DISA 2

Interactive Disassembler for QDOS and SMS2

by Jochen Hassler

Distribution
Albin Hessler Software
Im Zeilfeld 25
D-72631 Aichtal
Tel+Fax 0049 (0)7127 56280

Author
Jochen Hassler
Heilbronner Str. 7
D-73728 Esslingen
Tel. 0049 (0)711 3160212

Table of Contents

License Agreement

As a user of the DISA Software you agree with the following terms:

The DISA program is copyrighted by Jochen Hassler, which remains the owner of this software. The user is granted the non-transferable license to use this software on a single computer. The user may make safety and working copies for his own purposes. The user accepts, not to give away or sell any copy of the software. The manual is property of the user but is also copyrighted by Jochen Hassler and may not be copied.

This license becomes only valid if the filled in user registration form is sent back to Albin Hessler Software.

Limited warranty

The DISA program and this manual have been developed with great care. As we can not control all possible hard- and software constellations under which the program may be used, we can not give any warranty for the perfect functioning of this software at any time. Especially we are not responsible for resulting damages of a malfunction which may occur by any reason in conjunction with the usage of this software.
However we warrant for an error free diskette with an error free copy of the program. In case of any damage which occurs under normal use, we replace the disk without charge within a period of six months, provided the user registration has been sent back to Albin Hessler Software.

Software updates

Software updates are normally free if the original disk is sent to us together with the cost for the return postage, which can be covered by international reply coupons.

Software upgrades

In case of major changes to the program we are free to offer special upgrade versions to all registered users for a special price.

Getting started

DISA runs as a job under the extended Pointer Environment from QJUMP (see chapter **The Pointer Environment**). You must load **ptr_gen** V1.38 or later, and **wman** V1.33 or later, before you can start DISA. If these are not present, you will get an error report.

In addition, you may load resident **menu_rext** (from Jochen Merz). The File-Select menu will then be used on loading or saving files. If QPAC2 is active, DISA may sleep in the button frame.

DISA is started as a normal job, just by typing **EX DISA,** or by QPAC2 or Cueshell start button. DISA is written as romable, "clean" assembler code which does not change itself on run time. Thus it is possible to run DISA directly out of an Eprom, start it as a THING, or define a hotkey and start several jobs from the same code. There are no problems or restrictions on a 68020/68030 - system with cache.

The start syntax accepts a command string, e.g.:

EX flp1_DISA;"flp2_testcode"

NOTE: DISA is also available as a "Demo version". It is not possible to save data with a demo version. If you want to upgrade your demo version into a full version, please contact your distributor.

DISA-idea

DISA provides you with listings, regained from 68000-machine code, for assembling. This can not be done automatically, however. In this sense DISA is not an 'intelligent' disassembler.
The main problem on disassembling is to distinguish program code from data structures. It is nearly impossible to do this automatically, and it always remains erroneous. Therefore the DISA idea goes like this:
The user sees several lines of normal disassembly on the screen as if it were an editor. He himself decides if it is program code or some kind of data code. From within the context, this is rather easy and secure. The user should have a basic understanding of the Motorola-68000-assembler language, of course.
A graphic menu environment as well as numerous predefined functions facilitate this job. The result may immediately be seen, and, if necessary, be corrected.
The editing is done by attaching attributes to program lines, like **code** or **dc.w**, depending on how the line concerned and the following lines of the "_dis"-listing are to be presented.
All informations about the new listing are written to a special file, the so called "index-file". This index-file may be saved to disk and reloaded later.

The menu

DISA has been written as a mouse driven program. Program control is possible with arrow- and select keys too, but it is not as comfortable and more slowly (!) than with mouse control.

The selection keys are underlined or follow the normal convention. The left mouse button (HIT = SPACE) means select or deselect, the right mouse button (DO = ENTER) means direct action. NOTE: In general, the left and right mouse button have different reaction.

The mouse pointer normally looks like a little arrow pointing upwards. Sometimes it changes in shape. The meanings are:

- an arrow pointing downwards: text moving back
- a watch: DISA is very busy, it may last...
  This option can be cancelled immediately by a HIT (but with care!).
- an arrow pointing straight to the left: here you can set a marker
- an arrow pointing straight to the right: here you can set an M-label
- a little box with an arrow pointing down: put the code word to the next line down
- a little box with an arrow pointing up: put this code word back to the previous line again

If you do not use a mouse until now, please think it over again. It may be a good idea. Why not take the famous "SERMouse" (from Albin Hessler Software), and plug in a normal PC-mouse into ser2. All done!

The different menu items as an overview

Move sprite **[^F4]**
Move around the whole job on the screen.

Window sprite **[^F3]**
The job window grows or shrinks in size. You can have 5...32 lines, depending on your screen. On a normal QL screen there are 5...16 lines, on an Atari screen perhaps a little more.

Sleep sprite **[^F1]**
The job gets a button. This is only possible, if QPAC2 is installed. If not, there will be a nasty beep.

ESC sprite **[ESC]**
The job is cancelled and removed. All data will be lost. If you have not already saved the data, DISA kindly asks you to do so by presenting the FILES pull down menu. You may ignore this and exit with another [ESC].

STATUS window **[F1]**
Pull down menu to configure DISA at run time. All changes get valid immediately.

DISA menu item **[F2]**
Start the internal disassembler to reconstruct all labels and pointers. Be patient, this action will take a little time...

FILES window **[F3]**
Pull down menu to load and save data.
With a DEMO VERSION, it is not possible to save data.

SEARCH window **[F4]**
Pull down menu to search pointers, text, hex figures and to jump to lines and labels.
With a HIT: present pull down window
With a DO: repeat or continue last action

EDITOR menu item **[F5]**
with a HIT: switch between EDITOR and WINDOW mode with a DO: select EDITOR mode and jump to the next label

dc.b ... ascii menu items **[b,w,l,p,a,d,t,c,s]**
These menu items are used for editing the assembler code in EDITOR mode. They are explained in detail in chapter "Attributes for Editing".

STATUS menu [F1]

Pull down menu to configure DISA at run time. All the following items can also be configured using a Standard Configuration Program (Config or MenuConfig, added on your source disk).

Default device and path **[#]**
If the file can not be found under the given name, the default device and subdirectory path are added to the name. The TK2 defaults "DATAD$" and "PROGD$" are not affected at all.

