
RSCV_Compiler

A Tool To Update The *RSC_Vviewer* Application.

Version 2.3 (09/22/1991)

Written by François Menneteau

RSCV_Compiler copyright © 1991 by F. Menneteau
All Rights World Wide are Reserved.

Page — 2 —
Written in *THINK's LightspeedC*™
Portions © 1986, THINK Technologies, Incorporated.

RSCV_Compiler Documentation written and copyright © 1991 by F. Menneteau

INTRODUCTION

As the Macintosh family increases regularly, new systems and new ROMs are regularly provided by Apple with the new machines. Consequently new trap-calls and new low memory global variables are created.

The purpose of the *RSCV_compiler* application is to provide an elegant mean to update the *RSC_Viewer* application. Indeed, it is easier to update a plain text file than a "resource-form" of that file.

To sum up, RSCV_Compiler allows the user to update the ‘**GVAR**’ (global variables description) and ‘**TRAP**’ (trap description) resources of the *RSC_Viewer* application without using a Resource Editor.

THE FILES

So to update *RSC_Viewer* two files are provided. They contain a clear-editable description of the traps and global variables that are currently recognized. Their names are: **GlobalVariables**, and **ListOfTraps**. To modify them, see the corresponding chapters.

USER’S GUIDE

Launch the *RSCV_Compiler* application, select one of the two files, and when the application presents the second dialog box, search for the *RSC_Viewer*¹ application. When the application asks you if you want to replace *RSC_Viewer* answer **yes**. Do this operation twice, if you have updated the two files. That’s all.

Note that you can put the ‘**GVAR**’ and ‘**TRAP**’ resources in other applications than *RSC_Viewer*. But remember the Copyright.

¹ The name the application proposes is located in the ‘STR#’ resource ID 128 (first item). If you have rename *RSC_Viewer*, you can edit this resource and change the default name. Currently it is *RSC_Viewer 6.2*.

GLOBALVARIABLES FILE

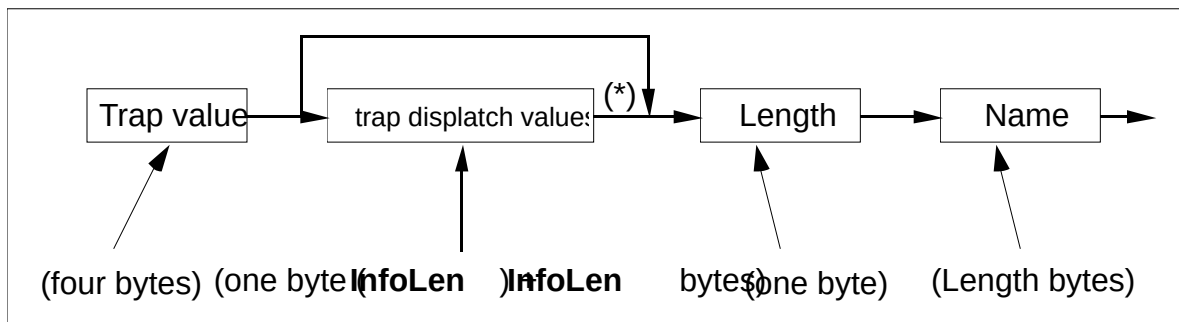
This file contains the description of the low memory global variables used by the Macintosh Systems.

To add or update a global variable, simply edit this file, find the place where to store it, and write it. You can change everything in that file (names, location of the addresses, etc) but two things:

- The word ‘**GVAR**’ at the beginning of the first line.
- And the format **\$xxx:<length>** where length (an hexadecimal value) is the size of the variable (otherwise *RSC_Viewer* won’t be able to show you their content) .

LISTOFTRAPS FILE

The format for that file is totally fixed. There must be the word ‘**TRAP**’ at the beginning of the first line, and the syntax of the other lines (trap description) is the followings:



(*) These are the values that are pushed onto the stack, or put in register D0 or A0 and allow the system to choose the right element of a dispatch trap.

As some traps can have some bits set (typically bit 9 and 10) and as those bits have different meanings (it depends on the trap actions, see Inside Macintosh books), it has been decided to use different trap values to differentiate them.

Further more, some traps act as dispatch traps. So, in order to distinguish them from the standard ones, new values have been introduced.

In addition, as there are different means to build a element of a dispatch trap (by pushing a long value into the stack or by putting a word or long value in register D0 or A0), we need to store an information that allow the disassembler and the assembler of the *RSC_Viewer* application to work properly.

To sum up, as a Macintosh trap is a word value which range is [A000...ABFF], *RSC_Viewer* uses the

high order byte of the trap value (i.e. the A digit) to handle the different trap types, and the high order 4 bits of the **infoLen** byte to handle the different dispatch selectors.

Trap types selector (high order byte of a trap value).

```

+--+--+--+--+
| | | |
+--+--+--+--+
|ST|CM|SC|DI|

```

ST = 1, DI = 0	this is a standard trap.
ST = 0, DI = 1	this is a dispatch trap.
ST = 0, DI = 0	this is an element of a dispatch trap.
ST = 1, DI = 1, CM = SC	reserved

Now we must handle the ASYNC, SYS, etc bits.

