Searduino Manual

C/C++ environment
Stubs
Simulator
... for Arduino
Version: 0.9.93b
Date: 04 dec 2015

Henrik Sandklef

Table of Contents

1 Introduction

Searduino is made to ease and speed up developing code for the Arduino boards. In short, with Searduino you get

- C/C++ interface - use C/C++ to program your Arduino boards
- Makefiles - easy to use Makefiles for inclusion in your project
- Stand alone program - build your Arduino code to run on your local computer instead
- Simulators - run your Arduino code in a simulator to test it
- Simulation API - write your own test cases in C/C++
- Java simulation API - write your tests in Python
- Arduino example import
- Python simulation API - write your tests in Python (obsoleted)

1.1 C/C++ interface

With Searduino you can use your favorite Editor to develop your program. Write plain C code and compile it either for any of the Arduino boards or for your local computer.

1.2 Makefiles

Autobuild and test your code with no user interaction. Searduino comes with Makefiles that makes it easy, and not much Makefile code from you, to use the features of Searduino. Using a Makefile from an example, shipped with Searduino, you should get started in just a few seconds.

The Makefiles come with support for:

- compiling and linking for the Arduino boards
- compiling and linking for the Arduino Simulator
- compiling and linking for Stand alone programs
- cleaning up
- uploading code to the board

1.3 Stand alone program

Build for and run your arduino program on your local computer. Testing code on the Arduino is tricky sometimes and it not easy to automate these tests. Executing your program locally and looking the printouts (write on the pins) make up a simple test. For more serious testing we suggest you use the simulator (C or Python API or the GUI).

1.4 Simulator GUI

bladi

1.5 Simulation API

bladi

1.6 Python simulation API

write your tests in Python

1.7 Arduino example import

You can import Arduinon examples and turn them into Searduino projects with Makefiles and everything set up for you. You can do this either using the graphical simulator (`Jearduino`) or a script (`searduino-arduino-ex2c`).

Exampe on how to use the script:

```
/opt/searduino/bin/searduino-arduino-ex2c --yes --shlib
--destination-dir /home/hesa/searduino/
/usr/share/arduino/examples/01.Basics/Blink
```

2 Background

The authors of Searduino loves developing code for Arduino. We love using Arduino and we believe that developing code for Arduino has been made significantly easier for not-so-experienced-developers with the Arduino IDE. However, for some of us it is easier to develop code in our favorite editors and build and upload via the command line.

Searduino was initially created to make it possible to automate the building of your arduino program which has to be done outside of the Arduino IDE. Once we had the build and linking up and running we quickly noticed that it wouldn't take that much to make it possible to turn your Arduino program into a program executing on your local computer. The writes and reads on pins in your Arduino program were 'translated' in to reads and writes on stdin/stdout, which we used to create a simple simulator communicating via a pipe. We saw the potential of the simulator and decided to write a "proper" API for it instead. To make the simulator more usable for quick checks and for people preferring GUIs we started to write a simulator GUI in Python, so we added a Python extension to the simulator API. After a while we started hacking on a GUI frontend in Java. And here we are right now....

Hope you like it! And feel free to join us!

3 Abbreviations

- Arduino program - a program written for the Arduino board. Uses only the Arduio and avr APIs.
- stub - Type of board. When building the software to run locally on your computer and not building for real Arduino boards we use the word stub we use this word. A better name would perhaps have been sim or simulaor but stub it is.
- Faked Arduino - library implementing the Arduino and avr APIs
- Streamed input/output - instead of a fullblown simulator GUI Searduino provides you with a stdin/stdout interface. This can be used to script (bash, Python..) your test cases. Using programs such as netcat you can also run the Arduino program on one PC and the test on another PC.
- Java interface - All of the simulation features are offered by a C API s well as via a Java API.
- local computer - the computer you're developing your code on
- Jearduino - The grphical simulator, written in Java.
- Pearduino - The grphical simulator, written in Python/gtk. Obsoleted.
- Python interface - Most of the simulation features are offered by a C API s well as via a Python API. This API is deprecated.