Comment Field **[F]**
This character will be used to separate opcode and comments within the same assembly line.

Comment Character **[O]**
This character will be used at the beginning of a line containing comments only.

QDOS / SMS2 **[Q],[M]**
decides whether the trap names and vector names of QDOS or of SMS2 are used. If both menu items are deselected, no automatic comments are generated.

(PC) **[P]**
decides, whether pc relative labels are listed with or without "(pc)", e.g. lea L0000(pc),a0 or lea L0000,a0. Each assembler has its own unique syntax...

>Bcc **[>]**
When this option is selected, all conditioned jumps are indented by two positions (according to Motorola conventions). This will facilitate the reading of listings, once you got used to it.

CAPS **[C]**
If this option is selected, opcode is displayed in capitals, otherwise in small letters.

a6=BASIC DEVICE **[B],[D]**
This fixes the meaning of address register a6:

BASIC: a6 points to the SuperBasic area
e.g. SuperBasic extension code
DEVICE: a6 points to the system variables
e.g. Device driver
deselected: a6 has no fixed meaning
e.g. job code

H/V **[H]**

decides whether labels pointing out of the code area before code start (V-label) or behind code end (H-label) are listed as equ-lines or not.

SOUND **[O]**

gives a BEEP with error and status messages. Switch it off if it makes you nervous.

Cmd1, Cmd2 **1],[2]**

these texts are automatically inserted into the header of a listing. They are meant as standard assembler directives. e.g. "section", "nolist", etc.

FILES menu [F3]

Pull down window to load and save data.

Within the files menu, the following conventions are valid:
Select the desired menu item with the left mouse button HIT:

- immediate input and line editing
  The default device and subdirectory path is tried if needed.

Select the desired menu item with the right mouse button DO

- if menu_rext is not active, immediate input and line editing as above
- if menu_rext is active, call the file select menu from menu_rext.

  When calling the file select menu, the default device and subdirectory path will be used (ref STATUS window). Closing the files menu will write back the setting of device and path last used and is taken as a new default on the next call.

1. Load program code from file [f]

Enter and edit the desired filename in the LOAD file window and press the right mouse button (DO). The file will be loaded. In the LOAD memory window file length (in Hex bytes) and absolute address in memory used will appear. In the SAVE file window a default file name for the disassembled listing (extension _dis) is proposed, and in the SAVE index window a default file name for the index file (extension _idx) is proposed. It is possible of course to alter and edit these names, extensions included. If no index file has been created in an earlier session, you will now exit the LOAD menu with a DO or ESC. DISA automatically creates a new index file on disassembling the whole file for the first time. Please be patient, this will perhaps take a little time.

2. Load program code from memory [m]

Instead of program code out of a file, it is possible to define directly an area in memory to be taken as program code.
Syntax: "startaddress endaddress" (both in Hex, no preceding "$") into the LOAD memory window. Then the same procedure as above. A memory area greater than 256 kb = $3FFFF bytes will not be accepted (?!).

3. Load an index file * See below [f]

A saved index file (see later) created earlier can be loaded in the LOAD file window. It is provided that a code file (or a memory area) is loaded, and that the index file really belongs to the code. In this case, no new index file will be created on exit. The file name of the index file is not of any interest, and the extension _idx is not necessary. It is not possible however to load an index file from DISA Version 1.03 or older into a DISA Version 2.00 or later because of some major differences in index file structure.

4. SAVE a disassembly [l]

Enter and edit the desired filename in the SAVE file window and press the right mouse button (DO). The file will be saved as a disassembler text file in the _dis file. No control characters or non printable characters are used at all (except LF, of course). If the whole file has been properly edited, every assembler can reassemble the text. The syntax needed by the own assembler version (Qmac, Quanta, C1,...) can be adjusted with the STATUS window, the ascii attribute and the assembler directives. Remark: There should be enough free space on your diskette. As a rule: _dis file length = 10 x code file length (or even more).

5. SAVE an index file [i]

Enter and edit the desired filename in the SAVE index window and press the right mouse button (DO). The file will be saved as an index file, normally ending with _idx. So it is possible to end a lengthy session, save all the work done and continue the other day.

*① Files -> load binary file.
ESC or Do window!
Default editor appears ([illegible])

② Files -> load
Files : enter idx file
ESC or do window
EDITOR appears. Index is loaded.

③ F5 -> edit mode

SEARCH menu [F4]

Pull down window to search pointers, text, Hex figures and to select line numbers and labels.

Within the SEARCH window, the following conventions are valid:

- pressing the left mouse button (HIT): select, but no action.
- pressing the right mouse button (DO): enter or edit item.
  exception: go to top / bottom is done immediately.
- DO the window: do the selected item(s).
  The action starts with the first line seen in the window. If you want to start action at the top of file, select the **UP** item in addition.
- quit with **ESC**: do nothing, close SEARCH window.

To continue the search, it is not necessary to pull down the SEARCH window. DO-ing the SEARCH menu item again will do it. The name of the SEARCH menu item will change according to the last action used.

The SEARCH functions can be used from within both the EDITOR mode and the WINDOW mode. The line found is centred in the window and highlighted with a red border. If nothing (more) can be found, there will be a nasty beep.

The SEARCH functions in detail:

Line **[e]**
The given line number is searched and centred in the window. If this line does not exist, the next nearest line will be presented.

Text **[t]**
The given text is searched. Upper case and lower case are not the same. The text can contain 1...8 characters.

Label **[l]**
A label is given as a line number (without preceding "L"). The code is disassembled and searched for the desired pointer(s). It is a good idea to start DISA action (menu item [F2]) first in order to get the line itself to which the pointers are pointing.

Figure **[f]**
The given hex number is searched. Enter as usual without preceding "$". The hex number can be a byte, a word or a long word. Words and longwords must be aligned to even addresses, bytes must not. With the mask [m] the bit wise AND is performed prior to the search. In this way it is possible to mask out individual bits or groups of bits. The internal disassembler of DISA uses this principle to analyse the code. Some examples how to use bit masks are described in section Tips and Tricks.

