Version 0.04

USERS GUIDE

INDEX

APPENDICES

1. INTRODUCTION

Welcome to the world of the Basic Asm Operating System, the new way of using your graphical calculator.
BAOS will let you use your calculator as a mini-computer, comparable to the computer-systems of a few decades ago.

However, currently BAOS is still in a (pre-) alpha state. This means that only some of the functionality of the Operating System is implemented and can be used. On the other hand, there will be updates on a regular base, and functionality will quickly extend.

This manual will cover everything you need to know about this Operating System, and you should not need anything else to work with it. But if you want more information, you can visit the official BAOS website, which is (as part of the BZC project) located at:

http://bzc.sf.net/

under the “BAOS” section.

You can find here:

- Always the latest news and updates on BAOS.
- The current phase of development (what kind functionality is implemented, and what will come)
- Developer information: How do different parts of BAOS work (for instance multitasking, or what kind of layout does the file-system use) and a complete reference of all internal (OS-only) and external (“user-space”, used by programs) functions and routines of BAOS.
- Always the latest release of BAOS (TiCalc.org for example, has sometimes release files pending a long time).

2. INSTALLING BAOS

2.1: Needed hard- and software.

To install and run BAOS you need at least:

- A Personal Computer or other kind of computer that can run Unix or Windows.
- A program to extract zip-files.
- On Unix Systems, you also need some basic developer tools (gcc, libc and a gnu-compatible make)

For running BAOS on a real calculator:

- A TI-83+ (SE) or TI-84+ (SE) graphical calculator. Note that only a TI-83+ is tested, and that the USB port on the TI-84+ (SE) calculators is known as trouble-maker.
- A “linking-cable”.
- Linking Software, preferable TILP or TI-Graph Link.
  TI-Connect is known to give problems when sending Operating Systems to “clean” calculators (that don't have an OS installed. This can happen when sending an OS fails because of a transmission error). TI Graph Link and TI-Connect can only be used on Windows computers and with TI-made linking cables.

For running BAOS inside an emulator:

- WabbitEmu, the only emulator known that is capable of running BAOS, you can download a beta release from http://www.revsoft.org/wabbitemu.zip
- For unix systems: Wine, which you can (if it doesn't come with your system) obtain from http://www.winehq.com/

2.2: Extracting and Compiling / Assembling

After downloading the latest release of BAOS from http://bzc.sf.net/ -> BAOS -> Releases, you should extract the .ZIP file to a temporarily folder or directory. Now you need to assemble BAOS:

On Unix Systems

On the command line, enter the temporarily directory and run a simple:

```
$ make
```

This command will first build some helper tools (assembler, squasher, etc) and after this is done, BAOS is being assembled and, when make is done, you should have 3 files in the current directory:

```
baos.8xu
baos.rom
baos.sym
```

On MS Windows Systems

Enter the temporarily folder with Windows Explorer and double-click on the make.bat file (if you don't see a make.bat file, double-click on the file with the name “make” and with a window and a gear as icon.

A window should pop-up, and BAOS is being assembled. At the end you are being asked to press a key, after checking there is no error message, you can press a key to close the window.

2.3: Installing BAOS.

On a real calculator

Connect the calculator to your computer and start your linking program. Chose inside you linking program to send over an OS and browse to your temporarily folder / directory. Inside this folder select a file named “baos” or “baos.8xu”. When using TI Graph Link, you should use the following menu: Connect -> Send Flash Software -> Operating System.

Now comes the difficult part: before sending over the OS, open the battery pocket of your calculator. Stand-by to remove any of the 4 batteries, and start the transfer.

Don't look at the LCD-screen of your calculator, but take a look at the screen of your computer. When the transfer is at 70%, immediately take out one battery of your calculator. After doing this, disconnect the link-cable and exit the linking program.