4 Supported boards and platforms

Supported Arduino boards

- Uno - `http://arduino.cc/en/Main/arduinoBoardUno`
- Mega - `http://arduino.cc/en/Main/ArduinoBoardMega`
- Due - `http://arduino.cc/en/Main/arduinoBoardDuemilanove`
- Leonardo - `http://arduino.cc/en/Main/arduinoBoardDuemilanove`

Supported Operating Systems

- GNU/Linux (source code supports both 32 and 64 bits. Binaries available)
- MacOS (source code supports both 32 and 64 bits. Binaries available)
- Windows (source code supports both 32 and 64 bits. Binaries NOT available at the moment)

5 Getting and installing Searduino

To use Searduino you need some software installed (see Required software). To developer searduino you need some additional tools in order to create the makefiles, configure scripts etc. The latter will not be discussed in this manual. Instead, we will focus on how to use, and build Searduino.

You have a couple of options to build and install the software. The options we provide, until software packages for your distributions are available, are:

- Install a prebuilt binary version
- Build and install from a released version
- Build and install from the source code repository

Note: The first option is easiest. Only downside is that you have to live with our fixed installation path (/opt/searduino)

For some OS/distributions we have prepared scripts for setup and build our software. To see if your distribution is supported clone the git repository and look for files in the bin folder coresponding to your dist. If you, as an example, are using Debian you can setup your development environment and build the software this way:

- `bin/setup-debian.sh`
- `bin/build-debian.sh`

Note: Check the content in the scripts above to see if the settings fits your environment

5.1 Required software

5.1.1 Debian based GNU/Linux distributions:

- gcc-avr
- g++
- binutils-avr
- avrdude
- avrprog
- avr-libc
- openjdk-6-jdk (or openjdk-7-jdk)

5.1.2 Windows:

- Arduino (we need the avrdue program in there)
- Win-AVR
- cygwin (gcc-core, gcc-g++, make)

5.1.3 Mac:

- Xcode
- MacPorts - http://www.macports.org/install.php
- and via MacPorts install avrdude, avr-libc, gcc-avr, gcc, make
- Java JDK

5.2 Binary releases

5.2.1 GNU/Linux

Create a installation directiory (e.g /opt/searduino)
`mkdir -p /opt/`

Go to the installation directory
`cd /opt/`

Download a release from
`http://download.savannah.gnu.org/releases/searduino/bin/`
E.g http://download.savannah.gnu.org/releases/searduino/bin/searduino-bin-0.4-x86.tar.gz

Unpack
`tar zxvf searduino-bin-0.4-x86.tar.gz`

5.3 Released version of the source code

Download from
`http://download.savannah.gnu.org/releases/searduino/`

5.3.1 Installing a released version

You must first configure the makefile etc by typing:
`./configure`

The configure script accepts several option. Type `./configure --help` to see them.

Note: The configure script cannot find the jni.h file needed when building java extension and the Jearduino simulator frontend. To help the configure script you need use both the CFLAGS and CXXFLAGS to point out the directory of the jni.h file.

and then continue with building

```
make
```

and then continue with installating

```
make install
```

Note: You can configure Searduino in several ways (e.g with or without Python). See Configuring Searduino for more information

5.4 Getting the latest source code

We try to keep the latest version in the repository working but there's no guarantee. If you want to play safe use the released versions (see above).

5.4.1 Getting a tgz/tar.gz file from the git repository

Download from
`http://git.savannah.gnu.org/gitweb/?p=searduino.git;a=snapshot;h=HEAD;sf=tgz`

Info on how to install below

5.4.2 Cloning git repository

Download from
`git clone git://git.savannah.nongnu.org/searduino.git`

5.4.3 Building the latest version

First, create tke configure script to set up the Makefiles
`make -f Makefile.git`

After this, you should follow the procedures for Installing a released version (see above).

5.5 Verify installation

5.5.1 With Arduino examples

To verify the Searduino installation we have developed a script. To use it, type:

```
cd /opt/searduino
```

```
./scripts/verify-install.sh
```

You can upload all built program to the uno boards by adding the option `--upload`

```
scripts/verify-install.sh --upload
```

Note: this only uploads to the Uno boards

Experimental feature!! You can also execute each shared library in the stream simulator, by using the option `--simulate`

```
scripts/verify-install.sh --simulate
```

Note: Since the Arduino code will execute for ever, you must stop the simulator each time it is loaded with a shread library. You stop the simulator by pressing pressing Ctrl-c

5.5.2 With the digpins example

Copy the digpins example directory.

```
cp -r /opt/searduino/share/searduino/example/digpins /tmp
```

Enter the digpins example directory.

```
cd /tmp/digpins
```

Make sure that the SEARDUINO_PATH in the Makefile points to your Searduino installation dir.