Abs.Adr **[a]**
The program code is disassembled and each instruction is examined if an absolute address is used. If the address mode "absolute short" or "absolute long" is found in this instruction, either as "source" or "destination", this line is presented. With this option, you can find the well known but unbeloved direct accesses to system variables ($28000) and the screen memory ($20000...$2FFFF), or to the QL hardware ($18021 etc.), or to the operating system ($0000...$BFFF). Or if the "#" is missing with "immediate data"...
You can specify a range to show abs.adr:
The line **Abs.Adr. C0 =>to: 180** for example presents all calls to QDOS vectors, which are in the range of $00C0 to $0180 (Minerva).
The line **Abs.Adr. 180 =>to: C0** instead for example presents all instructions using absolut addresses, but no QDOS vectors.

Mark No **[n]**
Enter the number as 0...9, A...F. The given marker is selected if set. Otherwise there will be a beep. With a number greater F (e.g. FF), all markers set are toggled around. This is the default setting.

illg **[i]**
The code is disassembled and scanned for illegal instructions. After editing, no illegal instruction should be found. This would indicate that there are unrecognised data areas or jump tables left unedited.

Up **[u]**
If Up is selected, an option is started off from line 1. Do-ing this item will jump directly to top.

Down **[d]**
Do-ing this item will jump directly to bottom.

0 **[0]**
Deselect all items.

Attributes to be edited

All the following notes refer to the EDITOR mode. You get into EDITOR mode by selecting the EDITOR menu item.

The purpose of the EDITOR mode is to tell DISA, were program code, data space, pointers, jump tables or strings are, in order to define the appearance of the disassembler listing, as explained in section **DISA Idea**. Attributes **dc.b...ascii** are attached to a line by selecting the desired attribute with the left mouse button and dropping it with the right mouse button at the appropriate line. It is immediately displayed at the start of the line. The attribute is valid for the actual line and all following lines, until a new attribute is given.

code attribute **[c]**
Display as program **code**. This is the default setting.

dc.b attribute **[b]**
Display as **dc.b**. All printable characters ($20...$7F) will be displayed as letters, figures or special characters. All non-printable characters are displayed as hex-ascii figures. New line is executed by Linefeed ($0A), by SPACE ($20), or latest after 40 character in one line. You can toggle to the so called "Only-Hexascii-Display" by **ascii** menu item (see later).

dc.w attribute **[w]**
Display as **dc.w**. 8 words are put in one line. A new line is performed with each new attribute (which can be a new dc.w), or with a pointer to a word.

dc.l attribute **[l]**
Display as **dc.l**. 4 longwords are put in one line. A new line is performed with each new attribute (which can be a new dc.l), or with a pointer to a longword. DISA automatically switches back to dc.w display if necessary.

text attribute [t]

Display as QDOS text, length count as **dc.w**, text bytes as **dc.b**. The text attribute must point exactly to the word counter of the text. If the word counter is not at the start of a line, please refer to the section **Enhanced Editing**. An editing error is assumed, if the length count is bigger than $01FF = 511 bytes or negative and a beep is generated. If all is right, DISA drops a **dc.w** at the length counter word, marks all following bytes as **dc.b**, calculates the end of the text (which is well known!), puts an end attribute **^^^^** at the end, and realigns the following lines so that the word after the end of text is on a new line. Thus the disassembler is synchronised. Beginning with the end attribute **^^^^**, normal code display is selected.

prel attribute [p]

Display as pc-relative pointer

```
e.g.    dc.w    L0020-*
        dc.w    L0024-*
        dc.w    L013E-*
        etc.
```

The pointer itself may be word, longword, or byte pointer.

There is a special pull down menu window to define prel attributes as dc.w, dc.l or dc.b.
You get into the pull down menu window by clicking the **prel** attribute with the right mouse button (DO). The prel attribute with the lowest line number in the EDITOR window is changed. If there are several prel at a time, scroll the lines up until the desired prel is the first one off the top of window. The actual line number with the prel attribute is displayed in the prel pull down menu for confirmation.

arel attribute [a]

Display as address-relative pointer

```
e.g.    dc.w     L0020-L0010
        dc.w     L0022-L0010
        dc.w     L013A-L0010
        etc.
```

The pointer itself may be word, longword, or byte pointer.

There is a special pull down menu window to define **arel** attributes as dc.w, dc.l or dc.b. To get into the pull down menu window see above (prel attribute). As base address the line number containing the **arel** attribute is taken. In the example above, this would be line number 0010. But things are not so easy. It often happens, that the line containing the **arel** attribute is really not the base address of a jump table. For this reason, the menu item **Offset** in the **arel** pull down menu is provided. To start with, you see the same number in the **Offset** item and the **Line** item (that means Offset = Line, base address = base of jump table). Now you can change the reference line in the **Offset** item. Consider the example above. Assume the reference line to be line number 0. If you change the number in the **Offset** item to 0, you will get the following display:

```
e.g.    dc.w     L0010-L0000
        dc.w     L0012-L0000
        dc.w     L012A-L0000
        etc.
```

There is another menu item in the **arel** pull down window with name **last**. This means a reference line of a previous arel attribute. If selected, the reference line is no longer the line containing the current **arel** attribute, but the line containing the last **arel** attribute with **last** item not selected. This is true, even if there are other attributes (like dc.w etc.) in between different **arel** attributes. Such sophisticated data structures are common to operating systems. But the actual reference line is calculated and displayed immediately in the **Offset** item to give you a chance...

ascii attribute **[s]**
ascii/alternate display. This is a special attribute because it is not an attribute for itself, but changes the display of other attributes.

- with "#immediate data" hex <-> ascii.
  e.g. move.l #$464C5030,d0 <-> #'FLP0',d0
  the #-data may be bytes, words or longwords

- with "'immediate data" signed <-> unsigned.
  e.g. moveq #-$01,d0 <-> #$FF,d0
  the #-data may be bytes, words or longwords

- with dc.b-Attribut ascii <-> hex.
  e.g. dc.b $00,$01,'AB' <-> dc:b $00,$01,$41,$42
  The dc.b is displayed as dc:b with Only-Hex-Display

- with link-instruction signed <-> unsigned.
  e.g. link a6,#-$00C0 <-> a6,#$FF40

- with prel-attribute 0 and -1 as data <-> as pointer
  e.g. on line 0020 dc.w $0000 <-> dc.w L0020-*

- with arel-attribute 0 and -1 as data <-> as pointer
  e.g. on line 0020 dc.w $0000 <-> dc.w L0020-L0020

defp attribute **[d]**
Display QDOS/SMS2 defined procedures and data structures.