Now put the battery back into your calculator, and close the pocket. After doing this, you can press the ON-key to turn on your calculator. You should be inside BAOS. If you see a message like this:

```
Waiting...
Please install calculator software now.
```

or

```
Error!
Press any key to turn unit off, then turn back on.
```

you failed to remove a battery in time, just get yourself at the first of the above 2 messages, and try to send the OS again using the following steps:

- Get yourself to the first message described above (“*Waiting... etc”*), by pressing a key to turn the calculator off, followed by pressing the ON-key.
- Start your linking program (I.e.: TiLP, TI-GRAPH LINK, TI-CONNECT)
- Start the transfer (by drag-and-drop the OS-file or select “Send Operating System from a menu, depending on the linking software you use).
- When you see a dialog with information, ignore the information but do not start the transfer yet.

See next page for further steps...

Now follow the next step carefully although they seem rather ridicules:

- Press the ON-key, you will see the Error message described at the previous page.
- Press ENTER, the calculator turns off.
- Press the ON-key the get the “Waiting...” message again.
- Repeat this ON-ENTER-ON sequence.
- Press the ON-key, you see the “Waiting...” message again.
- Start the transfer in your linking software.

Now, the transfer will start successfully and, when there is no transmission error, succeed.

To an emulator

As said above, it seems only WabbitEmu is able to run BAOS (TiLEm doesn't run it anyway). Just start WabbitEmu, and it will ask you for a rom file. Browse to the temporarily directory / folder, and select “baos” or “baos.rom”. That's all!

3. USING BAOS

3.1: Using the keypad.

Before you start using BAOS, you should know how the keypad is handled inside BAOS, which differs from how it is handled in TIOS. Basic rules for using the keypad are these:

- The Arrows have the normal arrow-function.
- The blue buttons under the LCD screen (“function keys”, Y= to GRAPH), have a special function, like the F1 to F12 on a PC.
- MODE, STAT and X,T,ø,*n* currently don't serve any purpose.
- DEL, CLEAR and ENTER are used like DELETE, BACKSPACE and ENTER on a PC.
- All other keys (except 2ND and ALPHA) are used as the green symbol above them (in lower-case).
- When you press 2ND together with one of these keys, you get the same, but in upper-case.
- When you press ALPHA together with one of these, you get the symbol, you should normally get in TIOS (numbers, mathematical symbols, etc)
- Instead of theta (ø), you get an @ ('at').

3.2: Starting and Shutting down BAOS.

Starting BAOS

To start BAOS, you simple press the ON-key.
Immediately after doing this, you will see the OS error-checking the filesystem, which will take a few seconds (depending of the amount of errors on the filesystem).
When there are too many errors on the filesystem (or there is no filesystem at all), the OS will automatically format the filesystem and copy a basic system to it. While formatting the filesystem, the following message will appear in the screen:

```
Formatting FS...
```

If a base-system is detected, the try to load and run the `/bin/init` program. This program will check for a file called `/inittab` with it's configuration stored in it. If `init` could not find this file, it will print an error message and halt the system (see also *forcing a clean system*). Under normal circumstances, you will see the following message, indicating that `init` is setting up your system.

```
INIT: Booting up
```

Depending on your configuration, you will probably see the following message immediately after it:

```
BAOS Shell v0.1
$ _
```

The dollar sign ('$') indicates that the system is ready. You can now enter a programs name to start executing it. See also *The command line*.

Forcing a clean system.

There are a few situations in which booting can fail because of the filesystem. In this case the system is probably in an endless loop of booting – rebooting – booting, etc.

In this case, it is possible to force a format of the filesystem:

- Pull out a battery.
- Press and hold the CLEAR button.
- Re-insert the battery.
- In case nothing happens, press ON.
- You will see the following message:

```
Wiping
Filesystem at
user request...
```

And a few seconds later:

```
Done. Press any
key to reboot...
```

Just press a key, and BAOS will boot up.

Shutting down BAOS

There are two ways of shutting down BAOS. You can choose to do it directly, using a special key-sequence, or start the `halt` program from the command line.

