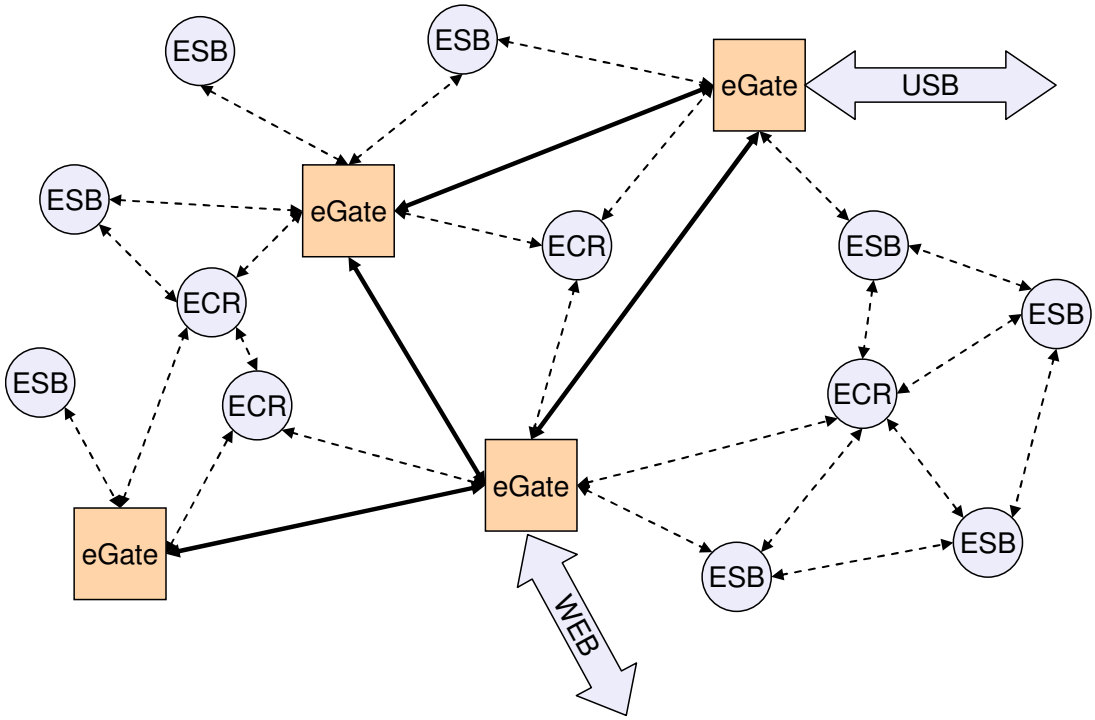
veritas
iustitia
libertas

Freie Universität Berlin

TinyOS documentation (for ScatterWeb)

A platform for teaching & prototyping wireless sensor networks



scatterweb.mi.fu-berlin.de

Computer Systems & Telematics

Freie Universität Berlin, Germany

Content

1 Introduction

This user manual shows how to use TinyOS on a PC and set up a connection to the components of the ScatterWeb-platform. Predominantly, the user manual is targeted at people who are experienced with TinyOS on other components. Additionally questions of new users are answered.

You will find more detailed information about downloading and installing TinyOS and the specific interfaces for the ScatterWeb. The ScatterWeb-interfaces are introduced and a short introduction into TinyOS on the basis of an example application is given. Subsequently, it explains how to compile TinyOS and flash the compiled code on the nodes.

The installation procedure is presented for an Windows environment with cygwin, but other operating systems are quite similar to use.

2 Components of the ScatterWeb

Three of the components of the ScatterWeb platform are prepared to be operated by TinyOS. If you are not familiar with the ScatterWeb, read the following instructions, otherwise continue with chapter 3 (see page 9) .

2.1 Embedded Chip Radio

By the many analog and digital IO-ports the Embedded Chip Radio (ECR) can be connected to other sensors, onboard are thermometers and vibration sensor. The Embedded Chip Radio uses a low-power MSP 430 controller and radio on 868 MHz.

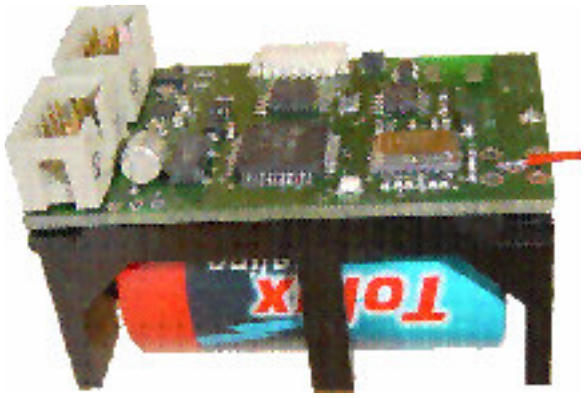


Figure 1: Embedded Chip Radio

2.2 Embedded Sensor Board

Figure 2: Embedded Sensor Board

The Embedded Sensor Board (ESB) is the component of the ScatterWeb, which is equipped with the largest number of sensors. These are temperature, vibration, infrared movement sensor, infrared transmitter / receiver, microphone / speaker, luminosity sensor and buttons. Communication is possible over radio (868 MHz), serial interface and I²C bus. Also equipped with the MSP 430 the device is optimized for low power consumption and with three standard AA batteries the device has an long life cycle of five years. This corresponds a use of 1 % (with sensor activities and wireless communication). The ESB can operate by a solar cell only. In the sleep mode (LPM4) the Embedded Sensor Board uses only 8 µA. Individual assemblies are possible - and all this at extremely low prices!

2.3 ScatterFlasher

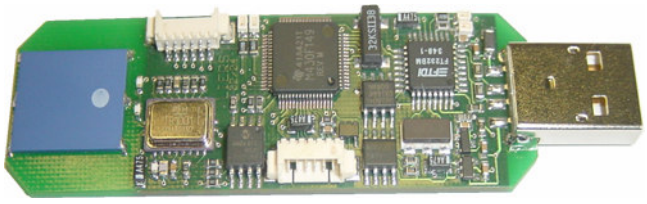


Figure 3: Scatter Flasher

The ScatterFlasher is the optimal gateway between your PC and the ScatterWeb. Connected to the USB Over the air Flashing, collecting sensor data, debugging and more is available. Also this component is equipped with a MSP 430, the radio is also working at 868 MHz.

3 Installation Instructions

3.1 Downloading the TinyOS-package

At the beginning you have to download the TinyOS package from the Website at the University of Berkeley (www.tinyos.net) on your own. The download is more than 100 MB. You can also use your already existing TinyOS-installation.

Please make sure in both cases that the compiler for the microcontroller MSP 430 used on our components is integrated or installs the compiler additionally if it is necessary.
You can also use the TinyOS package, which is linked on the page scatterweb.mi.fu-berlin.de in the menu TinyOS. The following guidance describes this installation, because in this case you get all required tools. Please store all downloaded files in one directory.

3.2 Installing the TinyOS-package

After the download the following files should be stored in one directory: Autorun.inf, Data1.cab, setup.exe, Setup.skin and TinyOS.msi (see figure 4).

Now start the installation by executing the file setup.exe.