The following structures are recognised and decoded:

1. BASIC definition list from BP_INIT/BI_INIPR
2. Rom header with $4AFB0001 - flag
3. Job header with $4AFB - flag
4. Thing header with THG% - flag
5. Config Blocks with <<QCFX>>01 - flag
6. Window definitions from WM_SETUP (wman)
7. Sprite/Blob/Pattern definitions from ptr_gen

The defp attribute must point exactly to the first word of the structure. How to find the first word of a structure is different from case to case, and sometimes it is really difficult. Some hints are written in section "Tips and Tricks". DISA recognises automatically the type of structure and checks the syntax.

Only error free structures are scanned and decoded. If the QDOS/SMS2 syntax is not met, there will be this famous beep.

delete attributes:
Attributes on a line can be deleted at any time. Deselect all attribute items and click with the right mouse button on the attribute in the line. The attribute will be removed.

Enhanced editing

There are four more editing functions when you are in EDITOR mode:

Cutting off code words
Joining code words
Setting and deleting markers
Setting and deleting M-Label

They are, however, not accessible during normal editing, to avoid their activation by mistake. They will become accessible by selecting a line (line turns green or black, depending on paper colour).

Cutting off code words

Often, during the process of disassembling of data areas, quite arbitrarily senseless lines may be generated (the disassembler doesn't know better...). Therefore the possibility to re synchronise the disassembler has been provided. This is needed, when you want an attribute to point exactly to a certain code word, for example with QDOS texts.
Just move the cursor in a selected (green) line to the second, third or so code word. The cursor will change into a small box with an arrow pointing to the lower left corner. Hitting the right mouse button will cut off the last word in this line and forces the disassembler to start disassembling from this word on. All lines newly disassembled are now also displayed in green. This may be repeated until only one word is left in a line.
(Try it ...).

Joining code words

This has the opposite effect from **Cutting off code words.**
Move the cursor to the first code word in a selected line. When the standard cursor changes into a small box with an arrow pointing to the upper right corner, you have found a code word that has formerly been cut off. Hitting the right mouse button moves the code word back to the preceding line, and all following lines will be newly disassembled.

Only those words may be joined, which have been cut off before.

Setting and deleting markers

Move the cursor to the right end of a selected line. The mouse pointer changes shape to an arrow pointing to the left. With a click on the right mouse button you can drop a new marker with number <==0...<==9, <==A...<==F, or remove a marker, if it was already there. There is a maximum of 16 markers possible. They are saved in the index file (see SAVE window), and restored on load.

Setting and deleting M-Label

On some occasions it may be useful to put a label on a line, even if no pointer is pointing to it. Move the cursor to the left end of a selected line. The mouse pointer changes shape to an arrow pointing to the right. With a click on the right mouse button you can drop an M-label, or remove one, if it was already there.

EDITOR menu item [F5]

Toggles between EDITOR- and WINDOW-mode.

In **WINDOW** mode you can see the disassembled code according to the attributes set. By changing between EDITOR- and WINDOW-mode, you have full control over the effects of your editing. If the result does not look like you feel it should, you can easily change it. From time to time, it is worth while to start the DISA action [F2] to calculate label and pointers, and to delete pointers pointing into the nirvana.

The display in **WINDOW** mode is controlled by the settings in the **STATUS** window (look there). Traps #0...#4, QDOS-vectors $C0...$180 and QDOS error codes -1...-25 are named. The syntax is taken from the Technical Guide (Sinclair-QL) and from the QDOS/SMS2 Reference Manual (Tony Tebby/Jochen Merz). A blank line is inserted after each "rts", "bra.l" and "jmp" instruction for clarity and readability. Known QDOS/SMS2 structures decoded with defp are written out with headlines and commentaries.

In WINDOW mode, the result of your editing is displayed in the same way as it is saved later in the "_dis" file. If you don't need a listing to work with, you need not save the "_dis" file itself. It is totally sufficient to save the index along with the original code file. This is faster, files are smaller, but the whole information is preserved.

Tips and Tricks

Let us assume that you want to disassemble a not too small program. Let us further assume that you already know that it contains some BASIC procedures, that it is a Pointer Environment program using the thing system and that it is configurable with the standard config level 1. This could be the QPAC2demo, which you can find on the DISA disk... OK, let's go!

After loading the code, DISA generates the index automatically (this is a data field where the attributes of the disassembly are stored. Initially the index is empty, i.e. set to **code** by default from the first byte). Now switch to the **WINDOW** mode by HITing the **EDITOR** item. There you can find the instruction "filetype 0". This means that the code was not a job (filetype 1), but a program for resident load. Is there a ROM-Flag $4AFB0001 at the beginning? Unfortunately no, so the program seems to be intended to be CALLed.

Now switch into the EDITOR-Mode. You can see some readable text strings. Don't follow the temptation to mark these with **text** too fast, as DISA may probably do this for you.
So you should first try to find out whether these text strings belong to predefined QDOS structures. If you edit such a structure with **defp**, all text passages belonging to this structure will be marked automatically. This is faster and additionally the structure itself will be commented by a header line.

Let us start with Things. Therefore you select (HIT) the **SEARCH** item, type the Thing-Flag "THG%" into the **Text** item of the SEARCH menu and let DISA search for it from the start. On line 2BE8 it will be found. But there it is not a thing header but "Immediate Data" of a "move.l" instruction. You may like to hit the **ascii** item to toggle from $54484725 to 'THG%' and v.v.. Go on searching by DOing (selecting with the right mouse button or ENTER) the **SEARCH** item, which actually has changed to **TEXT**.

On line 2C34 again such a flag is found. This is the first Thing header. Mark it with **defp**. Lines 2C34 to 2C4C will be edited with attributes according to the Thing header definition automatically. You can switch to the WINDOW mode to see it. There you also find a pointer to L0012. Go to line 0012. But STOP! It is better to mark line 2C4C (or somewhere nearby) to return and continue editing from here easily. HIT the line and move the pointer to the end of the line, then DO it. Now the marker <==0 is set.