Build blinker program for PC

```
make prog
```

Execute blinker

```
./blinker
```

The blinker program should run and print out (the printouts comes from the stub libraries). Interrupt the program by sending a signal, e g by pressing Ctrl-C.

Build blinker lib for use in simulator

```
make shlib
```

There should be a shared library file called `libdigpins.so` in the current directory. You can load this shared library (think of it as a plugin) in any of the simulators. We will load it in the stream simulator.

```
/opt/searduino/bin/searduino-stream-sim --arduino-code ./libdigpins.so
```

You should now see printouts from the Searduino simulator. You can stop the program by pressing Ctrl-C ('Control key' and 'c key' at the same time).

To load the library in the graphical simulator, type:

```
/opt/searduino/bin/searduino-jearduino.sh --arduino-code ./libdigpins.so
```

To load the entire project in the graphical simulator, type:

```
/opt/searduino/bin/searduino-jearduino.sh --searduino-project ../digpins
```

5.6 Configuring Searduino

5.6.1 Java support

By default Searduino builds a Java extension. To disable this support, configure with the option `--disable-java-extension`:
`./configure --disable-java-extension`:

5.6.2 Graphical Simulator support

By default Searduino builds a graphical simulator. To disable this support, configure with the option `--disable-jeardunio`:
`./configure --disable-jearduino`:

5.6.3 Python support

Searduino has an unmaintained Python extension and a Simulator GUI. These are diabled by default. To enable this support, configure with the option `--enable-python-extension`: `--enable-pearduino`:
`./configure --enable-python-extension --enable-pearduino`:

5.6.4 Enable unit tests with check

Configure with the option `--enable-unittest`
`./configure --enable-unittest`:
Note: The unit test software package check must be installed.

6 Using Searduino

In the previous chapter we looked a bit at the digpins example, so we now have some feeling for using Searduino. We will now proceed by writing our first Arduino program using Searduino.

6.1 Writing the first program

6.1.1 Let Searduino generate the first C file and Makefile

Let's assume you want to create a program and you want to store it in a directory called blinker. Then, all you have to do is to run the following command:

```
/opt/searduino/bin/searduino-builder --create blinker
```

Searduino will now create a main.c file and a Makefile. Let's have a look:

```
cd
cd searduino/blinker
ls -al
```

Let's build the example and run it in the simulator:

```
make
make sim-start
```

Let's build the example and run it on a board (Uno in this example):

```
make
make uno-upload
```

Nice :)

6.1.2 Write first C file on your own

There are a couple of steps needed to get a program built and loaded in to ether an Arduino board or any of the Searduino simulators. In short the steps are:

- write a main function
- call init() explicitly
- write an eternal loop in your program's main function

Open up your favorite editor (emacs?) and begin....

To use the Arduino functionality you need to include Arduino.h and searduino.h, so we need to add this to our file:

```
#include <Arduino.h>
#include <searduino.h>
```

When using Arduino IDE you've seen the `loop` function as the starting point for the program. With Searduino we're back to the normal C way of doing this with a `main` function, so we need to define a main function.

```
int main(void) { }
```

As with the loop function you're writing when you're using the Arduino IDE, the main function needs to never exit or return. It's a simple control loop (see `http://en.wikipedia.org/wiki/Embedded_system#Simple_control_loop`).

So a very simple main function looks like this

```
#include <Arduino.h>
#include <searduino.h>

void setup()

{
 pinMode(13, OUTPUT);

}


int main(void)

{
 init();

 setup();

 for(;;)

{
 digitalWrite(13, 1);
 delay(100);
 digitalWrite(13, 0);
 delay(100);

}

 return 0;

}
```

Note: this program sets pin 13 high and low with 0.1 secs interval. You don't need to connect a led to output pin 13, since pin 13 already has a built in led on the board.

6.1.3 Write the first Makefile on your own

Inporant settings in the Makefile

- SEARDUINO_PATH - should be set to the directory of your Searduino installation
- PROG - name of the program to build
- SHLIB - name of the shared library to build
- SRC_C - a list (separated with space) of C files to compile
- SRC_CXX - a list (separated with space) of C++ files to compile
- ARDUINO - should be set to the type of software you want to build (see **Build types** below)

Include the searduino makefile
You need to include some settings, targets and rules from Searduino. This is done by adding the following line to your Makefile.

```
include $(SEARDUINO_PATH)/share/searduino/mk/searduino.mk
```

A Makefile to build the code above can look like this:

```
SEARDUINO_PATH=/opt/searduino/
SRC_C=seardex.c
SRC_CXX=
ARDUINO=stub
PROG=seardex
SHLIB=seardex.so
include $(SEARDUINO_PATH)/share/searduino/mk/searduino.mk
```

Note: You don't have to use the makefiles provided by Searduino. The makefiles do however provide a lot of help (board settings etc).