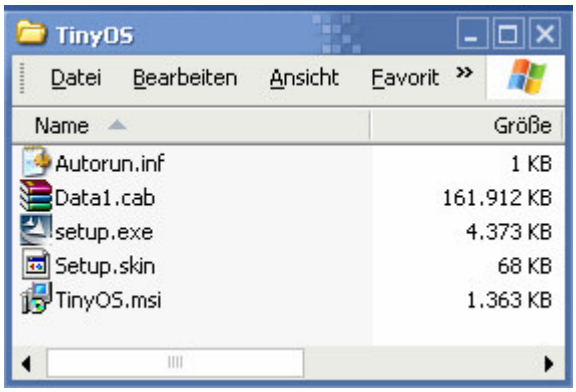


Figure 4: Files after the download

After eventually confirming a warning the installation program starts and a welcome screen appears (see figure 5).

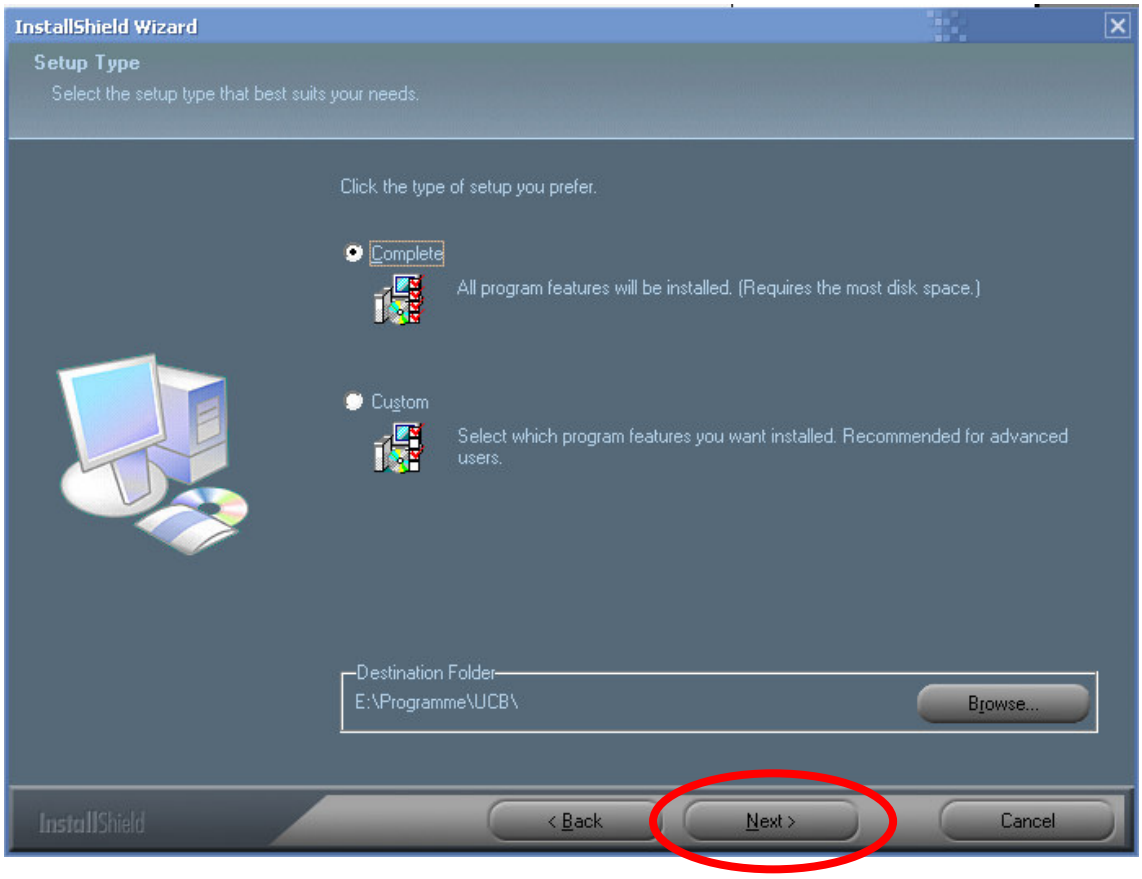


Figure 5: The starting screen of the installation program

If you like you can change the destination folder with one click on *Browse*. Otherwise the TinyOS package will install to \Program Files\UCB.

!	Please remember the name of the folder because the name has to be indicated in the further process of the installation.

Leave the type of setup adjusted to Complete and continue the installation by clicking on the *Next >* button.

i	By selecting the option Custom you can select in the further installation process more detailed installation options, this is not recommended and will not be treated in this guidance.

In the following screen (see figure 6) you have to confirm that you have read the license agreement for the JDK and Java Communications API. If you accept it, click on *Yes*.

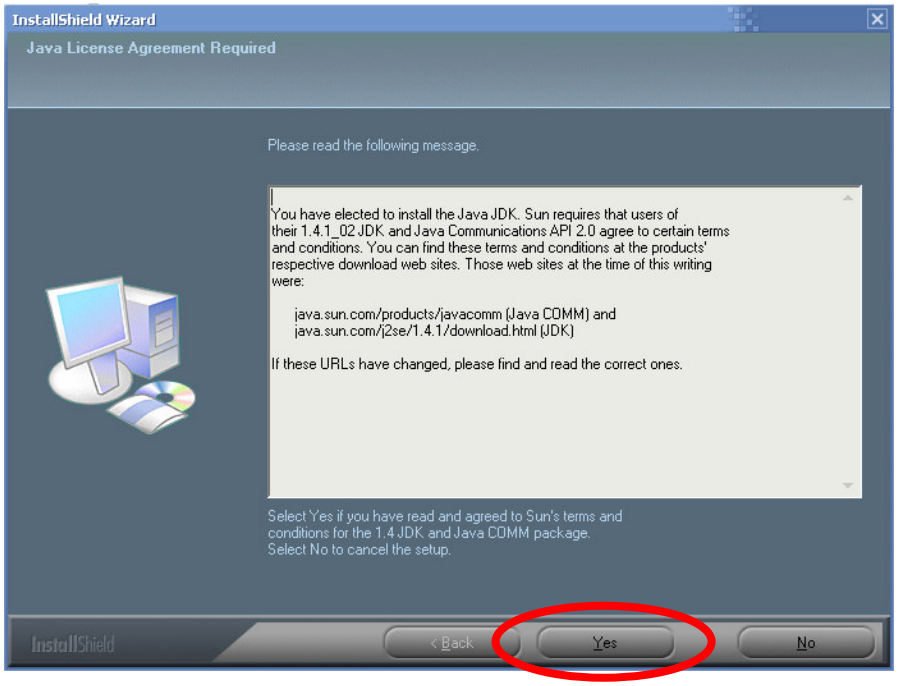


Figure 6: Accept the license agreements

On the final screen (see figure 7) all selected installation options are indicated again. You can start the installation now by clicking on *Next >*.