At line 0012 and below you can see the comment "Job Header" and the text "Things V1.01 " already edited. The first Thing Header at line 2C34 is an "executable Thing", Thing Type 1, and the pointer L0012 points to the job.
Now search for the next THG%-flag. You'll find it at line 2E92, and so on... Sometimes it may be necessary to bring the Thing flag to the line start (see Extended Editing). Sometimes you may find THG% flags which are already edited without you. This may happen with Things which form part of a linked list, as DISA scans and edits the whole list at once. Besides, I could not find any other program containing as much Things as QPAC2...

Now you may look for config blocks. Therefore you type "<<QCFX>>", which is the flag for config blocks, into the **Text** item of the **SEARCH** menu. Don't forget to HIT the ↑ ↑ ↑ ↑ item, to make DISA search from the beginning. The first flag will be found on line 1E4E. Mark it with **defp**. As with Things also then a lot of automatic editing will be done. Search again for "<<QCFX>>" and mark it with **defp** until DISA finds no more config flags. QPAC2 has many config blocks...

Next we'll try to find the definition list of SuperBASIC procedures. We already know that Basic commands are linked with the QDOS vector $110 (BP_INIT) and that a1 must point to the list. So we could search for the absolute address **Abs.Adr. $110 => to: $112** from the beginning. Unfortunately DISA is not yet synchronised so it is better (this generally applies to unknown code) to search for the instruction "movea.w $0100,an" as a hex figure. So type "30780110" into the **Figure** item, and then "F1FFFFFF" into the **Mask** item. Let DISA search from the beginning. DISA searches for a while, as the code must be disassembled, and finally finds it at line 31CE. Unfortunately there the address pointer is not loaded into register a1. Therefore we must look where the Label L31CC is jumped from and assume the address is set there. So we type "31cc" into the **Label** item of the **SEARCH** menu and start searching from the beginning again. Again it will take a while as the code must be disassembled again. Finally we find that a jump to subroutine to L31CC on line 2AF8. Directly before this instruction L3156(pc) is loaded to register a1, so this must be the Basic-Defproc-List. Entering 3156 into the **Line** item of the **SEARCH** menu will bring you there directly. It looks fine, so we HIT the line with **defp** selected. Then the list is edited automatically and you can see the result by switching to the **WINDOW** mode (HIT the **EDITOR** item).
The list tells us to have five Basic Procedures, though in fact it only has four (this is due to the BP_INIT arithmetics), "BT_SLEEP", "BT_HOTKEY", "BT_WAKE" and "BT_EXEC" and no functions.

Now it's time to start the **DISA** item. By this the whole code is disassembled and all labels are calculated according to the attributes actually already set. If DISA seems to be locked, HIT the **DISA** item twice again to force DISA to run.

Just for fun you may jump to line 3272 in **WINDOW** mode. There you can see the four Basic Procedures with a comment line giving the name of each procedure. Straight after that the Thing-Headers and parameter definitions can be found.

Now you've already invested a lot of time for editing the code, so it would be a good idea to make a safety copy. Therefore it is sufficient to save the index file, as all the work you've done is stored there. Select the **FILES** item to open the files menu. There you must HIT the **Index** item with the name of the index file. Naturally you can edit the name. If you select it with DO, the FILE-SELECT menu will appear but only if you've loaded the menu extensions menu_rext. It may happen that DISA runs automatically before saving the index to update the pointers.

Next come the most difficult but most important **defp** structures, window definitions (as processed by WM_SETUP). In case a program is developed using menu definitions generated with the EASYPTR development system (Albin Hessler Software) it is quite easy to find them. Just search for the text "menu". The EASYMENU header is described in the EASYPTR manual. Just after the header the window definition starts, and if you HIT there with **defp** selected, the whole definition is edited automatically. See example "Cueshell_Demo". QPAC2 window definitions are not generated using EASYMENU, but with the QPTR assembler macros. Here you may find the start of window definitions in a way similar to that described above for Basic procedures. Look after the Hex **Figure** $4EAA0004. This is the call to WM_SETUP which is window manager vector 4. There register a3 must point to the window definition. Search for the location where this register is set, there you should find the pointer to the start of the window definition. HIT there with **defp** selected, and if you don't get a beep it's OK.

In QPAC2 unfortunately it is a little bit more difficult as there register a3 is not set directly but selected from a list of pointers. This makes very compact and effective code, but is very hard to edit. There is no way other than analysing the code to understand how this works. Only then you have a chance to find all pointers to window definitions.

How? If you want to see it ready, just load the index file "Qpac2demo_idx", where all is already edited.

Now, how to find sprite, blob and pattern definitions. Most of them will form part of a window definition where DISA will find and edit them automatically. Sometimes there are also independent definitions, e.g. those used at run time to change the shape of the mouse pointer, etc... Look for the hex number $01000000 (QL 4 colours) or $01010000 (QL 8 colours). The chance to find a sprite, blob or pattern definition is rather good. Just try HITing with **defp** selected. If you get a beep, you're unlucky but nothing worse happened.

Now go to the code start and **SEARCH** for illegal instructions, selecting the **illg** item for the search. If you find such an illegal instruction, the code still needs some editing. Often you may have found a text, e.g. error messages... If it seems to be a QDOS text string with a leading length word, DO that word with **text** selected. Often another text may follow directly. Therefore the end marker will be set such that it points to the next location after the previous text. If it is also a text you just may DO that line again. If you get a beep it is very likely that there is not a standard text following. So probably normal code restarts there.

An other reason for illegal instructions may be jump tables. Also here there is no help other than analysing the code by hand and deciding whether it is likely to be a **prel** or an **arel** table. If such a piece of code is not of your special interest you may simply mark it **dc.w.** So you may avoid wrong labels if it is not a table but also risk for unknown labels if it is a table.

There is an other possibility for illegal instructions, those belonging to the 68020/68030 instruction set, which DISA actually does not know yet.

Now you may scroll once through the whole code in **WINDOW** mode. Have a special eye for "dead code", which is code apparently unused as not pointed to by a label. Often those pointers are hidden in jump tables not recognised yet. Naturally there may also be really "dead code", often if a program was assembled using libraries.

Also have a special look on the remarks **label not on code boundary** and **label on odd address**, which may appear as comment lines. Normally those labels result from data structures not edited correctly as **text, dc.w** or **dc.b**. To find the reason search for the label to find the origin and analyse the code there. If there really seems to be normal program code, then you may have to deal with a program altering its code at run time. This may also happen from within a configuration routine, so it must not be necessarily an impure program.

