## General Information

### DEFINITION OF THE CURRENT LINE

Most functions in ReSource operate only on the current line.  Because there is no real "cursor", it is necessary to define the "current line":

The "current line" is the first displayed disassembly line.  This may be displayed in one of five data types, "code", "bytes", "words", "longwords" or "ascii".

The "cursor position" refers to the first byte of the current line

When scrolling, ReSource moves one "disassembly line" at a time (minimum). This consists of one of the above types of lines, and possibly other text lines that are attached to the line of disassembly.  Consider the following:

```
        SECTION Lister000000,CODE
ProgStart
; This routine converts an ascii string to binary
; A0 -> null terminated string
; returns number in D0.L

AscToBin           MOVEM.L   D1-D6/A0-A6,-(SP)     ; save some registers
1$                 EQU       *-2
```

The above example consists of the following lines (in order):

1. A section statement.

2. ReSources' special program start label.  This may be used by code lines that reference the segment list.

3. Three full-line comments.

4. A user-defined blank line.  This can be done by creating a full-line comment, and specifying "; " as the string to use.

5. A code line, using both a label, and an end-of-line comment.

6. A hidden label.  These are quite rare in most programs.  Some "C" compilers use them in case statements.

When you use the "CURSOR/Relative/Next line" command, ReSource will move to the next disassembly line, and in the above example, this will be equivalent to scrolling nine "text" lines.  ReSource is *not* a text editor, or a word processor, even though the user interface is similar in some respects. Understanding the makeup of a disassembly line will help in making the best use of ReSource.

## General Information

### DATA TYPES

ReSource currently uses nine data types.  These are CODE, ASCII, BYTES, WORDS, LONGWORDS, SINGLE, DOUBLE, EXTENDED, and PACKED.  Examples follow:

Example CODE lines:

```
lbC00000C           MOVEM.L   D0/A0,-(SP)
                    MOVEA.L   (4),A6
                    LEA       (doslibrary.MSG,PC),A1
                    MOVEQ     #$27,D0
                    JSR       (_LVOOpenLibrary,A6)
```

Example ASCII lines:

```
Unknownoption.MSG   db        'Unknown option %c ignored',$A,0
snotfound.MSG       db        '%s not found',$A,0
                    db        'Cannot examine directory',$A,0
```

Example BYTE lines:

```
                    db        4
                    db        3,$23,14
                    dcb.b     14,0
                    dx.b      $1A0
                    ds.b      $1A0
```

Example WORD lines:

```
                    dw        4
                    dw        3,$23,14
                    dcb.w     14,0
                    dx.w      $D0
                    ds.w      $D0
```

Example LONGWORD lines:

```
                    dl        4
                    dl        3,$23,14
                    dcb.l     14,0
                    dx.l      $68
                    ds.l      $68
```

Example SINGLE, DOUBLE, EXTENDED and PACKED lines:

```
                    dc.s      2.0
                    dc.d      1.23E25
                    dc.d      200.0,-118.64,-0.0
                    dcb.d     7,0
                    dc.x      3.14159
                    dc.p      1.414213562
                    ds.s      4
                    ds.d      3
                    dx.x      2
                    dx.p      1
```

Double, Extended and Packed data types must be displayed in their native format--the numeric base may not be changed.

ReSource will try to automatically assign floating point data types during a code scan.  However, because of the uncertanties involved, some data references will need to be resolved by hand.

See also:

## General Information

### DATA TYPES

"numeric types"

### LINES, SCROLLING BACKWARDS...

For every byte in a file, ReSource maintains an attribute bit (#31) that specifies whether this byte is the start of a line or not. This information is required when scrolling backwards, in avoiding a jerky display. Under certain conditions, the maintainance of this bit can result in short delays.

For example, if you load a very large file as a binary image, and immediately set the data type at the start of the file say, to "ascii" or "code", and then move the cursor directly to the end of the file, ReSource is forced to scan the entire file, to properly determine which actual byte starts the last line in the file. When a scan of more than 1K is required, the wait pointer will appear, to let you know that ReSource is temporarily busy.

## General Information

### USING DYNAMIC STRING INDIRECTION

When supplying a string (to a string requester) that will become a label, symbol, or comment, you may use indirection to include in that string, some other predefined label, symbol, or comment. Indirection means a string is not used literally. Rather, it somewhat resembles a macro in that it "points" to other information, and is "expanded" when later encountered. Indirection must start with an "escape" character (just hit the "escape" key), and must be followed by a decimal digit between zero and three, inclusive. This is how you specify which type of string that you require:

```
0 = Label
1 = Symbol
2 = Full-line comment (the first one attached to a line only)
3 = End-of-line comment
```

This should be immediately followed by a hex, decimal, or binary number, specifying the offset in the current file to which the required string is attached. Another escape character is required immediately following the number, as a delimiter. You are then free to either supply more literal text, or you may use indirection several times within a single string. Consider the following program fragment:

```
lbW000140           DW        6
                    DW        8
                    DW        12
lbC000146           RTS

lbC000148           MOVEQ     #0,D0
                    RTS

lbC00014C           MOVEQ     #1,D0
                    RTS
```

The word values at label "lbW000140" correspond to the distance between "lbC000146" and "lbW000140" on the first line (6), the distance between "lbC000148" and "lbW000140" on the second line (8), and the distance between "lbC00014C" and "lbW000140" on the third line (12). This is what is commonly referred to as a "jump table". In order to make the program assemble properly, the values 6, 8, and 12 should be changed to symbols specifying their true meaning, rather than just the value that they were when the program was last assembled/compiled:

```
lbW000140           DW        lbC000146-lbW000140
                    DW        lbC000148-lbW000140
                    DW        lbC00014C-lbW000140
lbC000146           RTS

lbC000148           MOVEQ     #0,D0
                    RTS

lbC00014C           MOVEQ     #1,D0
                    RTS
```

The above example addresses the immediate problem of representing the offsets symbolically. However, if you later change the label "lbC000146" to "NullSub", for instance, the symbol "lbC000146-lbW000140" will cause an assembly error because the assembler will not be able to find the label "lbC000146". By using dynamic indirection, whenever the symbol at label "lbW000140" is displayed, ReSource will use whatever label is defined at that moment. (Because the escape

## General Information

**USING DYNAMIC STRING INDIRECTION**

character is not normally displayable, the following example will use a carat, "^", to represent the escape character.) For the line:

```
lbW000140           DW       lbC000146-lbW000140
```

instead of creating the symbol "lbC000146-lbW000140", you would instead specify "^0$146^-^0$140^" when asked by the "LABELS/Create single/Symbol" function.

Dynamic indirection is also used internally in ReSource. For example, the "SPECIAL FUNCTIONS/Convert specific EA's/" functions now use dynamic indirection when creating symbols.

Functions that access labels, symbols, and comments will normally support dynamic indirection in strings, unless the option "OPTIONS/Show/Symbols/" is OFF. In this case, the literal string will be used. Dynamic indirection is relatively safe to use with symbols. It is possible to use with labels and comments, however some problems may arise if care is not taken. Duplicate labels may easily be created if dynamic indirection is used in the specification for a label. It should be quite safe in any comments, however. Symbols containing dynamic indirection will not be stored in the EQUate or xref tables. It is expected that most strings using indirection will be symbols, that refer to labels, and therefore will not require to be EQUated or xref'd.

## General Information

### RELOC32

In Amiga loadable files (executables, fonts, device drivers, libraries etc.), there may be (and generally is) absolute pointers, to locations either within the file, or at least at some location at a set offset from the start of one of the hunks. When the program is executed (or used, in the case of fonts), these absolute pointers must contain the correct 32-bit memory address.

Because the Amiga is a multitasking system, programs must be able to be loaded to any memory address that happens to be available at the time of loading. So, the system used to make sure that the absolute 32-bit pointers will contain the correct addresses at run time is termed "reloc32" (relocation of 32-bit addresses).

Being able to see where relocation has been performed is extremely useful when disassembling a program. Particularly, in determining data types, whenever you see reloc32, you know for *certain* that the four bytes within the reloc32 area are not the start of a code line, and they are not ascii. In fact, you will find that there is only two uses of reloc32 in a program:

1. The first byte of the reloc32 is the start of a longword data item.

2. At an even number of bytes, generally 2,4 or 6 before the first byte of the reloc32, is the start of a line of code that uses absolute addressing.

If it were not for the relocation information, disassembling programs would be many times harder. ReSource makes much use of this information internally.

The following example lines will each produce a reloc32 area:

```
                    dl       1$
1$                  dl       ProgStart-4
                    dl       *-8
```

The following example lines will NOT produce reloc32 areas:

```
                    dl       1$-*          ;an offset, NOT an address
1$                  dl       0
                    dl       *-1$          ;another offset - NOT reloc32
                    dl       14*88
                    dl       $BFEC01
                    dl       $FC01AE
```

The last two lines above *are* absolute pointers, but the addresses are not relative to a location in this program, thus they will not be reloc32.

## General Information

### EFFECTIVE ADDRESS CONVERSIONS

These functions were specifically designed to be used when disassembling 'C' programs, in which an address register is frequently used as a base register for accessing data, throughout all or most of the program.  Let's use an example program here:

```
                    SECTION test000000,CODE
                    LEA       lbB00011E,A4    ; scanning has already been done
                    LEA       (0,A4),A1
                    MOVE.L    ($000C,A4),D0

                    ...       ; rest of code

                    SECTION test00011E,DATA
lbB00011E           db        'dos.library',0
                    dl        $00010001
                    END
```

In the above example program, the A4 register points to the start of the data segment.  There are three ways to tell ReSource where the A4 register points; in the above example, the "This operand" function could be used with the cursor at start of file, or the "This address" function could be used with the cursor at offset $00011E, or the "Specify" function could be used, supplying a parameter of "$11E".  Basically, with this function you are telling ReSource where the A4 register can be assumed to be pointing, relative to the start of the program.  After you do this, any effective address that involves a word offset to the A4 register, will be shown symbolically.  If you have selected "Convert (xx,An) EA's/Absolute", the effective addresses will be shown as absolute addresses.  Thus, the example program above will appear as:

```
                    SECTION test000000,CODE
                    LEA       lbB00011E,A4
                    LEA       lbB00011E,A1
                    MOVE.L    lbL00012A,D0

                    ...                   ; rest of code

                    SECTION test00011E,DATA
lbB00011E           db        'dos.library',0
lbL00012A           dl        $00010001   ; This label will be created after
                    END                   ; left-Amiga mouse-button is used.
```

On the other hand, if you had selected "Convert  (xx,An)  EA's/Relative",  the effective  addresses would be shown as symbolic word offsets, and our example program would look like this:

```
                    SECTION test000000,CODE
                    LEA       DT,A4               ; Note the new base label "DT"
                    LEA       (lbB00011E-DT,A4),A1
                    MOVE.L    (lbL00012A-DT,A4),D0

                    ...       ; rest of code

                    SECTION test00011E,DATA
DT
lbB00011E           db        'dos.library',0
lbL00012A           dl        $00010001      ; This label will be created after
                    END                      ; left-Amiga mouse-button is used.
```

## General Information

### EFFECTIVE ADDRESS CONVERSIONS

The advantage here is that labels can automatically be created where they could not before. With a little more work, our code could look like this:

```
                    SECTION test000000,CODE
                    LEA      DT,A4
                    LEA      (DOSName-DT,A4),A1
                    MOVE.L   (Mem_Parms-DT,A4),D0

                    ...                           ; rest of code

                    SECTION test00011E,DATA
DT
DOSName             db       'dos.library',0
Mem_Parms           dl       MEMF_CLEAR!MEMF_PUBLIC
                    END
```

If this conversion process is not used, it is likely that the program will not successfully re-assemble, as different assemblers will assemble the same source code into different length opcodes. For example, the Metacomco assembler normally does virtually no optimizing, and so the resulting program is often larger than the original, even if the source code has not been modified. For example, take the line:

```
                    MOVEA.L  (4),A6
```

If absolute long addressing is used, the instruction above will be 6 bytes long, whereas if absolute short addressing is used (as will be the case when you assemble with Macro68), it will be only 4 bytes long. Where you wish to do EA conversions in only a portion of a program, you can set the lower and upper limits. See:

SPECIAL FUNCTIONS/Convert (xx,An) EA's/Set lower limit
SPECIAL FUNCTIONS/Convert (xx,An) EA's/Set upper limit

You can further define conversions to only data references. See:

SPECIAL FUNCTIONS/Convert (xx,An) EA's/Data refs only
SPECIAL FUNCTIONS/Convert (xx,An) EA's/All references

Changing to absolute EA's will increase the size of the resulting program, unless you convert back to relative addressing when assembling (assuming that you will be using Macro68). Please note that this function may be used with ANY register as the base register.

See "SPECIAL FUNCTIONS/Specify Base Register".

If the label, "DT", is not defined by the user, ReSource will define it for you. It will be attached to the last DATA hunk in the file, or if no DATA hunks exists, the last BSS hunk, or failing that, the last CODE hunk. Be aware that a few programs define "DT" outside of, but relative to one of the hunks. In most cases, this does not present a problem. However, in a program with more than one hunk, when "DT" is defined as being at some negative offset from the start of any hunk other than the first, there is a problem. To ReSource, this will appear that the reference is relative to the previous hunk. When this situation arises, you must manually define the label "DT", equating it relative to the correct hunk:

```
SECTION             First,CODE
```

## General Information

### EFFECTIVE ADDRESS CONVERSIONS

```
... (first hunk contents)

SECTION             Second,DATA

DT                  equ       *-120
```

To summarize:

EA conversions are enabled with the functions:
SPECIAL FUNCTIONS/Convert (xx,An) EA's/This operand
SPECIAL FUNCTIONS/Convert (xx,An) EA's/This address
SPECIAL FUNCTIONS/Convert (xx,An) EA's/Specify

EA conversions are disabled with the function:
SPECIAL FUNCTIONS/Convert (xx,An) EA's/Disable

The address register to be used can be specified with the functions:
SPECIAL FUNCTIONS/Specify Base Register/A0
SPECIAL FUNCTIONS/Specify Base Register/A1
SPECIAL FUNCTIONS/Specify Base Register/A2
SPECIAL FUNCTIONS/Specify Base Register/A3
SPECIAL FUNCTIONS/Specify Base Register/A4
SPECIAL FUNCTIONS/Specify Base Register/A5
SPECIAL FUNCTIONS/Specify Base Register/A6
SPECIAL FUNCTIONS/Specify Base Register/A7

The upper and lower limits of the conversion can be set using:
SPECIAL FUNCTIONS/Convert (xx,An) EA's/Set lower limit
SPECIAL FUNCTIONS/Convert (xx,An) EA's/Set upper limit

Conversion can be further limited to data references only using:
SPECIAL FUNCTIONS/Convert (xx,An) EA's/Data refs only

Conversion can be enabled for all reference types using:
SPECIAL FUNCTIONS/Convert (xx,An) EA's/All references

## General Information

### MACROS

The macro facilities available in ReSource are quite extensive. There are currently a maximum of 57 macros available on line at one time. These are divided into 3 banks of 19 macros in the menus.

The 19th macro is special, in that it is executed when you first run ReSource, after the first file is loaded. For most people, this macro will often contain functions to set the various options to your personal taste, but may contain anything you care to put into it. The "PROJECT/Save Configuration" function *creates* macro #19, but you can create it too.

When first creating a macro, you will be asked to supply a name for it. This name will be shown in the MACRO menu immediately after, and if you save macros to a file, the names of the macros are also saved.

The functions:
```
"MACROS 1/Execute/     - Macros 1 -       "
"MACROS 2/Execute/     - Macros 2 -       "
"MACROS 3/Execute/     - Macros 3 -       "
```

are used to give each macros bank a name. These names are also saved when you save macros, and when you load the macro file later, these names will immediately appear in the menus.

The size of a macro is limited only by how much memory you have. To create a macro, select one of the sub-items in:

MACROS 1/Create/
MACROS 2/Create/
MACROS 3/Create/

If the current name for the macro is "-empty-", you will be asked to supply a new name for the macro. If you create a macro of zero length, the name for that macro will revert to back to "-empty-". To execute the macro, simply select the appropriate sub-item in:

MACROS 1/Execute/
MACROS 2/Execute/
MACROS 3/Execute/

You have control over the speed at which macros execute. The following chart indicates whether there will be a delay, and for how long, and whether the display will be refreshed between executing functions within the macro.

| Speed Selected | Delay | Screen Refresh | Titlebar refresh |
|---|---|---|---|
| Fastest | No | No | No |
| Very Fast | No | No | Yes |
| Fast | No | Yes | Yes |
| Slow | .1 sec | Yes | Yes |
| Very Slow | .5 sec | Yes | Yes |

To have complete control over execution of the macro, select "MACROS 1/Execution speed/Wait on mouse".

With this selected, you must press and release the left mouse button for each and every function within the macro to be executed. If you have selected "DISPLAY/Titlebar info/Function names" also, the titlebar will display the name of the function to be executed next. Combined with the "ShowMacros"

## General Information

**MACROS**

utility, this is an excellent way of finding bugs in a macro that you (or someone else) has created. Macros may be nested within macros, provided that the nesting depth does not exceed (approx.) 30.

While you are creating a macro, you might find that you have to execute some functions to continue the macro definition, but you don't want them included in the macro itself. When this situation arises, select "MACROS 1/Suspend learn/suspend", select the various functions that need to be done, then select "MACROS 1/Suspend learn/Normal" to continue the macro definition.

There are many functions that will make a macro fail, when it is executed. For example, using a cursor movement function that would place the cursor outside of the current file, will cause a macro 'fail'. A failed search will also cause a macro 'fail'. When executing a macro, if ReSource detects a macro 'fail', it searches forward in the macro definition for a "Conditional end" directive. If none is found, all macro processing aborts immediately. If one IS found, macro processing continues normally from that point in the macro definition. If, while searching for a "End conditional" directive, a "Start conditional" directive is found, the next "End conditional" is skipped. Thus, conditional macro processing may be nested.

There are five macro labels available. These are placed into the macro definition, and you can insert "goto previous macro label" and "goto next macro label" directives into the macro definition, which when found, will start a search either forward or backward, for the appropriate macro label. When found, macro processing will proceed normally from that point forward. Thus, you can loop a macro. If a search is made backwards for a macro label that is non-existent, macro processing will continue from the start of the macro definition. If a search is made forward for a macro label that is non-existent, the macro will exit, possibly to another macro that called this one. For example, the following macro will continuously scroll forward to the end of the file, then scroll backwards, one line at a time, to the start of the file, then forward to the end of the file again, etc., indefinitely, stopping only when you press rightAmiga-A (abort).

```
MACROS 1/Set macro label/#1
CURSOR/Relative/Next line
MACROS 1/Previous macro label/#1
MACROS 1/Directives/End conditional
MACROS 1/Set macro label/#2
CURSOR/Relative/Previous line
MACROS 1/Previous macro label/#2
MACROS 1/Directives/End conditional
MACROS 1/Previous macro label/#1
```

Notice that the directive "Start conditional" was not required in the above example, as conditional sections were NOT nested.

When you are creating a macro, and you are asked for a string (even the name of a file to load), if you select "Store" or press return after supplying a string, the string will be stored in the macro definition, and will be used when the macro is later executed, unless at the time that you execute the macro, "MACROS 1/Interactive/On" has been selected. In this case, you will be prompted for any strings that are requested during the execution of the macro. Normally, this will not be required, but it does give you more control over an executing macro, especially if it was created by someone other than yourself.

## General Information

**MACROS**

If, when creating a macro, you are asked for a string, and you want to force the user to input a string during the execution of the macro, you should select the "Use" gadget in the string requester. In this case, any string that you typed into the requester will still be used while creating the macro, however when the macro is executed, the user is forced to input a new string each time, even though 'Interactive' may be set to "OFF".

Instead of actually supplying a string literal to a string requester, you may instead use buffer indirection, to force the string to get copied from either the accumulator, or from one of the buffers A-M.

For example, if you wanted to create a comment, using the next encountered symbol, you could use the following macro:

CURSOR/Relative/Next symbol
CURSOR/Relative/Next byte
CURSOR/Relative/Previous line
STRINGS/Get/Symbol
LABELS/Create single/End-of-line comment (<esc><esc>)

Thus, the following code:

```
lbC002C68           MOVE.L    A6,-(SP)
                    MOVEA.L   (Intution_Base,PC),A6
                    MOVEA.L   (8,SP),A0
                    JSR       (_LVOOpenWindow,A6)
                    MOVEA.L   (SP)+,A6
                    RTS
```

would become:

```
lbC002C68           MOVE.L    A6,-(SP)              ; _LVOOpenWindow
                    MOVEA.L   (Intution_Base,PC),A6
                    MOVEA.L   (8,SP),A0
                    JSR       (_LVOOpenWindow,A6)
                    MOVEA.L   (SP)+,A6
                    RTS
```

Buffers A-M are simply 13 separate 240-byte string buffers, in which you can store strings. A more appropriate name might be 'text registers'.

Buffers L and M are special, in that if you select "STRINGS/Define string/M", if buffer L is not empty, the string contained in it will be used as the prompt, in the requester. This is handy during macros functions where you want to get a string from the user. The user can then see what the string is required for.

Normally, every function you put into a macro definition will be executed. This does not always have to be the case. The "MACROS 1/Commentary/" functions sets the commentary level. When creating a macro, if you set the commentary level to "None", it is like specifying "The functions following are absolutely essential to the execution of this macro". If you set the commentary level to "Full" during the creation of a macro, you are specifying "The functions following are by no means required, they are simply running commentary, perhaps explaining what is happening in the macro at this point". Thus, by setting the commentary level during the creation of a macro, you are letting ReSource know how important the functions are, that follow.