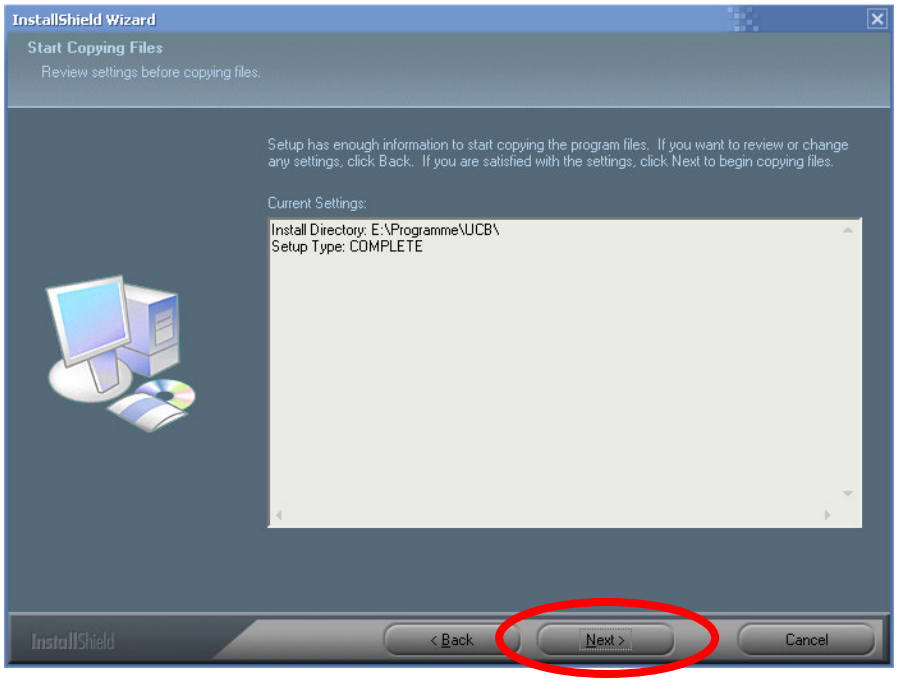


Figure 7: Final screen

After the installation the setup will now install cygwin and all necessary RPM´s (see figure 8). Confirm this with a click on *Continue*.

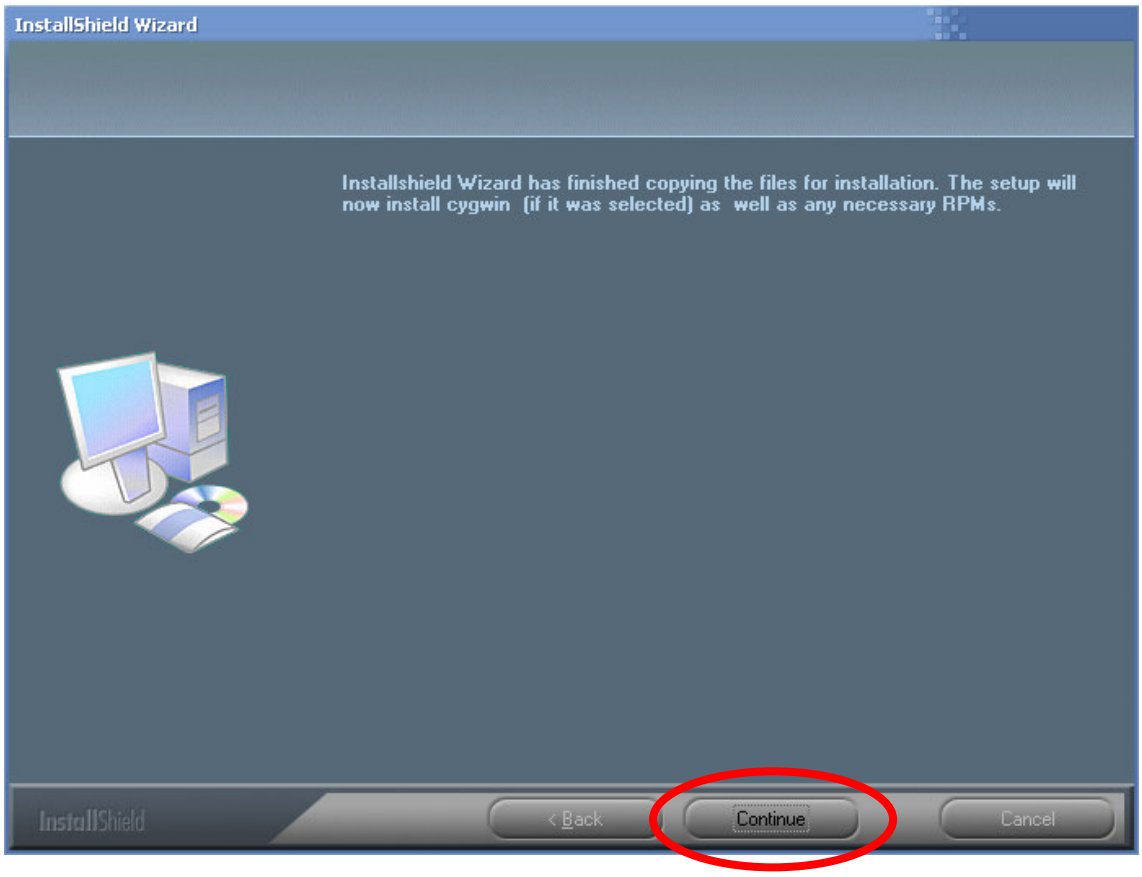


Figure 8: Installation of Cygwin

Now the Cygwin installation (see figure 9) starts. Cygwin is a program which makes a linux like user surface and associated tools available under Windows.

This installation is very extensive, in the further process several black windows will appear. Please wait until the installation is completed.

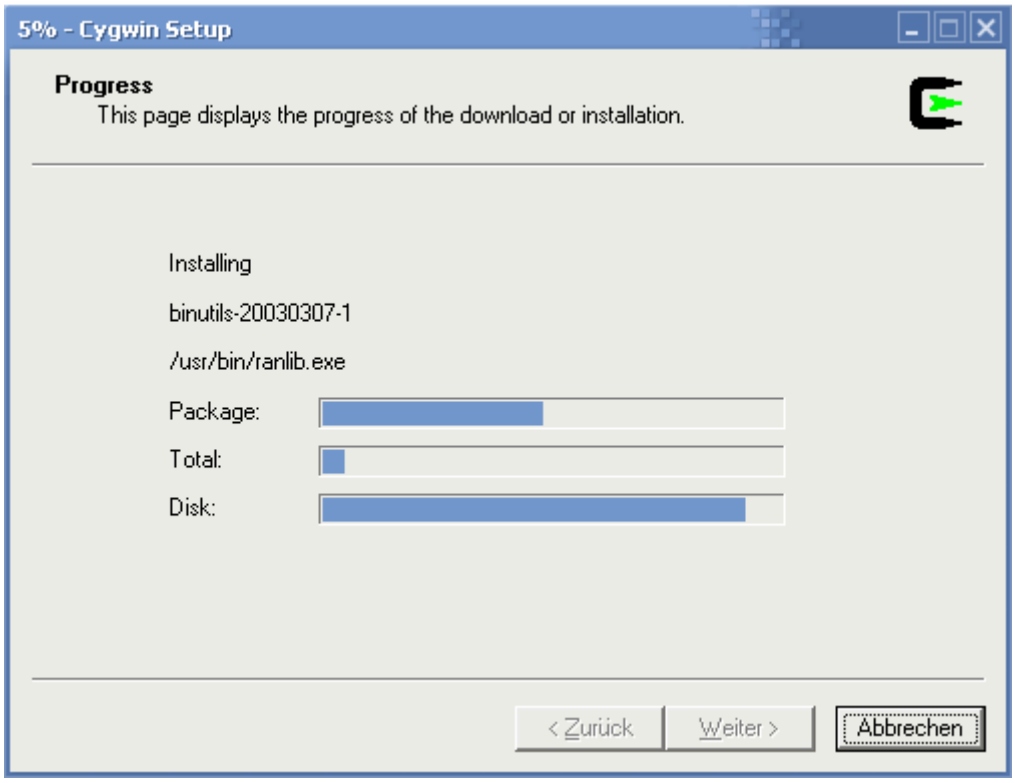


Figure 9: Cygwin-Installation

Subsequently, a short message box informs you about the successful ending of the installation.