When BAOS is running, you can shut it down, using one of these 2 sequences (which one you prefer):

- Press and hold the ON-key.
- Press 2ND once
- Release the ON-key

- Press and hold the 2ND-key.
- Press ON once
- Release the 2nd-key

NOTE: *it seems that the reaction of the OS take some time while powering off. If it doesn't work or BAOS restarts immediately, try pressing and releasing the buttons in a slower sequence.*

The alternative is to start the `halt` program. When you are at the command line (indicated by the dollar sign, '$'), simply type *halt* and press ENTER:

```
$ halt_
```

Using the command line, it is also possible to put BAOS into stand-by mode, by running `sleep` instead of `halt`. It seems to do exactly the same thing, but when you press the ON key, you will find the system in exactly the same state as before you executed `sleep`. You do not even have to wait for the filesystem check!
You can also run `reboot`, which will restart BAOS.

3.3: The command line.

As said before, you will find yourself at the command line after system start-up. From the command line, you are able to do everything BAOS can. Running a program is quite easy, just enter it's name and press ENTER. The program is started. When you're done and exit the program, you will see the typical dollar sign ('$'), waiting for your input. The `halt` and `sleep` commands from the previous paragraph are examples of running a program from the command line.

3.4: The BAOS FileSystem

BAOS features a directory-based filesystem to store programs and data. Everything you need to store for a long(er) time can be written to this filesystem in the same way you might do this on MS Windows or Unix-based Operating Systems.

Directory-based

When you take a closer look at the BAOS FileSystem (BAOSFS), you might find out that it differs from the way TIOS stores it's data. Instead of putting everything into one place, using different types of files (or '*variables*' as they are called by TIOS), you can organize things in BOASFS. This happens in the same way as you are used on modern computers, using directories (also known as '*folders*').

You can see directories as if they where boxes. You start with a single big one (the *root*-directory). Inside this big one, you can store objects (files), but also other, smaller, boxes (*sub*-directories). Inside these you can store more objects or, again, more boxes.
You can as much directories as you like, until the filesystem is full. Because each one occupies only 64 bytes, you can create almost an infinitive number of directories.

The second thing you might notice, is the fact that there are no different types of data. BAOS makes a difference between the following file-types:

- **Directories:** These can contain data (i.e. files and other directories).
- **Data files:** You can store data in these.
- **Executable files:** These are also known as *programs*, you can ask the system to load them into the RAM, and start executing them. In fact you do this when you type a commando (like `halt`).

However, to make things easier, you can give files of a special type a special name, or using an *extension*. The best example of this if the common used `.txt` extension for text-files, which indicate that there is plain text stored in the file. The name can be for example: `a_text.txt`.

NOTE: Users of MS Windows will notice a slash ('/') used to separate directories (commonly used within Unix and on the Internet), instead of the backslash ('\') which is normal in MS Windows.

NOTE: Because of the way the filesystem drivers work at this time, there is not yet any notification when the system is cleaning the flash. This means there can be hough timeouts when writing/saving files. Depending on the way the driver is called, parts of the system may keep running in this timeout (for example the cursor keeps blinking, screen-switching is still possible, other programs keep running, etc.). If this kind of a timeout appears, just wait a minute or so (remember "Garbage Collecting" can also take a very long time sometimes) before shutting down the system or taking out the batteries.

Command line tools

The use and administrate the filesystem, there are several command line tools installed by the system. Currently this is the only way to copy, delete and create files and directories. In the future, a Graphical User Interface (GUI) might be released with the same functionality.

While working with the filesystem, you can *enter* and *leave* directories. All commands have effect on files in you're current *working-directory* (the directory you entered the last time), unless you specify something else. See the example at the end of this paragraph.