## General Information

**MACROS**

The commentary level may be changed many times during a macro, and for tutorial macros, such as showing someone how to disassemble/zap a particular program, the commentary level should be set appropriately. When it comes time to execute the macro, if the commentary level is set to "Full" by the user before the macro starts executing, ALL functions within the macro will be executed normally. If the commentary level was set to something other than "Full", only those functions in the macro that were set to a commentary level lower than that presently set, will be executed; the rest will be skipped over. Examine the following example macro:

```
MACROS 1/Commentary level/Full
CURSOR/Relative/Next line
MACROS 1/Commentary level/Heavy
CURSOR/Relative/Next line
MACROS 1/Commentary level/Normal
CURSOR/Relative/Next line
MACROS 1/Commentary level/Light
CURSOR/Relative/Next line
MACROS 1/Commentary level/None
CURSOR/Relative/Next line
```

If you set the commentary level to "None" and execute this macro, the cursor will move down one line. If you set the commentary level to "Light", and execute this macro, the cursor will move down two lines. If you set the commentary level to "Full" and execute this macro, the cursor will move down five lines. Using the "SPECIAL FUNCTIONS/Dos command" function, you can use the "SAY" command, to add speech to macros, to give excellent running commentary to tutorial macros.

While executing a macro, if you have set the execution speed to "Wait on mouse", you can use other functions, perhaps to scroll backwards or forwards, to see what effect the last function had on the file. When you press the left mouse button to single-step the next function in the macro, the cursor address is restored in case you didn't return the cursor to the appropriate place within the file. This is necessary, as if this were not done, macro processing would not proceed normally from that point on.

And if you are single-stepping the execution of a macro, you may find it handy to have "DISPLAY/Titlebar info/Function names" selected. This way, the next function to be executed is displayed in the titlebar. This is also an excellent way of learning about ReSource functions - by single stepping one of the example macros, and watching carefully the names of the functions that get executed.

When you load a macro file, only the numbered macros in the file will actually overwrite macros already in ReSource. If the macro file only contains one macro, then only the one macro will be overwritten, and the rest will remain untouched. Thus, it is possible for a macro to load other macro files, giving an "overlay" effect.

While a macro is executing, if the "CANCEL" gadget is selected for a requester, the macro will abort immediately.

ReSource macro files may be disassembled, examined, edited, reassembled, and reused. "ShowMacros" is a support utility on the ReSource distribution disk.

It is used to disassemble a ReSource macro file into an assembly language source code file suitable for reassembly. The output is compatible with the

## General Information

**MACROS**

Macro68, CAPE and Metacomco assemblers, and the (reassembled) output file may be used by ReSource as a macro file immediately.  If you are using the Metacomco assembler, after assembly you may have to strip the loader information from the file.  To do this, simply load the file into ReSource as a load file, and then immediately save as a binary image file.

## General Information

### NUMERIC TYPES

By default, all numbers are shown in hexadecimal (base 16). You can individually change this to ASCII, DECIMAL, BINARY, or back to HEXADECIMAL on any line by selecting from the appropriate sub-item. It is possible to *globally* change the default to decimal for numbers less than 16, or less than 10, using the "DISPLAY/Decimal conversion/" functions.

The function "DISPLAY/Set numeric base/ASCII" will enable virtually any number at all to be shown as ASCII, providing that no non-valid ASCII characters are present. Examples follow:

```
                    dl        $4E
                    dl        $4E00
                    dl        $4E004E00
                    dl        $4E4E00
                    dl        $4E000000
                    dl        $444F53
                    dl        $444F5300
                    dl        $4B49434B
```

will, by using the ASCII numeric base, be shown as:

```
                    dl        'N'
                    dl        ('N'<<8)
                    dl        $4E004E00
                    dl        ('NN'<<8)
                    dl        ('N'<<24)
                    dl        'DOS'
                    dl        ('DOS'<<8)
                    dl        'KICK'
```

Note that the third line contained a non-valid ASCII character. Although some others contained zeroes in the lower byte(s), they can be safely shown as being an ASCII value shifted 8/16/24 to the left.

Double, Extended and Packed data types may not be shown as ASCII, or anything other than their native format. They are also not subject to the Decimal conversion function explained above.

Floating point data types are displayed in either decimal or scientific format depending on the value of the number. The display routines will try to provide the format that is clearest. Therefore, the value 100 will always be shown as "100.0" and not as "1E2".

A very short synopsys of the format specifications for floating point data types follows:

```
  Single-Precision binary real:

      Bit: 31 30      23 22      0
         +---+----------+----------+
         | S | Exponent | Mantissa |
         +---+----------+----------+

      Ranges (approximate):
         Maximum Positive Normalized   3.4 x 10(38)
         Minimum Positive Normalized   1.2 x 10(-39)
         Minimum Positive Denormalized 1.4 x 10(-45)
```

## General Information

**NUMERIC TYPES**

```
    Double-Precision binary real:

        Bit: 63 62      52 51       0
           +---+----------+----------+
           | S | Exponent | Mantissa |
           +---+----------+----------+

        Ranges (approximate):
           Maximum Positive Normalized    18 x 10(307)
           Minimum Positive Normalized   2.2 x 10(-308)
           Minimum Positive Denormalized 4.9 x 10(-324)

    Extended-Precision binary real:

        Bit: 95 94      80 79    63 62       0
           +---+----------+--------+----------+
           | S | Exponent |  Zero  | Mantissa |
           +---+----------+--------+----------+

        Ranges (approximate):
           Maximum Positive Normalized   6 x 10(4931)
           Minimum Positive Normalized   8 x 10(-4933)
           Minimum Positive Denormalized 9 x 10(-4952)

    Packed-Decimal (BCD) real:

              +------------ Sign of Mantissa  0=Positive, 1=Negative
              |    +------- Sign of Exponent       "           "
              |    |    +-- Used only for +/- Infinity or NAN(s)
              |    |    |
        Bit: 95   94   93  91  88 87  84 83  80 79  76 75  72 71  68
           +----+----+----+------+------+------+------+------+------+
           | MS | ES | YY | EXP2 | EXP1 | EXP0 |(EXP3)| XXXX | XXXX |
           +----+----+----+------+------+------+------+------+------+
                                                [     Don't cares    ]
                         +-- Implicit decimal point
                         |
            67        64 | 63                      0
           +-------------+-------------------------+
           | Mantissa 16 | Mantissa 15 - Mantissa 0 |
           +-------------+-------------------------+

  See also:
    "data types"
```

## General Information

### USING STRING BUFFER INDIRECTION

When supplying a string (into a string requester), rather than typing in the actual string, you may use "string buffer indirection". This means that if the string that you want to use is already in one of ReSource's string buffers (A-M), you can tell ReSource to use the string from one of those buffers. This is done by typing an escape character (either type "\e" or just press the escape key), followed by one of the letters A-M, representing the buffer from which the string will be gathered. Or, if the string is to come from the accumulator, two escape characters should be used.

For example, if you wanted to create a macro that searched for the string "START+", but rather than simply overwrite the current search string, it saved, and then restored it, the following macro would do this:

```
STRINGS/Get/Search string                       ;Get current search string
CURSOR/Normal search/Set search string  "START+"
CURSOR/Normal search/Find next occurrence
CURSOR/Normal search/Set search string  "\e\e"  ;Restore search string!
```

### USING SYMBOL BASES

Functions associated with the "SYMBOLS" gadget are used to create symbols. There are literally hundreds of structures, with many symbols defined for various offsets from these structures. Rather than use the actual value of a particular offset, symbols should be used. When assembling, this serves two purposes:

1. It is easier to remember "_LVOOpenLibrary" and "_LVOOpen" than the equivalent values "-$228" and "-$1E".

2. When a new operating system is released, you have only to reassemble with the appropriate include files for the new operating system release, and any changed offset values will be picked up immediately.

When disassembling, symbols give you information about what the original author intended the code to do. Before using the "SYMBOLS" gadget to select a function, you must first decide which symbol base should be used. Examine the following code:

```
                    MOVE.L    (4),A6
                    JSR       (-$C6,A6)
```

The first line is getting a pointer to the "exec" library, and putting it into the A6 register. The second line calls a library function, at an address that is $C6 bytes lower in memory than the base of the exec library. To find out the name of this library call, scroll so that the second line becomes the current line, click on the "SYMBOLS" gadget and then select "Library Offsets/Exec Library/All". The code will then become:

```
                    MOVE.L    (4),A6
                    JSR       (_LVOAllocMem,A6)
```

In this case, the symbol "_LVOAllocMem" has been created, replacing the number "-$C6". When reassembled, the resulting code will be same whether you use a symbol or the number.

## Menu Functions

### &H003C  PROJECT/Open load file

ReSource will present the ASL file requester. When you either double-click on a file name, select a file and press "return", or click on the "okay" gadget, ReSource will attempt to load the file as an executable, or "load" file. If there are any debug symbols in the executable, these will be stored, and will become labels at the appropriate places in the file. Relocation is performed where required.

If the file has overlays, ReSource will attempt to load the root hunk. Each debug symbol encountered is attached to the appropriate byte in the file, and for those that are nine characters long, and start with "lb", immediately followed by 'A', 'B', 'C', 'L', or 'W', the data type at the appropriate byte will be set to ASCII, Bytes, Code, Longwords, or Words, respectively.

Additionally, local labels, of the "1$" variety, and labels ending in "SUB" are recognized as code, and labels ending in ".MSG" are recognized as ASCII. Any BSS hunks or uninitialized data areas found are expanded fully. You may hilite areas where relocation has been performed by selecting "DISPLAY/Hiliting/Reloc32".

You are limited to files smaller than 16 Megabytes, and further limited by how much memory you have. To disassemble a 100K file, you will require approximately 850K of memory, of which at least 400K must be contiguous. If the file has large BSS hunks, this will greatly increase the memory requirements.

### &H0190  PROJECT/Open binary file

Similar to "PROJECT/Open load file", except that no translation of the file is performed; the file is read in "as-is". This allows you to see the loader information in executables, object files, etc. You can also load in text files, data base files, in fact any file whatsoever, memory permitting. If you select "PROJECT/Open load file" for a non-executable file, the file will be loaded as a binary image instead, therefore "PROJECT/Open binary file" is not really required except when you wish to force an "as-is" load for an executable file.

### &H0048  PROJECT/Restore file

Useful only when the current buffer was loaded using "PROJECT/Open binary file" or "PROJECT/Open load file".

This function will attempt to load a file, with the same name as the current file, using the same function that loaded the current file originally. Do not use this function if the current buffer was loaded from tracks, or directly from memory.

## Menu Functions

### &H0276  PROJECT/Dismble memory

Disassembles a block of memory directly.  You will be asked to supply a start and end address for the memory region to be disassembled.  The memory is NOT copied to a separate buffer, it is disassembled "as-is", which means that if you hold down the left mouse button, in some areas you can see the memory being dynamically updated.  It also means that you can modify memory directly.  Anywhere.  For each address that you supply, if the number starts with a "$", the number is assumed to be hexadecimal.  If it starts with "%", it is assumed to be a binary number.  If neither, decimal is assumed.  If you wish to work on a COPY of a block of memory, to avoid modifying the original, or perhaps because the original will not be around for long, disassemble the memory directly, and save the data with the "PROJECT/Save .RS" function.  Then use "PROJECT/Open load file" to load the ".RS" file normally.

### &H0273  PROJECT/Read tracks

You will be asked to supply the parameters for tracks to read.  The first parameter must be either "DF0:", "DF1:", "DF2:", or "DF3:" (lower case is okay).  This represents the drive that holds the disk to be read from.  The second parameter is the number of the cylinder to start reading from.  The third parameter is the number of the last cylinder to be read, plus one.  For example, if you wanted to read the first cylinder from the disk in DF0:, the parameters would be "DF0:  0 1".  The fourth parameter is optional, and represents the number of extra sectors to read.  For example, if you wished to read only the very first sector from DF1:, the parameters would be "DF1: 0 0 1".  If you wished to read the first sector from the directory track of DF2:, the parameters would be "DF2: 40 40 1".  The fifth parameter is also optional, and it represents the sector offset, to start the read.  For example, if you wished to read sectors nine and ten on cylinder 79 of the disk in DF3:, the parameters would be "DF3: 79 79 2 9".

## Menu Functions

### &H0299  PROJECT/O''lay binary image

This function prompts you for a filename.  ReSource will attempt to open the file, and read the contents, overlaying the current file, starting at the cursor position.  This is NOT the same as opening a file normally, as all labels, symbols, comments, data types, etc., stay as they are.  Only the actual contents of the executable itself are overwritten.

For example, you may wish to pass on a ".RS" file to someone, but because the program you have disassembled is copyrighted, you cannot legally distribute the normal ".RS" file, as it contains the executable.  By using this function, and inputting a filename of "*" (asterisk), ReSource will completely clear the executable within the current file.  You may then save to a ".RS" file, which may then be distributed, devoid of the executable.  If another person has previously purchased the program that you have disassembled, they may load it into ReSource, and use "SAVE/Save binary image/All", to save the executable to a file.  They then load the special ".RS" file which you created.  Next, they then use "PROJECT/O'lay binary image", supplying as a filename, the name of the file previously saved with the "SAVE/Save binary image/All" function.  This effectively puts the executable back into the ".RS" file.  The other save functions may then be used, to save to a .asm file, for example.

Please make sure the executable section IS cleared before you share ".RS" files that are disassemblies of copyrighted material.  Failure to do this may result in your meeting many new people - mostly lawyers.

## Menu Functions

### &H0306 PROJECT/Disassemble

This function does most of the work in converting "START+" references to label references. It can also recognize many types of "jump tables", especially those in programs created using any of the popular "C" compilers. It can be used on most executables, but before using, you (the user) should determine if a base register is being used to reference data and/or code. If so, use one of the "SPECIAL FUNCTIONS/Convert (xx,An) EA's/" functions to do the conversions, then go ahead and select the "PROJECT/Disassemble" function. The following guide is applicable to most executables that require disassembling:

1. Check for base register addressing. Select the appropriate EA conversion function from the SPECIAL FUNCTIONS menu.

2. If appropriate, execute a command language file.

3. Select the "PROJECT/Disassemble" function.

4. Move the cursor to the start of the file, and select the "CURSOR/ Relative/Next uncertain D/T" function. This function asks ReSource to show you places in the file where it needs some help in determining what type of data (code/ASCII/bytes/words/longwords) is there. At each location, change the data type if required, and then *without scrolling*, select the "CURSOR/Relative/Next uncertain D/T" function again. This tells ReSource that it can now be certain that the data type there is correct. Keep doing this until you hit the end of the file.

5. Select the "PROJECT/Disassemble" function again. If you changed anything in step 4 to code, this will convert more labels. Repeat steps 4 and 5 until you no longer find any uncertain data types in step 4.

6. Move the cursor to the start of the file, and use the "CURSOR/Relative/Next error line" function. ReSource will move the cursor to find places in the file that require attention. These may be unresolved "START+" references, odd address errors, bad references, unreferenced code/data, library calls that haven't yet had a symbol created for them, or instructions that are either illegal, or extremely unlikely to be used.

7. The file is probably ready for reassembly, however you may first want to carefully examine the code, and create meaningful labels and symbols. Even though the "PROJECT/Disassemble" function may have been used before, perhaps several times for any particular file, after changing any other data type to "code", it may be useful to use it again. For example, there may be a large portion of code with only one entry point, which up until now has been shown as ASCII. After changing to code, the "PROJECT/Disassemble" function should be able to resolve ALL the "START+" references for the entire subroutine, and all the subroutines that it calls, and so on.

### &H0045 PROJECT/Save .RS/Save

Use this function to save what you are currently working on, regardless of how far you have gotten in disassembling it. Virtually everything is saved, even your current position within the file. To re-open a ".rs" file, use the "PROJECT/Open load file" function.

## Menu Functions

**&H0045  PROJECT/Save .RS/Save**

**&H036C  PROJECT/Save .RS/Original**
**&H0370  PROJECT/Save .RS/Current dir**
**&H0368  PROJECT/Save .RS/Specify**

You can set up the environment in ReSource to specify where files are to be saved.  There are 3 different sets of functions that you can use to specify the behaviour of:

"PROJECT/Save .RS/Save"
"SAVE/Save binary image/All"
"SAVE/Save binary image/Partial"
"SAVE/Save .asm/All"
"SAVE/Save .asm/Partial"
"SAVE/Save executable/With Labels"
"SAVE/Save executable/No Labels"

1. Current dir
If enabled, the default path to save files to will be the same path as where the file was loaded from.  When the file requester is displayed, you will only see the file name.

2. Original
If enabled, the default path to save files will be the current directory.  When the file requester is displayed, you will see the path and the file name.  If the file was loaded from the current directory, the behaviour will be the same as for "Original".

3. Specify
You will be asked to supply a pathname, which will be used as the default path to save files to in future.  This pathname will be saved in the "Auto-configuration" macro when you select "PROJECT/Save Configuration".

## Menu Functions

### &H0322  PROJECT/Save Configuration

Use to save the current state of ReSource's various options, menu settings, contents of string buffers, etc.  ReSource does this by creating macro #19, and naming it "Auto-configuration", and then offering to save macros.  If the macros are saved to the file "S:RS.Macros", whenever you next run ReSource, it will run the new "Auto-configuration" macro, thereby setting up the various options to how they were when you used "Save config" last time.
Here is a complete list of what is restored when the Auto-configuration macro is eventually executed at startup:

1. The checked/unchecked state of all menu items.
2. The state of the four case flipping functions (code,data,registers,size qualifiers)
3. All user-defined symbol bases will be loaded to the same places as before.
4. The "comments column" number.
5. The custom path for saving ".asm" files to.
6. The custom path for saving ".rs" files to.
7. The custom path for saving executables to.
8. The custom path for saving binary images to.
9. The current search string.
10. The current binary search parameters.
11. The contents of the accumulator, and buffers A-M inclusive.

At the end of the Auto-configuration macro, it executes macro #57 (the last macro under the "MACROS 3" menu).  If there is any other functions that you would like executed at startup time, put them in macro #57.

### &H0049  PROJECT/About

Get information about the version number, author, agents, and people that helped in the development of ReSource.

### &H0357  PROJECT/     -<HELP>-

The function that you are using right now!!
The "Help" function allows you to get information about ReSource's functions quickly.  There are various keys used inside the "Help" function.  These are hard-bound, they cannot be rebound the way normal functions can.  They are:

| | |
|---|---|
| Exit | normal-ESC |
| Scroll down | normal-down arrow |
| Scroll up | normal-up arrow |
| Scroll to start of file | alt-up arrow |
| Scroll to end of file | alt-down arrow |
| Hilite word for hyper-help | left and right arrows |
| Hyper-help for hilited word | normal-return |
| Back out from hyper-help | normal-backspace |
| Page down | shift-down arrow |
| Page up | shift-up arrow |
| More help | alt-help |
| Next help screen | alt-right arrow |
| Previous help screen | alt-left arrow |

## Menu Functions

**&H0008  PROJECT/Quit**

Asks you for confirmation, then quits without saving.  If this function is used within an executing macro, it will NOT ask the operator for confirmation.

If any new macros have been defined, without being saved to a macro file, this fact will be mentioned in the quit requester.

See "PROJECT/Quit NOW"

**&H0325  PROJECT/Quit NOW**

Quits immediately, with no questions asked.

**&H001C  DISPLAY/Hiliting/BSS hunks/ON**

Data within BSS hunks and uninitialized data areas will be hilited.  The actual hiliting consists of light underlining of all opcodes.

**&H001B  DISPLAY/Hiliting/DATA hunks/ON**

Everything within data hunks will be hilited.  Data hunks often will contain only data constants, but may also contain code.  The actual hiliting consists of light underlining of all opcodes.

**&H001A  DISPLAY/Hiliting/CODE hunks/ON**

Everything within code hunks will be hilited.  Code hunks often will contain only code, but may also contain data constants.  The actual hiliting consists of light underlining of all opcodes.

**&H005E  DISPLAY/Hiliting/Chip load hunks/ON**

Data within code or data hunks that MUST be loaded into chip memory (only) will be hilited.  Data falling into this category will usually be graphics or sound data, to be accessed directly by the blitter, copper or DMA.  The type of hiliting used will be full hiliting of the opcode field.

**&H005D  DISPLAY/Hiliting/Fast load hunks/ON**

Data within code or data hunks that MUST be loaded into fast memory (only) will be hilited.  The type of hiliting used will be full hiliting of the opcode field.

**&H0023  DISPLAY/Hiliting/Reloc32/ON**

All "RELOC32" pointers are hilited.  The actual hiliting consists of using inverse video, for the characters that make up the reloc32 reference.

## Menu Functions

### &H0151  DISPLAY/Hiliting/Symbol scan/ON

Hilites lines that have been scanned.  "Scanned" or "parsed" lines are those that have been the target of either the "LABELS/Create single/Label - fwd ref" or "LABELS/Create multiple/All" functions, or those lines scrolled over while holding down the leftAmiga key, as well as the LMB.

The type of hiliting in this case will be light underlining, of the first whitespace area.

### &H0152  DISPLAY/Hiliting/Data type uncertain/ON

For most labels that ReSource creates automatically, it is certain of the data type that it assigns.  Sometimes though, it cannot be 100% sure, and this function will hilite all lines that fall into this category.