It has now created a shortcut on your desktop and in the start menu in the folder cygwin to start Cygwin.

3.3 Downloading the ScatterWeb specific interfaces

To make the system operable with our components there are extensions and drivers to enhance the system on our website. These extensions make it possible to use our components in your software. Supported interfaces are introduced in the following chapter 4 (see page 17).

You will find these software elements under scatterweb.mi.fu-berlin.de in the menu TinyOS. Select the component in the list on the download page, which you would like to use, and download the software.

For each components an own interface structure is existing, adapted to the appropriate hardware. This guidance treats a merging of this extensions for the Embedded Sensor Board (ESB). Merging the extensions for the further components takes place similarly.

After downloading you find a file named <platform>.exe (e.g. ESB.exe). Start this with a doubleclick. After confirming a possible safety warning, you will see the following screen (see figure 10):

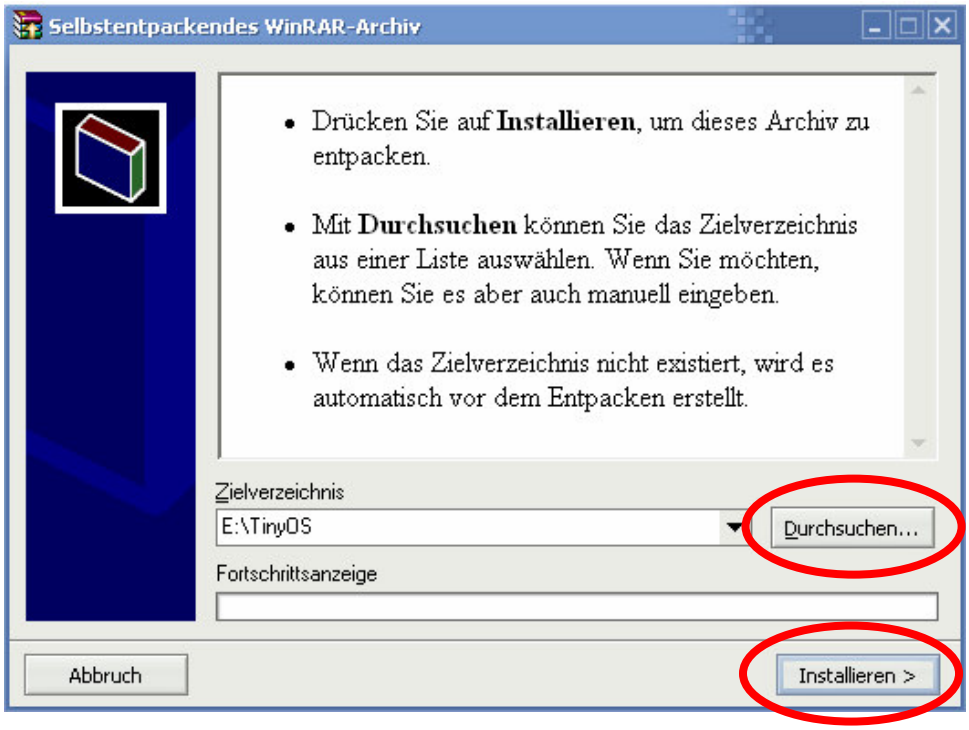


Figure 10: Extensions install

With one click at *Durchsuchen* you can select a folder, into the files will be unpack.

Please remember the name of the folder because the name must indicated in the further process of the installation again.

By clicking on *Installieren >* the contained files will be unpacked.

3.4 Installing the ScatterWeb specific interfaces

Open now the selected folder. You can find several files with the ending .target (e.g. ESB.target) in this folder, a file esw.extra and a further folder named <platform> (e.g. ESB).

Now open additionally the folder which you indicated for the installation of the TinyOS package (see page10). According to standard this folder is called UCB and is in the folder program files. Now change in the subfolder \cygwin\opt\tinyos-1.x\tools\make.

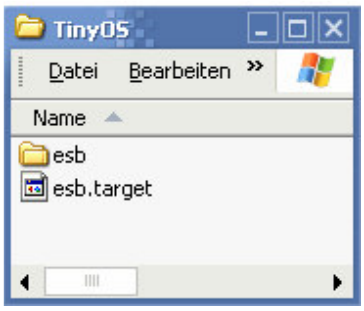


Figure 11: Folder with unpacked files

Depending on the current TinyOS version the file name (tinyos-1.x) can slightly deviate.

Copy all files with the ending target (e.g. ESB.target) and the file esw.extra into this folder and overwrite already existing files if necessary.

Change now in the subfolder \cygwin\opt\tinyos-1.x\tos\platform and copy the folder <platform> (e.g. ESB).

You have to repeat these steps for all components, which you would like to use.

You can delete the folder with the downloaded files now.

Now all preparations are made in order to use TinyOS. If you would like to get further information about the ScatterWeb specific interfaces, read chapter 4. In order to learn how to write your own applications and programming with TinyOS you should continue with chapter 4 (see page 30). In order to use TinyOS and already existing applications, jump to chapter 6 (see page 39).

Advanced users can load the current source code from TinyOS from the CVS, by typing
cvs -z3: d:pserver:anonymous@cvs.sourceforge.net:/cvsroot/tinyos co tinyos-1.x
in the Cygwin bash (without line breaks).

4 Operating Instructions

In this chapter we present the interfaces with which you can work if you would like to access certain ScatterWeb specific components in your programs.

How you use these interfaces is described, in chapter 5 (see page 30) more specifically.

An interface lists a number of functions, which you can use to develop your own application. For example you can switch the LEDs on and off with the interface LED on the components.
There is still a large number of further interfaces, which are delivered with TinyOS.

4.1 LEDs

\tinyos-1.x\tos\interfaces\Leds.nc:

```
interface Leds {
  async command result_t init();
  async command result_t redOn();
  async command result_t redOff();
  async command result_t redToggle();
  async command result_t greenOn();
  async command result_t greenOff();
  async command result_t greenToggle();
  async command result_t yellowOn();
  async command result_t yellowOff();
  async command result_t yellowToggle();
  async command uint8_t get();
  async command result_t set(uint8_t value);
}
```

The interface LEDs make functions available in order to control the LEDS on the components. It was contained in TinyOS, but adapted by us on our components.

The associated standard implementation is LedsC.

```
async command result_t init();
```

Function: The LEDs are set into an operational condition.

