QBASIC

This manual describes the BASIC-Interface between QD and QLiberator. In order to be able to use it, you require the editor QD (Version 5 or Version 6) and the BASIC compiler QLiberator (preferably Version 3.35 or more recent). It is throughout this manual assumed that you already read the manuals of QD and QLiberator - it still describes the various steps in setting it up and using it in detail. A reference to the other manuals is made wherever required.

The principle: QBASIC is a resident extension, which has to be loaded in the way resident extension always have to be loaded: using the LRESPR command. Before you do this, you have to configure this extension to make it fit into your system setup. After you LRESPRed **QBASIC_rext**, you will find that a new Thing for QD has been installed. It's called QBASIC, of course. See also the QD manual for extension things. When a BASIC program is typed into QD (or loaded, of course), then this Thing can be called by pressing F10, which then presents a menu which allows you to call the parser. The parser checks the program for syntax errors. In case it is sytax-error-free, the parser generates a file which can then be compiled by QLiberator (similar to the work files created by QSAVE and the LIBERATE command). The program is then compiled and executed, if the compilation is successfull. QD is immediately available again - you can edit the source program while the compiled code is running - you can also execute multiple programs from the source-text.

How to do it: first of all the file **QBASIC_rext** has to be configured. Execute the program **CONFIG** or **MenuConfig** and load in the file **QBASIC_rext**. You will find thre configuration items which all have to do with QLiberator:
QLiberator thingname: QLIB. If you have QLiberator installed as a Thing (or you will install it as a Thing), then enter here the name the Thing has got. There are various possibilities how QLiberator could be loaded as a Thing - if you don't know any of these then you probably do not have the possibility (i.e. THING extension) to do it. It is not a disaster, as QBASIC's search for a Thing is very quick, and it then searches for QLiberator in a file - you will not notice the Thing search.
QLiberator filename: flp1_QLIB_obj. This is the full filename of the QLiberator file in your system. If you use the QLiberator master disk, then it is flp1_QLIB_obj. If you have QLiberator somewhere on your harddisk, then specify the full filename on the harddisk.
QLiberator options: empty. Here you can define the various compiler options which can be passed to QLiberator for compilation (for example, -NOWINDS). Have a look at the details in the QLiberator manual for the various compiler options.

That's it - not a lot to setup. The so configured file can be loaded now - remember, it's a resident extension.

Starting QD with BASIC facilities if you want to start a QD with the BASIC interface, then you have to tell this QD right at the beginning. This is very simple: just pass a key with the parameter string (a detailed description of various options is given in the QD manual). Examples:

```
EXEP QD;"\T QBASIC"                    if QD has been loaded resident before or
EX flp1_QDM"\T QBASIC"          starts QD from floppy disk.
```

Both starts QD and uses the QBASIC Thing. If you'd like to have a full BASIC environment, for example the HyperHelp BASIC System in QD, then use

```
EXEP QD;"\TQBASIC \Hflp1_basic_help_ \E_bas \Dflp1_basic_"
```

The \T defines the Thing QBASIC, \H defines the directory for the help information (if you have it on harddisk, specify the full subdirectory), \E defines the default extension _BAS ('cause you want to load and save BASIC programs, probably), and \D defines the default directory for those files, e.g. flp1_BASIC_.

Whatever you did to execute QD, let's assume you have now a QD running and you see the QBASIC text in the menu item next to F10. If not, then you've definitely done something wrong.

Within QD, you can load, save, edit or enter BASIC program. As a special feature, you can create line-numberless program! Of course, GOTO will not work anymore if there are no line numbers.

If you want to compile the BASIC program which is currently held in QD, then press **F10**. If nothing happens, then the QBASIC thing is not present or not properly installed (can't think of a way of not properly installing it). But, be positive and let's assume it works: you will see a menu which allows you to choose between the following options: change the pre-defined compiler-options (e.g. to set -NOWIND option - see QLiberator manual) or if you just want to compile the program or both compile and execute on success.