Now some additional hints, which can be useful for you when working with DISA:

1. Often you'll not be interested in a complete program, but only a small part, e.g. a single routine which you want to study and understand. In such a case you don't need to save the whole listing in a "_dis" file. You can also save only a part of it. Select the FILES menu and set the first memory address which initially is set to 000000 (i.e. code start) to the line you want to start with and the second number which initially is set to the code length (i.e. pointing to the end of the code) to the end line you want. Make sure that the third number which gives the base address of your code in memory is not changed. If you select SAVE now, only the listing within the given boundaries is saved to file. Before you should have deselected the **H/V** item in the **STATUS** menu.This is to avoid long list of H- and V-labels which you don't need in this case. To restore the old (complete) boundaries, reset the first number to 0 and the second number (naturally you've forgotten the old value) to a very large amount (e.g. 3FFFF), it is automatically restricted to the initially loaded length of the file.

2. Sometimes you may be interested in a certain group of assembler instructions. Therefore the mask in the SEARCH menu is intended to be of some help. This works like this: when searching a hex number, DISA ANDs the mask with a number taken from the code and compares it with the hex number to search for. Assume you want to find all calls to window manager vectors in the QPAC2demo. These calls are in the form "jsr $00xx(a2)" or "jmp $00xx(a2)". In hex this is "4EEA00xx" or "4EAA00xx". The wman-vectors are in the range $0004 ... $007F. Enter "4EAA0000" as the hex number to search for and "FFAFFF80" as mask. Then you will find all wman calls subsequently. As you're an assembler programmer you certainly will soon find some other useful examples...

3. Sometimes DISA may stutter when scrolling in WINDOW-mode. Why? If you load a very large code file into DISA, e.g. $C000 bytes (the Minerva operating system seems to be very popular), and then jump to a location far at the end without editing, then anytime DISA is forced to scroll back or forth it must look for the next attribute.This may take a certain time if the next attribute is very far away. To avoid this you can simply set an attribute somewhere before the code you're actually interested in. Then scrolling in this area will happen much faster.

The Pointer Environment

The pointer environment is a system extension to QDOS. Since it appeared first with QRAM it has been further developed and enhanced. Therefore you should only use a recent version.
The Pointer Environment consists of three parts:

Pointer Interface	file 'ptr_gen'
Window Manager	file 'wman'
Hotkey System	file 'hot_rext'

All three parts must be loaded residently after the computer is switched on in the following order. If Toolkit 2 and Lighting are used, they must be initialised before:

```
100 _lngINIT
110 TK2_EXT
120 LRESPR 'flp1_ptr_gen'
130 LRESPR 'flp1_wman'
140 LRESPR 'flp1_hot_rext'
```

The Pointer Interface

The Pointer Interface is a system extension to QDOS. It installs the pointer device which makes it possible to move a pointer on the screen. Generally this is a small item (sprite) of max. 64 x 48 pixels. This sprite may be moved with a mouse or the cursor keys as well. Thus for instance

- a menu for program control may be managed
- a pencil in a paint program may be moved
- objects may be marked and moved around etc. ...

Although mainly intended for the use with a mouse, the pointer interface allows all control to be done with the keyboard alone. But a mouse gives more flexibility and comfort. For the standard QL the QIMI Mouse Interface as well as our SERMouse serial mouse driver is available. The mouse may even simulate the cursor keys when moved while the left button is pressed. Thus e.g. it may be used in any Editor (ED, QUILL etc.).

The Pointer Interface includes two SuperBASIC extensions CKEYON and CKEYOFF:

CKEYON

The mouse pointer can be moved with the cursor keys.

CKEYOFF

The mouse pointer can not be moved with the cursor keys.

Multitasking with the pointer interface

QDOS allows several programs to run at the same time (multi-tasking). But standard QDOS has no option to save the window content of a program (destructible windows). Each program must have a separate option to restore its windows. CTRL C is used to switch around the keyboard queues, but the window content is not restored and the program on top destroys the windows of other programs.
The pointer interface - by help of an extended channel definition - saves and restores the window content automatically when the programs are switched around. This extended channel definition is the reason why software which does not use the system routines correctly (e.g. by manipulating the channel definition blocks directly) does not run under the pointer environment. The Hotkey System and Cueshell offer several ways to adapt badly behaving software (especially the PSION suite). But all serious software suppliers now offer program versions that run with the pointer environment.
Programs (jobs) are switched around with CTRL C as with standard QDOS, but now the pointer interface switches around the pile of primary windows that belong to every job and no longer the simple keyboard queue.

The Pointer Interface distinguishes between two different types of windows of a job.
1. The primary window is the first window channel opened for this job.
2. All subsequently opened window channels are called secondary windows.
The channel definition does not only keep the window size but also the window outline which is equal or beyond the window size area. Pointer requests are restricted to the outline area while all standard input/output is restricted to the window size area.
The primary window outline may cover the whole screen, while a secondary outline is restricted to be within its primary outline.

To keep the new window definition compatible with standard QDOS, window channels are managed by the Pointer Interface in two different ways.
With unmanaged windows (in general windows of those programs which are not written for the Pointer Interface) it watches automatically that the primaries outline always covers all window channel areas opened for this program.
When a primary window is marked to be managed (also called well behaved) by setting the outline explicitly, then the program itself has to make sure that all its windows are within its primary windows outline. Pointer requests are only possible on managed windows. All managed windows of a program (job) form a hierarchical structure. The Pointer Interface allows pointer requests only to the latest window declared to be managed.

Programmers access

In the pointer interface there are some system routines implemented as

TRAP #3 $6C-$7F

to which the assembler programmer has access. The technical description is in the QPTR manual.
Our EASYPTR development system comes with a set of SuperBASIC extensions which give access to the system routines of the pointer interface for the Super-BASIC programmer. For the C-programmer a set of library routines is available.

The Window Manager

The Window Manager is a set of vectored routines to manage the internal structure of a menu-window. As these can be used by several programs they are loaded as a system extension together with the pointer interface.

Structure elements

The internal structure of a menu window is built by several elements:

Main window

This is the total area within the outline. This area is defined by size, border, shadow and pointer sprite.

Information sub-windows

These can be situated at any place within the main window. They can be used for general information purposes, division of the window, colour design, ...
Info-windows are ignored by the pointer request.