Parameter: None

Return value: Always *SUCCESS*

```
async command result_t redOn();
```

Function: The red LED switches on

Parameter: None

Return value: Always *SUCCESS*, even if no red LED is available

```
async command result_t redOff();
```

Function: The red LED switches off

Parameter: None

Return value: Always *SUCCESS*, even if no red LED is available

```
async command result_t redToggle();
```

Function: The red LED switches

Parameter: None

Return value: Always *SUCCESS*, even if no red LED is available

```
async command result_t greenOn();
```

Function: The green LED switches on

Parameter: None

Return value: Always *SUCCESS* even if no green LED is available

```
async command result_t greenOff();
```

Function: The green LED switches off

Parameter: None

Return value: Always *SUCCESS*, even if no green LED is available

```
async command result_t greenToggle();
```

Function: The green LED switches

Parameter: None

Return value: Always *SUCCESS*, even if no green LED is available

```
async command result_t yellowOn();
```

Function: The yellow LED switches on

Parameter: None

Return value: Always *SUCCESS*, even if no yellow LED is available

```
async command result_t yellowOff();
```

Function: The yellow LED switches off

Parameter: None

Return value: Always *SUCCESS*, even if no yellow LED is available

```
async command result_t yellowToggle();
```

Function: The yellow LED switches

Parameter: None

Return value: Always *SUCCESS*, even if no yellow LED is available

```
async command uint8_t get();
```

Function: Queries the status of the LEDs

Parameter: None

Return value: Returns a 8-bit value, its last bit represents the red LED, whose next to last bit the green LED and its third last bit the yellow LED. If the appropriate bit is set, the LED is switched on, otherwise it is switched off. This value is equivalent to a number between 0 and 7.

```
async command result_t set(uint8_t value);
```

Function: All LEDs set

Parameter: The parameter is a 8-bit value, its last bit represents the red LED, whose next to last bit the green LED and its third last bit the yellow LED. If the appropriate bit is set, the LED is switched on, otherwise it is switched off. This value is equivalent to a number between 0 and 7.

Return value: Always *SUCCESS*

4.2 Beeper

\tinyos-1.x\tos\platform\esb\Beeper.nc:

```
interface Beeper {
  async command result_t init();
  async command result_t on();
  async command result_t off();
  async command result_t toggle();
  async command uint8_t get();
}
```

The interface Beeper makes functions available with which the Beeper on the components can be controlled. The return values depend on whether the used component has a Beeper or not.

The associated standard implementation is BeeperC.

```
async command result_t init();
```

Function: Initialize the Beeper

Parameter: None

Return value: Always *SUCCESS* returns, otherwise always *FAIL* with components with no Beeper

```
async command result_t on();
```

Function: Switches on the Beeper

Parameter: None

Return value: Always *SUCCESS* returns, otherwise always *FAIL* with components with no Beeper

```
async command result_t off();
```

Function: Switches off the Beeper

Parameter: None

Return value: Always *SUCCESS* returns, otherwise always *FAIL* with components with no Beeper

```
async command result_t toggle();
```

Function: Switches the Beeper

Parameter: None

Return value: Always *SUCCESS* returns, otherwise always *FAIL* with components with no Beeper

```
async command uint8_t get();
```

Function: Readout the current status of the Beeper

Parameter: None

Return value: 0 or 1 returns, according to whether the Beeper is switched on (1) or switched off (0).

4.3 Sensors

\tinyos-1.x\tos\platform\esb\SensorsC.nc:

```
configuration SensorsC {
  provides {
      interface ADC as Pir;
      interface ADC as Vib;
      interface ADC as Mic;
      interface ADC as Tem;
      interface ADC as Ext;
      interface ADC as Bat;
      interface ADC as Rxp;
      interface Button;
      interface IR;
      interface SplitControl;
      interface TemperatureDevid;
  }
}
```

\tinyos-1.x\tos\interfaces\ADC.nc

```
interface ADC {
  async command result_t getData();
  async command result_t getContinuousData();
  async event result_t dataReady(uint16_t data);
}
```

```
\tinyos-1.x\tos\interfaces\SplitControl.nc
interface SplitControl {
  command result_t init();
  event result_t initDone();
  command result_t start();
  event result_t startDone();
  command result_t stop();
  event result_t stopDone();
}
\tinyos-1.x\tos\platform\esb\TemperaturDevid.nc
interface TemperatureDevid {
  async command uint16_t get();
}
\tinyos-1.x\tos\platform\esb\Button.nc
interface Button {
  async event result_t pressed();
}
\tinyos-1.x\tos\platform\esb\IR.nc
interface IR {
  async event result_t detected();
}
```

The interface Sensors makes functions available, in order to be able to query the different sensors. If a sensor on a component is missing, then the value FAIL and/or 0 is always returned.

The associated standard implementation is SensorsM.

```
async command result_t SplitControl.init();
```

Function: Initialize the sensors

Parameter: None

Return value: Always *SUCCESS*

```
event result_t SplitControl.initDone();
```

Function: Terminates the initialization

Parameter: None

Return value: Always *SUCCESS*

```
async command result_t SplitControl.start();
```

Function: The sensors switches on

Parameter: None

Return value: Always *SUCCESS*

```
event result_t SplitControl.startDone();
```

Function: Terminates the starting procedure

Parameter: None

Return value: Always *SUCCESS*

```
async command result_t SplitControl.stop();
```

Function: The sensors switches off

Parameter: None

Return value: Always *SUCCESS*

```
event result_t SplitControl.stopDone();
```

Function: Terminates the stop procedure

Parameter: None

Return value: Always *SUCCESS*

```
async command result_t Pir.getData();
```

Function: Reads the counter of the movement sensor

Parameter: None

Return value: At components with movement sensor and not locked status *SUCCESS*, otherwise *FAIL*

```
async command result_t Pir.getContinuousData();
```

Function: Scans the counter of the movement sensor

Parameter: None

Return value: At components with movement sensor *SUCCESS*, otherwise *FAIL*

```
async event result_t Pir.dataReady(uint16_t data);
```

Function: One calls, as soon as the desired value was determined

Parameter: The current value of the movement sensor

Return value: Always *SUCCESS*

```
async command result_t Vib.getData();
```

Function: Reads the counter of the vibration sensor

Parameter: None

Return value: At components with vibration sensor and not locked status *SUCCESS*, otherwise *FAIL*

```
async command result_t Vib.getContinuousData();
```

Function: Scans the counter of the vibration sensor

Parameter: None

Return value: At components with vibration sensor *SUCCESS*, otherwise *FAIL*

```
async event result_t Vib.dataReady(uint16_t data);
```

Function: One calls, as soon as the desired value was determined

Parameter: The current value of the vibration sensor

Return value: Always *SUCCESS*

```
async command result_t Mic.getData();
```

Function: Reads the counter of the microphone

Parameter: None

Return value: At components with microphone and not locked status *SUCCESS*, otherwise *FAIL*

```
async command result_t Mic.getContinuousData();
```

Function: Scans the counter of the microphone

Parameter: None

Return value: At components with microphone *SUCCESS*, otherwise *FAIL*

```
async event result_t Mic.dataReady(uint16_t data);
```

Function: One calls, as soon as the desired value was determined

Parameter: The current value of the microphone

Return value: Always *SUCCESS*

```
async command result_t Tem.getData();
```

Function: Reads the current temperature

Parameter: None

Return value: At components with temperature and not locked status *SUCCESS*, otherwise *FAIL*

```
async command result_t Tem.getContinuousData();
```

Function: Scans the current temperature

Parameter: None

Return value: At components with temperature *SUCCESS*, otherwise *FAIL*

```
async event result_t Tem.dataReady(uint16_t data);
```

Function: One calls, as soon as the desired value was determined

Parameter: The current value of the Termometers

Return value: Always *SUCCESS*

The conversion of the temperature depends on the used hardware. You find functions for conversion in 0 on page 44.

```
async command result_t Ext.getData();
```

Function:	Reads the tension of the external voltage supply
Parameter:	None
Return value:	At components with external source of tension and not locked status *SUCCESS*, otherwise *FAIL*

```
async command result_t Ext.getContinuousData();
```

