnset() macro.

1.38 OptiMUI2E

If you take a look at mui.ma, 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.

1.39 Bugs

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

* Using pre-analyzed macro files asks for some precautions (see mode pre-analysis and mode preprocessor).
* The number of arguments for a macro is limited to 32.
* Mac2E does not verify that the preceding limitation is respected.

1.40 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
Version 4.0 :   - completely reprogrammed for Amiga E v3.0 with using of
                    optimization options, of hasing method provided with
                    Amiga E v3.0, of OO programming, of new very fast
                    algorithms for string handling and of a better (and above
                    all faster) processing of I/O
                - pre-analysis are about 50% faster and preprocessing is about
                    twice faster than with Mac2E v3.x
                - PreMac2E was merged to Mac2E which is now the only executable
                    file (see  calling Mac2E  for its new calling syntax)
                - it's possible to give Mac2E macro files pre-analyzed or
                    not, and in this last case, it will perform pre-analysis
                    when needed "on the fly" (see  preprocessor mode )
                - it's possible to put comments inside the body of macros, even
                    when this body extends over several lines (see
                     comments in the body )
                - it's possible to extend a macro call over several lines
                    (see  handling of argument passing )
                - it's possible to give an E list as macro parameter (see
                     special characters )
                - Mac2E can be interrupted at any moment by pressing Ctrl-C
                - addition of a "DEBUG" mode to Mac2E (see  DEBUG )
                - Mac2E checks the double declarations of macros
                - Mac2E detects the macro dependency cycles
                - Mac2E hasn't any more limitation about macro name size and
                    macro body size
                - Mac2E reports accurately write errors
                - a few minor bugs fixed
                - Mac2E is now GiftWare ! (see  distribution )
                - optiMUI2E v1.2 was rewritten cleaner for Amiga E v3.0
                - update of mui.ma according to MUI v2.2
                - the constant definitions were moved to mui.m (see
                     mui.m )
                - new MUI_TRUE constant introduced (see  mui.m )
                - two macros interesting to look at (see  mui.ma )
```

1.41 Future

```
    For Mac2E v3.0, I wrote :
<<
    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!
>>
    So now, I'm only waiting for your suggestions :-).
```

1.42 Distribution

All the files of this distribution stay under the copyright of the author (Lionel Vintenat). You are allowed to modify them only for your STRICTLY PERSONAL usage.

The only exceptions are the files "mui.m", "muimaster.m", "mui.ma", "Readme.mui" and the icons. You can of course use the sources provided in this archive for your personal programs : they are here for that !

This archive can be freely distributed by any thinkable ways (ftp server, BBS, public domain collection, etc), as long as the two following conditions are respected :

1) No person gains ANY benefit from this distribution. In particular, if Mac2E is spread on a floppy disk, this one can't be sold for more than 4$ US (or equivalent), and if it is spread on a CDROM, this one can't be sold for more than 30$ US (or equivalent). No other type of sale (with benefit) can be made IN ANY CASES without the author's authorization. The only exceptions are the Fred Fish's and aminet CDROMs, who they (and only them !) can include Mac2E in their collections without asking me the permission first. In particular, this excludes DEFINITIVELY to France Festival Distribution the right to distribute Mac2E (I insist HEAVILY on this point...). But maybe this won't annoy too much Mr Serge Hammouche, who doesn't hesitate to call (openly on some French nets) French PD programers incapable...

2) This archive is distributed IN IT'S ENTIRETY, and without MODIFICATIONS compared to the original version on aminet. This means in particular that if you make a translation of the documentation or of the catalog in a new language, or if you fix some bugs and re-compile the executable, you MUST send them to me in order I EVENTUALLY (pretty sure in fact for translations) redistribute a new version of this program. The file structure this archive must have is given in the "ReadMe.first" file of this distribution.

Any distribution of Mac2E which doesn't respect the two previous conditions without my authorization is ILLEGAL.

All the begining of this paragraph may seem very strict, even close to the paranoia, but knowing the dubious practices of people like Serge Hammouche who sells for HUDGE prices (under translation excuse) some freely distribuable softwares without even informing the authors, I think it's necessary to protect my rights. I make programs for my pleasure without any claim to earn money, and I'm happy if they may help other persons, but that some people make money from my work : NO ! The previous limitations doesn't aim in any way serious people like Fred Fish, the Montréal Amiga Club, the aminet system operators, or all the clubs of fascinated people, who, them, really support the Amiga domain public. They only aim the organizations with discutable pratices like France Festival Distribution.

Moreover, I cannot be held responsible for the use of this program and any damages that it may cause directly or not. I want this to be clear : YOU USE IT AT YOUR OWN RISK !

However, I think I tested and I made people test it enough to say that it doesn't contain any serious bug.

Finally, this program is distributed under giftware concept. In other words, you must send me a gift if you use Mac2E ! :)

Indeed, I make programs for my pleasure and by need. My goal is certainly not to earn money with. However to distribute a program ask for some extra work (documentations, installer script, etc), so I'd like to receive a feedback from those who will use Mac2E. In fact, any sign of life will be VERY appreciated, even a simple e-mail or a postcard. What I am the more interested in is contact with other persons. But to help you to choose my gift, here are some suggestions :) :

- a (free !) registration to a shareware program
- one of your realizations (program, module, animation, picture, etc) if it isn't easy to retrieve it from aminet
- some sourves
- some E, C or assembly sources which are closed to the programation of the system, or even False or BrainFuck sources (they can be closed of anything, provided they work ! :))
- some money, hummmm, why not ? :)
- your old 1.3 RKM (or better 2.0)

I insist on the fact that it's very frustrating to make the effort to put his program in the freely distribuable domain wihtout never receiving any feedback, just guessing that some people use your program ! So please support the giftware concept, everybody break even : the author is happy to receive a feedback, and it costs almost nothing to the users.

1.43 The author

You can reach me by snail-mail at my family address :

Lionel Vintenat
3 impasse Boileau
Lotissement Les Termes
87270 COUZEIX
FRANCE

You can also reach me on the internet. My e-mail address is vintenat@irit.fr. This one will probably be reliable until September 1994 included, but I won't have access to it cause of holydays/military service until this September month. So don't rely too much on a quick reply if you use this address ! If you really want to reach me until there, the best is to directly write to me to my family address (don't forget to do it with a nice postcard from you country 8)).

The best solution is for my point of view to wait for new signs of life from me on the net starting September 1994, because I will have at this moment a new internet address reliable for all the new school year.

1.44 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 : if you see strange sentences in this documentation, it's not his fault, it probably comes from my personal updates :-)
- to Marc Schröer for the German translation of the documentation
- to all those who have sent to me some bug reports and some encouragements, and especially among them Jan Hendrik Schulz who have gave me a lot of very good ideas for the Amiga E-MUI gateaway (above all the solution to the set() macro problem)
- 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!

1.45 Index