6.1.4 Building the program for your local computer

To build your software to be executed on your local computer, and not for a real Arduino board:

make sure the the variable **ARDUINO** in the Makefile is set to **stub**.

and type:

```
make clean
make prog
```

To run the program

```
./seardex
```

6.1.5 Building a shared library for use in the simulators

To build your software to be executed on your PC: make sure the the variable **ARDUINO** in the Makefile is set to **stub**.

and type:

```
make clean
make shlib
```

To run the code in the stream simulator

```
/opt/searduino/bin/searduino-stream-sim --arduino-code ./seardex.so
```

To run the code in the Jearduino simulator

```
/opt/searduino/bin/searduino-jearduino.sh --arduino-code ./seardex.so
```

6.1.6 Building the program for UNO

To build your software to be executed on your PC:

make sure the the variable **ARDUINO** in the Makefile is set to **uno** and type:

By setting ARDUINO to uno the Searduino makefiles will use the settings for building and uploading for the Arduino UNO board.
To build the program, all we have to do now is to type:

```
make clean
make prog
```

To upload and run the program on the Arduino UNO board:

```
make upload
```

You should now be able to see the built in led (pin 13) flash. If not, the author of this document need to his homework.
You can also upload to the board with the safe-upload makefile target. Searduino scans your program for objects known to cause problems on the Arduino boards before uploading to the board.

6.2 Searduino Makefile variables

You can fine tune the compilation and linking of your Arduino programs using the Searduino Makefile variables.

Variable	Description
`USER_C_FLAGS`	Adds the value of the variable to both Arduino and simulation C compilation
`USER_CXX_FLAGS`	Adds the value of the variable to both Arduino and simulation C++ compilation
`USER_LD_FLAGS`	Adds the value of the variable to both Arduino and simulation C/C++ linking
`USER_STUB_C_FLAGS`	Adds the value of the variable to simulation C compilation

USER_STUB_CXX_FLAGS	Adds the value of the variable to simulation C++ compilation
USER_STUB_LD_FLAGS	Adds the value of the variable to simulation linking
USER_ARDUINO_C_FLAGS	Adds the value of the variable to Arduino C compilation
USER_ARDUINO_CXX_FLAGS	Adds the value of the variable to Arduino C++ compilation
USER_ARDUINO_LD_FLAGS	Adds the value of the variable to Arduino linking

6.3 Setting the USB port to use for uploading to Arduino boards

Searduino tries to find the USB port to use automatically for you. In some cases, e g if you have more than one Arduino board attached yo your computer, it is not possible for Searduino to know which port you aim to program.
You can bypass Searduino's method of finding correct device. You do this by the environment variable `USER_PORT`.

6.3.1 Setting USER_PORT in bash

Here's an example of how to instruct Searduino to use `/dev/ttyACM0` for programming.
`export USER_PORT=/dev/ttyACM0`
After this you can invoke make on your makefile as usual.

6.3.2 Setting USER_PORT in a Makefile

Here's an example of how to instruct Searduino to use `/dev/ttyACM0` for programming.
Add the following to your Makefile, before you include the file `searduino.mk`
`USER_PORT=/dev/ttyACM0`
After this you can invoke make on your makefile as usual.

6.4 Checking version of Searduino

In your makefile you can specify the earliest version you want to use. If this requirement is not met, including the file `searduino.mk` will yield an error. Set the variable `REQUESTED_SEARDUINO_VERSION` to the version you (at least) want, e g 0.60:

```
REQUESTED_SEARDUINO_VERSION=0.60
```

6.5 I2C devices

Searduino provies an API to write software to simulate I2C devices.

6.5.1 Registering an I2C device

To register an I2C device you us the function `int seasim_i2c_add_device (unsigned int device_nr, const char *setup_fun)`.

The argument (device_nr) corespond to the device number of the device you want to simulate and the name of the function that setups up your I2C code.
When calling the i2c_add_device function Searduino will find the setup function you provide and call it.

6.5.2 Build the I2C device

To build your I2C code and prepare it to be loaded by the simulator, simply write a Makefile and follow the procedures as in the section “Building a shared library for use in the simulators”.