The type of hiliting in this case will be full hiliting of the first whitespace area.

### &H0153  DISPLAY/Hiliting/Data type known/ON

For most labels that ReSource creates automatically, it is certain of the data type that it assigns.  Sometimes though, it cannot be 100% sure. This function will hilite those that ReSource was 100% sure of, and also those lines where the user has specifically set the data type.

The maintenance of "data type known" information is very important in ReSource.  For example, if a label is created, and the data type at that label is set to "ascii" by ReSource, and then at some later time, a line is parsed, where there is another reference to the same location, but this time ReSource decides that the data type there must be code, it will actually change the data type, if "data type known" attribute bit has not been set at that location, unless the "data type uncertain" bit is also set.

ReSource also uses the "data type known" bit to determine line starts.  For example, ascii is often displayed in strings of more than one byte long.  In determing how many bytes to display on one line, ReSource checks for bytes that have the "data type known" bit set, and when it finds one, will start it on a new line, even if it is set to "ascii" data type also.

The "data type known" bit can be cleared by selecting either the "DISPLAY/Set data type/Unknown" or "LABELS/Remove single/All" function. The type of hiliting used will be full hiliting of the first whitespace area in the line.

### &H0154  DISPLAY/Hiliting/Internally-produced refs/ON

Hilites label definitions where the label name itself was created by ReSource ("shop" labels), rather than defined by the user.  This differs from the 'DISPLAY/Hiliting/"Shop" labels' function in that only the label definition is hilited, not references to the label.

The type of hiliting in this case will be light underlining of the label definitions.

## Menu Functions

### &H02B9  DISPLAY/Hiliting/Uninitialized data/ON

When selected, any uninitialized data areas will be hilited.

The type of hiliting used will be full hiliting in the second whitespace area.  This is the spaces or tab between the opcode and the first operand.

### &H02CB  DISPLAY/Hiliting/DCB override/ON

This function will hilite any areas that have been selected by "DISPLAY 2/DCB override/Set".  The type of hiliting used will be light underlining of the first whitespace area.

### &H02CD  DISPLAY/Hiliting/Symbols/ON

This function will hilite all symbols.  The type of hiliting used will be heavy underlining of all symbols.

### &H034E  DISPLAY/Hiliting/"Shop" labels/ON

All labels that were originally created by ReSource ("shop" labels) will be hilited.  The type of hiliting used will be light underlining of each every shop label, be they the label definition, or a reference to the label definition.

### &H034F  DISPLAY/Hiliting/Custom labels/ON

All labels that were originally created by the user ("custom" labels) will be hilited.  The type of hiliting used will be light underlining of each every custom label, be they the label definition, or a reference to the label definition.

## Menu Functions

**&H000E  DISPLAY/Set data type/Code**
**&H000B  DISPLAY/Set data type/ASCII**
**&H0017  DISPLAY/Set data type/Bytes**
**&H0014  DISPLAY/Set data type/Words**
**&H0013  DISPLAY/Set data type/Longs**
**&H03BA  DISPLAY/Set data type/Single**
**&H03BB  DISPLAY/Set data type/Double**
**&H03BC  DISPLAY/Set data type/Extended**
**&H03BD  DISPLAY/Set data type/Packed**

When you are certain of which type of data that you are looking at, select from one of the sub-menu items: Code, ASCII, Bytes, Words, Longwords, and the floating point types Single, Double, Extended and Packed.
Under certain conditions, ReSource will override the previously set data type. These are detailed below:

1. Attempts to set the data type to ASCII or CODE in a BSS or uninitialized data area will instead set the data type to BYTES.  Regardless of the setting of the attributes bits, data in a BSS hunk or in an uninitialized data area will *always* show as "ds.b", "ds.w" or "ds.l".

2. If a line is displayed such that a reloc32 longword will overlap to the next line, the data type at the start of that line is changed to bytes/ words, effectively preventing the overlap.    The floating point types will scale down in a similar manner.

If a data type of "code", "word", "longword" "single", "double", "extended" or "packed" is selected at an odd address, a DisplayBeep() is generated.

If a label is defined for an uninitialized data or BSS area, and the data type for the previous word was set to longword, ReSource will override the previous data type in order to avoid hiding the label.  This will also be true for other combinations of data type sizes, such as when the previous byte had its data type set to words.  This means that you will not see any hidden labels in a BSS or uninitialized data area.

Double, Extended and Packed data types behave in a slightly different manner than other data types:  They may not be assigned symbolic names, and must be displayed in their native format--the numeric base may not be changed.

See also:
  "data types" and "numeric types"

**&H03B3  DISPLAY/Set data type/Unknown**

This function is used to remove a previously-set data type.  This may occur when you find a reference to somewhere, set the data type there and create a label, only to discover later that the original reference was bogus.  You may then use "LABELS/Remove single/Label" to remove the label, and "DISPLAY/Set data type/Unknown" to remove the data type setting.  Actually, you can do both of these at the same time, by using the "LABELS/Remove single/All" function.

**&H01A5  DISPLAY/Set Numeric base/Hexdecimal**

Display numbers on current line as hex.  See "numeric types".

## Menu Functions

**&H005C  DISPLAY/Set Numeric base/Decimal**

Display numbers on current line as decimal.  See "numeric types".

**&H002D  DISPLAY/Set Numeric base/Binary**

Display numbers on current line as binary.  See "numeric types".

**&H0022  DISPLAY/Set Numeric base/ASCII**

Display numbers on current line as ASCII, if possible.
See "numeric types".

**&H016A  DISPLAY/Decimal conversion/None**
**&H016B  DISPLAY/Decimal conversion/<10**
**&H016C  DISPLAY/Decimal conversion/<16**

By default, all numbers are shown in hexadecimal.  You can have numbers less than 16, or numbers less than 10, shown in decimal throughout the file, by selecting one of the appropriate sub-items:

| | |
|---|---|
| None: | Do not do any decimal conversions. |
| 10: | Globally convert numbers 0 to 9 inclusive, to decimal. |
| 16: | Globally convert numbers 0 to 15 inclusive, to decimal. |

**&H0175  DISPLAY/Blank lines/Unconditional branches/ON**
**&H0176  DISPLAY/Blank lines/Conditional branches/ON**
**&H02B1  DISPLAY/Blank lines/DBRA instructions/ON**
**&H02B2  DISPLAY/Blank lines/DBcc instructions/ON**
**&H02CF  DISPLAY/Blank lines/Entry points**
**&H02A4  DISPLAY/Blank lines/Returns/ON**
**&H02A3  DISPLAY/Blank lines/Calls/ON**

To better define subroutines, and logical blocks of code, ReSource can insert blank lines in the source code.  You may select whether a blank line is inserted after each conditional branch, unconditional branch, return, or subroutine call.  Additionally, selecting "DISPLAY/Blank lines/Entry points" will create a blank line where data ends, and code starts.

You may select more than one, in which case a blank line will appear after any line which falls into any of the selected categories.  A line will never be followed by more than one blank line, regardless of how many selected categories it falls into.

**&H01C5  DISPLAY/Cursor address/Relative**

The cursor address will be shown in the title bar, as a hexadecimal offset from the start of the file.

## Menu Functions

### &H01C6  DISPLAY/Cursor address/Absolute

The cursor address will be shown in the title bar, as a hexadecimal absolute machine address. This allows you to quickly find the cursor position in memory with other utilities, such as MetaScope, perhaps to carry out large scale modifications.

### &H01A6  DISPLAY/Titlebar info/Filename

Information is displayed in the title bar in several fields. From the left, the first is the program name. The second is a decimal number, representing the cursor position relative to the file size, and is a percentage. The third field is a hexadecimal number, representing the cursor position, and can be shown either as an offset from the start of the file, or as an absolute memory location. The fourth field is the one that this function relates to. By selecting this function, the current file name will be shown. If user feedback is ON, informative messages will sometimes be shown in this field also.

## Menu Functions

### &H01A7  DISPLAY/Titlebar info/Attributes

The last field of the title bar will display information representing the current attributes of the cursor byte.  This information is NOT required for general program usage, and was originally included for debugging purposes only, and left in to satisfy those that want to know how the attributes table in ReSource works.  The attributes table is four times the size of the current file, plus 16 bytes.  It is used internally as a large bitmap, to store information about each and every byte in the current file.  Because it is four times the file size, 32 bits are available for each byte in the current file.  For each character in the string displayed, if lower case, the bit that that character represents is clear, if upper case, the bit is set.

Going from the left, the bit definitions are:

| | | |
|---|---|---|
| Bit 31 | S: | Start of a line of code/data |
| Bit 30 | B: | Bss hunk |
| Bit 29 | D: | Data hunk |
| Bit 28 | C: | Code hunk |
| Bit 27 | R: | reloc32 area |
| Bit 26 | F: | First byte of reloc32 area |
| Bit 25 | L: | Label attached to this byte |
| Bit 24 | S: | Symbol attached to this byte |
| Bit 23 | F: | Full-line comment attached to this byte |
| Bit 22 | E: | End-of-line comment attached to this byte |
| Bit 21 | P: | Parsed previously (symbol scan) |
| Bit 20 | I: | Internally referenced |
| Bit 19 | L: | Low priority.  The data type has been set, but can be overridden by internal routines.  The data type can always be overridden by the user.  This bit was generally set when ReSource had to decide on a data type to set for a byte, but when it did so, it wasn't really sure that it was correct. |
| Bit 18 | A: | Show numeric operands as ASCII if possible.  Decimal conversion is on a higher priority than this bit. |
| Bit 17 | B: | Show numeric operands as Binary.  Decimal conversion is on a higher priority than this bit. |
| Bit 16 | D: | Show numeric operands as Decimal if possible. |
| Bit 15 | F: | Data in this hunk is to be loaded into FAST memory only. |
| Bit 14 | C: | Data in this hunk is to be loaded into CHIP memory only. |
| | : | [Next 4 bits are reserved for future use] |
| Bit  9 | U: | User should check the data type that has been set, as ReSource wasn't really sure that it chose the correct one.  Use |

## Menu Functions

**&H01A7  DISPLAY/Titlebar info/Attributes**

```
                    "CURSOR/Relative/Next uncertain D/T" to find the next occurrence.
                    If such a line is already the current line when you use that
                    function, it will clear this bit.

  Bit  8     H:  High priority.  The data type has been set either by the user,
                 or by ReSource when it was quite certain that it set the
                 data type correctly.  This does not mean that it cannot make
                 mistakes, it simply means that if some operation attempts to set
                 the data type for this byte, it will NOT override the currently
                 set data type.  This will not ever stop the user from redefining
                 the data type, however.

  Bit  7     C:  This line is code.

  Bits 6/5/4/3   dddd:  Data type is unknown (usually defaults to code)
                 dddD:  Data type is single
                 ddDd:  Data type is bytes
                 ddDD:  Data type is double
                 dDdd:  Data type is words
                 dDdD:  Data type is extended
                 dDDd:  Data type is longwords
                 dDDD:  Data type is packed
                 Dddd:  Data type is bytes, within a BSS hunk or
                        uninitialized data area.
                 DddD:  Data type is ASCII.
                 DdDd:  Data type is single, within a BSS hunk or
                        uninitialized data area.
                 DdDD:  Data type is double, within a BSS hunk or
                        uninitialized data area.
                 DDdd:  Data type is extended, within a BSS hunk or
                        uninitialized data area.
                 DDdD:  Data type is packed, within a BSS hunk or
                        uninitialized data area.
                 DDDd:  Data type is words, within a BSS hunk or
                        uninitialized data area.
                 DDDD:  Data type is longwords, within a BSS hunk or
                        uninitialized data area.

  Bit  2     S:  Start of a new section.
```

**&H0246  DISPLAY/Titlebar info/Accumulator**

The third field of the title bar will display information representing the contents of the ACCUMULATOR. This is a 240 byte buffer, used as a central number and string processor, which will find most of its uses within macros.

Only the first 30 characters or so can be shown in the title bar. However, often the string will not be longer than that anyway. For more information on the accumulator, see the functions within the "STRINGS" menu.

## Menu Functions

**&H03B2  DISPLAY/Titlebar info/Function names**

The fourth field in the titlebar will display the name of the last-executed function.  A special case is while executing a macro, with the execution speed set to "Wait on mouse".  In this case, the *next* function to be executed will be displayed in the titlebar.

**&H0195  DISPLAY/Wrap/ON**
**&H0196  DISPLAY/Wrap/OFF**

With wrap on, any lines longer than the current display width will be shown on several lines.  Otherwise, any lines longer than the current display width will be truncated.

**&H02BC  DISPLAY/Block-fill/Set start**
**&H02BD  DISPLAY/Block-fill/Set end**
**&H0156  DISPLAY/Block-fill/Fill**

To use the "Block fill" function, the start and end addresses must be specified by selecting "DISPLAY/Block fill/Set start" and "DISPLAY/Block fill/Set end" at the appropriate locations in the file.  The block-fill start and end addresses will default to the actual start and end of the file.

Selecting "DISPLAY/Block-fill/Fill will copy the data type attributes of the first byte (as defined by "Set start", from the start location up to, but not including, the end location.

**&H0024  DISPLAY/Set Counter**
**&H0025  DISPLAY/Reset Counter**

In ReSource's title bar, the offset from the start of the file is normally shown near the left edge.  Sometimes, you may wish to know offsets from something other than the start of the file.  To do this, move to where you wish to calculate offsets relative from, and select "DISPLAY/Set Counter". To see normal offsets again, select "DISPLAY/Reset Counter".

**&H01A2  DISPLAY/Flip case/CODE**

If code is currently being displayed using upper case, it will instead be shown in lower case, and vice versa.  Note that searches are case sensitive.

**&H01A3  DISPLAY/Flip case/data**

If data is currently being displayed using upper case, it will instead be shown in lower case, and vice versa.

**&H0314  DISPLAY/Flip case/REGISTERS**

If registers are currently being displayed using upper case, they will be instead shown in lower case, and vice versa.

## Menu Functions

### &H0366  DISPLAY/Flip case/SIZE SPECIFIERS

If size qualifiers are currently being displayed using upper case, they will be instead shown in lower case, and vice versa.

### &H02C2  DISPLAY 2/DCB override/Set
### &H02C3  DISPLAY 2/DCB override/Clear

These functions set or clear an attribute bit for the current line determining whether the DCB directive will be used on this line if there is suitable data.  This bit works in conjunction with the global DCB option (see "OPTIONS/Show.../DCB statements") to comprise an exclusive-OR function:  If global DCB's are ON, this bit will turn them off locally.  Conversely, if global DCB's are set to OFF, this bit will enable them locally.

### &H02C4  DISPLAY 2/Mult constants override/Set
### &H02C5  DISPLAY 2/Mult constants override/Clear

These functions set or clear an attribute bit for the current line determining whether multiple constants per line will be displayed, should there be suitable data.  This bit works in conjuction with the global multiple constants option (see "OPTIONS/Show.../Multiple constants" to comprise an exclusive-OR function:  If global multiple constants are ON, this bit will turn them off locally.  Conversely, if multiple constants are set to OFF, this bit will enable them locally.

### &H02C2  DISPLAY 2/DCB override/Set
### &H02C3  DISPLAY 2/DCB override/Clear

These functions set or clear an attribute bit for the current line determining whether the DCB directive will be used on this line if there is suitable data.  This bit works in conjunction with the global DCB option (see "OPTIONS/Show.../DCB statements") to comprise an exclusive-OR function:  If global DCB's are ON, this bit will turn them off locally.  Conversely, if global DCB's are set to OFF, this bit will enable them locally.

### &H0384  DISPLAY 2/Set comments column

You will be asked to supply a number, that will be used as the starting column for all end-of-line comments.  If a line finishes after the specified comment starting column, only one space will be used before the "" is displayed.

## Menu Functions

**&H0373 SYMBOLS/Custom bases/Use (1)**
**&H0374 SYMBOLS/Custom bases/Add (1)**
**&H0375 SYMBOLS/Custom bases/Clear (1)**
**&H0376 SYMBOLS/Custom bases/Use (2)**
**&H0377 SYMBOLS/Custom bases/Add (2)**
**&H0378 SYMBOLS/Custom bases/Clear (2)**

The custom symbol bases give you a way of creating symbol bases from within ReSource itself. The idea here is that you firstly select and clear one of the custom symbol bases, and select "use" whenever you wish to create a symbol using that custom symbol base. The first time you select "use", ReSource will ask you for the name of the symbol to use. On subsequent symbols, ReSource will compare the symbol value, to those for the symbols that have already been added to this custom symbol base. If there is a match, you will not have to type in the symbol name again, instead ReSource will create the symbol name using the same name as was used last time. When you use "add", the currently selected symbol and its symbol value will be added to the base. If there is already a symbol in the base with the same symbol value, it will be removed, and replaced by this symbol and symbol value. When you select "clear", it clears all symbols from that particular custom symbol base.

**&H0050 SYMBOLS/Select field/First (default)**
**&H0051 SYMBOLS/Select field/Second**
**&H0312 SYMBOLS/Select field/Third**
**&H0313 SYMBOLS/Select field/Fourth**

Examine the following line of code:

```
                    MOVE.L    #-$0228,$0022(A6)
```

If you wished to create a symbol for the second number on this line, "$0022", you should first select "Second". The setting of "Select field" will go back to "First" with virtually any function you use, therefore when creating a symbol for the second field in a line of code, use "SYMBOLS 1/Select/Second" only just prior to actually creating the symbol. "Third" and "Fourth" fields may be necessary when disassembling 68020 effective addresses.

**&H0270 SYMBOLS/Object/Accumulator**
**&H0268 SYMBOLS/Object/Cursor**

When creating symbols using ReSource's inbuilt symbol bases, the number, used to find the string will come from the cursor line. You can instead have ReSource get the number from the accumulator, by selecting "Accumulator". In this case, the output string will be placed back into the accumulator. To set it back, select "Cursor".

## Menu Functions

### &H002C  CURSOR/Remember

"Push" the current cursor location on ReSource's location stack.
Other functions that call this function are:

"CURSOR/Relative/Next hilite"
"CURSOR/Relative/Next error line"
"CURSOR/Relative/Next D/T change"
"CURSOR/Relative/Previous D/T change"
"CURSOR/Absolute/End of file"
"CURSOR/Absolute/Start of file"
"CURSOR/Absolute/Forward reference"
"CURSOR/Absolute/Second forward reference"
"CURSOR/Absolute/Backward reference"
"CURSOR/Normal search/Set search string"
"CURSOR/Pattern search/Set search pattern"
"CURSOR/Buffer search/Set search string"
"CURSOR/Search/Search from start of file"
"CURSOR/Search/Search from end of file"

To "pop" a cursor location, you use the function:
"CURSOR/Absolute/Previous location"

The stack is cleared when you load a new file, or when you select:
"CURSOR/Clear loc stack"

The stack is NOT stored within ".RS" files.

### &H005F  CURSOR/Clear Loc stack

Clear the cursor location stack.  See "CURSOR/Remember"

### &H0011  CURSOR/Relative/Next byte

The cursor location is incremented.  If at the last location within the current file, a macro "fail" will result.

### &H0012  CURSOR/Relative/Previous byte

The cursor location is decremented.  If at the first location within the current file, a macro "fail" will result.

### &H000A  CURSOR/Relative/Next line

Scroll forward one line of code/data.  If at the last line of code/data in the file already, a macro "fail" will result.  The speed at which this scroll is done is set by the "CURSOR/Scrolling speed" settings.

### &H0009  CURSOR/Relative/Previous line

Scroll backwards one line of code/data.  If at the first line of code/data in the file already, a macro "fail" will result.  The speed at which this scroll is done is set by the "CURSOR/Scrolling speed" settings.

## Menu Functions

### &H0186  CURSOR/Relative/Next label

Move cursor to the next location that has a label attached to it.  If one cannot be found, a macro "fail" will result.

### &H0187  CURSOR/Relative/Previous label

Move cursor to the previous location that has a label attached to it.  If one cannot be found, the start of the file will become the new cursor position.

### &H019E  CURSOR/Relative/Next symbol
### &H019F  CURSOR/Relative/Previous symbol

Move cursor to the next/previous location that has a symbol attached to it. If one cannot be found, a macro "fail" will result.  Note that this location may be in the middle of a line of code.  Where the actual operand is stored will become the new cursor position.

In a macro, if you wish to find the start of a line that contains a symbol, use the following sequence after you have found the symbol:

CURSOR/Relative/Next byte
CURSOR/Relative/Previous line

### &H0197  CURSOR/Relative/Next Section

Move cursor to the start of the next section (hunk).  If this is the last section in the file, a macro "fail" will result.

### &H0198  CURSOR/Relative/Previous Section

Move cursor to the start of this section (hunk).  If already at the start of a section, move to the start of the previous section.  If already at the start of the first section, a "macro fail" will result.

### &H01A0  CURSOR/Relative/Next reloc32

Move cursor to the start of the next relocated longword.  If one cannot be found, a macro "fail" will result.  Note that this may be in the middle of a line of code.

### &H01A1  CURSOR/Relative/Previous reloc32

Move cursor to the start of the previous relocated longword.  Note that this may be in the middle of a line of code.  If one cannot be found, the start of the file will become the new cursor position.

### &H000D  CURSOR/Relative/Next page

Moves cursor forwards approximately three quarters of a screenful.

## Menu Functions

**&H000C  CURSOR/Relative/Previous page**

Moves cursor backwards approximately three quarters of a screenful.

**&H00F9  CURSOR/Relative/Skip forward**

Move cursor backward approximately 4K.