Loose menu items

These are special areas for program control. When the pointer is within a loose item, a border is drawn around the item. Any loose menu item is directly coupled with an action routine, which is processed automatically when the item is selected (->HITs and DOs).

Application sub-windows

For free use by the application program. Application sub-windows are, like loose items, subject of the pointer request, i.e. any application sub-window has a hit routine, where the application program can decide for its own purposes what to do, if the pointer is within the window.
Application sub-windows may have a regular menu structure also processed by the window manager. This menu may be larger than the window size. Hidden parts are scrolled or panned into the visible area by help of pan and/or scroll bars and arrows. The menu may consist of several independent sections and/or be able to be split in several sections (and joined again).

Control elements/Events

The window manager offers standardised elements to control basic functions of a menu.

key	event
SPACE	select
HIT (left mouse key)	select
ENTER	do action
DO (right mouse key)	do action
ESC	cancel
F1	help
CTRL F1	sleep
CTRL F2	wake
CTRL F3	change window size
CTRL F4	move window

Programmers access
The technical description of the window manager routines can be found in the QPTR manual.
Our EASYPTR development system offers an easy way to design menu-windows and structures on the screen and to control them from within SuperBASIC as well as C- and assembler written programs.

The Hotkey System

The Hotkey System is intended to connect various actions (starting jobs, inserting strings, picking jobs,...) to a single key press (together with the ALT key). Some key presses do have a standard meaning (although configurable):

ALT ENTER	recovers the last string stuffed into keyboard queue
ALT SPACE	recovers the last string stuffed into the hotkey buffer
ALT SHIFT SPACE	recovers the previous string stuffed into the hotkey buffer

The hotkey buffer can be used by any program, e.g. a filename can be written into the buffer by one program and inserted into the keyboard queue of an other program.

Beyond these, up to totally 128 Hotkeys can be defined.

A Hotkey may lead to one of the following actions:

- a string is inserted into the current keyboard queue as if it where typed on the keyboard directly
- a string is stuffed into the hotkey buffer
- an executable program is started
- an existing job is picked to the top
- a sleeping program is wakened
- SuperBASIC is picked on top and a command sequence inserted into the command line
- an executable program is loaded and marked to be an executable Thing for later use
- an executable Thing is started.

SuperBASIC extensions in the Hotkey system

Some SuperBASIC extensions to use and define Hotkeys are part of the system. Most of them are functions and are usually used in a *boot* program. If necessary they can be used directly from the command line.

A description of the commands follows. The syntax description follows the usual conventions. Alternative parameters are put between curly brackets, optional parameters between square brackets.

Extra parameters:

These parameters can be set where **[,extra[,...]]** is to be found in the description. Their purpose is to adapt difficult programs.

P ask for memory size

P,kByte the program is given the memory size given in kByte

The P option is mainly intended to be used with the PSION suite programs, which otherwise would grab all available memory.
Example: EXEP 'flp1_quill',P,64

I the program is marked to be impure (not romable).

Example: ERT HOT_RES ('c','flp2_c1mon',I)

U the primary window of the specified program is marked to be unlockable, that means it will destroy other windows as in standard QDOS.

Example ERT HOT_LOAD1 ('s','flp2_mysuperwatch',U)

G opens a guardian window (primary window) for this program, which covers the whole screen.

G,width,height,x-origin,y-origin
opens a guardian window (primary window) in the given size and position.

The G parameter is mainly intended for programs which do not use the standard screen input/output routines, e.g. if the program writes directly into the screen buffer.

An additional command string parameter, as with EX in Toolkit 2, can be given with the functions HOT_RES, HOT_CHP, HOT_LOAD and HOT_THING.
Example ERT HOT_RES ('l','flp1_linker';'-with flp2_link')

Hotkey System SuperBASIC commands

ERT

ERT

Automatic error report procedure to be used with functions. If the function return value is set to the error code, ERT will redirect it to the command channel.

EXEP

EXEP **{filename}{thing [, extra[,...]]**

The job or thing is started.

HOT_CHP

err = HOT_CHP **(key$, filename [, extra[, ...]])**

The file is loaded into the common heap area. A new copy of the program is started every time when ALT+ key$ is pressed.
Example: ERT HOT_CHP('q', 'flp2_QD')
Variant:
err = HOT_CHP1 **(key$, filename [, extra[, ...]])**
Only one copy of the program can be started. (For not romable programs.)

HOT_CMD

err = HOT_CMD **(key$, commandline$)**

Picks SuperBASIC and stuffs commandline$ into the SuperBASIC command channel #0.
Example: ERT HOT_CMD ('F','print free_mem','')

HOT_DO

HOT_DO **{key$}{name}**

Executes the action defined for this Hotkey (key$) or Thing (name) linked to this Hotkey, as if it was typed on the keyboard.

HOT_GO

HOT_GO

Starts the hotkey job. Then all defined hotkeys are active.

HOT_KEY

err = HOT_KEY (key$, string$[, string$[, ...]])

The string(s) is(are) stuffed into the keyboard queue when the Hotkey is pressed.
An empty string will cause a line feed
Example: ERT HOT_KEY ('Y', 'Yours sincerely','')

HOT_LIST

HOT_LIST [#channel]

Gives a list of all actually defined Hotkeys on the given channel (default: #1).

HOT_LOAD

err = HOT_LOAD (key$, filename [, extra[, ...]])

A new copy of the program is loaded and started.Then, every time when ALT+ key$ is pressed a new copy is started with the same code.
Example: ERT HOT_LOAD ('s', 'flp2_easysprite_exe')
Variant:
err = HOT_LOAD1 (key$, filename [, extra[, ...]])
Only one copy of the program can be started.

HOT_NAME$

name$ = HOT_NAME$ (key$)

Returns the name of a hotkey.

HOT_OFF

err = HOT_OFF ({ key$ }{ name })

The hotkey is switched off (not removed).

HOT_PICK

err = HOT_PICK **(key$, jobname)**

The job is picked to the top.

HOT_REMV

err = HOT_REMV **({ key $}{ name })**

Removes a hotkey.

HOT_RES

err = HOT_RES **(key$, filename[, extra[, ...]])**

The file is loaded into the resident procedure area. A new copy of the program is started every time when ALT+ key$ is pressed.
Example: ERT HOT_RES ('s', 'flp2_easysprite_exe')
Variant:
err = HOT_RES1 **(key$, filename[, extra[,...]])**
Only one copy of the program can be started. (For not romable programs .)