7 Testing your Arduino code in Searduino

Apart from making it easier to develop code for the Arduino boards, a major goal with Searduino is to be able to test your code (logically) on your local host. This chapter introduce you to how to write test code using the C API for the simulator (in a way you're writing a simulator within your test code).

7.1 Writing C code to test your application

It's easy to test your code using the simulator interface (seasim.h). Below you can find an example on this.

7.1.1 Introducing the code

Assume that we have written some code to give us the distance to an object of some kind. The function is called get_distance (`uint8_t get_distance(void)`) and returns the sum of the analog pin 11 and 12. We want to test this function locally, and not on a real Arduino board, by writing values to the pins 11 and 12 and check of our function returns the correct value.

This is not a rocket science or Nobel prize winning function but it serves well as an example.

7.1.2 Your Arduino code

```
#include <Arduino.h>
#include <searduino.h>

uint8_t get_distance(void)

{
 uint8_t ret = (analogRead(11) + analogRead(12));
 return ret ;

}
```

7.1.3 Your test code

```
#include <Arduino.h>
#include <searduino.h>
#include <seasim.h>


void setup(void)

{
 pinMode(13, OUTPUT);

}



int main(void)
```

```
{
 uint8_t i = 0;
 uint8_t j = 0 ;
 int ctr=0;
 init();

 setup();

 for(i=0;;i++)

{
 /* Set the analog pin 11 & 12,
 which is used by get_distance() */
 seasim_set_generic_input(11, i, INPUT);
 seasim_set_generic_input(12, j, INPUT);

 /*
 printf ("get_distance()=%.4d (i:%.4d j:%.4d ctr:%.4d)\n",
 get_distance(), i, j , ctr++);
 */

 if (get_distance()!=(uint8_t)(i+j))

{
 printf(" ERROR: %d != %d. %d succeeded before this one failed\n",
 get_distance(), (uint8_t)(i+j), ctr);
 return 1;

}
 if (i==255) j++;
 if (j==255) break ;
 ctr++;

}

 printf ("%d tests passed\n", ctr);
 return 0;

}
```

7.1.4 Your Makefile

```
SEARDUINO_PATH=/opt/searduino/
SRC_C=distance.c test_distance.c
SRC_CXX=
ARDUINO=stub
PROG=distance
SHLIB=distance.so
USER_C_FLAGS=-I/opt/searduino/include/searduino/seasim -I/opt/searduino/include/searduino/
-I/opt/searduino/include/searduino/arduino
```

```
include $(SEARDUINO_PATH)/share/searduino/mk/searduino.mk
```

7.1.5 Executing the test

First we need to build the test program.
`make clean prog`

Then simply execute the test program
`./distance`

7.2 Writing Java code to test your application

7.3 Test your application as a stand alone binary

7.4 Test your application in the stream simulator

7.5 Test your application in the Jearduino simulator

8 Build for different targets

With Searduino it's (relatively) easy to compile your program for various boards. You decide what targets to build for with the ARDUINO variable in the Makefile. The following values of that variable are implemented.

Build types

- uno - builds software for the Arduino UNO board
- mega - builds software for the Arduino Mega board
- mega2560 - builds software for the Arduino Mega 2560 board
- due - builds software for the Arduino Mega board
- leonardo - builds software for the Arduino Leonardo board
- stub - builds software for the PC

If your Makefile is actually called Makefile, and not has a suffix like Makefile.something, you can use special make targets: **Special make targets**

- make uno - builds software for the Arduino UNO board
- make mega - builds software for the Arduino Mega board
- make mega2560 - builds software for the Arduino Mega 2560 board
- make due - builds software for the Arduino Mega board
- make leonardo - builds software for the Arduino Leonardo board
- make stub - builds software for the PC
- make uno-upload - uploads software to the Arduino UNO board
- make mega-upload - uploads software to the Arduino Mega board
- make mega2560-upload - uploads software to the Arduino Mega 2560 board
- make due-upload - uploads software to the Arduino Mega board
- make leonardo-upload - uploads software to the Arduino Leonardo board
- make stub-upload - uploadssoftware to the PC

9 Various Searduino functions/macros

Searduino provides a small set of macros for you. When building for your local computer they are enabled, and when building for the Arduino hw they are disabled.