**&H00FA  CURSOR/Relative/Skip Backward**

Move cursor forward approximately 4K.

## Menu Functions

### &H02F6 CURSOR/Relative/Next error line

The purpose of the "CURSOR/Relative/Next error line" function is to find areas that MAY not be disassembled properly.  There are currently 10 types of "errors" that can be detected.  Each of these can be independantly enabled or disabled (via the "OPTIONS/Error detection/" submenu items):

1. Code terminate :  This is where code finishes, and data starts, where the last line of code was NOT one of the "normal" code terminators, such as "RTS", "RTE", "BRA", "JMP", etc.

2. Missing label :  The code or data line immediately following a valid code terminating line (such as "RTS", "RTE", "BRA", "JMP" etc) should have a label.

3. Bad alignment :  Word or longword data lines should not start on an odd address.

4. Code reference : An instruction that normally references code, is referencing data instead.  For example, what a "DBRA" instruction references must NEVER be data.

5. Data reference : An instruction that normally references data is referencing code instead.  For example, what a "CLR" instruction references must NEVER be code.

6. START+ : There should be no "START+" references left in the disassembly at save .asm time.

7. AFLINE :  Any lines producing an "AFLINE" instruction are detected.

8. Library calls : Any JSR or JMP instruction with an effective address of the "Address register indirect with displacement" type, where the address register is A6, and the offset is an even multiple of minus 6, is detected. These are usually calls to system libraries.

9. Illegal code : Illegal opcodes, and extremely unusual opcodes.  "Unusual" opcodes include the "TAS" instruction, branches with an odd offset, ADDQ and SUBQ with the stack pointer as the destination, and an odd number for the source, using the "EXG" for exchanging a register with itself, using "LINK" or "UNLK" with the stack pointer, etc.

10. Sym/EQU values : Whenever a symbol is created, the symbol and symbol value are stored in the "equates table".  If you have somehow managed to give two different values the same symbol name, any lines containing the symbol with the symbol value that differs from that stored in the equates table, will be detected.

The purpose of the "CURSOR/Relative/Next error line" function is to find areas that MAY not be disassembled properly.  There will be cases where the code/data has been disassembled correctly, but will still get picked up.

Note that when a "Calculate .asm size" or "Save .asm" is performed, lines that are picked up by the above function are pushed onto the cursor location stack.  If "error comments" are enabled, you will also get some extra full-line comments in your source code, complaining about the errors that were detected.

## Menu Functions

### &H003D  CURSOR/Relative/Next unparsed code

Move cursor forward to the next location that satisfies both of the following conditions:

1. The data type has been set to CODE
2. This location has NOT been scanned for forward references previously.

This function is normally used in conjuction with the "LABELS/Create multiple/All" function.

### &H0001  CURSOR/Relative/Next D/T change

Moves cursor forward to the next change of data type.  For example, if the cursor line is currently displaying code, the cursor will move forward to the next line of bytes/words/longwords/ascii.

### &H0002  CURSOR/Relative/Prev D/T Change

Moves cursor backward to the first encountered change of data type.  For example, if the cursor line is currently displaying code, the cursor will move backward to a line of bytes/words/longwords/ascii.

### &H0044  CURSOR/Relative/Next uncertain D/T

Move cursor forward to the next location that has the "Uncertain" bit set in the attribute table.  This bit indicates that ReSource really wasn't sure of the data type of this byte.  It is a good idea to check these locations after using the "PROJECT/Disassemble" function.

### &H0046  CURSOR/Relative/Next backward ref

Once you have used the "CURSOR/Absolute/Backward reference" function, you can use this function to get to other backward references.

### &H02CE  CURSOR/Relative/Next hilite

Move cursor forward to the next line which has any form of hiliting.  By changing the type of hiliting, in the DISPLAY/Hiliting" menu, you can tailor the search for various attribute types to your own needs.

### &H0016  CURSOR/Absolute/End of file

Move cursor to the start of the last line of the file.  Be aware that there may be a delay, while ReSource properly determines the start of the last line.

### &H0015  CURSOR/Absolute/Start of file

Move cursor to the start of the file.

## Menu Functions

### &H0040  CURSOR/Absolute/Forward reference

If there is a reference to a position within the current file and within the current line of code, move cursor to the location being referenced.  If there are two forward references within the current line, the first is used.  An exception to this is if you have selected any of the "SYMBOLS 1/Select field/:" functions other than "First".

It is also possible to forward-reference from symbols which themselves reference labels, i.e.:

```
                    ADD.W     D0,D0
                    MOVE.W    JumpBase(PC,D0.W),D0
                    JMP       JumpBase(PC,D0.W)

JumpBase            DW        FirstOffset-JumpBase
                    DW        SecondOffset-JumpBase
                    DW        ThirdOffset-JumpBase
                    DW        FourthOffset-JumpBase
FirstOffset  ...
```

Any of the symbols on the "DW" lines, are each defined as the distance between two labels.  Each of the symbols on these lines is evaluated by subtracting the offset of "JumpBase" from "FirstOffset", or "SecondOffset", etc.

### &H0041  CURSOR/Absolute/Second forward reference

Similar to "CURSOR/Absolute/Forward reference", except that the second reference is selected, rather than the first, as default.

### &H0047  CURSOR/Absolute/Backward reference

Starting from the start of the file, search for references to the cursor location.  If there are several references, you can use "CURSOR/Relative/Next backward reference" to get to successive references.

### &H003F  CURSOR/Absolute/Previous location

"Pop" the cursor location from the top of the cursor location stack.  If the stack is empty, a macro "fail" will result.

### &H02EF  CURSOR/Absolute/Swap W/Previous

Swaps the cursor location with the one on the top of the cursor location stack.

### &H01AD  CURSOR/Absolute/Specify offset

You will be asked to supply an offset representing how far from the start of the file, the cursor location should be.  The number may be specified in hexadecimal, decimal, or binary.  If hexadecimal, the string must start with "$", if binary, it must start with "%".

## Menu Functions

**&H01AE  CURSOR/Absolute/Specify label**

You will be asked to supply the name of a label.  If the label is used within the current file, its location will become the new cursor location.  Note that this may be in the middle of a line of code.  If there is more than one label with the name that you supply, the first label bearing that name will be used.  See "CURSOR/Label search/Find next occurrence".

**&H01BC  CURSOR/Absolute/Specify symbol**

You will be asked to supply the name of a symbol.  If the symbol is used within the current file, its location will become the new cursor location.

Note that this may be in the middle of a line of code.  Also note that if there is more than one symbol of that name, ReSource will pick one closest to the start of the file.  See "CURSOR/Symbol search/Find next occurrence".

**&H01AF  CURSOR/Absolute/Specify percentage**

You will be asked to supply a decimal percentage of the current file, where the cursor will be positioned.  In this case 0 is the beginning of the file and 99 is the end.  The actual positioning will not be exact.  It will generally be placed a little before the required percentage.

**&H01B2  CURSOR/Copy/Clip #1   &H01B5  CURSOR/Paste/Clip #1**
**&H01B3  CURSOR/Copy/Clip #2   &H01B6  CURSOR/Paste/Clip #2**
**&H01B4  CURSOR/Copy/Clip #3   &H01B7  CURSOR/Paste/Clip #3**
**&H01B8  CURSOR/Swap/Clip #1**
**&H01B9  CURSOR/Swap/Clip #2**
**&H01BA  CURSOR/Swap/Clip #3**

ReSource has 3 internal cursor "clipboards".  You can copy a cursor location to any of these, and then later "paste" one of these locations, which in effect moves the cursor to where it was when you used the "copy" function.

Also, you can swap cursor locations, between the current, and a stored cursor location.  The contents of these cursor "clipboards" are cleared when you load a new file.

## Menu Functions

**&H003A  CURSOR/Scrolling speed/Very fast**
**&H0039  CURSOR/Scrolling speed/Fast**
**&H0038  CURSOR/Scrolling speed/Normal**
**&H0037  CURSOR/Scrolling speed/Slow**
**&H0036  CURSOR/Scrolling speed/Very slow**

Whenever you use either "CURSOR/Relative/Next line" or "CURSOR/Relative/Previous line" functions, the display will scroll one text line, N pixels at a time:

| Scrolling Speed | Lines Scrolled | Notes |
|---|---|---|
| Very fast | 8 | Exceptionally fast scrolling, good for scanning through a file. |
| Fast | 4 | Reasonably fast scrolling, allows scanning, but difficult to read. |
| Normal | 2 | Reasonably smooth scrolling, allowing you to read text fairly easily WHILE it is scrolling. |
| Slow | 1 | Extremely smooth scrolling, allowing you to easily read text WHILE it is scrolling. |
| Very Slow | 1 | Exceptionally smooth scrolling, with a delay, excellent for slowly browsing through a file. |

Note that by using the mouse to scroll, you can select any of the above mentioned scrolling rates.

**&H001E  LABELS/Create single/End-of-line comment**

You will be asked to supply a string of less than 240 characters, which will become a comment attached to the end of the cursor line.  Only one end-of-line comment is allowed per line; if you create one on a line that has already got one, it will replace the old comment.

**&H001F  LABELS/Create single/Full-line comment**

You will be asked to supply a string of less than 240 characters, which will become a comment to be attached to the start of the cursor line.  Multiple full-line comments are allowed, and successive comments will be displayed AFTER previously defined full-line comments.  Normally, full-line comments will be displayed starting with a semicolon (";"). However, if you start the string with a semicolon, the logic is reversed, and it becomes an extra line of code/data.

Thus, you can insert extra lines of code/data, before saving the source code to a text file.  By creating a label that starts with a semicolon for a given line, when that code is assembled, that line of code/data is effectively treated as a comment only, therefore you can both insert and effectively remove code/data using ReSource, during the disassembly process.

## Menu Functions

### &H001D  LABELS/Create single/Label

You will be asked to supply a string of less than 240 characters, which will become a label to be attached to the cursor line.  Duplicate labels are NOT allowed, unless they start with a semicolon (";"), or an asterisk ("*").

Creating a label that starts with "*" or ";" effectively makes the rest of the line a comment.  To safely create unique labels during macros, use the "STRINGS/Define/Acm", and then the "STRINGS/Put/Label" functions.  String indirection is supported in labels, symbols and comments.

When creating local labels, the uniqueness check is different to that done for normal labels.  The scope of local labels (and hence the scope of the duplicate checking) is limited to between the most previous, and next, normal label.  ReSource supports local labels of the "1$" type, and also user-friendly local labels as used in Macro68 (".loop", ".strcpy" etc).

### &H0034  LABELS/Create single/Label - fwd ref

The current line of code will be searched for references to positions within the current file.  If any are found, ReSource will make a decision on which type of data is at the position referenced.  It will then set the data type (unless it has already been set), and create a label at that offset (unless a label has already been defined for that location). This new label will be immediately used for all references to that location, which of course includes the reference within the current line.  If there are no references, or there is a reference, but it is outside of the range of the current file, then this function will do nothing.  Normally, this function will only be used within macros, as the "PROJECT/Disassemble" normally will create all necessary labels.

## Menu Functions

### &H0006  LABELS/Edit single/Symbol

You will be asked to supply a string, which will replace one of the numbers on the current line.  A symbol may be created for the first, second, third of fourth number on the current line.  To select which one, you must first select one of the following:

"SYMBOLS 1/Select field/First (default)"
"SYMBOLS 1/Select field/Second"
"SYMBOLS 1/Select field/Third"
"SYMBOLS 1/Select field/Fourth"

As noted above, the selection defaults to the first number.
For example, if you supplied the string "MyColdStartData", the following line:

```
                    MOVE.L    D0,(4,A0)
```

would become:

```
                    MOVE.L    D0,(MyColdStartData,A0)
```

If there are no numbers in the current line, a macro "fail" will result.
If the user creates a symbol which is currently also defined as a label, a DisplayBeep() will be generated.

If an attempt is made to create a symbol, using the same name as a previously defined symbol, but for a different symbol value, the symbol creation will fail, and a DisplayBeep() will be generated.

See "OPTIONS/Allow.../EQUate value checks

### &H0043  LABELS/Edit single/End-of-line comment

You will be asked to supply a string of less than 240 characters, which will become a comment to be attached to the end of the cursor line.  Only one end-of-line comment is allowed per line; if you create one on a line that has already got one, it will replace the old comment.  Selecting "Edit" rather than "Create" end-of-line comment will allow place the existing comment in the string requester.

## Menu Functions

### &H0042  LABELS/Edit single/Full-line comment

You will be asked to supply a string of less than 240 characters, which will become a comment to be attached to the start of the cursor line, but after any existing full-line comments.  Multiple full-line comments are allowed, and successive comments will be displayed AFTER previously defined full-line comments.

Normally, full-line comments will be displayed starting with a semicolon (";"). However, if you start the string with a semicolon, the logic is reversed, and it becomes an extra line of code/data.  Thus, you can insert extra lines of code/data, even before saving the source code to a text file. By creating a label that starts with a semicolon for a given line, when that code is assembled, that line of code/data is effectively treated as a comment only, therefore you can both insert and effectively remove code/data using ReSource, during the disassembly process.

When "Edit" is selected rather than "Create" full-line comment, the first existing full-line comment is replaced with the one that you supply.  If you need to edit a full-line comment other than the first one, use the "LABELS/Edit single/Rotate F/L comments" function first.

### &H003E  LABELS/Edit single/Label

Similar to the "LABELS/Create single/Label" function, except that if there is already a label/symbol/comment defined, you will have the chance to edit that string, rather than type the entire new string into the requester.

Like all requests for a string, you can use indirection, so that ReSource gets the string from either the accumulator, or from one the buffers A-M.

## Menu Functions

### &H0359  LABELS/Create single/Symbol
### &H0007  LABELS/Edit single/Symbol - dest

You will be asked to supply a string, which will replace one of the numbers on the current line.  A symbol may be created for the first, second, third of fourth number on the current line.  To select which one, you must first select one of the following:

"SYMBOLS 1/Select field/First (default)"
"SYMBOLS 1/Select field/Second"
"SYMBOLS 1/Select field/Third"
"SYMBOLS 1/Select field/Fourth"

As noted above, the selection defaults to the first number.
For example, if you supplied the string "MyColdStartData", the following line:

```
                    MOVE.L    D0,(4,A0)
```

would become:

```
                    MOVE.L    D0,(MyColdStartData,A0)
```

If there are no numbers in the current line, a macro "fail" will result.
If the user creates a symbol which is currently also defined as a label, a DisplayBeep() will be generated.

If an attempt is made to create a symbol, using the same name as a previously defined symbol, but for a different symbol value, the symbol creation will fail, and a DisplayBeep() will be generated.

See "OPTIONS/Allow.../EQUate value checks

### &H0358  LABELS/Edit single/Rotate F/L comments

Rotate the order of the full-line comments on the current line.  This may be required when you wish to edit the second or subsequent full-line comment on a line.

### &H0184  LABELS/Replace single/Label

Create a "shop" label at the cursor position.  A "shop" label is one that is 9 characters long, begins with "lb", has the data type immediately following, and ends with the code offset.  It may also be of some other length, and end with ".MSG".

The "data type known" attribute will be set also.

### &H0005  LABELS/Remove single/Label

Removes the label from the current line.  This does *no* changes of data type.  See "DISPLAY/Set data type/Unknown"

## Menu Functions

### &H0155  LABELS/Remove single/Symbol

Removes a symbol from the current line.  Use "SYMBOLS 1/Select field/:" to select which symbol to remove.

### &H037A  LABELS/Remove single/EQUate

You will be asked for the name of a symbol.  If there is an entry for that symbol in the equates table, it will be removed.

### &H02F5  LABELS/Remove single/Symbol & EQUate

Removes the symbol from the current selected field, as well as the equate table entry for that symbol, should there be one.

### &H0004  LABELS/Remove single/End-of-line comment

Removes the end-of-line comment from the current line.

### &H0003  LABELS/Remove single/Full-line comment

Removes the first full-line comment attached to the current line.

See "LABELS/Edit single/Rotate comments"

### &H000F  LABELS/Remove single/All

Removes any labels, symbols, and comments from the current line, and then sets the data type to "unknown".  This is useful when a bogus reference was made, as when ReSource (or you) thought code was being scanned, and it was actually data or ASCII.

### &H0029  LABELS/Create multiple/Reloc32 only

For each reloc32 pointer within the current file, determine and set the data type being referenced, and create a label at that location.  This function will be called automatically whenever you load a load file that has at least one reloc32 pointer, unless you override this option by selecting "OPTIONS/Allow../Auto labels/OFF".

## Menu Functions

### &H003B  LABELS/Create multiple/All

This function is used internally in the "PROJECT/Disassemble" function. It remains accessible for users that want more control over the disassembly process.

Starting at the current line, the "LABELS/Create single/Label - fwd ref" function is executed, and if ReSource thinks that there was valid code, the "CURSOR/Relative/Next line" function is executed, and the entire function loops, otherwise exits.

Care must be taken when ReSource finishes parsing a section of code to ensure that the next section selected is really code. Otherwise, there is a chance that what ReSource thinks is valid code is really ASCII, and invalid labels will be created. The best way to accomplish this is to select "CURSOR/Relative/Next unparsed code" before selecting this function again. When creating "shop" ASCII labels, if no valid text can be found to use in the label name, instead of creating a label starting with "lbA", ReSource creates the label "ascii.MSG", or "ascii.MSG0", or "ascii.MSG1", etc.

References to the START+$FFFFFFFC and START+$FFFFFFF8 are shown as "ProgStart-4" and "ProgStart-8" respectively. If the label "ProgStart" is not defined anywhere, it will be displayed as a label attached to the first byte of the current file. You can still have some other label attached to the start of the file.

### &H03B6  MACROS 1/End macro

Use this function to end the creation of a macro. Pressing rightAmiga-A (abort) will also end a macro.

### &H0203  MACROS 2/Execute/(rename MACROS 2)

You will be asked to supply a string, of less than 24 bytes, which will be displayed at the top of the "MACROS 1/Execute/" and "MACROS 1/Create/" submenu items.

### &H0201  MACROS 1/Execute/(rename MACROS 1)

You will be asked to supply a string, of less than 24 bytes, which will be displayed at the top of the "MACROS 2/Execute/" and "MACROS 2/Create/" submenu items.

### &H0326  MACROS 3/Execute/(rename MACROS 3)

You will be asked to supply a string, of less than 24 bytes, which will be displayed at the top of the "MACROS 3/Execute/" and "MACROS 3/Create/" submenu items.

### &H016D  MACROS 1/Load macros

Asks you to select a file, which will be loaded as a ReSource macro file. This normally done automatically at startup, loading the "S:RS.Macros" file.

## Menu Functions

**&H016E  MACROS 1/Save all macros**

Asks you to select a file, which will be used to save all currently defined macros to.  This function is called within the "PROJECT/Save configuration" function.  Normally, you will use the "S:RS.Macros" file, as ReSource will always load this file during startup.

**&H035A  MACROS 1/Execution speed/Fastest**
**&H0200  MACROS 1/Execution speed/Very fast**
**&H019B  MACROS 1/Execution speed/Fast**
**&H019C  MACROS 1/Execution speed/Slow**
**&H01A4  MACROS 1/Execution speed/Very slow**
**&H019D  MACROS 1/Execution speed/Wait on mouse**

The "MACROS 1/Execution speed/:" functions allow you to control the speed at which macros will be executed.  Normally, you will probably want macros to execute as fast as possible, however if you require more feedback as to what exactly the macro is doing, you may select a slower speed.  In particular, the "Wait on mouse" selection allows you to "single step" a macro, and if you also have the "DISPLAY/Titlebar info/Function names" function selected, you will see the name of the function that will be executed next, in the titlebar.  When you click the LMB, that function will execute, and the titlebar will again display the name of the next function to execute.

The "Fastest" selection will disable all types of output during macro execution, including the updating of the titlebar.  If "Very fast" is selected, only the titlebar will be updated, the rest of the screen will not be.  This normally represents a 5% difference in macro execution speed.

Selecting "Fast" will allow screen updates.  Selecting "Slow" will give a short delay after each screen update.  Selecting "Very slow" will give a more substantial delay after each screen update.

**&H0220  MACROS 1/Commentary/Full**
**&H0221  MACROS 1/Commentary/Heavy**
**&H0222  MACROS 1/Commentary/Normal**
**&H0223  MACROS 1/Commentary/Light**
**&H0224  MACROS 1/Commentary/None**

Sets the commentary level in macros.

## Menu Functions

**&H0210  MACROS 1/Directives/Start conditional**
**&H0211  MACROS 1/Directives/End conditional**

While creating a macro, you may wish to use conditional execution of parts of the macro. Whenever a function is executed that generates a "macro fail", the macro processor searches forward in the macro for and "end conditional".

If it does not find one, the macro will exit, ignoring all of the functions that were skipped over while searching for the "end conditional".
If, while searching for an "end conditional", a "start conditional" is found, the macro processor for searches for the matching "end conditional" for the just-found "start conditional", and then continues searching for the original "end conditional". In short, you may nest conditionals by placing the "start conditional" and "end conditional"s at the appropriate places in a macro.
Example:

The following macro will perform the equivalent of:
"If the word "MOVE" is found, and there is a label on that line, parse the line, else create a shop label, else go to the start of the file":

```
CURSOR/Normal search/Set search string   "MOVE"
MACROS 1/Directives/Start conditional
CURSOR/Normal search/Find next occurrence
MACROS 1/Directives/Start conditional
STRINGS/Get/Label
LABELS/Create single/Label - fwd ref
MACROS 1/Next macro label/#1
MACROS 1/Directives/End conditional
LABELS/Replace single/Label
MACROS 1/Set macro label/#1
MACROS 1/Next macro label/#2
MACROS 1/Directives/End conditional
CURSOR/Absolute/Start of file
MACROS 1/Set macro label/#2
```

**&H022C  MACROS 1/Interactive/ON**

While creating a macro, you may be presented with various requesters, that require a string. When you run the macro, the requesters do not normally appear, instead the functions that require strings get them from inside the macro body. By selecting "MACROS 1/Interactive/ON", you can force ReSource to display requesters during macro execution, allowing you to view the strings that were used during the macro creation, and you may even change them for this macro invocation.