The following tools are installed by default:

ls

`ls` is probably the easiest command to understand. It simply prints out the name of all the files in your current working directory.

cd

`cd` is used to enter a specific directory. Directories can be relative (seen from the current working directory, using `..` for the *parent*-directory (the directory where you're current working directory is stored in), for example: `../some/dirs`, or absolute (a full path from the root directory, for example: `/a/path/to/a/dir`.
Under normal circumstances, you might wish to go one step at the time, using often `..` or a single *sub*-directory.

mkdir

`mkdir` works just like cd, only instead of entering a directory, you create one.

cp

`cp` is something completely different. With this tool, you can copy a file from one place to another. These places do not have to be different directories, you can also copy a file to a different name (because you want to make a backup, for example from `text.txt` to `backup.txt`), and stay in the same directory. Just like all other tools, you can use both relative as absolute paths.
`cp` currently does not support copying full directories.

rm

You should be careful with this tool. `rm` can delete files, and even full directories. However, when you delete a full directory, you will be warned.

Example:

Maybe it is time for a short example. **Bold** text is typed by the user:

```
BAOS Shell v0.1
$ ls
../
./
bin/
inittab
$ mkdir
Create dir:
> test
$ ls
../
./
bin/
inittab
test/
$ cd
Change to:
> test
$ ls
../
./
$ cp
Source:
> ../inittab
Destination:
> a_file
```

Continuing...

```
$ ls
../
./
a_file
$ cd
Change to:
> ..
$ rm
Delete file:
> test
Delete all files
inside directory
[Y/N] y
$ ls
../
./
bin/
inittab
$ halt
```

3.4: Switching screens.

Another feature of BAOS is the virtual screen functionality. BAOS has 4 logical text-screens, and 1 logical graphical screen. Each screen can have it's own data (text or images), which means that multiple programs can be active and generate output at once, each on it's own screen.
Under normal circumstances, after starting BAOS you are on the first logical text-screen (or just: screen 1), which means that the data of screen 1 is being displayed on the LCD display. The data on other screens are stored inside your calculator's RAM until you switch screens.

Screen	First logical text screen	Second logical text screen	Third logical text screen	Fourth logical text screen	Logical graphical Screen
Short Name	Screen 1	Screen 2	Screen 3	Screen 4	Screen 5
Function Key	F=	WINDOW	ZOOM	TRACE	GRAPH

Switching your active screen

To switch your active screen (The logical screen that is being displayed on the (physical) LCD screen), you need to do the following:

- Press and hold the ON-key.
- Press the function key belonging to the screen you want to switch to (see the table above) once.
- Release the ON-key
- The new active screen is being displayed at the LCD screen.

Note about the Graphical Screen

There is a major difference between the graphical screens and the text screen. The text screens are operated by BAOS itself, and immediately react on switching to it.

The graphical screen, however, needs additional software to be used. To use the graphical screen, you need a Window Manager from a GUI, or any other third-party Graphical Screen Manager. Without this software running, you won't notice any screen switching is being done. You won't even get an empty screen, because there is nothing to tell the OS to wipe it clean!

4. NOTES ON THIS RELEASE

4.1: Current stage.

Currently, BAOS is in a early alpha stage. This means that the Operating System lacks much functionality. However, bit by bit BAOS is progressing towards a real Operating System.
At this time there is (more or less) everything that is needed to run simple programs (like the included text-editor). The BAOS Shell takes care of starting these programs.

Because there is not yet any linking functionality, every program you might want to execute must be stored within the Operating System at assemble time. See Appendix B for more information on how to store your own programs into the filesystem.

Some programs are build and can be reached from within BAOS by default. They are described in the next paragraphs.

4.2: The “cat” program.

This is a very simple program, which let you enter some text, after which is displays the same text underneath it. To leave the program, type exit and press the ENTER-key.

4.3: The “ps” program.

This program is a tool to easily watch the process-table (for more information, see the Developers section of BAOS, http://bzc.sf.net/ -> BAOS -> OS Documentation. You will find it under “Multitasking”.)

The process table has $20 (= 32) entries, trough which you can “walk” by pressing the left or right arrows.

On the top of the screen you find the name of the process (if given any), or “empty”, when there is no process stored at this location.

Direct underneath the name, you find the entry-number you are displaying, you can (as said above) change this by pressing the left or right arrow.