HOT_SET

err = HOT_SET **([newkey$,]{key$}{name}**

Reset a hotkey previously switched off with HOT_OFF. Additionally a new key may be defined.

HOT_STOP

HOT_STOP

Removes the hotkey job. Then all defined hotkeys are inactive. The hotkey job can be removed via the Jobs list in Cueshell or RJOB of TK2 too.

HOT_STUFF

HOT_STUFF **string$**

Stuff the given string into the hotkey buffer.

HOT_THING

err = HOT_THING (key$, thingname)

An executable thing is started. Things can be part of files which are loaded residently (e.g. QPAC1/QPAC2).

HOT_TYPE

err = HOT_TYPE (key$)

Returns the type of a hotkey.

-8	last line recall
-6	stuff keyboard queue with previous stuffer string
-4	stuff keyboard queue with current stuffer string
-2	stuff keyboard queue with defined string
0	pick SuperBASIC and stuff command
2	do code
4/5	execute Thing
6	execute file
8	pick job
10/11	wake or execute Thing
12	wake or execute file
-7	Hotkey not defined

HOT_WAKE

err = HOT_WAKE (key$, jobname)

The job is wakened. If the job does not exist, the function will try to start a Thing with the given name.

The Thing System

Things can be various type of things, e.g.

- a job waiting to be executed from a Hotkey (an executable thing)
- a printer driver which can be used by all jobs
- a piece of code or data for a certain program.

The Hotkey system is implemented as a Thing and is bound together with the Thing system extension in the hot_rext file which is an integral part of the Pointer Environment.
The main purpose of the Thing system is to provide a standardised definition format for various things and to make them accessible through the system variables.
Any Thing is identified by its name and thus can be accessed easily from any program.
The Thing system controls installation, usage and proper termination of Things.
Cueshell would not run without the Hotkey Thing.

Menu Extensions

The menu extensions 'menu_rext' come with DISA by license of JOCHEN MERZ SOFTWARE.

The FILE SELECT window

The FILE SELECT window is always shown when the user is required to enter or select a filename. Here you can enter the filename either directly or edit a suggested one by selecting the menu option directly beneath the request. Beneath this are two menu options with which you can recall the contents of the HOTKEY buffer and all previous contents. Just select the menu option and the contents will be written to the "suggested" area. Confirm with OK and the input will be accepted by the system. You can also edit the name, of course.

The rest of the window concerns the current drive. Depending on the size of the window, one or two sub-windows are shown. If you only see one sub-window, this will contain the filenames and sub-directories. If you see two windows, the right hand (larger) one will only show the filenames, the smaller one on the left only the subdirectories. In these windows all the files are sorted alphabetically. The files will be taken from the current drive and must all have the correct ending (if any). The endings for sub-directories are ignored, because subdirectories don't really have endings. Now you can edit the drive and/or the ending.

If you press ENTER at the directory menu option, a further window is overlaid, from which you can select predefined devices and sub-directories. If you just select a directory, the list of files will be updated. You can also "update" it by selecting the lightning symbol or by WAKEing up the job to which the window belongs, should this be hidden.

If you press ENTER at the endings menu option, this will be deleted if it wasn't already empty. This is simpler and quicker than having to select it and then deleting the four characters. If it was already empty, then a window overlay will show some suggested endings.

Above the current directory is a list of available devices, e.g. MDV or FLP. There are also drive numbers from 1 to 4 listed. To select MDV1_, press M and 1, and the directory window displays MDV1_.

Behind the directory name you will see an arrow "<-". By selecting this option you can retrace a step back along the subdirectory tree without having to edit anything. So if you're in directory flp1_paul_texts and select this once, then you get to flp1_paul_. The next time you select it, you get flp1_. If the current drive has true subdirectories (e.g. Miracle's Harddisk or the QL Emulator drivers) then you'll find the subdirectories of the file names marked with a "->" arrow. As already mentioned, subdirectories are always listed, the endings don't have to correspond. If you select such a subdirectory, then you'll "get in", i.e. the name will be taken over for the directory and the file list read in again.

But to get back to the list of files: You can select any file you like. SPACE accepts the name as "suggested". ENTER takes the name and carries out an OK automatically. If the window is too small to show all the suitable files, the normal scroll arrow will appear in order to scroll the next batch of names up the screen. You can also select files or directories by pressing the character which is in front of the name.

At the right you will see a scrolling bar. Move to this area, press SPACE and the area will be shown relative to the total area. Press ENTER and the window will split, enabling you to control the two parts independently from each other. Move to the split and press ENTER to join the window together again.

You can also preselect the eight different subdirectories suggestions in the Directory Select menu. Make the necessary changes to the MENU_rext file using CONFIG.

The DIRECTORY SELECT window

This window allows the selection of different drives resp. subdirectories. The user can preset the eight most used subdirectories. The most popular drive systems (MDV, RAM, FLP and WIN) are selectable immediately. Just move to a menu option and press SPACE or ENTER to select. Use the CONFIG program to preset the subdirectories on the MENU_rext file.

The EXTENSION SELECT window

This window serves to select the most usual file endings. Move to a menu option and press SPACE or ENTER to select. You can also use the number keys 0 to 9 to select the menu options. All ten endings may be set by the user using CONFIG.

The ITEM SELECT window

When you see a window with one to three menu option, you can make your selection by pressing SPACE or ENTER. You could also press the first letter of the option.

The REPORT ERROR window

The only thing you can do here is press OK to show that you've taken notice of the error reported.

The READ STRING window

You are asked to enter a string or filename. Under certain circumstances you may be offered a suggested name. You can either press ENTER to accept the suggestion, edit it as usual using the cursor keys or just enter a new string.

The VIEW FILE window

This window enables you to view a file. You can scroll one line by pressing SPACE or the left mouse button while the pointer is over the view-window. You can scroll one page by pressing ENTER or the right mouse button. The lightning-symbol lets you start again viewing the file from the beginning. Selecting WRAP causes any line, which exceeds the permitted width, to be continued on the next line.

The BUTTON

If a program is in Button mode, then you can wake this up by moving to the button area and pressing ENTER. If the button is not positioned inside the button frame, you can move the button by using SPACE.