## Menu Functions

**&H0247  MACROS 1/Set macro label/#1**
**&H0248  MACROS 1/Set macro label/#2**
**&H0249  MACROS 1/Set macro label/#3**
**&H024A  MACROS 1/Set macro label/#4**
**&H024B  MACROS 1/Set macro label/#5**
**&H024C  MACROS 1/Next macro label/#1**
**&H024D  MACROS 1/Next macro label/#2**
**&H024E  MACROS 1/Next macro label/#3**
**&H024F  MACROS 1/Next macro label/#4**
**&H0250  MACROS 1/Next macro label/#5**
**&H0251  MACROS 1/Previous macro label/#1**
**&H0252  MACROS 1/Previous macro label/#2**
**&H0253  MACROS 1/Previous macro label/#3**
**&H0254  MACROS 1/Previous macro label/#4**
**&H0255  MACROS 1/Previous macro label/#5**

Macro labels allow you to create control constructs within macros.  For example, using "MACROS 1/Set macro label/#1" is in effect defining a label with the macro body.  If you want the macro to loop back to this label, you would use the "MACROS 1/Previous macro label/#1" function, at some point past where "MACROS 1/Set macro label/#1".  Similarly,
"MACROS 1/Next macro label/#1" will skip forward, to the next defined macro label #1.

**&H01FE  MACROS 1/Suspend learn/Suspend**

While creating a macro, you may have to use some functions in order to continue the macro creation, but you do not want the functions actually recorded into the macro.  In this situation, you can select "suspend", do what you have to do, and then turn "suspend" off again, to continue the macro creation as normal.

**&H0209  STRINGS/Get/Label**

If there is a label defined at the cursor location, it will be copied to the accumulator.

**&H020A  STRINGS/Get/Symbol**

Copy the symbol from the selected field on the current line to the accumulator.

**&H020B  STRINGS/Get/Symbol - dest**

Copy the symbol from the second field on the current line to the accumulator.

**&H020E  STRINGS/Get/Symbol value**

Copy the symbol value from the selected field on the current line to the accumulator.

**&H020F  STRINGS/Get/Symbol value - dest**

Copy the symbol value from the second field on the current line to the accumulator.

## Menu Functions

### &H020C  STRINGS/Get/End-of-line comment

Copy the end-of-line comment attached to the current line to the accumulator.

### &H020D  STRINGS/Get/Full-line comment

Copy the first full-line comment attached to the current line to the accumulator.  See "LABELS/Edit single/Rotate F/L comments".

### &H0232  STRINGS/Get/Search string

Copy the current search string to the accumulator.

### &H0279  STRINGS/Get/Filename

Copy the current filename to the accumulator.

### &H0240  STRINGS/Get/Save .asm name

If the current file has already been saved as source code (to a ".asm" file), copy the filename used to the accumulator.  If the current file has not already been saved, the name will be the same as the current filename, with ".asm" appended.

### &H0241  STRINGS/Get/Save .RS name

If the current file has already been saved to a .RS file, copy the filename used to the accumulator.  If the current file has not already been saved, the name will be the same as the current filename, with ".rs" appended, unless ".rs" is already appended.

### &H0244  STRINGS/Get/Macros filename

Copy the current macros filename to the accumulator.

### &H021F  STRINGS/Get/Keytable filename

Copy the current keytable filename to the accumulator.

### &H0243  STRINGS/Get/Cursor byte

Write the data where the cursor is to the accumulator in hexadecimal, decimal, or binary, depending on the setting of "STRINGS/Accumulator/".  This may be a byte, word, or longword, depending on the setting of "STRINGS/Operand size".

### &H0245  STRINGS/Get/Cursor offset

The currently displayed cursor location is written into the accumulator in the current numeric base as a longword (the "operand size" is set to longwords).

## Menu Functions

### &H0245  STRINGS/Get/Cursor offset

### &H027B  STRINGS/Get/Accumulator length

The length of the string currently in the accumulator is written into the accumulator in the current numeric base.  The 'operand size' setting is not affected, but is used to decide how many leading zeroes to use.

### &H027C  STRINGS/Get/Partial save size

The number representing the distance in bytes, between the partial save end and the partial save start is written into the accumulator in the current numeric base.  The 'operand size' is set to longwords.

### &H0242  STRINGS/Get/File

You will be asked to supply the name of a file, of which up to 240 characters will be copied into the accumulator.  The data is copied "as-is", no hex to ASCII translation is performed.  If you need to load a file that is larger than 240 bytes, use the "STRINGS/Input buffer/Load", and then you can read parts of it using the "STRINGS/Input buffer/Read $$$" functions, where "$$$" can be line, byte, word, longword or character.

### &H02C6  STRINGS/Get/Attribute bits

The attribute bits for the current location are copied into the accumulator, using the current numeric base (hex/decimal/binary).  The definitions for these bits are described under "DISPLAY/Titlebar info/Attributes".

### &H0213  STRINGS/Put/Label

If it is possible to use the string within the accumulator as a label on the current line without creating a duplicate label, it will be done.  Else, if it is already used as a label, then a digit is appended to it, and again it is tested to see if a label already exists of that name.  This continues until a unique label is created, then that label is attached to the current line.

This is a safe way of creating labels during macros.

If the label in the accumulator is of the local type, it will be used with no duplicate check being done.

### &H02C7  STRINGS/Put/Attributes

If the accumulator contains a valid number, that number will overwrite the attributes for the current location.  The definitions for these bits are described under "DISPLAY/Titlebar info/Attributes".  This function is within the "dangerous" category.  Do not experiment with this function on files that you haven't got a backup of.

## Menu Functions

### &H02D2  STRINGS/Put/Base register

The support for base register-relative addressing allows any address register to be used.  This can be done by using either "STRINGS/Put/Base register", or "SPECIAL FUNCTIONS/Specify Base Register".  To use this function, you must first place a number between 0 and 7 inclusive, into the accumulator.  The current base register in use will be visible in the "SPECIAL FUNCTIONS/ Convert (xx,An) EA's" function; you may see "(xx,A6)" instead of "(xx,A4)", for example.  For use in macros that must be able to calculate and enter the base register number, this function may be used rather than SPECIAL FUNCTIONS/Specify base register".

### &H0205  STRINGS/Edit functions/Clip start

You will be asked to supply a string.  Starting from the left, each character in the supplied string is compared to the character in the accumulator at that position, and if they are equal, the next character is compared, and so on.  When either the end of one of the strings is found, or when a mismatch is found, the characters in the accumulator that matched are deleted.  The question mark ("?") is used as a wildcard, and you can use the asterisk once only, to "delete everything up to and including" the following character. Examples follow:

| Accumulator | User-supplied string | Resulting accumulator |
|---|---|---|
| MOVE.L | M | OVE.L |
| MOVE.L | MUVE.L | OVE.L |
| MOVE.L | MOV.L | E.L |
| MOVE.L | move.l | MOVE.L |
| MOVE.L | M?V | E.L |
| MOVE.L | ??? | E.L |
| MOVE.L | *. | L |
| MOVE.L | *M | OVE.L |
| MOVE.L | ?MOVE.L | OVE.L |
| _LVOAllocMem | _LVO | AllocMem |
| $00000004 | $000000 | 04 |

### &H0206  STRINGS/Edit functions/Clip end

Similar to the "STRINGS/Edit functions/Clip start" function, except that clipping is performed on the end of the accumulator string.  The question mark is still used as a wildcard, but the asterisk does not have a special meaning in this function.  Examples follow:

| Accumulator | User-supplied string | Resulting accumulator |
|---|---|---|
| MEMF_PUBLIC | ?PUBLIC | MEMF |
| MEMF_PUBLIC | ??? | MEMF_PUB |
| MEMF_PUBLIC | ?LC | MEMF_PUBLI |
| meMcollAOVL_ | OVL_ | meMcollA |

## Menu Functions

### &H0207 STRINGS/Edit functions/Prepend

You will be asked to supply a string, which will be prepended to (added to the beginning of) the accumulator, providing that the resulting string is no longer than 240 characters. Examples follow:

| Accumulator | User-supplied string | Resulting accumulator |
|---|---|---|
| PUBLIC | MEMF_ | MEMF_PUBLIC |
| AllocMemSUB | _ | _AllocMemSUB |
| PUBLIC | CHIP | CHIPPUBLIC |

### &H0208 STRINGS/Edit functions/Append

You will be asked to supply a string, which will be appended to (added to the end of) the accumulator, providing that the resulting string is no longer than 240 characters. Examples follow:

| Accumulator | User-supplied string | Resulting accumulator |
|---|---|---|
| PUBLIC | MEMF_ | PUBLICMEMF_ |
| AllocMemSUB | _ | AllocMemSUB_ |
| PUBLIC | CHIP | PUBLICCHIP |
| AllocMem | SUB | AllocMemSUB |

### &H0278 STRINGS/Edit functions/Reverse

The contents of the accumulator are reversed. Examples follow:

| Accumulator | Resulting accumulator |
|---|---|
| PUBLIC | CILBUP |
| AllocMemSUB | BUSmeMcollA |
| AllocMem | meMcollA |

### &H027A STRINGS/Edit functions/Lower case

Any upper case characters in the accumulator are changed to their lower case counterparts. Examples follow:

| Accumulator | Resulting accumulator |
|---|---|
| PUBLIC0123 | public01243 |
| AllocMemSUB | allocmemsub |
| AllocMem | allocmem |

## Menu Functions

**&H025D  STRINGS/Operand size/Byte**
**&H025E  STRINGS/Operand size/Word**
**&H025F  STRINGS/Operand size/Longword**

Some functions in ReSource require to know whether to work at the byte, word, or longword level.  For example, "STRINGS/Get/Cursor value" could get a byte, word, or longword from the cursor position, and this is your way of specifying at which level you wish those functions to work.  Generally, they specify the size of the numeric operand in the accumulator.  Some functions that write a numeric value into the accumulator will set the operand size themselves, others that get the value out of the accumulator need to know the operand size.

**&H029C  STRINGS/Accumulator/Hex**
**&H029D  STRINGS/Accumulator/Decimal**
**&H029E  STRINGS/Accumulator/Binary**

Options control whether to write the results of any maths functions to the accumulator in hex, decimal, or binary.

**&H0216  STRINGS/Maths functions/Increment**

The increment function adds one to the accumulator.  If the resulting number develops a carry (using the current operand size), a macro 'fail' will result.  Thus, loops in macros can exit after a given number of iterations, rather than a failed search, etc.

**&H0217  STRINGS/Maths functions/Decrement**

The decrement function subtracts one from the accumulator.  If the resulting number develops a carry (using the current operand size), a macro 'fail' will result.  Thus, loops in macros can exit after a given number of iterations, rather than a failed search, etc.

**&H022E  STRINGS/Maths functions/Add**

You will be asked for a number, to be added to the accumulator.  If a carry results, a macro 'fail' will occur.

**&H022F  STRINGS/Maths functions/Subtract**

You will be asked for a number, to be subtracted from the accumulator.  If a carry results, a macro 'fail' will occur.

**&H0230  STRINGS/Maths functions/Multiply**

You will be asked for a number, to be multiplied by the number within the accumulator.  If a carry results, a macro 'fail' will occur.  Full 32-bit multiplies are supported, however the setting of 'operand size' is used when determining the size of the operands.

## Menu Functions

### &H0231  STRINGS/Maths functions/Divide

You will be asked for a number.  The contents of the accumulator will be divided by the number that you specify.  The divide function does NOT operate on 32-bit operands, but otherwise will work at the currently set operand size.  If the current operand size is set to longwords, the divide will be performed on the lower 16 bits only.

### &H025C  STRINGS/Maths functions/Negate

The number within the accumulator is negated, using the currently set operand size.  If the result is zero, a macro 'fail' will result.

### &H02F2  STRINGS/Maths functions/Clear

Place a zero character ("0") into the accumulator.

### &H025B  STRINGS/Maths functions/Logical NOT

The number within the accumulator is NOT'd in bitwise fashion, using the currently set operand size.  If the result is zero, a macro 'fail' will result.

### &H0258  STRINGS/Maths functions/Logical AND

You will be asked to supply a number, which will be bitwise AND'd with the contents of the accumulator, and the result is stored back into the accumulator.  If the result is zero, a macro 'fail' will result.

### &H0259  STRINGS/Maths functions/Logical OR

You will be asked to supply a number, which will be bitwise OR'd with the contents of the accumulator, and the result is stored back into the accumulator.  If the result is zero, a macro 'fail' will result.

### &H025A  STRINGS/Maths functions/Exclusive OR

You will be asked to supply a number, which will be bitwise exclusive-OR'd with the contents of the accumulator, and the result is stored back into the accumulator.  If the result is zero, a macro 'fail' will result.

### &H02D1  STRINGS/Maths functions/ROL

You will be asked for a number, which is used in determining how many times to rotate left the accumulator.  The accumulator size is used to determine whether 8, 16, or all 32 bits will be rotated.  If the accumulator does not contain a valid numeric value, a macro 'fail' will result.

## Menu Functions

### &H02D0  STRINGS/Maths functions/ROR

You will be asked for a number, which is used in determining how many times to rotate right the accumulator.  The accumulator size is used to determine whether 8, 16, or all 32 bits will be rotated.  If the accumulator does not contain a valid numeric value, a macro 'fail' will result.

### &H02F0  STRINGS/Maths functions/LSL

You will be asked for a number, which is used in determining how many times to logical shift left the accumulator.  The accumulator size is used to determine whether 8, 16, or all 32 bits will be shifted.  If the accumulator does not contain a valid numeric value, a macro 'fail' will result.

### &H02F1  STRINGS/Maths functions/LSR

You will be asked for a number, which is used in determining how many times to logical shift right the accumulator.  The accumulator size is used to determine whether 8, 16, or all 32 bits will be shifted.  If the accumulator does not contain a valid numeric value, a macro 'fail' will result.

### &H02F4  STRINGS/Maths functions/ASL

You will be asked for a number, which is used in determining how many times to arithmetic shift left the accumulator.  The accumulator size is used to determine whether 8, 16, or all 32 bits will be shifted.  If the accumulator does not contain a valid numeric value, a macro 'fail' will result.

### &H02F3  STRINGS/Maths functions/ASR

You will be asked for a number, which is used in determining how many times to arithmetic shift right the accumulator.  The accumulator size is used to determine whether 8, 16, or all 32 bits will be shifted.  If the accumulator does not contain a valid numeric value, a macro 'fail' will result.

## Menu Functions

| | | | |
|---|---|---|---|
| **&H0233** | **STRINGS/Define string/A** | **&H023A** | **STRINGS/Define string/H** |
| **&H0234** | **STRINGS/Define string/B** | **&H023B** | **STRINGS/Define string/I** |
| **&H0235** | **STRINGS/Define string/C** | **&H023C** | **STRINGS/Define string/J** |
| **&H0236** | **STRINGS/Define string/D** | **&H023D** | **STRINGS/Define string/K** |
| **&H0237** | **STRINGS/Define string/E** | **&H023E** | **STRINGS/Define string/L** |
| **&H0238** | **STRINGS/Define string/F** | **&H023F** | **STRINGS/Define string/M** |
| **&H0239** | **STRINGS/Define string/G** | **&H0225** | **STRINGS/Define string/Acm** |

These functions allow to you define the contents of either the accumulator, or one of the string buffers A-M. Whenever you are asked for a string using the simple string requester (NOT the large file requester), you can either type in a string literal, or input an escape character (just press the escape key), followed by either another escape character, or one of the letters A-M inclusive. If the second character is the escape character, the string being requested will be copied from the accumulator instead. If the second character that you typed (after the escape character) is one of the letters A-M, then the string being requested will instead be copied from the appropriate buffer A-M. This applies even when you are defining the contents of one of the string buffers. This method of using indirection should be particularly handy in sophisticated macros.

The maximum string length in any buffer (including the accumulator) is 240 characters.

| | | | |
|---|---|---|---|
| **&H027F** | **STRINGS/Swap with buffer../A** | **&H0286** | **STRINGS/Swap with buffer../H** |
| **&H0280** | **STRINGS/Swap with buffer../B** | **&H0287** | **STRINGS/Swap with buffer../I** |
| **&H0281** | **STRINGS/Swap with buffer../C** | **&H0288** | **STRINGS/Swap with buffer../J** |
| **&H0282** | **STRINGS/Swap with buffer../D** | **&H0289** | **STRINGS/Swap with buffer../K** |
| **&H0283** | **STRINGS/Swap with buffer../E** | **&H028A** | **STRINGS/Swap with buffer../L** |
| **&H0284** | **STRINGS/Swap with buffer../F** | **&H028B** | **STRINGS/Swap with buffer../M** |
| **&H0285** | **STRINGS/Swap with buffer../G** | | |

The contents of the accumulator is swapped the with contents of the chosen buffer [A-M].

## Menu Functions

### &H02A9  STRINGS/Input buffer/Load
### &H02AB  STRINGS/Input buffer/Restart
### &H02AA  STRINGS/Input buffer/Read line

The Input and Output buffers can be used to buffer textual information between a file and the accumulator within ReSource.  To transfer information into ReSource, you load the input buffer, using "Load", and transfer a line at a time to the accumulator, using "Read line".  Each time you read a line from the input buffer, it replaces the contents of the accumulator. Successive uses of "Read line" will get successive lines from the input buffer.  A line means all text up to a line feed, to a maximum of 240 characters.  The line feed is NOT transferred, as the line will be null terminated within the accumulator.  To start reading lines from the start of the buffer again, use "Restart".  For example, you may have a list of labels that you wish to put into the current file.  Use "Load" to get the list of labels into the input buffer, then create a macro something like:

```
        CURSOR/Relative/Next label
        STRINGS/Input buffer/Read line
        STRINGS/Put/Label
        MACROS 1/Previous macro label/#1
```

The first function will advance the cursor to the next label.
The second function reads the next label to use into the accumulator.
The third function will replace the current label with the one in the accumulator.
The last function ("MACROS 1/Previous macro label/#1") is optional.  It will make the function repeat for the entire file, or the entire list of labels, whichever ends first.

Instead of reading an entire line, you may read just a byte, word, longword or character from the input buffer.  Bytes, words and longwords are read as-is, and translated into a string, using the current accumulator numeric type. Characters are read as-is, and placed into the accumulator without translation.

### &H0385  STRINGS/Input buffer/Read byte

Read a byte from the input buffer, and place it in the accumulator using the currently defined accumulator numeric type (decimal/hex/binary).

### &H0386  STRINGS/Input buffer/Read word

Read a word from the input buffer, and place it in the accumulator using the currently defined accumulator numeric type (decimal/hex/binary).

### &H0387  STRINGS/Input buffer/Read longword

Read a longword from the input buffer, and place it in the accumulator using the currently defined accumulator numeric type (decimal/hex/binary).

### &H0388  STRINGS/Input buffer/Read character

Read a character from the input buffer, and insert it directly into the accumulator, without translation.

## Menu Functions

**&H0388  STRINGS/Input buffer/Read character**

**&H02AC  STRINGS/Output buffer/Save**
**&H02AE  STRINGS/Output buffer/Clear**
**&H02AD  STRINGS/Output buffer/Append accumulator**

The output buffer is used to hold information that you wish to save from ReSource. The idea here is that you append the contents of the accumulator to the end of the output buffer, and eventually save the entire contents of the output buffer to a file. For example, you might wish to create a list of labels for the current file (see "STRINGS/Input buffer/:"). The following macro will do this:

```
        CURSOR/Relative/Next label
        STRINGS/Get/Label
        STRINGS/Edit functions/Append
        STRINGS/Output buffer/Append accumulator
        MACROS 1/Previous macro label/#1
```

Note: Prior to actually running this macro, you should ensure that the cursor is at the start of file, and use the function "STRINGS/Output buffer/Clear", to ensure the output buffer is empty.

The first line will move the cursor to the next label in the file. The second function will put the current label into the accumulator. The third function will prompt you for a string to append to the accumulator. A line feed (ASCII value 10) is required here, to do this, first click inside the requester string gadget, hold down the ctl key, and press "J", then press return or click on the "OKAY" gadget. Rather than use ctl-J, you may instead use "\n". The fourth function, "STRINGS/Output buffer/Append accumulator", will add the current contents of the accumulator to the end of the Output buffer. Memory is dynamically allocated for the output buffer, so maximum size is only determined by how much memory you have. The fifth function is optional, use it if you want the macro to loop until all labels in the current file are added to the list.

Once this is done, use "STRINGS/Output buffer/Save" to save the current contents of the output buffer. You will be asked for a filename to use. In practice, most people will probably combine several pieces of information for each line that gets appended to the output buffer, such as label, followed by its offset from the start of the file, or its data type, etc.

**&H016F  reserved**

When you select this function, it is identical to selecting the same menuitem that was last selected. Functions that were used by pressing keys are not repeated, just menu items. So, it's a good idea to bind this function to a key that is very easy to get to - like the space bar.