The next thing displayed is the PID, or *process identity*, a unique number that identifies the process.

Next is the state: 1 means running, 0 means sleeping. You will find that most of the time processes are sleeping, because they wait for the user to press a key, or just have a time-out.

The next two: `stackptr` (Stack Pointer) and `prgm cntr` (Program Counter) are technical, for which I refer to the Developers section of BAOS (see above).

Next we have the screen-number the process is running on, see paragraph 3.5 of this manual for more information)

At last we find the Flash page, this process has currently access to.

Note that because of the way the multitasking engine of BAOS works, you can't monitor some stats of the `ps`-process itself. You will see the "*`This process`*" message instead. However, if you would have 2 `ps`-processes (running on different virtual screens), they would be able to monitor each other's stats.

To leave the program, press the MODE/QUIT-key, you are back at the shell.

4.4: The text-editor

From this release on, BAOS features a simple text editor, to edit (and of course create) text files. Just start the editor by typing `edit` at the command line.

When the editor is started, you will see an empty screen with the cursor in the top-left corner. You can now start typing text.

Moving around

You can use the arrow keys to move around in your text. (Up to go up one row, left/right to go to the previous/next character. When you press left at the start of a row, the cursor will be places at the last character of the previous row.)

When you type something within the text, the editor moves all text at the right of the cursor one position to the right and inserts the character at the location of the cursor (like you might be used from a text editor or word processor on a computer).

To delete a character to the left of the cursor, use CLEAR. To delete the character *on* the cursor, use DEL.

The menu

When you press the MODE/QUIT-key, you enter the menu.
You can choose here to load or save a file, or exit the text editor, by pressing the number corresponding with the menu entry (i.e. Press 1 to load a file).

When you load or save a file, you will be asked for the path to it. When you do not give a directory (relative or absolute), the editor will search for / save the file in the current working directory.
You are warned when you try to overwrite an existing file (by giving the name of a file that already exists).

NOTE: See the second note in paragraph 3.4 (*The Baos FileSystem*)

When you load a file, the cursor is placed at the beginning of it (first row, first column).

When a file is too large, you are being warned that is not possible to insert more text without deleting something else first. If you try to load a file that is too big, you get an error message. However, you need to have a *very* big file to get this kind of messages.

When for some reason the saving or loading of files fails, you will get an appropriate error message.

Use the menu to exit the editor.

5.

OTHER RESOURCES AND CONTACT INFORMATION

If this manual just wasn't enough for you, you may find it useful to go to the BAOS website, located at http://bzc.sourceforge.nl/ -> BAOS. You can find here the latest news, the current status of the project and much information for developers.

Still not enough? In that case, there are several email-addresses you can sent mail to*:

bugs@hofhom.nl	Do you find a bug in BAOS? Please notify us at this email-address.
ideas@hofhom.nl	Do you think you have *the* idea to give BAOS it's break-trough? Please send it to this address. Other comments (both positive as negative) are also welcome.
help@hofhom.nl	You still have a question that keeps being unanswered? You ask it by mail, to make sure it gets inside these manuals.

*: To get a quick reaction and to make sure you don't get inside a spam-filter, please put “BAOS” in the subject of your email massage.

Appendix A
A GUIDE TO PROGRAMMING FOR BAOS

A.1: The problem

When the BAOS project was started, the goal was to create a Operating System that was capable of doing some real “computer”-like things. One thing was really needed to get such an experience, was multitasking (or, for educated people: multi*programming*). However, this creates 2 problems:

1. The switching of programs will create a significant delay at execution time. Not to mention that if you have 3 programs running, every program will theoretically have one third of the available CPU cycles at max.

2. Since there are multiple programs running at the same time, you cannot have one place in memory where programs are loaded, which means that a program is not aware where it will be executed at assemble-time.

