# e4thcom - Terminal for Embedded Forth Systems

## Abstract

*e4thcom is a terminal program for embedded Forth Systems that supports **conditional and unconditional uploading of source code**. It runs on 32 BIT Linux Systems, X86 and Raspberry/ Raspbian.*

*The latest version **e4thcom 0.4.4** comes with a number of modifications and extensions:*

- *executable for Raspberry/Raspbian added*
- *amforth , 4e4th ,  mecrisp and **noForth** (new) targets are supported now*
- *target specific code was factored out into small Forth Source Code files (plug-in)*
- writing a *plug-in should be easy for experienced Forth users*
- *a cross assembler interface was added*
- a *MSP430 cross assembler for/from noForth is included (to be used and as an example)*
- a *file with register and bit identifiers for the MSP430G2553 MCU is included (to be used and as an example)*

## Forth on Embedded Systems

Forths interactivity is an important advantage on embedded systems. Any simple terminal program can be used to get in touch with the system and explore its structure and functionality.

But - as soon as the first program is written and ready for testing - an easy way to upload the source code files is missing. A tool to overcome this deficiency is the e4thcom terminal program.

## The e4thcom Terminal

e4thcom is a minimalistic terminal program for 32 Bit Linux Systems (X86 and Raspberry Pi/ Raspbian). It sends terminal input without a local echo to the connected target (Forth System) and displays characters received from the target in the terminal window. This is the terminal-mode of the program.

Entering the character # as the first one of a new line activates the control-mode of the program. Terminal input is then not forwarded to the target system but catched until the enter key is pressed and is then interpreted as a terminal directive.

## Terminal Direktives

Terminal directives temporarily change the operational mode of the terminal. Two directives are defined for uploading of source code files. They are modelled after the corresponding words *include* and *require* from the  Forth Standard and named *#include* and *#require*. Both expect to get the name of a source code file as the only parameter:

```
   #include <file name>  \ unconditional uploading

   #require <file name>  \ conditional uploading
```

Both directives can also be included in source code files, one directive per line, the rest of the line is silently discarded.

The file name lookup is done at the path `cwd:cwd/target:cwd/mcu:cwd/lib`. cwd is the directory, that was the active one (current working directory) at program start. The underlaying concept is, to store project-specific source code in the cwd directory, target-forth-specific in cwd/target, hardware-specific in cwd/mcu and target-, mcu- and project-independent code in cwd/lib.

When directives are entered at the terminal, the abbreviations #i and #r can be used.

**Uploading of Source Code**

Source code files are red line by line and the text is interpreted as a terminal directive or uploaded to the target.

After sending a character to the target, the terminal waits for the echo and displays it in the terminal window. The transmission is aborted with an error message, if no echo is received.

At the end of a line, the terminal waits for the target to process that line and return with a message. If no error occured, the next line is red from the source. Otherwise the transmisson is aborted with an error message.

Comments starting with a \ char - are not uploaded but only displayed in the terminal window.

When the end of a source code file is reached, the file is closed and control is given back to the calling level. Finally, when the last file is closed, the terminal mode is activated again.

**Conditional Uploading of Source Code**

Before uploading a file, the terminal checks, whether a word, that equals the file name, already exists in the targets dictionary. The upload directive is ignored, if that is the case. Otherwise the file is uploaded.

After uploading a file, the terminal again checks if now a word exists in the target dictionary, that equals the file name. If that is not the case, a NOOP-Word with that name is created in the target dictionary.

With this approach, the number of dictionary entries for file names can be minimized, by using - whenever adequate - the name of a prominent word, defined in a file, as the name for that file.

**Installing e4thcom**

e4thcom is distributed as a tar.gz archive. For testing or as a user without admin rights unpack the archive in a directory of your choise, e.g. in your home directory or on the desktop. You will then find the e4thcom binary and related files in a new directory named e4thcom-xyz. The e4thcom binary in this directory is the one for X86 Linux Systems.

When installing e4thcom on a Raspberry Pi with Raspbian OS please copy the binary from the Raspbian subdirectory to the e4thcom directory.

As an admin you can make e4thcom available to all system users by unpacking the archive to the /opt directory, changing owner and group with `chown -R root:root /opt/e4thcom-xyz` and creating links named e4thcom and forthbox in /usr/local/bin to the corresponding files in the /opt/e4thcom-xyz directory.

**Starting e4thcom**

To start the e4thcom terminal program, open a terminal window in a (project) folder of your choice and enter the command

```
/path/to/e4thcom [options]
```

The following options are supported:

| | |
|---|---|
| **-t** [4e4th, amforth, mecrisp, noforth] | the target system to connect to *(option changed)* |
| **--xas** | enables a cross assembler *(new)* |
| -b [B1200, B2400, B4800, B9600, B19200, B38400, B57600, B115200] | the baudrate to be used |
| -d [device] | the serial device, e.g. ttyS0, ttyUSB0, ttyACM0 |
| -h | displays the terminals help text |

The connection to the target system is established during the program start. This may fail if the chosen device is not available or used by another process (e.g. by a modem manager). The program is then terminated with an error message. After fixing the cause of failure or waiting for the modem manager to give up, a successful start of the program should be possible.

**Babel ...**

All target systems have some peculiarities in their transmission protocol concerning the messages returned after processing a line of uploaded code.

Up to version 0.3.4 e4thcom was split in an executable file and a number of target specific image files, named after the targets. Starting with version 0.4 there is only one image file for all targets. The target specific code was factored out and is now loaded as a plug-in at program start.