Function:	Scans the tension of the external voltage supply
Parameter:	None
Return value:	At components with external source of tension *SUCCESS*, otherwise *FAIL*

```
async event result_t Ext.dataReady(uint16_t data);
```

Function:	One calls, as soon as the desired value was determined
Parameter:	The current value of the external voltage supply
Return value:	Always *SUCCESS*

You receive the tension in volts by the following computation:
$$U = \frac{Wert}{4096} * 3V * \frac{52}{11}$$

```
async command result_t Bat.getData();
```

Function:	Reads the tension of the batteries
Parameter:	None
Return value:	At components with batteries and not locked status *SUCCESS*, otherwise *FAIL*

```
async command result_t Bat.getContinuousData();
```

Function:	Scans the tension of the batteries
Parameter:	None
Return value:	At components with batteries *SUCCESS*, otherwise *FAIL*

```
async event result_t Bat.dataReady(uint16_t data);
```

Function:	One calls, as soon as the desired value was determined
Parameter:	The current value of the batteries
Return value:	Always *SUCCESS*

You receive the tension in volts by the following computation:
$$U = \frac{Wert}{4096} * 3V * 2$$

```
async command result_t Rxp.getData();
```

Function:	Reads the receiving power of the transceiver
Parameter:	None
Return value:	At components with transceiver and not locked status *SUCCESS*, otherwise *FAIL*

```
async command result_t Rxp.getContinuousData();
```

Function:	Scans the receiving power of the transceiver
Parameter:	None
Return value:	At components with transceiver *SUCCESS*, otherwise *FAIL*

```
async event result_t Rxp.dataReady(uint16_t data);
```

Function:	One calls, as soon as the desired value was determined
Parameter:	The current value of the receiving power
Return value:	Always *SUCCESS*

```
async command uint16_t TemperatureDevid.get();
```

Function:	Reads the devid of the current temperature
Parameter:	None
Return value:	At components with temperature devid the current value, otherwise 0

```
async event result_t Button.pressed();
```

Function: One calls, as soon as the button was pressed

Parameter: None

Return value: Always *SUCCESS*

```
async event result_t IR.detected();
```

Function: One calls, as soon as infrared radiation was detected

Parameter: None

Return value: Always *SUCCESS*

4.4 UART (serial interface)

\tinyos-1.x\tos\interfaces\HPLUART.nc:

```
interface HPLUART {
  async command result_t init();
  async command result_t stop();
  async command result_t put(uint8_t data);
  async event result_t get(uint8_t data);
  async event result_t putDone();
}
```

The interface HPLUART makes functions available the enable you to communicate with the serial interface on the lowest level.

The associated standard implementation is *HPLUARTM*.

```
async command result_t init();
```

Function: Activated and initialize the serial interface

Parameter: None

Return value: Always *SUCCESS*

```
async command result_t stop();
```

Function: The serial interface switches off

Parameter: None

Return value: Always *SUCCESS*

```
async command result_t put(uint8_t data);
```

Function: Sends data of the length of a byte over the serial interface

Parameter: The data byte which can be sent

Return value: Always *SUCCESS*

According to specification it should be waited after sending for the fact that the Event putDone() arises, before a byte can be sent again. In the ScatterWeb is this not compulsory necessary, because the processing is blocked, until the data were sent.

```
async event result_t get(uint8_t data);
```

Function: One calls, as soon as a byte data became to receive

Parameter: The received data byte

Return value: Always *SUCCESS*

```
async event result_t putDone();
```

Function: One calls, as soon as put() is final

Parameter: None

Return value: Always *SUCCESS*

4.5 BPS of the serial interface

```
\tinyos-1.x\tos\platform\esb\HPLUARTBPS.nc:
interface HPLUARTBPS {
  async command result_t set(uint8_t value);
  async command uint8_t get();
}
```

The interface HPLUARTBPS makes it possible to specify the transmission rate of the serial interface. According to standard it is preset on 115200 bps.

The associated standard implementation is *HPLUARTBPSC*.

```
async command result_t set(uint8_t value);
```

Function: The bps of the serial interface sets

Parameter: A value, which represents the bps

Return value: Always *SUCCESS*

The values result as follows:
ESWBPS_2400, ESWBPS_19200, ESWBPS_38400, ESWBPS_57600, ESWBPS_115200

```
async command uint8_t get();
```

Function: Reads the bps of the serial interface

Parameter: None

Return value: A value, which the bps represented (see set)

4.6 Radio

\tinyos-1.x\tos\platform\esb\RadioCRCPacket.nc

```
configuration RadioCRCPacket {
  provides {
      interface StdControl as Control;
      interface BareSendMsg as Send;
      interface ReceiveMsg as Receive;
  }
}
```

\tinyos-1.x\tos\interfaces\StdControl.nc

```
interface StdControl {
  command result_t init();
  command result_t start();
  command result_t stop();
}
```

\tinyos-1.x\tos\interfaces\BareSendMsg.nc

```
interface BareSendMsg {
  command result_t send(TOS_MsgPtr msg);
  event result_t sendDone(TOS_MsgPtr msg, result_t success);
}
```

\tinyos-1.x\tos\interfaces\ReceiveMsg.nc

```
interface ReceiveMsg {
  event TOS_MsgPtr receive(TOS_MsgPtr m);
}
```

The configuration RadioCRCPacket connects the three interfaces StdControl, BareSendMsg as well as ReceiveMsg and makes thereby all functions available in a collection, which are needed for the use of the radio module.

The associated standard implementation is RadioCRCPacketM.

```
command result_t init();
```

Function: Initialize the radio module

Parameter: None

Return value: Always *SUCCES*S

```
command result_t start();
```

Function: Starts the radio module

Parameter: None

Return value: Always *SUCCES*S

```
command result_t stop();
```

Function: Stops the radio module

Parameter: None

Return value: Always *SUCCES*S

```
command result_t send(TOS_MsgPtr msg);
```

Function: A message sends over the radio module

Parameter: msg: Pointer on the message, which is to be sent

Return value: *SUCCESS*, if the message will be sent, otherwise *FAIL*

After the call of this function, if *SUCCESS* was returned, the event *sendDone* is called.

```
event result_t sendDone(TOS_MsgPtr msg, result_t success);
```

Function: One calls, as soon as sending the message was completed

Parameter: msg: Pointer on the message, to be sent should

success: *SUCCESS*, if the message were sent

Return value: Always *SUCCES*S

```
event TOS_MsgPtr receive(TOS_MsgPtr m);
```

Function: One calls, as soon as a package became to receive

Parameter: m: Pointer of the received package

Return value: Pointer of the next received package

4.7 Transmitting power

\tinyos-1.x\tos\interfaces\Pot.nc:

```
interface Pot {
  command result_t init(uint8_t initialSetting);
  command result_t set(uint8_t setting);
  command result_t increase();
  command result_t decrease();
  command uint8_t get();
}
```

The interface Pot makes functions available, with which the transmitting power of the transceiver can be steered.