The first problem just exists, however a program is able to *sleep*, while it's not active (for example it waits for a key to be pressed), and can this way consume only a little bit of the CPU cycles available. There is also the advantage of the *screen manager* which let programs continue executing while they would otherwise be waiting for the LCD-driver. Thus this problem is not that bad.

The other problem is really a problem. Very fundamental things (like the program's *origin*, defined by the *.org* line) can't be defined anymore.
BAOS offers a solution called *program loading*. When a program is loaded, the Operating System copies it to some free space inside the memory, and after this is done, the program gets parsed for special op-codes (like JP and CALLS). These are modified to fit to the new conditions, after which the program is started.

A.2: The layout

As you probably can imagine, programs that are written for BAOS, have a different layout than those that are written for TI-OS. There are some basic rules about them, depending on which kind of assembler you use (TASM or Z80ASM, the last one is the opensource assembler which is used to assemble the Operating System). These are three formats for a simple program, containing the biggest issues (and is therefore highly inefficient):

TI-OS:

```
#include "ti83plus.inc"

.org $9D93

    LD HL, start
    PUSH HL
    RET

start:
    LD HL, pgm_data
    B_CALL(puts)
    RET

pgm_data:
.db "Program loading works!!\n", 0

.end
```

BAOS, Z80ASM:

```
#include "baos_z80asm.inc"

    LDCODE HL start
    PUSH HL
    RET

start:
    LDPTR HL pgm_data
    CALL printstr
    RET


pgm_data:
.db "Program loading works!!\n", 0

pgm_end:
```

BAOS, TASM:

```
#include "baos_tasm.inc"

    LDCODE(HL, start)
    PUSH HL
    RET

start:
    LDPTR(HL, pgm_data)
    CALL printstr
    RET


pgm_data:
.db "Program loading works!!\n", 0

pgm_end:

.end
```

A.3: The Rules

You can see that the layout indeed differs from that where you might be used to. To explain this layout, lets list all the rules that are needed to program for BAOS:

1. You don't use .org anywhere.
2. You use a special header file, that comes with BAOS.
3. You separate the program code from the program data (numbers, strings, textures, anything that is not executable) and put the special *pgm_data* label between them. You must always use this label, even in the case that you do not have any data.
4. At the end of the data you put the *pgm_end* label. this one is also required.

<continues on the next page>

The following rules are a bit difficult to get used to, but there is a logical reason for every rule.

5. When using normal arithmetic, you use the LD command to load a 16-bit number (a.k.a. imm16) into a 16-bit register:

   LD HL, 0

   LD DE, 0

   LD BC, 0

6. It is forbidden to load a 16-bit number value between $4000 and $7fff into an index register:

   LD IY, $5000

   LD IX, 20500

   LD IX, 0

7. To load a pointer to data (between *pgm_data* and *pgm_end*) into a normal 16-bit register, you use the LDPTR macro. Index registers can be used normally:

   LD HL, string1

   LD IX, string1

   LDPTR(HL, some_data) (TASM)

   LDPTR DE sprite1 (Z80Asm, note: no comma).

8. If you like using hacks, or think that JP is going to take over the world and you are prepared to fight back, you should use the LDCODE macro to load a code label (every label before *pgm_data*). This is also needed for index registers.

```
LD HL, writebyte

LDCODE(DE, my_routine)     (TASM)
PUSH DE
RET
```

These are the basics of BAOS programming. Currently there is no library support, and you have access to all the OS routines (which you can find on the developers section of the BAOS website, http://bzc.sf.net/ -> BAOS -> Function Reference).

Because BAOS has not yet any linking functionality, you must store your programs currently inside the OS itself, which will copy them to the filesystem when you run it for the first time. You can read more about this in appendix B.

Appendix B

Storing your programs in BAOS at Assemble-time

Because BAOS has not yet any linking functionality, you need to 'link' your own applications into BAOS at build time, in order to find them back at in the filesystem at run-time.

The best way to do that is to integrate them into the BAOS build process.

In order to do this, you need to put your source code in the `src/userspace` directory of the BAOS package. After doing this, you need to change two files.