This function is most useful when a particular of the file that you are disassembling requires a particular symbol base used many times. It may be very rarely that you use many symbol bases, so it isn't really worth rebinding a key to that function. So, you simply select the function from the menus, scroll to the next line that needs that symbol base again, and press<space bar>  to use that symbol base again (assuming that you have got "repeat command" bound to the space bar).

## Menu Functions

### &H01C9  */DOS command

You will be asked for a string, which will be given as the operand to the Dos "execute" function.  Anything that you normally type into a CLI window, you can give as a command.  Any output will go to the CLI window from which you started ReSource, unless you redirect the output elsewhere.  The "Run" command must be in your C: directory (or resident).  This function is not available if you started ReSource from the icon.

### &H0226  */ZAP/Zap

You will be asked for a number/string, which will overwrite the byte/word/longword/characters at the cursor position.  The current setting of 'operand size' is used.  Using "PROJECT/Open binary file", then this function, followed by "SAVE/Save binary image/ALL", you can quickly modify a file, without having to go through the disassemble/edit/reassemble/link cycle.

However, when using this method of modification, you are restricted as to the size of changes that may be legally made.  For example, you cannot insert extra code or data, nor can you delete it.  You may "NOP" it instead, though.  If the string that you supply starts with a single quote ("'"), the following characters are treated as a string, and are copied to the cursor position "as-is".

If the string that you supply starts with "$", the following number is assumed to be hexadecimal.  If the string that you supply starts with "%", the number that follows is assumed to be binary, otherwise it is assumed to be a decimal number.

If the string you supply starts with "l.", "w." or "b.", the number following will be treated as a longword, word, or byte, respectively.  In this case, the accumulator operand size does not matter.  The case of the letter "l", "w", or "b" may be upper or lower.

See also:
  "*/ZAP/Zap2"

## Menu Functions

**&H0679  */ZAP/Zap2**

"Zap2" is functionally equivalent to "Zap", but has a much better user interface.

When you call "Zap2", it tries to make an educated guess about whether it's looking at code or data, and if data, the appropriate size.  It then puts the code/data into the string gadget, and waits for you to either edit the string, or change the mode/data type.

If the Mode=CODE, ReSource will try to assemble the string.  If the Mode=DATA, ReSource will attempt to convert the string to binary data.  Any problems will be reported, and you will be given a chance to edit the string, or abort the zap.

In data mode, only as much of the binary data as defined by the Data Size gadgets will be written to the file.  If you define the size as Byte and edit the string as "$12345678", only $78 will be written.

If the Data Type is set to ASCII, the data will be copied literally to the string gadget buffer until a null is reached, for up to 127 bytes.  If the data wasn't really text, this will be difficult to read.  When "OK" is pressed, the entire string, exactly as shown including 1 null byte, will be written back to the file.

Note that "Zap2" isn't compatible with macros.  If "Zap2" is called while learning a macro, it will immediately jump to "Zap", instead.  If "Zap2" is called while a macro is running, it will cause a macro fail.

"Zap2" requires simpleasm.library to be available for the CODE mode to work.

See also:
  "*/ZAP/Zap"

**&H022A  */Screen/Front**
**&H022B  */Screen/Back**

"Front" moves ReSource's screen in front of all other screens.  "Back"  moves ReSource's screen behind all other screens.

**&H0277  */Set task priority**

You will be asked for a number, representing the priority at which to set the current task.  For example, if you have a program being assembled in the background, and you don't want it to slow down the operation of ReSource, you might set the task priority to one or two.

## Menu Functions

**&H0296  */Origin/Set**
**&H0298  */Origin/Specify**
**&H0297  */Origin/Clear**

There is sometimes a requirement to disassemble a block of code, as if it is loaded to some particular area of memory, other than that to which it is now loaded.  For example, if you disassemble Kickstart(TM), and save the binary image, then later load this into ReSource, it will not disassemble properly unless ReSource "thinks" that its origin is correct.  To do this, load the file, then use the "Specify" option.  Both the "Specify" and "Set" set the origin to an absolute address; the "Set" option is the same as "Specify", except that it assumes that the current loading address is to be used.  These options are really only useful if the file is loaded as a binary image, or from track, or memory.  If loaded from an executable, the reloc32 information will ensure that proper relocation is done where needed.  It is because no reloc32 information is available when opening files other than load files, that this function is required.  To remove the origin, so that it uses whatever actual address the file is loaded to, use the "Clear" option.

**&H004F  */Convert (xx,A4) EA''s/This operand**

Enables (xx,An) type Effective Address conversions, and sets "DT" to the operand of the current line.  See "EA conversions".

**&H0028  */Convert (xx,A4) EA''s/This address**

Enables (xx,An) type Effective Address conversions, and sets "DT" to the cursor address.

**&H0019  */Convert (xx,A4) EA''s/Specify**

You will be asked to supply a number.  ReSource will then use this in resolving address register relative Effective Addresses.  The number that you supply is assumed to be the offset from the start of the current file, where the currently selected address register will point to at runtime.

By using this function, you not only set the new offset for (xx,An) conversions you also enable conversions to be done.  To disable (xx,An) conversions, use "SPECIAL FUNCTIONS/Convert (xx,An) EA's/Disable".

To tell ReSource which address register to use for conversions (only one can be used), use "SPECIAL FUNCTIONS/Specify Base Register/", and the appropriate register A0-A7.

**&H0020  */Convert (xx,A4) EA''s/Set lower limit**

For (xx,An) Effective Address conversions, the lower limit will be set to the current cursor position.

**&H0021  */Convert (xx,A4) EA''s/Set upper limit**

For (xx,An) Effective Address conversions, the upper limit will be set to the current cursor position.

## Menu Functions

**&H02AF  */Convert (xx,A4) EA''s/Relative**
**&H02B0  */Convert (xx,A4) EA''s/Absolute**

When using the "Convert (xx,An) EA's" functions, the original effective address can be converted to either a symbol form of the original, or it can be changed to an absolute reference. Selecting "Relative" will mean that lines such as:

```
                    CLR.L     (-$FEA2,A4)
```

would be displayed something like:

```
                    CLR.L     (START+$4D6E-DT,A4)
```

By selecting "Absolute", it would instead show as:

```
                    CLR.L     (START+$4D6E)
```

**&H02DB  */Convert (xx,A4) EA's/Disable**

Disables base register conversions.

**&H034D  */Convert (xx,A4) EA''s/Data refs only**
**&H0350  */Convert (xx,A4) EA''s/All references**

If "Data refs only" is selected, whenever base register conversions are done, only those that are the result of a data reference will be converted. Selecting "All references" allows all references to be converted (both data and code).

| | |
|---|---|
| **&H02D3  */Specify Base Register/A0** | **&H02D7  */Specify Base Register/A4** |
| **&H02D4  */Specify Base Register/A1** | **&H02D8  */Specify Base Register/A5** |
| **&H02D5  */Specify Base Register/A2** | **&H02D9  */Specify Base Register/A6** |
| **&H02D6  */Specify Base Register/A3** | **&H02DA  */Specify Base Register/A7** |

This function permits you to select the register to be used for base register conversions. See also "STRINGS/Put/Base register".

## Menu Functions

**&H0056  */Convert specific EA''s/Set base #1**
**&H0057  */Convert specific EA''s/Set base #2**
**&H0058  */Convert specific EA''s/Set base #3**
**&H0059  */Convert specific EA''s/Cvert W/base 1**
**&H005A  */Convert specific EA''s/Cvert W/base 2**
**&H005B  */Convert specific EA''s/Cvert W/base 3**

Use this function to convert numbers into symbols that represent the distance between two labels.  You may need to do this where there is a table of words, where each word represents the distance between some base address, and the entry point of some code to execute.  Examine the following:

```
                    MOVE.W    (lbW001170,PC,D0.W),D0
                    JMP       (lbW001170,PC,D0.W)

lbW001170           dw        8
                    dw        $18
                    dw        $38
                    dw        $40
                    dw        $24BC
                    dl        Cannotopendev.MSG
                    dw        $603E
                    MOVE.L    #Notenoughmemo.MSG,(A2)
                    BRA.B     START+$11BE

                    MOVE.L    #CannotopenAmi.MSG,(A2)
                    BRA.B     START+$11BE

                    MOVE.L    #Writeprotecte.MSG,(A2)
                    BRA.B     START+$11BE

                    MOVE.L    #Diskchangedno.MSG,(A2)
                    BRA.B     START+$11BE

                    MOVE.L    #Seekerror.MSG,(A2)
                    BRA.B     START+$11BE

                    MOVE.L    #Disksizeerror.MSG,(A2)
                    BRA.B     START+$11BE

                    MOVE.L    #Verifyerror.MSG,(A2)
                    BRA.B     START+$11BE
```

In the above code, there is a "jump table" starting at label "lbW001170".  It consists of four words.  The first word, of value 8, represents the distance between the label "lbW001170" and some code to execute.  To make sense of this jump table, and allow it to reassemble correctly, scroll to the label "lbW001170", and select the "SPECIAL FUNCTIONS/Convert specific EA's/Set base #1" function.  We know that this is the "base" of the jump table because of the instruction:

```
                    JMP       (lbW001170,PC,D0.W)
```

Next, we do the actual conversions.  Starting at the first word of the jump table, select "*/Convert specific EA's/Cvert W/base 1", scroll to the next line, and continue for each of the four words in the jump table.  The result will be:

```
                    MOVE.W    (lbW001170,PC,D0.W),D0
                    JMP       (lbW001170,PC,D0.W)
```

## Menu Functions

**&H0056  */Convert specific EA''s/Set base #1**
**&H0057  */Convert specific EA''s/Set base #2**
**&H0058  */Convert specific EA''s/Set base #3**
**&H0059  */Convert specific EA''s/Cvert W/base 1**
**&H005A  */Convert specific EA''s/Cvert W/base 2**
**&H005B  */Convert specific EA''s/Cvert W/base 3**

```
lbW001170           dw        lbC001178-lbW001170
                    dw        lbC001188-lbW001170
                    dw        lbC0011A8-lbW001170
                    dw        lbC0011B0-lbW001170
lbC001178           MOVE.L    #Cannotopendev.MSG,(A2)
                    BRA.B     START+$11BE

                    MOVE.L    #Notenoughmemo.MSG,(A2)
                    BRA.B     START+$11BE

lbC001188           MOVE.L    #CannotopenAmi.MSG,(A2)
                    BRA.B     START+$11BE

                    MOVE.L    #Writeprotecte.MSG,(A2)
                    BRA.B     START+$11BE

                    MOVE.L    #Diskchangedno.MSG,(A2)
                    BRA.B     START+$11BE

                    MOVE.L    #Seekerror.MSG,(A2)
                    BRA.B     START+$11BE

lbC0011A8           MOVE.L    #Disksizeerror.MSG,(A2)
                    BRA.B     START+$11BE

lbC0011B0           MOVE.L    #Verifyerror.MSG,(A2)
                    BRA.B     START+$11BE
```

Notice that the labels "lbC001178", "lbC001188", "lbC0011A8" and "lbC0011B0" were created, and that the data type was set to code.

**&H01BD  */Data type set/Code**
**&H01BE  */Data type set/ASCII**
**&H01BF  */Data type set/Bytes**
**&H01C0  */Data type set/Words**
**&H01C1  */Data type set/Longwords**
**&H01C2  */Data type set/Automatic**

Affects how the "SPECIAL FUNCTIONS/Convert specific EA's" functions work. When the data type must be determined, it can be automatic, or you can force it to be set to Bytes, Words, Longwords, ASCII, or Code.

**&H01C3  */Convert to../Absolute**
**&H01C4  */Convert to../Offset**

Determines the type of symbol created when the "SPECIAL FUNCTIONS/Convert specific EA's" functions are used, either an Absolute or Offset value.

## Menu Functions

**&H029F  */Reloc32/Delocate**
**&H02A0  */Reloc32/Relocate**

Use these functions to specifically delocate or relocate the current file. This is only useful if the file was loaded from an executable, where reloc32 information is present. Be aware that relocation is performed immediately after the "PROJECT/O'lay binary image" function, when a file is actually loaded.

**&H02EB  */Run .rcl file**

This function asks you for the name of a file, which is expected to be a compiled ReSource command language file. If the file can be opened successfully, it will be executed by ReSource's new command language interpreter. The contents of the file itself determine exactly what will happen next.

The compiled command language file supplied with this release should be used immediately after loading any Aztec "C" program, specifically those compiled with the V3.6 compiler. It will be able to recognize many of the standard "C" functions that get compiled into Aztec "C" programs.

The command language interpreter has built-in functions that allow it to be programmed to recognize a sequence of information, and if all tests were successful, a sequence of actions are taken. The sequence of actions might include creating labels, symbols, setting data types, creating comments, etc.

There are no current plans to make the command language itself public knowledge. It is highly complex, and very difficult to write in. Also, the form of the language is still undergoing changes.

**&H02BE  SAVE/Partial save/Reset start**

Sets the partial save start to the start of the file.

**&H021C  SAVE/Partial save/Set start**
**&H021D  SAVE/Partial save/Set end**

Set the start/end of the "partial save" area. The functions that will be affected are:

SAVE/Save .asm/Partial
SAVE/Calculate .asm size/Partial
SAVE/Save binary image/Partial
SAVE/Save tracks/Partial
STRINGS/Get/Partial save size
SAVE/Save to memory/Partial

**&H02BF  SAVE/Partial save/Reset end**

Sets the partial save end to the end of the file.

## Menu Functions

### &H01C7  SAVE/Save binary image/All

Save the current buffer contents "as-is", to a file.  If the file was loaded with "PROJECT/Open load file", and was NOT loaded as a binary image file, the resulting saved file will NOT be the same as that originally loaded, because the loader information, and relocation information, has been stripped.  If you wish to load a file, and then save it exactly as it was loaded, you should use "PROJECT/Open binary file".  Even though you may have disassembled memory directly, or read tracks from a floppy disk, it does NOT stop you from saving the binary image.

Once something is loaded into ReSource, it can be saved as source code, as a binary image file, directly to memory, or it can be written directly to a floppy disk, to a specific sector or track.  If you originally loaded the file using "PROJECT/Open load file", you can also use "SAVE/Save executable/".  This gives fantastic flexibility for general hacking, for as little as one byte of the current buffer can be saved to memory, to a binary image file, or as source code.  Track writes must be in multiples of 512 bytes, however the clipping is done for you.

Save "Partial" is similar to the above function, except only the data between the partial save start and the partial save end is saved.

### &H0272  SAVE/Save binary image/Partial

Similar to the "SAVE/Save binary image/All" function, except that only the area between the partial save start and partial save end, will be saved.

## Menu Functions

**&H036E  SAVE/Save binary image/Original**
**&H0372  SAVE/Save binary image/Current dir**
**&H036A  SAVE/Save binary image/Specify**

You can set up the environment in ReSource to specify where files are to be saved.  There are 3 different sets of functions that you can use to specify the behaviour of:

"PROJECT/Save .RS/Save"
"SAVE/Save binary image/All"
"SAVE/Save binary image/Partial"
"SAVE/Save .asm/All"
"SAVE/Save .asm/Partial"
"SAVE/Save executable/With Labels"
"SAVE/Save executable/No Labels"

1. Current dir
If enabled, the default path to save files to will be the same path as where the file was loaded from.  When the file requester is displayed, you will only see the file name.

2. Original
If enabled, the default path to save files will be the current directory. When the file requester is displayed, you will see the path and the file name.  If the file was loaded from the current directory, the behaviour will be the same as for "Original".

3. Specify
You will be asked to supply a pathname, which will be used as the default path to save files to in future.  This pathname will be saved in the "Auto-configuration" macro when you select "PROJECT/Save Configuration".

**&H0018  SAVE/Save .asm/All**

You will be asked to supply the name of a file to save source code to, for the entire buffer contents.  The output will be very similar to how you see it on the display.  The "equates" or "xref" table will be saved first.

If "OPTIONS/Allow.../Error comments" is enabled, lines on which an error has been detected will have a full-line comment appended to them, describing the type of error detected.

**&H0185  SAVE/Save .asm/Partial**

Similar to the "SAVE/Save .asm/All" function, with the following exceptions:

1. Only the lines between the partial save start, and the partial save end are saved.  See "SAVE/Partial save/Set start".

2. No "equates" or "xref" table is created, nor does any of the header information that normally is saved in a ".asm" file saved.  You get just the lines that you selected, and nothing else.

## Menu Functions

**&H036B  SAVE/Save .asm/Original**
**&H036F  SAVE/Save .asm/Current dir**
**&H0367  SAVE/Save .asm/Specify**

You can set up the environment in ReSource to specify where files are to be saved.  There are 3 different sets of functions that you can use to specify the behaviour of:

"PROJECT/Save .RS/Save"
"SAVE/Save binary image/All"
"SAVE/Save binary image/Partial"
"SAVE/Save .asm/All"
"SAVE/Save .asm/Partial"
"SAVE/Save executable/With Labels"
"SAVE/Save executable/No Labels"

1. Current dir
If enabled, the default path to save files to will be the same path as where the file was loaded from.  When the file requester is displayed, you will only see the file name.

2. Original
If enabled, the default path to save files will be the current directory.  When the file requester is displayed, you will see the path and the file name.  If the file was loaded from the current directory, the behaviour will be the same as for "Original".

3. Specify
You will be asked to supply a pathname, which will be used as the default path to save files to in future.  This pathname will be saved in the "Auto-configuration" macro when you select "PROJECT/Save Configuration".

**&H01BB  SAVE/Calculate .asm size/All**

Calculates the size of the source code file that would result if you had used the "SAVE/Save .asm/ALL" function.

**&H021E  SAVE/Calculate .asm size/Partial**

Calculates the size of the source code file that would result if you had used the "SAVE/Save .asm/Partial" function.

**&H027D  SAVE/Save to memory/All**
**&H027E  SAVE/Save to memory/Partial**

You will be asked to supply a memory address to save the contents of the current buffer.  The buffer will be saved "as-is"; NOT as disassembled source.  Care should be taken with this function, as a system crash will likely result if you supply the wrong address.  By finding out where the current buffer is in memory, (by using "DISPLAY/Cursor Address/Absolute") it is possible to copy one part of the current buffer to another part.  This will require judicious use of "SAVE/Save to memory/Partial".

Save "Partial" is as above, except that only the data between the partial save start and the partial save end is saved.

## Menu Functions

**&H0274  SAVE/Save tracks/All**
**&H0275  SAVE/Save tracks/Partial**

You will be asked for a string representing where to start writing tracks. The first parameter you give must be either "DF0:", "DF1:", "DF2:", or "DF3:". The second parameter is the starting cylinder to save to.  The third parameter is optional, and represents an offset, in sectors, where the track write is to start.  For example, to save sector #3 on cylinder 5 on DF2:, the parameters required would be "DF2: 5 3".  To save to the first sector on DF0:, the parameters required would be "DF0: 0".  Whole sector writes ONLY are done, clipping is performed automatically.

Save "Partial" is as above, except that only the data between the partial save start and the partial save end is saved.  Furthermore, only even multiples of 512 bytes will be written.  For example, if the partial save size is 1027 bytes, only two sectors (1024 bytes) will be written to the disk.

**&H0275  SAVE/Save tracks/Partial**

You will be asked for a string representing where to start writing tracks. The first parameter you give must be either "DF0:", "DF1:", "DF2:", or "DF3:". The second parameter is the starting cylinder to save to.  The third parameter is optional, and represents an offset, in sectors, where the track write is to start.  For example, to save sector #3 on cylinder 5 on DF2:, the parameters required would be "DF2: 5 3".  To save to the first sector on DF0:, the parameters required would be "DF0: 0".  Whole sector writes ONLY are done, clipping is performed automatically.

Save "Partial" is as above, except that only the data between the partial save start and the partial save end is saved.  Furthermore, only even multiples of 512 bytes will be written.  For example, if the partial save size is 1027 bytes, only two sectors (1024 bytes) will be written to the disk.

**&H02B6  SAVE/Save screen**

This function will prompt you for the name of a file.  If the file can be opened, it will save the currently displayed text to that file.  Use this to send a screen dump to the printer, or any other file.  Note that this can be used at the completion of a "save .asm", to get a printout of the source code profile.  Almost any function will cause a display refresh, so this must be the first function used.  If you have to re-activate ReSource's window for any reason, in order to avoid refreshing the display (and erasing this information in doing so), place the mouse pointer at the extreme left edge of the screen, then press the LMB.

## Menu Functions

**&H0307  SAVE/Save executable/With Labels**
**&H0365  SAVE/Save executable/No Labels**

This function attempts to create an executable program from the file currently being disassembled.  Not all programs can be changed back into an executable successfully.  The main problem here is with programs that have reloc32s pointing to somewhere outside the referenced hunk.  Also, programs that contain overlays will definately not be recreated successfully.

This function does NOT reassemble the current disassembly, and it is because of this that a complete disassembly is NOT necessary to create a good executable.  It is expected that the main use of this function will be performing minor modifications/zaps to executable programs.  This was possible with previous versions of ReSource, however the user was forced to load the file as a binary image, which made it difficult to find the place in the file where the modification was necessary.  Now, you may load a program using the "PROJECT/Open load file" function, disassemble only as much as necessary to find the part that requires modification, do the necessary zaps, and use this function to recreate the modified executable.  Labels may be saved in SYMBOL hunks, which may be read and used by debuggers.