Macro	Description
`SEARDUINO_STUB`	SEARDUINO_STUB is set when compiling for stub (not Arduino boards)
`SEARDUINO_ARDUINO`	SEARDUINO_ARDUINO is set when compiling for Arduino boards
`SEARDUINO_LOOP`	On the Arduino boards this is the same as a for loop (`for(;;)`). When using this macro your code can be paused in the simulator.
`SEARDUINO_DEBUG((msg))`	The macro (if not building for Arduino boards) takes what is inside the parenthesies and passes that to printf. It does a bit more than that, but put simply, that's what it does. Example use: `SEARDUINO_DEBUG(("The variable x is:%d",x));` *Note:* **You must use double paranthesises!**
`searduino_usb_init()`	If building for Arduino boards this macro initialises the USB Device (USBDevice). If building for simulator this macro is not doing anything. Example use: `searduino_usb_init();`
`SEARDUINO_FLUSH_USB`	Flushes the serial buffer. When using USB Devices you must use this every now and theh, typically in once per loop Example use: `SEARDUINO_FLUSH_USB();`

10 Simulators

With Searduino you can easily build your code for use with:

- stub program - the Arduino fuctions print when they are being called (stdout by default)
- stream - same as with stub, but now also with a listening (stdin) thread to which you can send commands (such as settig digital input pin 2 to 1).
- Jearduino - a simulator GUI (Java).
- pardon - simulator interface (Python). Obsoleted and unmaintained.
- Pearduino - a simulator, written in C++/Qt/Qml. Obsoleted and unmaintained.
- python scripts - your own Python scripts, using the Python simulator interface (pearduino)

How to use each of the above is explained below in separate sections.

10.1 Stub

Use this way if you want to run your program stand alone, with no way of giving input to it. As soon as your program sets a pin you will see a printout on stdout.

To build a stand alone stubed program set the following variables in your `Makefile` (before you include `searduino.mk`):

```
ARDUINO=stub
PROG=somename
```

Note: You must NOT have the variable SHLIB set in the Makefile

Check out the example as found in `example/python-digcounter`. Edit the Makefile (`Makefile.digcounter`) and make sure to set the variables as described above. After you're done, type:

```
make -f Makefile.digcounter clean
make -f Makefile.digcounter
```

and possibly also

```
make -f Makefile.digcounter check
```

10.1.1 Stub output syntax

With this mode set Searduino print messages to a stream (default to stdout) for the function calls where some hardware is set.

You switch on and off this mode as many times you want during execution using the functions:

```
void searduino_enable_streamed_output(void)
void searduino_disable_streamed_output(void)
```

Directive	Example	Description
`dpin:<pin>:<value>`	`dpin:1:0`	Digital output pin 1 is 0
`dmode:<pin>:<mode>`	`dpin:1:0`	Mode of digital pin 1 is 0
`apin:<pin>:<value>`	`apin:2:1.123`	Analogue pin 2 is 1.123

10.2 Streamed simulator

With the program searduino-stream-sim you can test your Arduino program and give input data to it using stdin.

10.2.1 Preparing your arduino code for the simulator

First of all you must build your Arduino code as a shared library. To do this you must set the following variables in your `Makefile` (before you include `searduino.mk`):

```
ARDUINO=stub
SHLIB=libyourcode.so
```

Note: You must NOT have the variable PROG set in the Makefile

After this you must do a clean build:

```
make -f Makefile.digcounter clean
make -f Makefile.digcounter
```

10.2.2 Preparing the simulator

Next thing to do is to make sure that your system can find all the shared libraries. Type:

```
export LD_LIBRARY_PATH=/opt/searduino/lib
```

By doing this we tell the system to look for libraries in `/opt/searduino/lib`, which is where we assume you've installed searduino in.

We're now ready to launch the simulator, but let's do a quick check before we proceed. Let's verify that the dynamic loader will find all the libraries needed by pearduino (Searduino's Python library). On GNU/Linux and similar system do:
`ldd /opt/searduino/lib/pearduino.so`

We are, as before, assuming you've installed Searduino in /opt/searduino. ldd (a tool to print out dynamic link dependencies) will print out a list of the libraries pearduino depends on. Make sure that you see no printouts waring you of missing libraries (ldd reports this by saying "not found").

If this went ok, we're finally ready to proceed by invoking pardon.