CM = 0, SC = 0	for normal trap or those with the ASYNC/IMMED bits.
CM = 1, SC = 0	for trap with the CASE/MARK bits.
CM = 0, SC = 1	for trap with the SYS/CLEAR bits.
CM = 1, SC = 1	for trap with the ASYNC/HFS bits.

To sum up:

Trap type:	the selector is:
standard trap and those with the ASYNC/IMMED bits	0x8
standard trap with the SYS/CLEAR bits	0xA
standard trap with the CASE/MARK bits	0xC
standard trap with the ASYNC/HFS bits	0xE
dispatch trap and those with the ASYNC/IMMED bits	0x1
dispatch trap with the SYS/CLEAR bits	0x3
dispatch trap with the CASE/MARK bits	0x5
dispatch trap with the ASYNC/HFS bits	0x7
disp elem trap and those with the ASYNC/IMMED bits	0x0
disp elem trap with the SYS/CLEAR bits	0x2
disp elem trap with the CASE/MARK bits	0x4
disp elem trap with the ASYNC/HFS bits	0x6
reserved (special use for PMgrOp)	0x9
reserved (do not use it)	0xF

Dispatch selectors.

At present we distinguish five means to build a dispatch trap:

MOVE.W	#value, A0	(307C xxxx)	
MOVE.W	#value, D0	(303C xxxx)	
MOVE.W	#value, -(SP)	(3F3C xxxx)	
CLR.W	-(SP)	(4267)	(used when value is null)
MOVE.L	#value, D0	(203C xxxx xxxx)	
MOVEQ	#value, D0	(70xx)	(used when value is <255)

MOVE.L #value, -(SP) (2F3C xxxx xxxx)

They are coded in the high order 4 bits of the **infoLen** byte. The low order 4 bits are used to store the size of the trap dispatch value.

infoLen (1 byte): +--+--+--+--+--+--+--+--+--+
 | | | | | | | | | |
 +--+--+--+--+--+--+--+--+--+
 |WL|register|| real size |

WL = 0 the dispatch value is coded as a WORD value.
 WL = 1 the dispatch value is coded as a LONG value.
 register = 000 If you don't know how the selector is build.
 register = 001 If the dispatch value is stored in register A0.
 register = 010 If the dispatch value is stored in register D0.
 register = 011 If the dispatch value is pushed onto the stack.

To sum up:

If the mean is:	the selector is:
A0 [W]	0x1
D0 [W]	0x2
-(SP) [W]	0x3
D0 [L]	0xA
-(SP) [L]	0xB

We give here, for each dispatch trap currently recognized, the corresponding dispatch selector. If you find a mistake, please tells me and I shall correct it.

Trap name	Value	Means
AliasDispatch	0xA826	0xA
CommToolboxDispatch	0xA08B	0xA
DebugUtil	0xA08D	0xA
FontDispatch	0xA854	0xA
HFSDispatch	0xA060	0xA
HighLevelFSDispatch	0xAA52	0x2
InternalWait	0xA07F	0x1
IdleState	0xA485	0xA
MemoryDispatch	0xA05C	0xA
MIDIDispatch	0xA800	0x3
OSDispatch	0xA88F	0x3
Pack0	0xA9E7	—
Pack1	0xA9E8	0x3
Pack2	0xA9E9	0x3
Pack3	0xA9EA	0x3
Pack4	0xA9EB	—
Pack5	0xA9EC	—
Pack6	0xA9ED	0x3
Pack7	0xA9EE	0x3
Pack8	0xA816	0x2
Pack9	0xA82B	0x2
Pack10	0xA82C	—
Pack11	0xA82D	0x2

Pack12	0xA82E	0x3
Pack13	0xA82F	0x2
Pack14	0xA830	0x2
Pack15	0xA831	0x2
PaletteDispatch	0xAAA2	0x2, A
PPC	0xA0DD	0xA
QDExtensions	0xAB1D	0xA
ResourceDispatch	0xA822	0xA
ScripUtil	0xA8B5	0xB
SCSIDispatch	0xA815	0x3
SerialPower	0xA685	0xA
SlotManager	0xA06E	0xA
Shutdown	0xA895	0x3
SoundDispatch	0x8000	0xA
SysError	0xA9C9	0x2
TEDispatch	0xA83D	0x3

Examples.

- 8xxx standart trap (ex: **80230CDisposHandle**) [A023]
- 1xxx dispatch trap (ex: **19EA15Pack0**) [A9EA, SFpackage]
- 0xxx element of a dispatch trap (ex: **09EA\x(3101)09SFPutFile**) [Pack0 dispatch]
(or: **0B1D\x(A400040016)0BPixMap32Bit**)
[trap dispatch values for AB1D: QDExtensions]
- Axxx trap with SYS/CLEAR (ex: **A04008ResrvMem**)[A040]
- Cxxx trap with CASE/MARK (ex: **C03C0BEqualString**)[A03C]
- Exxx trap with ASYNC/HFS (ex: **E20004Open**)[A000]
- 7xxx dispatch trap with ... (ex: **72600AFSDispatch**)[A060]
- 6xxx elem ... with ASYNC/... (ex: **6060\x(A101)06OpenWD**)

Notes:

- **GetTrapAddress** and **SetTrapAddress** are also treated internally, so they are considered as standard traps.
- The maximum length for the Trap dispatch value is **16 bytes** (where 16 is coded as zero), it seems to be widely sufficient...