**&H036D  SAVE/Save executable/Original**
**&H0371  SAVE/Save executable/Current dir**
**&H0369  SAVE/Save executable/Specify**

You can set up the environment in ReSource to specify where files are to be saved.  There are 3 different sets of functions that you can use to specify the behaviour of:

"PROJECT/Save .RS/Save"
"SAVE/Save binary image/All"
"SAVE/Save binary image/Partial"
"SAVE/Save .asm/All"
"SAVE/Save .asm/Partial"
"SAVE/Save executable/With Labels"
"SAVE/Save executable/No Labels"

1. Current dir
If enabled, the default path to save files to will be the same path as where the file was loaded from.  When the file requester is displayed, you will only see the file name.

2. Original
If enabled, the default path to save files will be the current directory.  When the file requester is displayed, you will see the path and the file name.  If the file was loaded from the current directory, the behaviour will be the same as for "Original".

3. Specify
You will be asked to supply a pathname, which will be used as the default path to save files to in future.  This pathname will be saved in the "Auto-configuration" macro when you select "PROJECT/Save Configuration".

## Menu Functions

**&H004B  SAVE/Tabs/Real tabs**
**&H004C  SAVE/Tabs/Spaces**

Use to select between real tabs (ASCII value 9) and spaces being used in the output source code as separators.

**&H0192  SAVE/Symbol table/None**
**&H0193  SAVE/Symbol table/XREF**
**&H0194  SAVE/Symbol table/EQUate**

When creating symbols, either automatically or manually, the value and name of the symbol is stored, so that when the source code is saved, an equate table, or XREF table can be generated. If this function is set to "EQUate", a symbol table similar to the following will precede the source code in the output file:

```
AG_OpenLib          EQU       $00030000
AO_DOSLib           EQU       $00008007
AT_Recovery         EQU       $00000000
_LVOAlert           EQU       $FFFFFF94
_LVOClose           EQU       $FFFFFFDC
_LVOCreateProc      EQU       $FFFFFF76
_LVOCurrentDir      EQU       $FFFFFF82
```

If, instead, this function is set to "XREF", a symbol table  similar  to  the following will preceed the source code in the output file:

```
                    XREF      AG_OpenLib
                    XREF      AO_DOSLib
                    XREF      AT_Recovery
                    XREF      _LVOAlert
                    XREF      _LVOClose
                    XREF      _LVOCreateProc
                    XREF      _LVOCurrentDir
```

## Menu Functions

### &H0053  KEY BINDINGS/Rebind key

Every function that you see in ReSource's menus can be bound to a key, except "abort" which is permanently bound to rightAmiga-A.  Both Amiga-keys are supported, you may use shift, alt, ctrl, lAmiga, rAmiga, shift-alt, shift-ctrl, alt-ctrl, or shift-alt-ctrl, in combination with any non-qualifier key, as a distinct key to which to bind a function.  You may, of course, also use non-qualifier keys by themselves.  Thus, you have complete control over which keys do what.  After re-binding some keys, you will probably want to save the keytable.  If you save the keytable as "S:RS.keytable", it will be loaded every time you run ReSource.

You may want to create several keytables, suitable for doing differing program types (C, assembler, etc.). If a keytable file is loaded, ALL current key bindings will be overwritten by the bindings present in the keytable.  If you don't load any keytable, and ReSource couldn't find "S:RS.Keytable" when you first ran it, some keys will still be bound:

| | |
|---|---|
| right-Amiga Q | PROJECT/Quit |
| right-Amiga O | Open load file |
| up arrow | CURSOR/Relative/Next line |
| down arrow | CURSOR/Relative/Previous line |
| right arrow | CURSOR/Absolute/Fwd reference |
| left arrow | CURSOR/Absolute/Previous location |
| shift-up arrow | CURSOR/Relative/Previous page |
| shift-down arrow | CURSOR/Relative/Next page |
| HELP | PROJECT/     -<HELP>- |

To rebind a key, select "KEY BINDINGS/Rebind key", and follow the  prompts  in the title bar.

### &H0638  KEY BINDINGS/Report key

<RESERVED>

### &H0054  KEY BINDINGS/Save key table

Save the current keytable.  See "KEY BINDINGS/Rebind key".

### &H0055  KEY BINDINGS/Load key table

Load a keytable.  See "KEY BINDINGS/Rebind key".

## Button Window Functions

### &H037F  CURSOR/Binary search/Any alignment

When the "CURSOR/Binary search/" functions are used, all matches will be "found", not just those starting on a word boundary.

### &H0382  CURSOR/Binary search/Find next occurrence

Using the currently defined search parameters, search forward through the buffer, for a match.  Note that it is the buffer itself that it searched, not a disassembly of the buffer.

### &H0383  CURSOR/Binary search/Find previous occurrence

Using the currently defined search parameters, search backward through the buffer, for a match.  Note that it is the buffer itself that it searched, not a disassembly of the buffer.

### &H037D  CURSOR/Binary search/Set search parameters

You will be asked to enter a string, which will be used in the "CURSOR/Binary search/Find next occurrence" and "CURSOR/Binary search/Find previous occurrence" functions.  You may use the same type of parameters as you would supply to a "dc.b" directive in assembler source code.  For example, if you wished to search for a null-terminated "dos.library" string, you would use:

```
'dos.library',0
```

If you wanted to search for a "NOP" instruction, you would use:

```
$4E71
```

You can use decimal, hex or binary numbers.  They may be preceeded by "b.", "w." or "l.", to force following numbers to be seen as byte, words, or longwords, respectively.  For example, if you wanted to search for the longword values 1,2,3, followed by the string "dos.library" followed by the bytes $45 and $7E, you would use:

```
l.1,2,3,'dos.library',b.$45,$7E
```

If you don't specify any of the "b.", "w." or "l." size specifiers, the values that you supply will default to the smallest size that those values will fit into.  Thus, "123" "$FF" and "000000001" will be treated as bytes, and "$200" and "12345" will be treated as words.  An exception to this is with binary numbers.  If you specify a binary number that consists of more than 16 digits, it will be treated as a longword value, even if they are all "0" digits.

### &H037E  CURSOR/Binary search/Word aligned only

When the "CURSOR/Binary search/" functions are used, only matches that start on a word boundary will be "found".

## Button Window Functions

**&H034B  CURSOR/Buffer search/Find next occurrence**
**&H034C  CURSOR/Buffer search/Find previous occurrence**

Using the currently defined search string, the buffer is searched for a matching string. Note carefully here that it is the *buffer* that is searched, not a *disassembly* of the buffer as is the usual case. For example, if the string "dos.library" were currently displayed as:

```
                    db        'dos.lib'
                    db        'ra'
                    db        'ry',0
```

a normal search would be able to find "dos.lib", or "ra", or even "'ry',0". But because the buffer search searches the buffer itself, it would find the string "dos.library", regardless of how it was being displayed. And because the buffer is not disassembled in the search process, it searches around 110 times faster than a normal search.

**&H0352  CURSOR/Label search/Find next occurrence**
**&H0353  CURSOR/Label search/Find previous occurrence**
**&H0351  CURSOR/Label search/Specify label**

The "Label search" functions allow you to find labels by searching the label definitions themselves, rather than disassembling each line of code looking for label definitions. This way, labels can be found very quickly (around 500 times faster than performing a normal search). By selecting either "find next occurrence" or "find previous occurrence", you are able to find every occurence of the label. In the case of local labels, there may be quite a few.

**&H01AC  CURSOR/Normal search/Find nearest occurrence**

Similar to "CURSOR/Normal search/Find next occurrence", except that search proceeds both forwards and backwards alternately, line for line, until a match is found, or until the entire buffer has been searched.

**&H01AA  CURSOR/Normal search/Find next occurrence**

Using the currently defined search string, search forward for the next occurrence of that string. The string is searched for as-is, including question marks, asterisks, etc., which in a pattern search take on special meaning. This function will search a disassembly of the current buffer, not the buffer itself. This means that the state of disassembly will affect the search. For example, if you are searching for the string "dos.library", it will only be found if that data type was set to "ascii" at the appropriate location. See "CURSOR/Buffer search/Find next occurrence".

**&H01AB  CURSOR/Normal search/Find previous occurrence**

Similar to "CURSOR/Normal search/Find next occurrence", except that search proceeds backwards.

## Button Window Functions

### &H0215  CURSOR/Normal search/Search accumulator

Using the currently defined search string, search the accumulator, and if a match is not found, perform a macro fail, else exit normally.

### &H0257  CURSOR/Normal search/Search this line

Using the currently defined search string, search the current line only, and if a match is not found, perform a macro fail, else exit normally.

### &H004D  CURSOR/Normal search/Set search string

You will be asked to supply a string, which will be used in future searches of the "normal" and "buffer" type.

Like all other string requests, string indirection may be used.

### &H01A9  CURSOR/Pattern search/Find nearest occurrence

Using the currently defined search string, search for the nearest occurrence of that string.  ARP wildcard characters are expanded (not used literally). This is a bi-directional search.

### &H004E  CURSOR/Pattern search/Find next occurrence

Similar to the "CURSOR/Normal search/Find next occurrence" function, except that wildcard characters are expanded (not used literally).

### &H01A8  CURSOR/Pattern search/Find previous occurrence

Using the currently defined search string, search backward for the previous occurrence of that string.  ARP wildcard characters are expanded (not used literally).

### &H0214  CURSOR/Pattern search/Search accumulator

Using the currently defined search string, search the accumulator, and if a match is not found, perform a macro fail, else exit normally.  ARP wildcard characters are expanded (not used literally).

### &H0256  CURSOR/Pattern search/Search this line

Using the currently defined search string, search the current line only, and if a match is not found, perform a macro fail, else exit normally.  ARP wildcard characters are expanded (not used literally).

## Button Window Functions

### &H0677  CURSOR/Pattern search/Set pattern string

You will be asked to supply a string, which will be used in future searches of the "pattern" type.

Like all other string requests, string indirection may be used.

### &H0380  CURSOR/Search/Exact match
### &H0381  CURSOR/Search/Ignore case

Selecting "Case Sensitive" will cause most search functions to perform a case-sensitive search.  Selecting "Ignore Case" will cause a search to be performed without regard to case.  Only the following functions are affected:

"CURSOR/Normal search/Find next occurrence"
"CURSOR/Normal search/Find previous occurrence"
"CURSOR/Normal search/Find nearest occurrence"
"CURSOR/Pattern search/Find next occurrence"
"CURSOR/Pattern search/Find previous occurrence"
"CURSOR/Pattern search/Find nearest occurrence"
"CURSOR/Buffer Search/Find next occurrence"
"CURSOR/Buffer Search/Find previous occurrence"
"CURSOR/Binary search/Find next occurrence"
"CURSOR/Binary search/Find previous occurrence"

See also:
"CURSOR/Search/Search from start of file"
"CURSOR/Search/Search from end of file"
"CURSOR/Search/Search from current position"

### &H03B9  CURSOR/Search/Search from current position

This function resets "CURSOR/Search/Search from start of file" and "CURSOR/Search/Search from end of file".  Search functions will then work in the normal manner of finding the next or previous occurrance of a string from the current position.

## Button Window Functions

### &H03B8  CURSOR/Search/Search from end of file

Selecting this will begin the next search at the end of the file.  You would use this function when you want to find the last occurrance of a string. This function will only affect searches done in the reverse direction. These include:
"CURSOR/Normal search/Find previous occurrence"
"CURSOR/Pattern search/Find previous occurrence"
"CURSOR/Buffer Search/Find previous occurrence"
"CURSOR/Binary search/Find previous occurrence"

When you execute the search, the file will be searched from the end. After the search, the FROM gadget will revert to CURRENT so that you can search for the previous instance.

After a search following the use of this function, your old position wll be found on the position stack, and may be recalled by the "CURSOR/Remember" function.

See also:
"CURSOR/Search/Search from start of file"
"CURSOR/Search/Search from current position"

### &H03B7  CURSOR/Search/Search from start of file

Selecting this will begin the next search from the start of the file.  You would use this function when you want to find the first instance of a string. This function will only affect searches done in the forward direction.  These include:
"CURSOR/Normal search/Find next occurrence"
"CURSOR/Pattern search/Find next occurrence"
"CURSOR/Buffer Search/Find next occurrence"
"CURSOR/Binary search/Find next occurrence"

When you execute the search, the file will be searched from the beginning. After the search, the FROM gadget will revert to CURRENT so that you can search for the next instance.

After a search following the use of this function, your old position wll be found on the position stack, and may be recalled by the "CURSOR/Remember" function.

See also:
"CURSOR/Search/Search from end of file"
"CURSOR/Search/Search from current position"

## Button Window Functions

**&H0355  CURSOR/Symbol search/Find next occurrence**
**&H0356  CURSOR/Symbol search/Find previous occurrence**
**&H0354  CURSOR/Symbol search/Specify symbol**

The "symbol search" functions allow you to find symbols by searching the symbol definitions themselves, rather than disassembling each line of code looking for symbols.  This way, symbols can be found very quickly (around 500 times faster than performing a normal search).  By selecting either "find next occurrence" or "find previous occurrence", you are able to find every occurence of the symbol.

Note that the forward reference of a label as in:

```
                    LEA       (doslibrary.MSG,PC),A1
```

is _not_ a symbol.

**&H03A4  DISPLAY/Blank lines/Calls/OFF**
**&H039F  DISPLAY/Blank lines/Conditional branches/OFF**
**&H03A1  DISPLAY/Blank lines/DBcc instructions/OFF**
**&H03A0  DISPLAY/Blank lines/DBRA instructions/OFF**
**&H03A2  DISPLAY/Blank lines/Entry points/OFF**
**&H03A3  DISPLAY/Blank lines/Returns/OFF**
**&H039E  DISPLAY/Blank lines/Unconditional branches/OFF**

Used to disable a particular type of blank line.  See the "ON" function for details.

**&H039C  DISPLAY/Hiliting/"Shop" labels/OFF**
**&H038F  DISPLAY/Hiliting/BSS hunks/OFF**
**&H0392  DISPLAY/Hiliting/Chip load hunks/OFF**
**&H0391  DISPLAY/Hiliting/CODE hunks/OFF**
**&H039D  DISPLAY/Hiliting/Custom labels/OFF**
**&H0390  DISPLAY/Hiliting/DATA hunks/OFF**
**&H0397  DISPLAY/Hiliting/Data type known/OFF**
**&H0396  DISPLAY/Hiliting/Data type uncertain/OFF**
**&H039A  DISPLAY/Hiliting/DCB override/OFF**
**&H0393  DISPLAY/Hiliting/Fast load hunks/OFF**
**&H0398  DISPLAY/Hiliting/Internally-produced refs/OFF**
**&H0394  DISPLAY/Hiliting/Reloc32/OFF**
**&H0395  DISPLAY/Hiliting/Symbol scan/OFF**
**&H039B  DISPLAY/Hiliting/Symbols/OFF**
**&H0399  DISPLAY/Hiliting/Uninitialized data/OFF**

Used to switch off the particular type of hiliting.  See the matching "ON" function for details.

## Button Window Functions

|  |  |  |  |
|---|---|---|---|
| **&H002E** | **MACROS 1/Create/(#1)** | **&H0165** | **MACROS 1/Create/(#11)** |
| **&H0030** | **MACROS 1/Create/(#2)** | **&H0167** | **MACROS 1/Create/(#12)** |
| **&H0032** | **MACROS 1/Create/(#3)** | **&H01CA** | **MACROS 1/Create/(#13)** |
| **&H0157** | **MACROS 1/Create/(#4)** | **&H01CB** | **MACROS 1/Create/(#14)** |
| **&H0159** | **MACROS 1/Create/(#5)** | **&H01CC** | **MACROS 1/Create/(#15)** |
| **&H015B** | **MACROS 1/Create/(#6)** | **&H01CD** | **MACROS 1/Create/(#16)** |
| **&H015D** | **MACROS 1/Create/(#7)** | **&H01CE** | **MACROS 1/Create/(#17)** |
| **&H015F** | **MACROS 1/Create/(#8)** | **&H01CF** | **MACROS 1/Create/(#18)** |
| **&H0161** | **MACROS 1/Create/(#9)** | **&H01D0** | **MACROS 1/Create/(#19)** |
| **&H0163** | **MACROS 1/Create/(#10)** |  |  |

Start recording a macro.

|  |  |  |  |
|---|---|---|---|
| **&H002F** | **MACROS 1/Execute/(#1)** | **&H0166** | **MACROS 1/Execute/(#11)** |
| **&H0031** | **MACROS 1/Execute/(#2)** | **&H0168** | **MACROS 1/Execute/(#12)** |
| **&H0033** | **MACROS 1/Execute/(#3)** | **&H01E4** | **MACROS 1/Execute/(#13)** |
| **&H0158** | **MACROS 1/Execute/(#4)** | **&H01E5** | **MACROS 1/Execute/(#14)** |
| **&H015A** | **MACROS 1/Execute/(#5)** | **&H01E6** | **MACROS 1/Execute/(#15)** |
| **&H015C** | **MACROS 1/Execute/(#6)** | **&H01E7** | **MACROS 1/Execute/(#16)** |
| **&H015E** | **MACROS 1/Execute/(#7)** | **&H01E8** | **MACROS 1/Execute/(#17)** |
| **&H0160** | **MACROS 1/Execute/(#8)** | **&H01E9** | **MACROS 1/Execute/(#18)** |
| **&H0162** | **MACROS 1/Execute/(#9)** | **&H01EA** | **MACROS 1/Execute/(#19)** |
| **&H0164** | **MACROS 1/Execute/(#10)** |  |  |

Execute macro.

**&H022D  MACROS 1/Interactive/OFF**

While creating a macro, you may be presented with various requesters, that require a string.  When you run the macro, the requesters do not normally appear, instead the functions that require strings get them from inside the macro body.  By selecting "MACROS 1/Interactive/ON", you can force ReSource to display requesters during macro execution, allowing you to view the strings that were used during the macro creation, and you may even change them for this macro invocation.

**&H01FF  MACROS 1/Suspend learn/Normal**

While creating a macro, you may have to use some functions in order to continue the macro creation, but you do not want the functions actually recorded into the macro.  In this situation, you can select "suspend", do what you have to do, and then turn "suspend" off again, to continue the macro creation as normal.

## Button Window Functions

| | | | |
|---|---|---|---|
| **&H01D1** | **MACROS 2/Create/(#1)** | **&H01DB** | **MACROS 2/Create/(#11)** |
| **&H01D2** | **MACROS 2/Create/(#2)** | **&H01DC** | **MACROS 2/Create/(#12)** |
| **&H01D3** | **MACROS 2/Create/(#3)** | **&H01DD** | **MACROS 2/Create/(#13)** |
| **&H01D4** | **MACROS 2/Create/(#4)** | **&H01DE** | **MACROS 2/Create/(#14)** |
| **&H01D5** | **MACROS 2/Create/(#5)** | **&H01DF** | **MACROS 2/Create/(#15)** |
| **&H01D6** | **MACROS 2/Create/(#6)** | **&H01E0** | **MACROS 2/Create/(#16)** |
| **&H01D7** | **MACROS 2/Create/(#7)** | **&H01E1** | **MACROS 2/Create/(#17)** |
| **&H01D8** | **MACROS 2/Create/(#8)** | **&H01E2** | **MACROS 2/Create/(#18)** |
| **&H01D9** | **MACROS 2/Create/(#9)** | **&H01E3** | **MACROS 2/Create/(#19)** |
| **&H01DA** | **MACROS 2/Create/(#10)** | | |

Start recording a macro.

| | | | |
|---|---|---|---|
| **&H01EB** | **MACROS 2/Execute/(#1)** | **&H01F5** | **MACROS 2/Execute/(#11)** |
| **&H01EC** | **MACROS 2/Execute/(#2)** | **&H01F6** | **MACROS 2/Execute/(#12)** |
| **&H01ED** | **MACROS 2/Execute/(#3)** | **&H01F7** | **MACROS 2/Execute/(#13)** |
| **&H01EE** | **MACROS 2/Execute/(#4)** | **&H01F8** | **MACROS 2/Execute/(#14)** |
| **&H01EF** | **MACROS 2/Execute/(#5)** | **&H01F9** | **MACROS 2/Execute/(#15)** |
| **&H01F0** | **MACROS 2/Execute/(#6)** | **&H01FA** | **MACROS 2/Execute/(#16)** |
| **&H01F1** | **MACROS 2/Execute/(#7)** | **&H01FB** | **MACROS 2/Execute/(#17)** |
| **&H01F2** | **MACROS 2/Execute/(#8)** | **&H01FC** | **MACROS 2/Execute/(#18)** |
| **&H01F3** | **MACROS 2/Execute/(#9)** | **&H01FD** | **MACROS 2/Execute/(#19)** |
| **&H01F4** | **MACROS 2/Execute/(#10)** | | |

Execute macro.

| | | | |
|---|---|---|---|
| **&H0324** | **MACROS 3/Create/(#39)** | **&H0343** | **MACROS 3/Create/(#11)** |
| **&H033A** | **MACROS 3/Create/(#2)** | **&H0344** | **MACROS 3/Create/(#12)** |
| **&H033B** | **MACROS 3/Create/(#3)** | **&H0345** | **MACROS 3/Create/(#13)** |
| **&H033C** | **MACROS 3/Create/(#4)** | **&H0346** | **MACROS 3/Create/(#14)** |
| **&H033D** | **MACROS 3/Create/(#5)** | **&H0347** | **MACROS 3/Create/(#15)** |
| **&H033E** | **MACROS 3/Create/(#6)** | **&H0348** | **MACROS 3/Create/(#16)** |
| **&H033F** | **MACROS 3/Create/(#7)** | **&H0349** | **MACROS 3/Create/(#17)** |
| **&H0340** | **MACROS 3/Create/(#8)** | **&H034A** | **MACROS 3/Create/(#18)** |
| **&H0341** | **MACROS 3/Create/(#9)** | **&H0327** | **MACROS 3/Create/(#19)** |
| **&H0342** | **MACROS 3/Create/(#10)** | | |

Start recording a macro.

| | | | |
|---|---|---|---|
| **&H0323** | **MACROS 3/Execute/(#39)** | **&H0331** | **MACROS 3/Execute/(#11)** |
| **&H0328** | **MACROS 3/Execute/(#2)** | **&H0332** | **MACROS 3/Execute/(#12)** |
| **&H0329** | **MACROS 3/Execute/(#3)** | **&H0333** | **MACROS 3/Execute/(#13)** |
| **&H032A** | **MACROS 3/Execute/(#4)** | **&H0334** | **MACROS 3/Execute/(#14)** |
| **&H032B** | **MACROS 3/Execute/(#5)** | **&H0335** | **MACROS 3/Execute/(#15)** |
| **&H032C** | **MACROS 3/Execute/(#6)** | **&H0336** | **MACROS 3/Execute/(#16)** |
| **&H032D** | **MACROS 3/Execute/(#7)** | **&H0337** | **MACROS 3/Execute/(#17)** |
| **&H032E** | **MACROS 3/Execute/(#8)** | **&H0338** | **MACROS 3/Execute/(#18)** |
| **&H032F** | **MACROS 3/Execute/(#9)** | **&H0339** | **MACROS 3/Execute/(#19)** |
| **&H0330** | **MACROS 3/Execute/(#10)** | | |

Execute macro.