The target specific plug-in files are small Forth Source Code Files so that support for other targets can be easily added by experienced Forth Users.

**Cross Assembling**

Starting with version 0.4 e4thcom comes with a cross assembler interface. A target specific cross assembler, written in Forth, can be included by the target specific plugin at program start. e4thcom must be called with the **--xas** option to enable the assembler interface. Using the cross assembler requires that the words **code** and **end-code** are defined in the targets dictionary. Both do not need to have any e4thcom related properties.

e4thcom 0.4.4 comes with a MSP430 cross assembler for noForth that is based on the file aux430ass.f from noForth1210.

When a cross assembler is loaded and enabled, the words **code** **end-code** and **\xas** are interpreted as terminal directives (in source code files only, not when entered at the terminal command line).

A line of source code, that starts with **\xas** is not uploaded to the target but interpreted by the cross assembler (the terminals internal Forth interpreter). This can be used to extend the cross assembler, e.g.

to define short macros or identifiers for MCU resources

```
   \xas 021 equ P1OUT  01 equ BIT0
```

or to load identifiers with **`MCU:name`** from a file name.efr

```
    \xas   MCU: MSP430G2553
```

The search order to find name.efr is project-dir:e4thcom-dir.

A line of source code starting with the phrase **`code name`** enables the terminals cross assembler mode. The phrase "code name" is then forwarded to the target to create a header in its dictionary and the following source code up to **`end-code`** is then assembled into a buffer in the terminals memory. Finally **`end-code`** disables the cross assembler mode, comma-compiles the assembled code into the targets memory and is send to the target to close the code definition.

```
    code led1.on ( -- ) BIT0 # P1OUT & .b bis next end-code
```

**The terminals number base in the cross assembler mode is hex.** Its Forth interpreter is case sensitive. So is the cross assembler. I prefere **lower case for operators** and **upper case for register and bit identifiers**.

**With a little help ...**

e4thcom does not have a command line history and does not support extended command line editing. It's intentionally designed minmalistic. But, in accordance with the UNIX Philosophy, one can combine it with a tool, that adds missing capabilities. Such a tool is the ForthBox.

**e4thcom in a ForthBox**

The ForthBox is a GTK+ GUI for Forth Interpreters and Forth Terminals. It is a virtual terminal emulator with some add-ons that make editing and uploading/execution of Forth source code files a bit more comfortable.

A preferences dialog lets the user chose the client to be started in the terminal emulator, the project folder to be used, the (external) editor and the parameters for the serial link.

To start the e4thcom terminal program in a ForthBox the following commands can be used:

`/path/to/e4thcom/e4thcom -i forthbox`      or      `/path/to/e4thcom/forthbox`

The preferences dialog pops up immediately after the program start. After chosing the required options and pressing the OK-Button e4thcom is started in the ForthBox terminal window. If the e4thcom client aborts with an error message, consider the hints given earlier concerning the possible reasons and try again.

The ForthBox has not yet been released on its own and documentation is still missing. So some hints are given here:

• The toolbar allows the user to choose a file for editing or uploading by means of the  Edit- resp. the Execute-Button.

• Next to this two buttons there is a Menu-Button, that gives access to the global list of recently used files, which is managed by the OS. All files recently chosen for editing are also listed here.

• A file, chosen for editing, is forwarded to the external editor, that was chosen in the preferences dialog.

• With the first entry in the recent files menu of the Execute-Button, the terminal client (in this case e4thcom) can be restarted if required.

• A separate command line with text selection and text completion can be activated underneath the terminal window by clicking on the check-button in the lower right corner of the ForthBox window.

• The preferences dialog reads its data from a config file named .forthbox . You can copy it into your project directories and edit it according to your preferences. The content should be self-explanatory. Search order is project-directory:e4thcom-directory.

**ForthBox on the Raspberry Pi/Raspbian**

When starting the ForthBox on the Raspberry you may see a number of GLib warnings. Ignore it. The ForthBox works as expected. (The same phenomenon occures with other Gtk applications.)

When connecting to the Raspberry via ssh -X and starting the ForthBox for the first time it may fail with an error message from the X-server. Reset the terminal with `reset [Enter]` and try again. You will then succeed.

**What's left to do :**

• Adding device locking to prevent concurrent device access from different processes. If another process uses the same device as the e4thcom terminal, it may snap away received chars so that the communication with the target will not work properly.

*The problem here seems to be, that not all programs, that use serial devices, use device locking or don't do it the same way.*

• Adding more targets and cross assemblers.

You can do it yourself and/or contribute. *The new plug-in structure should make it easy. The plug-in files and the cross assembler file that come with this release should be useful examples.*

**Annotation:**

e4thcom was implemented with MINFORTH Plus, a MINFORTH derivative, and was tested with the following target systems:

Arduino Duemilanove : amforth 4.9 and 5.1 *Test results for newer versions are welcome !*

TI MSP430 Launchpad : 4e4th 0.34, mecrisp 1.1, noFORTH V&C 141218

Questions, error notes and suggestions for improvements or additional targets are welcome and should be forwarded to manfred.mahlow@forth-ev.de .

*Thanks to Andreas Knochenhauer for creating MINFORTH, which was a great starting point to create my own Forth development environment MINFORTH Plus.*

*Thanks to Albert Nijhof and Willem Ouwerkerk for supporting me to make e4thcom ready for noForth and for agreeing that the noForth cross assembler could be adapted to and can be distributed with e4thcom.*

Last Revision: MM-150103