10.2.3 Launching the simulator

You need to pass the arduino code to load by using command line arguments, here how to do it:
`/opt/searduino/bin/searduino-stream-sim --arduino-code /some/dir/libyourcode.so`

10.2.4 Streamed simulator input syntax

Directive	Example	Description
`dpin:<pin>:<value>`	`dpin:13:1`	Set digital pin 13 to 1
`apin:<pin>:<value>`	`apin:7:1.123`	Set analogue pin 7 to 1.123

10.2.5 using I2C sw devices

To plugin a I2C device to the simulator you add the following to the simulator command line argument:
`--i2c-code /some/dir/youri2c-code.so`

10.2.6 Scripting with bash

TBD

10.2.7 Scripting over the network

TBD

10.3 Jearduino

Jearduino is a GUI frontend for Searduino. It is written in Java and provides all the features of the other simulators as well as some extra.

Jearduino is covered in a separate manual but we will go through how to start Jearduino since we hope that the simulator will be easy enough to understand - if not, please read the Jearduino manual.

10.3.1 Preparing the simulator

The procedure for doing this is the same as described in the section "Streamed simulator"

10.3.2 Launching the simulator

```
/opt/searduino/bin/searduino-jearduino.sh
```

Jearduino accepts some command line switches:
`--board BOARD` - sets the board to use when starting up
`--arduino-code` - sets the code to execute
`--searduino-project` - sets the Searduino project to use
`--build` - build the code when starting up
`--start` - start executing the code asap... if build was ok

10.4 Pardon simulator (obsoleted)

With Searduino you can test your Arduino program in the Python simulator (written in Python using Gtk).

10.4.1 Preparing your arduino code for the simulator

The procedure for doing this is the same as described in the section "Streamed simulator"

10.4.2 Setting up the Python environment

Now, the shared library is ready for use by python. It's almost time to start the simulator. But there's some few more things to do before we're there. First, we must tell Python where to look for the Searduino Python library called Pearduino. Using bash, as most do on GNU/Linux, BSD, cygwin systems, you type:

```
export PYTHONPATH=/opt/searduino/lib
```
assuming you've installed Searduino in */opt/searduino/*.

10.4.3 Preparing the simulator

The procedure for doing this is the same as described in the section “Streamed simulator”

10.4.4 Launching the simulator

```
/opt/searduino/bin/pardon
```

Pardon will ask you to point to the shared library containing the arduino code you want to execute in the simulator. Browse your way to the file and click ok. Now pardon should be executing your binary.

If you want to pass the arduino code to load by using command line arguments, here how to do it:

```
/opt/searduino/bin/pardon --arduino-code /some/dir/libyourcode.so
```

10.5 Python scripts

With Searduino you can write test code for your Arduino program in Python. Searduino comes with a Python library, called pearduino, for this.

Until we’ve written this section, we refer to the example `example/python-digcounter/simple-hw.py`.

10.6 xxx simulator

11 Using Arduino examples

Searduino comes with a program, called arduino-ex2c, the can convert an Arduino example (.ino) to a C file that you compile with Searduino. Searduino also comes with all the Arduino examples.
Let's assume you want to work with the Arduino example called Blink and that you want to create a shared library (for use in the simulators). Do the the following:

```
/opt/searduino/bin/arduino-ex2c --shlib --searduino-path /opt/searduino/ \
 /opt/searduino/share/examples/arduino/1.Basics/Blink/
```

If you want a stand alone program instead, do:

```
/opt/searduino/bin/arduino-ex2c --prog --searduino-path /opt/searduino/ \
 /opt/searduino/share/examples/arduino/1.Basics/Blink/
```

If you want to build for Uno instead, do:

```
/opt/searduino/bin/arduino-ex2c --uno --searduino-path /opt/searduino/ \
 /opt/searduino/share/examples/arduino/1.Basics/Blink/
```

You should now have a directory called Blink. Go to this dorectory and type:

```
make
```

Note: If you only want to create the C file, simply pass the ino file insetad the directoy.

12 Turning your code into a library

In bigger projects it might be useful to put some pieces of code together and turn it into a library. This is typically useful when you like to use some code in various different other projects. This chapter will guide you how to do this.

12.1 Preparing the Makefile

A Makefile for building libraries is somewhat similar to the usual Searduino Makefile as you use when building your normal Arduino programs.