The associated standard implementation is PotM.

```
command result_t init(uint8_t initialSetting);
```

Function: Initialize the transmitting power

Parameter: The parameter initialSetting determines the initialization size

Return value: If the parameter lies in the range of values *SUCCESS*, otherwise *FAIL*

The range of values lies between 0 and 100.

```
command result_t set(uint8_t setting);
```

Function: Set the transmitting power to a certain value

Parameter: The parameter setting determines the setting size

Return value: If the parameter lies in the range of values *SUCCESS*, otherwise *FAIL* (see init)

```
command result_t increase();
```

Function: Increases the transmitting power by a stage

Parameter: None

Return value: If the increase does not lead to the fact that the range of values will leave *SUCCESS*, otherwise *FAIL* (see init)

```
command result_t decrease();
```

Function: Reduces the transmitting power by a stage

Parameter: None

Return value: If the reduction does not lead to the fact that the range of values will leave *SUCCESS*, otherwise *FAIL* (see init)

```
command uint8_t get();
```

Function: Reads the current transmitting power

Parameter: None

Return value: The current value of the transmitting power

4.8 NodeType

\tinyos-1.x\tos\platform\esb\NodeType.nc:

```
interface NodeType {
  async command uint8_t get();
}
```

The interface NodeType makes it possible to determine the types of the node.

The associated standard implementation is *NodeTypeC*.

```
async command result_t get();
```

Function: Reads the type of the node

Parameter: None

Return value: A value, which represents the types of the node

The values result as follows:
ECR, ESB, ESF

5 First steps – Programming TinyOS

The following chapter is to arrange first steps for you.

If you like to use an application with TinyOS, that is already existing, copy these into a new file in \tinyos-1.x\apps\ and continue you with chapter 1 (see page 39).

5.1 Preparations

Change into the file \tinyos-1.x\apps \ and provide you there a new subfolder, which should carry the name, as your application is to be called.

In this example we call the application MyApp.

This file MyApp now provides you to three files with the following names: makefile, MyApp.nc and MyAppM.nc.

The two latter files can also be called differently, however for the sake of clarity offer themselves to keep these conventions.

5.2 The makefile

The Makefile instructions stand for the compiler. All Makefiles are identically developed for simple applications, you can thus always use this source code to take over and have only the name of application to accordingly adapt.

```
COMPONENT=MyApp
include ../Makerules
```

5.3 The configurations

In the configuration one is specified, which interfaces in application are used and addressed under what name.

The configurations are located in the file MyApp.nc. The interface StdControl must always be implemented, so that the use of TinyOS can be started and terminated. Minimum contents of a configuration look thus as follows:

```
configuration MyApp { }
implementation {
  components Main, MyAppM;
  Main.StdControl -> MyAppM;
}
```

If you want to run a function periodically, you need to add a timer. Thus the configuration looks as follows:

```
configuration MyApp {
}
implementation {
  components Main, MyAppM, TimerC;
  Main.StdControl -> TimerC.StdControl;
  Main.StdControl -> MyAppM;
  MyAppM.Timer -> TimerC.Timer[unique("Timer")];
}
```

5.4 The application

Actual applications are located in the file MyAppM.nc. In the minimum case it looks as follows:

```
module MyAppM {
  provides {
      interface StdControl;
  }
}
implementation {
  command result_t StdControl.init() {
      return SUCCESS;
  }
  command result_t StdControl.start() {
      return SUCCESS;
  }
  command result_t StdControl.stop() {
      return SUCCESS;
  }
}
```

The function init() is called, before application is started. You start via call of start(), for terminating stop() is called. Their program can be written thus in the simplest case completely into the function start(). The programming language, which is used, is called nesC.

If you want to call a function periodically, the realization takes place for example as follows by means of the timer:

```
module MyAppM {
  provides {
      interface StdControl;
  }
  uses {
      interface Timer;
  }
}
implementation {
  command result_t StdControl.init() {
      return SUCCESS;
  }
  command result_t StdControl.start() {
      call Timer.start(TIMER_REPEAT, 1000);
      return SUCCESS;
  }
  command result_t StdControl.stop() {
      call Timer.stop();
      return SUCCESS;
  }
  event result_t Timer.fired() {
      return SUCCESS;
  }
}
```

In the function start() the timer is started, which ensures in this example each second that the function Timer.fired() is called. From there, you are able to program the part of the application, which regularly is to be called.

For further references to programming for TinyOS please consider the sources of information specified in the appendix.
In TinyOS further applications in the file apps, which are able to serve you as collecting main, lie.
Sample programs, which use special functions of our hardware, you find www.scatterweb.net on our Website.

6 Last steps – Compile and Flash TinyOS

They can either only compile TinyOS and your application only or compile and flash them in one run.

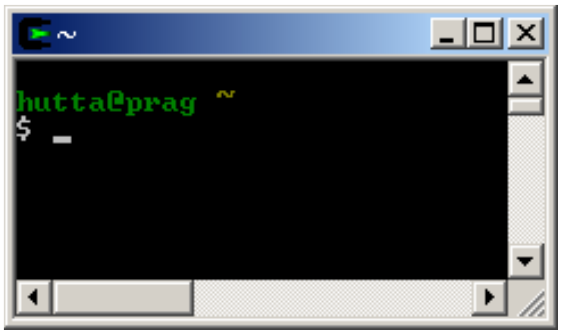


Figure 12: Starting picture Cygwin

For compiling or flashing of TinyOS and your application start by using the appropriate linkage in the starting menu and/or on the Desktop Cygwin. This announces itself with accompanying starting picture. In the appropriate places you can find user and computer names.

6.1 Compile TinyOS

Change by input of the instruction

```
cd "/opt/tinyos-1.x/apps/MyApp"
```

into the file, in which your application is put down. By input of

```
make <Platformname>
```

thus for example

```
make esb
```

start the compiling procedure.

The following message confirms the correctly locked procedure:

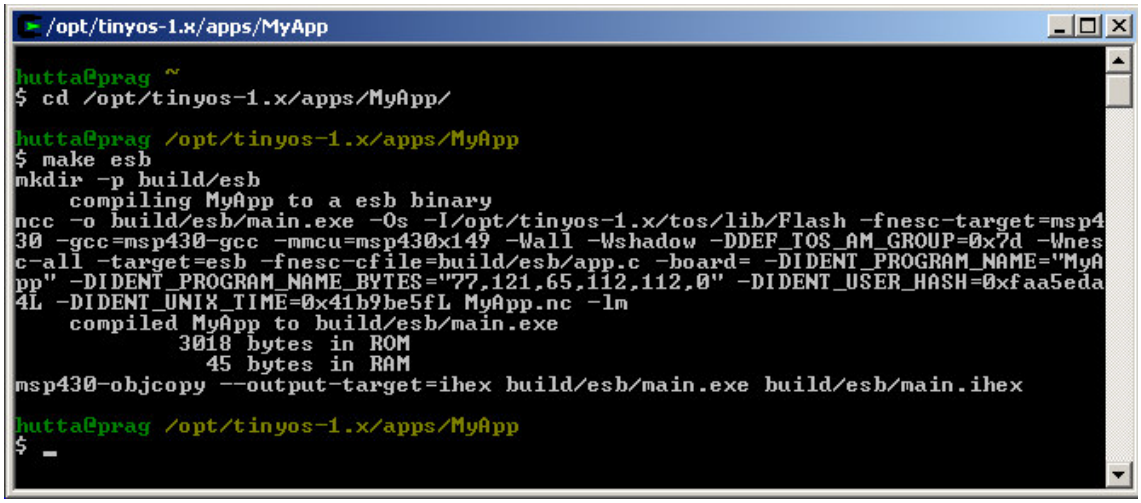


Figure 13: Completed compiler run

If the compiling procedure was not successful, don't ignore the warnings or errors.