Let's have a go:

```
FOR x=32 TO 191
 PRINT x,CHR$(x)
END FOR x
INPUT wait$
```

Press **F10 Execute** now and off we go. A QBASIC window appears at the top left corner of the screen (it is under QLiberator, and QLiberator always appears there, even on higher screen resolutions). After a very short moment, QLiberator appears and compiles the program. As it is very unlikely that it finds an error in the program listed above unless you made a typing mistake, it will also execute the program. The program prints the characters and waits for ENTER or RETURN to be pressed. Do it and the program disappears.

Let's deliberately generate an error: add the following line to the program:

```
q==1
```

and move the cursor somewhere on the screen (but leave it within the QD window) to see the effect, then press **F10 Execute**. The cursor will jump immediately over the second '=', as the parser thinks there is a syntax error (and is perfectly right, isn't it?).

Or try the following:

```
PRINT HEX$(12345,16
```

and try again **F10 Execute**. The cursor is put at the end of the line, as the parser is missing a closing bracket, and assumes it to be missing after the last parameter. Of course, it could be missing after the first parameter, but it is more unlikely. But, as we know, the HEX$ function requires two parameters, and the parser has guessed right. Another example:

```
PRINT LEN('123':PRINT LEN('4567')
```

The cursor is put after the string '123', directly over the colon. The parser knows that something is missing here (close bracket, of course), and the new statement is treated only if the previous syntactic expression is okay. You see, the parser is quite good in guessing where mistakes are.

Things which the parser does not like

Lines which are longer than the maximum line length to which QD is configured (i.e. lines which start with the → character). That's not accepted! If you require longer lines (default setting is 160), then re-configure QD.

Let's carry on with the explanation of QBASIC: please execute the first sample (error-free please) program which prints out the characters. Do not press ENTER. Have a look at the job list. You will find a job called "RAM1_". As you have not named the program which you compiled and executed, it just gets the name where the source file is created: RAM1_. Naming it is very easy: save the program before you compile it (you should do this anyway, as it could crash and then everything you have entered is gone!). Let's call the file **RAM1_TEST_BAS**. Recompile and guess how the job is named - just "TEST"! _BAS is only the file-extension which helps you recognising the file in a big file list, and RAM1_ is the drive, which has nothing to do with a job name either. You see, QBASIC tries to give jobs a sensible name if possible.

Now have a look at the directory of RAM1_. You will see three files (or maybe more):

TEST_BAS	you just saved it!
TEST_OBJ	the executable program generated by QLiberator.
TEST_ERR	contains usually warnings and errors (if there are any, but the program was okay.

The **OBJ**-file is executable, you can execute it as often as you like (use EX, QX, QPAC 2's files menu or Cueshell, for example) without recompiling it via QD every time.
The **ERR**-file contains no errors, otherwise QBASIC would have shown you the error file automatically. Easy to force an error: add the following line to the progam and try to compile and execute it:

```
fred 500
```

with the assumption, that you have not defined a procedure called FRED somewhere in your system, the compiler reports "ambigous name". Before doing this, it terminates compilation and wait for the SPACE key to be pressed. Rather superflous, but this can be patched. You will find a program which patches QLIB_obj to suppress to pause for SPACE - more later. Okay, press SPACE now and a window appears which shows you the error. You can scroll through the error file to see all the errors, if there are more. This time, there's not a lot to scroll through. You should not try to execute a program which did not compile without errors - this could lead to a crash.

On to the warnings: write a short program like:

```
a=3
OPEN #a,con
CLS #a
INPUT #a,a$
```

and try to execute this. QLiberator will wait for the SPACE keypress unless you patched it already 'cause it is so boring, then QBASIC opens a selector box with three choices: back to the Editor brings you back into QD, so that you can have a look at possible problems. Try execution tries to execute, and if it fails, then it fails. The compiler should trap run-time-errors, which could be generated by the warnings (which is explained very detailed in the QLiberator manual). You can also view the warning files - exactly like in an error case the _ERR-file is displayed.

Patch QLiberator

You will find a file called **PATCHQLIB_bas** on the QBASIC disk. This automatically patches QLIB_obj to remove the annoying SPACE request in error case. It also changes the colours so that it better fits in a green/white environment, which seems to be the most common. Just LRUN the program, but please change the working copy and NOT the master disk. Patch only version 3.35 (don't get confused, V3.36 contains a QLIB_obj V3.35, only the runtimes are 3.36).