The make file (MS Windows users)

To include your own source in the BAOS build, you have to edit the `make.bat` file inside the package. Open it with a text editor, and scroll down until you see something like:

```
rem ***** PUT HERE YOU'RE OWN PROGRAMS ******
```

Just use the template on these lines to add your own files. However, this way (which is currently the only supported way) they are assembled with the *z80asm* assembler. Over this, scroll down some more and copy-paste the same after the second

```
rem ***** PUT HERE YOU'RE OWN PROGRAMS ******
```

That's all.

The make file (Unix users)

For Unix users it's even easier.

Open the file `src/userspc/Makefile`, and append the name of your source file(s) to the last line of the `OUTPUT_FILES` files, only use `.h` instead of the `.asm` extension (of course the actual source files should still the `.asm` extension). That's all, the *z80asm* assembler will be used to assemble your source.

Including your programs in the filesystem

The next step is adding the assembled programs to the filesystem. This is done by the build process, using the `src/userspc/userspc.in` file as input. This file is more or less a list of every program that should be included.

Just scroll down to the end of this file and add for every program you want to add:

`third_party_`*Program name*`=`*Program name*`,`*Program name*`.h`

Note: Your assembly source file should also have the name of your program, followed by `.asm` of course.

Note2: Do not use spaces or tabs anywhere on this line.

Example:

You want to add the program *myprog*:

- You put the `myprog.asm` source file into `src/userspc`.
- You add the following line to `src/userspc/userspc.in`:

  ```
  third_party_myprog=myprog,myprog.h
  ```

For MS Windows users:

- Insert the following lines at the two marked places in the `make.bat` file:

```
..\tools\z80asm myprog.asm -o myprog.bin
..\tools\bin2h myprog.bin myprog.h
```

For Unix users:

- Add your file to the OUTPUT_FILES variable in src/userspc/Makefile. After doing this, the file might look like this (but that depends on the version of BAOS), the bold text is added:

```
OUTPUT_FILES =  demo.h fs.h test.h echo.h \
                recv.h iic.h \
                init.h firstrun.h sh.h halt.h sleep.h reboot.h \
                cp.h rm.h mkdir.h cd.h ls.h \
                edit.h ps.h \
                inittab.h myprog.h
```

Now simply run *make* or the `make.bat` file to rebuild BAOS.

Appendix C

The /inittab configuration file

As said before (in paragraph 3.2), when BAOS boots the `init` program is started after some time, which will set up your system. `init` uses the `/inittab` file as configuration. However, you can edit this file to your own wishes using the text editor that comes with BAOS.

Start the text editor and open the `/inittab` file, you will see something similar to this:

```
21/bin/sh
22/bin/sh
23/bin/sh
24/bin/sh
End
```

This seems rather strange and difficult at first sight, but it's actually rather easy. What `init` actually does is just start some programs. See the following table:

inittab configuration layout:

	Field 1 (one digit)	**Field 2** (one digit, only available in run-mode 2)	**Field 3** (a full path)
Layout:	Execution mode	Virtual Screen	Program to start
Example:	2	1	/bin/sh

The virtual screen is just the screen the program will run on (remember there is a shell running on every virtual screen, this is done by `init`, using the above lines). The path is just the full path to the executable file of the program. By default all programs reside in the directory `/bin`.

The execution mode can be one of the following:

0. i`nit` will start this program and wait with further execution until the program is finished.
1. `init` will start the program and immediately continue execution.
2. `init` will start the program on the specified virtual screen and immediately continue execution.

Note: In execution mode 0 or 1 the virtual screen field is ignored.

Example:

You want that instead of a normal shell, the text editor is automatically started at screen 4.

The steps you take:

- You want to let the program run at a specified virtual screen, so you use execution mode 2.
- You choose for screen 4
- You add the path to the text editor.

The complete file will look as follows:

```
21/bin/sh
22/bin/sh
23/bin/sh
24/bin/edit
End
```

Reboot and enjoy your text editor!