After compiling you can find in the folder MyApp a subfolder named build. In this folder you can find a further subfolder with the name of your platform. There you will find a file named main.hex. Now you can transfer it with a suitable flashtool on the node.

6.2 Flash TinyOS

For common compiling and re-flashing in one go use Cygwin by input of the instruction

```
cd "/opt/tinyos-1.x/apps/MyApp"
```

thus changing into the folder in which your application resides. By input of

```
make <Platformname>_flash
```

thus for example

```
make esb_flash
```

start the compiling procedure with following flashing.

Guarantee that you attached the component to be flashed over an JTAG interface to the parallel port of your computer.

The following message confirms the correctly locked procedure:

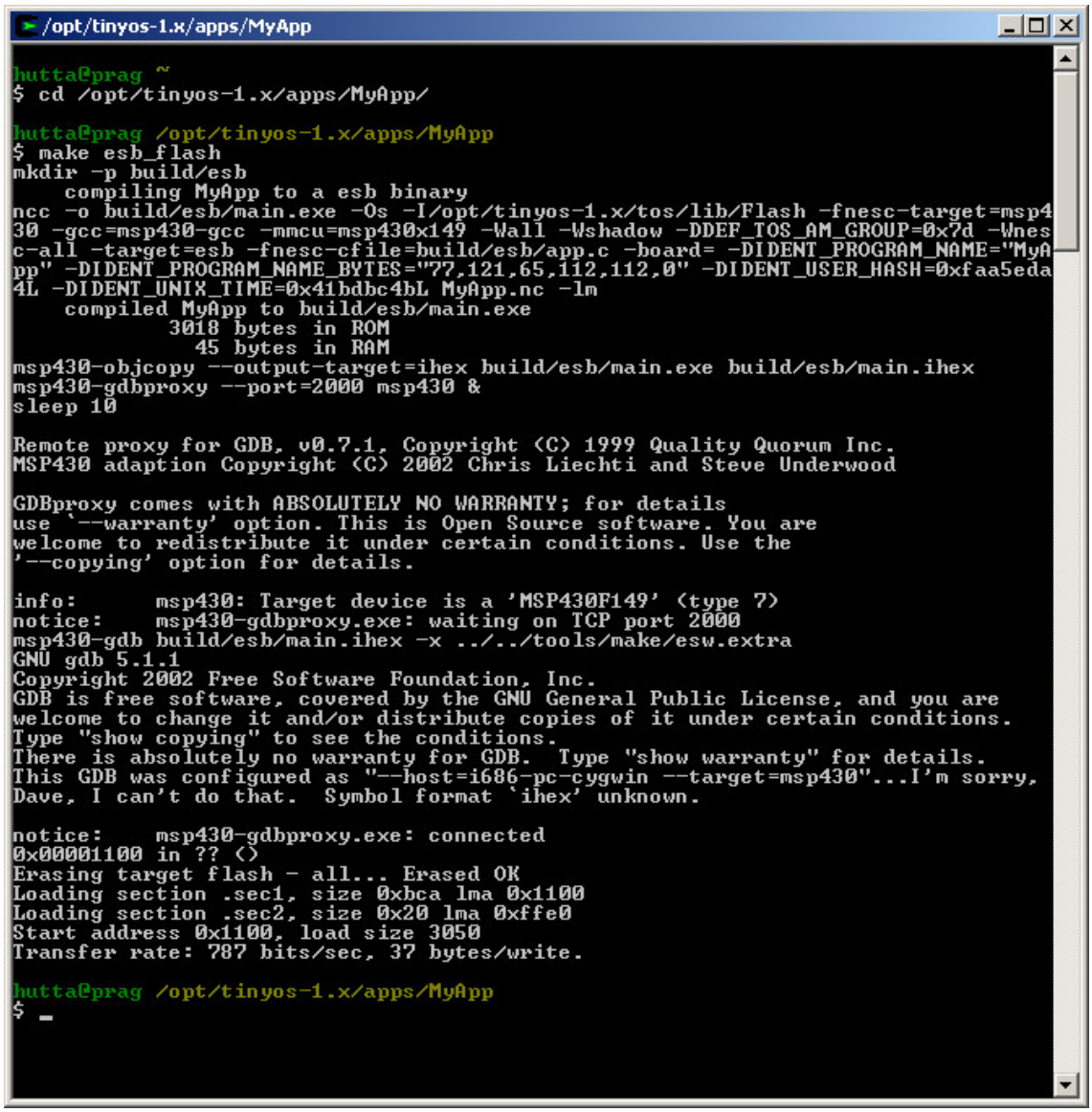


Figure 14: Message after the flashing

The first part of the message is identically to when simply compiling, the second gives information whether the flashing was successful.

Appendix A – Quick Installation Guide

1. Load yourselves the files of the TinyOS package from scatterweb.mi.fu-berlin.de (category TinyOS) down.

2. Start the installation (setup.exe) and install you for TinyOS with the standard attitudes.

3. Load the extensions for ScatterWeb (dependent on your hardware) of scatterweb.mi.fu-berlin.de and unpack them.

4. Copy the file <platformname>.target from \cygwin\opt\tinyos-1.x\tools\make and the file <platform contraction> from \cygwin\opt\tinyos-1.x\tos\platform.

5. Place the applications in a subfolder of \cygwin\opt\tinyos-1.x\apps.

6. If you start Cygwin, change into the folder in which your application lies, and compile it for TinyOS by issuing make <platform contractions>.

7. Alternatively you can compile for TinyOS on your hardware by issuing make <platformname>_flash.

Appendix B – Temperature conversion

If you want to convert the value which the temperature sensor returns into ℃ you can use the following code fragment:

For Embedded sensor board:

```
void loadTemperature(char* temp, int value) {
  char nk = (value & 0x0080);
  char vk = (value >> 8);
  if(vk < 0) temp[0] = '-'; else temp[0] = '+';
  if(vk & 0x80) vk = ~vk;
  temp[4]='.';
  temp[3] = 0x30 + (vk % 10);
  vk = vk / 10;
  temp[2] = 0x30 + (vk % 10);
  vk = vk / 10;
  temp[1] = 0x30 + vk;
  if(nk) temp[5] = '5'; else temp[5] = '0';
}
```

The parameter VALUE is the value, which the sensor supplies, if one queries the temperature. In temp the temperature level is written in ℃, first place comes the sign, the three following indications takes up the integral portion, it follows the point and afterwards a right-of-comma position.

Appendix C – Information

If you need resuming information about the ScatterWeb platform or assistance for problems, find to support here:

In the Internet at scatterweb.mi.fu-berlin.de

By E-Mail over the ScatterWeb mailing list. Information and registration is available via http://lists.spline.inf.fu berlin.de/mailman/listinfo/scatterweb, the E-Mail address reads scatterweb@lists.spline.inf.fu-berlin.de.

Information and assistance to TinyOS:

www.tinyos.net

Detailed guidance nesC:

http://www.tinyos.net/tinyos-1.x/doc/nesc/ref.pdf

TinyOS-Tutorial:

http://www.tinyos.net/tinyos-1.x/doc/tutorial/

Information to Cygwin:

www.cygwin.com

Information to the MSP 430 on the sides of the manufacturer:

http://www.ti.com

Programming tools for MSP 430::

http://www.iar.com