## Button Window Functions

### &H03AF  OPTIONS 1/Abs size specifiers/Longword/OFF

Used to disable the particular type of size qualifier.  See the "ON" function for details.

### &H0363  OPTIONS 1/Abs size specifiers/Longword/ON

If enabled, any absolute longword effective address will have the ".L" size qualifier displayed:

```
                    MOVEA.L   (4),A6          ;no size qualifier
                    MOVEA.L   (4).L,A6        ;uses size qualifier
```

### &H03B0  OPTIONS 1/Abs size specifiers/Optimize/OFF

Used to disable the particular type of size qualifier.  See the "ON" function for details.

### &H0364  OPTIONS 1/Abs size specifiers/Optimize/ON

If enabled, any absolute longword effective address which could be optimized to an absolute word effective address will have the ".W" size qualifier displayed.  Note that the ".L" size qualifier will only be shown if the OPTIONS 1/Abs size specifiers/Longword" switch is currently on, and the ".W" will only be shown if the "OPTIONS 1/Abs size specifiers/Word" switch is currently on.

### &H03AE  OPTIONS 1/Abs size specifiers/Word/OFF

Used to disable the particular type of size qualifier.  See the "ON" function for details.

### &H0362  OPTIONS 1/Abs size specifiers/Word/ON

If enabled, any absolute word effective address will have the ".W" size qualifier displayed:

```
                    MOVEA.L   (4),A6          ;no size qualifier
                    MOVEA.L   (4).W,A6        ;uses size qualifier
```

### &H02BB  OPTIONS 1/Allow/Auto labels/OFF
### &H02BA  OPTIONS 1/Allow/Auto labels/ON

This function determines whether the "LABELS/Create multiple/Reloc32" function is called automatically after every "Open load file".

## Button Window Functions

**&H037C  OPTIONS 1/Allow/EQUate value checks/OFF**
**&H037B  OPTIONS 1/Allow/EQUate value checks/ON**

If enabled, whenever a symbol is created, the symbol and symbol values are compared to those in the equates table. If there is a clash of symbol names (where the same symbol has two different symbol values) a macro fail will result, and the symbol will NOT be created. See: "CURSOR/Relative/Next error line"

**&H03B5  OPTIONS 1/Allow/Error comments/OFF**
**&H03B4  OPTIONS 1/Allow/Error comments/ON**

If enabled, "error comments" will be inserted into the saved ".asm" file. These are actually full-line comments that ReSource adds, to lines for which an error has been detected. The types of errors detected are set by the "OPTIONS/Error detection/:" functions.

For example, if you had "OPTIONS/Error detection/Library calls" enabled, and you saved a file containing the line:

```
                    JSR       (-$126,A6)
```

it would have a full-line comment appended:

```
                    JSR       (-$126,A6)
;fiX "_LVO" type symbol expected
```

The word "fiX" is meant to give you something that you can search for easily, in your text editor.

**&H0219  OPTIONS 1/Allow/Reference recognition/OFF**
**&H0218  OPTIONS 1/Allow/Reference recognition/ON**

If this option is set to ON, references to memory locations within the current file are recognized. For example, assuming that the absolute address of the start of the current file is at $200000, the following lines might be displayed with Reference recognition ON:

```
                    dl        $48638335
                    dl        START+$1097
                    dl        START+$3086
```

whereas if reference recognition was OFF, it would be displayed as:

```
                    dl        $48638335
                    dl        $201097
                    dl        $203086
```

Reference recognition is ON by default. If any pointer (32 bits) is in a reloc32 area, the reference will be recognized regardless of the setting of this option. If you want to disable only some references, you could use a macro that gets the symbol value, and creates a symbol using the value as the symbol itself:

STRINGS/Get/Symbol value
LABELS/Create single/Symbol

## Button Window Functions

### &H0295  OPTIONS 1/Assembler/Cape

This function specifies output suitable for the CAPE(TM) assembler.  Having selected this function does not guarantee CAPE(TM) compatible output - there are various other options that will result in non-compatible output, such as "OPTIONS/Pseudo opcodes/PUSH/POP/ON".

### &H02B8  OPTIONS 1/Assembler/Macro68

This function specifies output suitable for the Macro68(TM) macro assembler.

### &H0294  OPTIONS 1/Assembler/Metacomco

This function specifies output suitable for the Metacomco(TM) assembler. Having selected this function does not guarantee Metacomco compatible output - there are various other options that will result in non-compatible output, such as "OPTIONS/Pseudo opcodes/PUSH/POP/ON".

### &H018B  OPTIONS 1/Show/Chip-load info/OFF
### &H018A  OPTIONS 1/Show/Chip-load info/ON

If this option is set to ON, where a section statement is displayed, and that section either is forced to load into chip memory, or is forced to load into fast memory, a ",CHIP" or ",FAST" will be appended to the section statement.

Not all assemblers support this parameter for the section statement, so you may want to switch this option OFF if your assembler can't handle it.

### &H035C  OPTIONS 1/Show/Data comments/OFF
### &H035B  OPTIONS 1/Show/Data comments/ON

If enabled, comments will be created for every line, consisting of the hex bytes that make up that line.  This may be useful when comparing disassemblies of two files, where you want to compare not only the mnemonics and effective addresses, but the exact opcodes that make up the mnemonics and effective addresses also.

## Button Window Functions

**&H01B1  OPTIONS 1/Show/DCB statements/OFF**
**&H01B0  OPTIONS 1/Show/DCB statements/ON**

If this option is set to ON, "dcb" type statements will be used where appropriate.  For example, the following data:

```
                    db       0
                    db       0
                    dl       7
                    dl       8
                    dl       8
                    dl       8
                    dl       8
                    dl       8
                    dl       8
```

will be shown as:

```
                    dcb.b    2,0
                    dl       7
                    dcb.l    6,8
```

This is useful for shrinking the required .asm file  size,  especially  where there are large areas of zeroes.

**&H018F  OPTIONS 1/Show/End statement/OFF**
**&H018E  OPTIONS 1/Show/End statement/ON**

If this option is set to ON, the "END" statement will be displayed as the last line in the file.

**&H0181  OPTIONS 1/Show/End-of-line comments/OFF**
**&H0180  OPTIONS 1/Show/End-of-line comments/ON**

If this option is set to ON, end-of-line comments will be displayed. Normally, you will only turn end-of-line comments off when comparing ".asm" files, where you are looking for something other than comment differences.

**&H0183  OPTIONS 1/Show/Full-line comments/OFF**
**&H0182  OPTIONS 1/Show/Full-line comments/ON**

If this option is set to ON, full-line comments will be displayed. Normally, you will only turn full-line comments off when comparing ".asm" files, where you are looking for something other than comment differences.

**&H0189  OPTIONS 1/Show/Hidden labels/OFF**
**&H0188  OPTIONS 1/Show/Hidden labels/ON**

If this option is set to ON, hidden labels (labels attached to a byte in the middle of a line of code/data) will be displayed.

## Button Window Functions

**&H02A8  OPTIONS 1/Show/Label colons/OFF**
**&H02A7  OPTIONS 1/Show/Label colons/ON**

This option will display colons ,":", after all labels.

**&H017D  OPTIONS 1/Show/Labels/OFF**
**&H017C  OPTIONS 1/Show/Labels/ON**

If this option is set to ON, labels will be displayed.
Normally, you will only turn labels off when comparing ".asm" files, where you are looking for something other than label changes.

**&H029A  OPTIONS 1/Show/Leading zeroes/OFF**
**&H029B  OPTIONS 1/Show/Leading zeroes/ON**

Use these options to specify whether or not you wish the leading zeroes on hex values to be shown, e.g., "$0000003A" versus "$3A".

**&H02C9  OPTIONS 1/Show/Multiple constants/OFF**
**&H02C8  OPTIONS 1/Show/Multiple constants/ON**

For byte, word, and longword lines, this option allows you to have many data constants displayed on each line, rather than just one.  ReSource will limit the number of constants to your display width.

**&H035D  OPTIONS 1/Show/New Syntax/OFF**
**&H035E  OPTIONS 1/Show/New Syntax/ON**

Select old/new syntax.  Old syntax is suitable only for disassembly programs written specifically for the 68000 CPU.  If you plan to use the "Metacomco" or "CAPE" assemblers, then you will be stuck with using old syntax.  If you own the "Macro68" assembler, you are free to use new syntax for all 68000, 68010, 68020 and 68030 programs.

Many old assemblers incorrectly specified byte size branches as "Bcc.S" and word size branches as "Bcc.L".  Although this is incorrect, in order to output code that will assemble properly, ReSource must use the same mnemonics instead of the proper "Bcc.B" for a byte size branch, and "Bcc.W" for a word size branch.  The correct mnemonics will be used with new syntax.

**&H035E  OPTIONS 1/Show/New Syntax/ON**

Select old/new syntax.  Old syntax is suitable only for disassembly programs written specifically for the 68000 CPU.  If you plan to use the "Metacomco" or "CAPE" assemblers, then you will be stuck with using old syntax.  If you own the "Macro68" assembler, you are free to use new syntax for all 68000, 68010, 68020 and 68030 programs.

Many old assemblers incorrectly specified byte size branches as "Bcc.S" and word size branches as "Bcc.L".  Although this is incorrect, in order to output code that will assemble properly, ReSource must use the same mnemonics instead of the proper "Bcc.B" for a byte size branch, and "Bcc.W" for a word size branch.  The correct mnemonics will be used with new syntax.

## Button Window Functions

### &H0171  OPTIONS 1/Show/Offsets/OFF

Turns off the displaying of offsets on lines that have no labels.

See "OPTIONS/Show.../Offsets/ON"

### &H0170  OPTIONS 1/Show/Offsets/ON

On lines that do not have a label, display the offset of the start of that line, as a 6-digit hex number.  This can be handy when comparing two saved ".asm" files, as it allows you to compare not only the disassembled code, but how long each instruction was also.

For example, after reassembling a file, you may find that the file length differs, yet the saved ".asm" of the reassembled file appears to be identical to the original disassembly.  What is probably happening in this case is that your assembler made different optimizations than the assembler which created the original program.  For example, the "movea.l (4),A6" instruction can be either 4 or 6 bytes long, depending on which effective address was used.

### &H018D  OPTIONS 1/Show/Section statements/OFF
### &H018C  OPTIONS 1/Show/Section statements/ON

If this option is set to ON, section statements will be displayed.

### &H02A6  OPTIONS 1/Show/Separate labels/OFF
### &H02A5  OPTIONS 1/Show/Separate labels/ON

Under normal conditions, if a label is longer than 20 characters, it is broken at that point, and the remainder is shown on the next line.  This option forces the entire label, regardless of its length, to be shown on one line.

## Button Window Functions

**&H035F  OPTIONS 1/Show/Strict Mnemonics/OFF**
**&H0360  OPTIONS 1/Show/Strict Mnemonics/ON**

There are several instructions that are created by assemblers, which can in the source code use a more generic name.  For example, many assemblers will accept the "MOVE" instruction where a "MOVEA" instruction should have been used.  Similarly, where a "ORI" instruction should be used, many assemblers will accept just "OR" instead.  Macro68 can operate in either strict or relaxed mode.  This means that in "relaxed" mode, if you use an instruction that doesn't make complete sense, it will try to guess as to what you really meant.  In strict mode, you will be expected to supply the correct instructions every time.  ReSource supports the strict and relaxed modes as used by Macro68:

| Strict mnemonic | Relaxed mnemonic |
|---|---|
| ADDA | ADD |
| ADDI | ADD |
| ANDI | AND |
| CMPA | CMP |
| CMPI | CMP |
| DIVS.W | DIVS |
| DIVU.W | DIVU |
| EORI | EOR |
| LINK.W | LINK |
| MOVEA | MOVE |
| MULS.W | MULS |
| MULU.W | MULU |
| ORI | OR |
| SUBA | SUB |
| SUBI | SUB |

**&H017F  OPTIONS 1/Show/Symbols/OFF**
**&H017E  OPTIONS 1/Show/Symbols/ON**

If this option is set to ON, symbols will be displayed.
Normally, you will only turn symbols off when comparing ".asm" files, where you are looking for something other than symbol differences.

**&H03AB  OPTIONS 2/Error detection/AFLINE/OFF**
**&H03A7  OPTIONS 2/Error detection/Bad alignment/OFF**
**&H03A8  OPTIONS 2/Error detection/Code reference/OFF**
**&H03A5  OPTIONS 2/Error detection/Code terminate/OFF**
**&H03A9  OPTIONS 2/Error detection/Data reference/OFF**
**&H03AD  OPTIONS 2/Error detection/Illegal code/OFF**
**&H03AC  OPTIONS 2/Error detection/Library calls/OFF**
**&H03A6  OPTIONS 2/Error detection/Missing label/OFF**
**&H03AA  OPTIONS 2/Error detection/START+/OFF**

Used to disable the particular type of error detection.  See the "ON" function for details.

## Button Window Functions

**&H030F  OPTIONS 2/Error detection/AFLINE/ON**
**&H030B  OPTIONS 2/Error detection/Bad alignment/ON**
**&H030C  OPTIONS 2/Error detection/Code reference/ON**
**&H0309  OPTIONS 2/Error detection/Code terminate/ON**
**&H030D  OPTIONS 2/Error detection/Data reference/ON**
**&H0379  OPTIONS 2/Error detection/EQU values/ON**
**&H0311  OPTIONS 2/Error detection/Illegal code/ON**
**&H0310  OPTIONS 2/Error detection/Library calls/ON**
**&H030A  OPTIONS 2/Error detection/Missing label/ON**
**&H030E  OPTIONS 2/Error detection/START+/ON**

Set/clear types of errors to detect.  See:
"CURSOR/Relative/Next error line" for details.

**&H0676  OPTIONS 2/Interface/Delayed refresh/OFF**
**&H0675  OPTIONS 2/Interface/Delayed refresh/ON**

Making ReSource Intuition compatible has slowed down TitleBar rendering, and consequently functions that update the TitleBar frequently.  This function was added to cope with this problem.

When this switch is set, and the following conditions are met, the TitleBar will only be updated once per page (~256 bytes).  Note that the refresh will not necessarily happen on a page boundary.

1. The function being executed is a Normal or Pattern search, a Backward Reference, or "PROJECT/Disassemble",
2. A message does not need to be displayed, or
3. The TitleBar does not need to refreshing for other reasons.

This default for this switch is OFF.

**&H0173  OPTIONS 2/Interface/Display Beep/OFF**
**&H0172  OPTIONS 2/Interface/Display Beep/ON**

Many functions demand attention from the user, and a DisplayBeep() is used for this purpose.  Normally, this will make the screen flash.  However, if you have run the program "BeepIt", or a similar DisplayBeep() replacement, you will hear an audible "beep" instead.  If you don't want the flash/beep, then switch it off.

**&H002B  OPTIONS 2/Interface/Feedback Delays/OFF**
**&H002A  OPTIONS 2/Interface/Feedback Delays/ON**

If you have User feedback set to "OFF", then the setting of this function is irrelevant.  If feedback is ON, you may occasionally find that feedback messages are being displayed too fast for you to read them.  This is especially so when opening a load file, since many informative messages can be displayed in between raster scans, which means that you don't actually get to see some of the messages at all.

If you set feedback delays to "ON", there will be a one second delay after each message is displayed, which should be plenty of time to read each message.  While you hold the menu button down, feedback messages are skipped altogether.

## Button Window Functions

**&H0027  OPTIONS 2/Interface/User Feedback/OFF**

Stops ReSource from offering informative title-bar messages.

**&H0026  OPTIONS 2/Interface/User Feedback/ON**

Allows ReSource to offer informative messages when appropriate.

**&H02B5  OPTIONS 2/Interface/Verbose saves/OFF**
**&H02B3  OPTIONS 2/Interface/Verbose saves/ON**

The "SAVE/Save .asm" functions give you a profile of the source code as it is saved.  This may be disabled by selecting the "OPTIONS/Allow.../Verbose saves/OFF" function.  Each entry in the displayed table is counted as that condition occurs during the save, and the totals are displayed continuously.  It is possible to get a printout of this and any other screen in ReSource, using the new "SAVE/Save screen" function.  Speed penalty for enabling this function is approx. 10%.

**&H038E  OPTIONS 2/Pseudo opcodes/BLO/BHS/OFF**
**&H038D  OPTIONS 2/Pseudo opcodes/BLO/BHS/ON**

If enabled, "BCC" instructions are replace by "BHS", and "BCS" instructions are replace by "BLO".

**&H038A  OPTIONS 2/Pseudo opcodes/PUSH/POP/OFF**
**&H0389  OPTIONS 2/Pseudo opcodes/PUSH/POP/ON**

If enabled, instructions of the "MOVE.L" type that move a register from/to the stack are replaced by the "PUSH" and "POP" pseudo-opcodes, as used in the Macro68(TM) assembler.

**&H038C  OPTIONS 2/Pseudo opcodes/PUSHM/POPM/OFF**
**&H038B  OPTIONS 2/Pseudo opcodes/PUSHM/POPM/ON**

If enabled, instructions of the "MOVEM.L" type that move registers from/to the stack are replaced by the "PUSHM" and "POPM" pseudo-opcodes, as used in the Macro68(TM) assembler.

**&H03B1  OPTIONS/Error detection/EQU values/OFF**

Used to disable the particular type of error detection.  See the "ON" function for details.

## Button Window Functions

**&H02F7  SYMBOLS/Load user symbols/#1**
**&H02F8  SYMBOLS/Load user symbols/#2**
**&H02F9  SYMBOLS/Load user symbols/#3**
**&H02FA  SYMBOLS/Load user symbols/#4**
**&H02FB  SYMBOLS/Load user symbols/#5**
**&H02FC  SYMBOLS/Load user symbols/#6**
**&H02FD  SYMBOLS/Load user symbols/#7**
**&H02FE  SYMBOLS/Load user symbols/#8**
**&H02FF  SYMBOLS/Load user symbols/#9**
**&H0300  SYMBOLS/Load user symbols/#10**
**&H0301  SYMBOLS/Load user symbols/#11**
**&H0302  SYMBOLS/Load user symbols/#12**
**&H0303  SYMBOLS/Load user symbols/#13**
**&H0304  SYMBOLS/Load user symbols/#14**
**&H0305  SYMBOLS/Load user symbols/#15**

Load a user-defined symbol base.  If you select one that has already been loaded, it will replace the previous one.  When you use the "PROJECT/Save Config" function, it also saves the names of the user-defined symbol bases, as those given when you first loaded them.  So, to ensure that ReSource will always be able to load your user-defined symbol bases at startup, when loading a symbol base, supply a meaningful pathname, rather than just the filename by itself (as you might do if the symbol base were in the current directory at the time).

**&H02DC  SYMBOLS/User-defined symbols/#1**
**&H02DD  SYMBOLS/User-defined symbols/#2**
**&H02DE  SYMBOLS/User-defined symbols/#3**
**&H02DF  SYMBOLS/User-defined symbols/#4**
**&H02E0  SYMBOLS/User-defined symbols/#5**
**&H02E1  SYMBOLS/User-defined symbols/#6**
**&H02E2  SYMBOLS/User-defined symbols/#7**
**&H02E3  SYMBOLS/User-defined symbols/#8**
**&H02E4  SYMBOLS/User-defined symbols/#9**
**&H02E5  SYMBOLS/User-defined symbols/#10**
**&H02E6  SYMBOLS/User-defined symbols/#11**
**&H02E7  SYMBOLS/User-defined symbols/#12**
**&H02E8  SYMBOLS/User-defined symbols/#13**
**&H02E9  SYMBOLS/User-defined symbols/#14**
**&H02EA  SYMBOLS/User-defined symbols/#15**

Create a symbol, using a user-defined symbol base.  The user-defined symbol base must have been previously loaded using "SYMBOLS 1/Load user symbols/". For details on the format of a user-defined symbol base, see the accompanying text file "UserSymbols.doc".