Directive	Example	Description
SEARDUINO_PATH	Should be set yo your Searduino installation root directory.	/opt
SRC_C	The C files in the library you want to build	dir-a/filea.c filec.c
SRC_CXX	The C++ files in the library you want to build	dir-a/filea.cpp filec.cpp
H_FILES	The header files to install. Directory structure is kept.	dir-a/monkey.h donkey.h
BOARDS_TO_BUILD	The boards you want to build for. Defaults to all supported boards.	uno leonardo
PACKAGE	A name for your library.	tvout
USER_C_FLAGS	Flags added to the compiler when compiling C files	-Imydir
USER_CXX_FLAGS	Flags added to the compiler when compiling C++ files	-Imydir
USER_LD_FLAGS	Flags added to the linker when linking the build C/C++ files	-lc

And finally, you need to include some Makefile stuff:

```
include $(SEARDUINO_PATH)/share/searduino//mk/library-functions.mk
```

12.2 Building and installing the library

This is really easy, assuming the code itself is ok.

To build/compile you simply type:

```
make
```

To install you simply type:

```
make install
```

12.3 Using the library

To use the library and heades in another Searduino project you only need to do the following:

```
USER_C_FLAGS=-I<path>
USER_CXX_FLAGS=-I<path>
USER_LD_FLAGS=-L<path>
```

And of course the path should be set to where the headers and libs of your new lib were installed.

13 Debugging Arduino code

14 Examples

14.1 Example Makefile

```
SEARDUINO_PATH=/opt/searduino
PROG=blinker
SRC_C=main.c
SRC_CXX=
SHLIB=blinker.so
ARDUINO=stub
include $(SEARDUINO_PATH)/share/searduino/mk/searduino.mk
```

14.2 Example C code

```
#include <Arduino.h>
#include <searduino.h>

void setup()

{
 pinMode(13, OUTPUT);

}


int main(void)

{
 init();

 setup();

 for(;;)

{
 digitalWrite(13, 1);
 delay(100);
 digitalWrite(13, 0);
 delay(100);

}

 return 0;
```

```
}
```

14.3 Building the example

14.3.1 Building a stand alone program for host system

To build and execute the code above for your host system, all you have to do is type:
`make`
And to run the program, you simply have to invoke it.
`./blinker`
When the program is executing you can see printouts from the Arduino functions as implemented by Searduino.

14.3.2 Building the code for use in the simulators

To build and execute the code above for your host system and to be run in a simulator you should type:
`make shlib`
And to run the program in the stream simulator, you type
`/opt/searduino/bin/searduino-stream-sim --arduino-code ./arduino-code.so`
When the program is executing you can see printouts from the Arduino functions as implemented by Searduino. You can also set input values for your arduino code to read. Read more about the syntax in a separate chapter.
If you want to run your code in the GUI simulator, type:
`/opt/searduino/bin/searduino-pardon.sh --arduino-code ./arduino-code.so`

14.3.3 Building the code for the Arduino boards

To build and execute the code above for the Arduino hosts, UNO in this example, you have to adjust the makefile a bit.
Set the `ARDUINO` variable to `uno`, assuming you want to build for the Arduino UNO board. The next thing to do is to type:
`make clean make prog`
And to run the upload the program to the board, type
`make upload`
You can also upload to the board with the safe-upload makefile target. Searduino scans your program for objects known to cause problems on the Arduino boards before uploading to the board.

15 Write your own simulator

15.1 Writing a simulator in C/C++

Writing a GUI frontend in C/C++ on top of Searduino is pretty straight forward. We suggest you look into the source code of the streamed simulator (a command line simulator which is part of Searduino): `http://git.savannah.gnu.org/cgit/searduino.git/tree/simulators/stream`

All functionality offered by Searduino can be found in the header file called seasim.h: `http://git.savannah.gnu.org/cgit/searduino.git/tree/faked-arduino/include/seasim/seasim.h`

15.2 Writing local tests in Python

For now, we would like to refer to pardon in the Searduino source treee for en example on how to write a simulator in Python.

16 Future and possible enhancements

It should be possible to add more compiled languages to use to program the Arduino boards as long as there's a backend compatible with the C backend of the avr-gcc. It will not (easily t least) be possible to add byte compiled or interpreted lagnuages, so Java, Perl, Python and friends are out of the questions. At least for the moment.

Contact us if you want support for your favorite lagnuage.

17 FAQ

What does undefined reference mean?

In short it means that the linker can't find all the needed "function implementation" for your binary/library. If you're compiling your Arduino code for local execution it means you're using a function that has not yet been implemented in Searduino.

You can ask the Searduino folks to do it or you can implement it your self. the Arduino board?

Sim code works, but not on the Arduino?

One of the thing we've experienced when developing Searduino is that Arduino crashes when you're having printf and similar calls in your code. Instead of using printf in your code you can use the debug macros (e.g SEARDUINO_DEBUG) in Searduino.