
biicode docs Documentation

Release 3.0.2

biicode

July 12, 2015

1	Biicode	3
1.1	Installation	3
1.2	Getting started	8
1.3	Dependencies	12
1.4	Publishing	16
1.5	Custom build configuration	17
1.6	Adapt your library	24
1.7	Advanced Usage	30
1.8	Examples	44
1.9	Integrations	191
1.10	Reference	205
1.11	Release notes	230
1.12	FAQs	242
1.13	Troubleshooting	243
2	Arduino	247
2.1	Installation	247
2.2	Getting started	250
2.3	Arduino commands	253
2.4	How to	256
2.5	Examples	265
2.6	Troubleshooting	273
3	Raspberry Pi Cross Compilation	275
3.1	Installation	275
3.2	Getting started	277
3.3	RPi commands	280
3.4	How to	283
3.5	Examples	284
3.6	Troubleshooting	303

4	Node.js	305
4.1	Getting started	305
4.2	How to	307

Here's all the information to get started using biicode with your projects:

- **User Guide:** For C and C++ programmers. Guides to get started, learn how to use the libraries available in biicode, manage your projects with your usual IDE's, integrate with other tools, publish blocks, control your dependencies. There's also a full reference guide.
- **Arduino Docs:** specifics to develop C and C++ projects with Arduino and biicode. Includes a getting started guide.
- **Raspberry Pi Docs:** specifics to C and C++ cross-compiling and native development for Raspberry Pi boards using biicode.
- **Node.js Docs:** specific tools, commands and examples to develop Node.js projects with biicode.

Biicode

A multi-platform C and C++ dependency manager.

1.1 Installation

Biicode is a file-oriented Dependencies Manager for C and C++ developers. Install both biicode and the C/C++ tools to get started.

1.1.1 Install Biicode

Download [Biicode Installer](#) and double-click the downloaded package. Open the terminal and **make sure biicode is installed**:

```
~$ bii --version
```

Check alternative installations for:

- *Debian based distributions*
- *Arch based distributions*
- *Running biicode from source*

1.1.2 Install C/C++ tools

Then install required tools like CMake and MinGW or GCC:

```
~$ bii setup:cpp
```

If any problem installing C/C++ tools, check *[how to install C/C++ tools manually](#)*.

The setup command installs programs in a directory called `.biicode` in your home directory.

Execute again to make sure the tools are installed:

```
~$ bii setup:cpp
CMake 3.0.2 already installed
gcc 4.8.2 already installed
g++ 4.8.2 already installed
```

You're now ready to *get started*.

1.1.3 Debian based distributions

Alternative install for Debian based distributions (Ubuntu, Raspbian)

Use the `apt-get` program to install biicode through the APT repository:

Quick install:

```
wget http://apt.biicode.com/install.sh && chmod +x install.sh && ./install.sh
```

Execute **bii setup:cpp** to make sure you've got all tools required.

Step by step install:

```
# 1. Create a file named '/etc/apt/sources.list.d/biicode.list' and put the li

Ubuntu 12:
    deb http://apt.biicode.com precise main

Ubuntu 13:
    deb http://apt.biicode.com saucy main

Ubuntu 14:
    deb http://apt.biicode.com trusty main

Debian Wheezy:
    deb http://apt.biicode.com wheezy main

# 2. Add our public key executing:
sudo wget -O /etc/apt/trusted.gpg.d/biicode.gpg http://apt.biicode.com/keyring

# 3. Execute apt-get update:
sudo apt-get update

# 4. Execute apt-get install:
sudo apt-get -y install biicode

# 5. Execute bii setup:cpp to make sure you've got all tools required.
bii setup:cpp
```


1.1.4 Arch based distributions

Alternative install for Archlinux based distributions (Manjaro, Arch Linux ARM, etc)

Biicode maintains a package at the Arch User Repository (AUR). Install it using your preferred package manager:

```
$ sudo yaourt -S biicode
```

The package is maintained in the AUR, so your package manager will notify you automatically when we update the package.

1.1.5 Run biicode from source

The most flexible way to make a package for your distro is running biicode from source. Also, if you are developing biicode, testing new feature or helping to resolve a bug, you may need to run biicode directly from source.

Follow this guide at GitHub to [run biicode from source](#).

1.1.6 Install C/C++ tools manually

Install, set up and verify some **tools to build C and C++ projects with biicode**.

Follow these steps if something failed during the automatic installation explained before. If you experience any issues, please [contact us at our forum](#), we'll try to solve your problem as soon as possible. Linux

Install the required development tools as root:

```
$ sudo apt-get install build-essential cmake
```

That's all! MacOS

You need to get installed both XCode Developer Tools and CMake:

1. The XCode Developer Tools

```
$ xcode-select --install
```

2. Download and install the appropriate [version of CMake](#) for your Mac OSX.

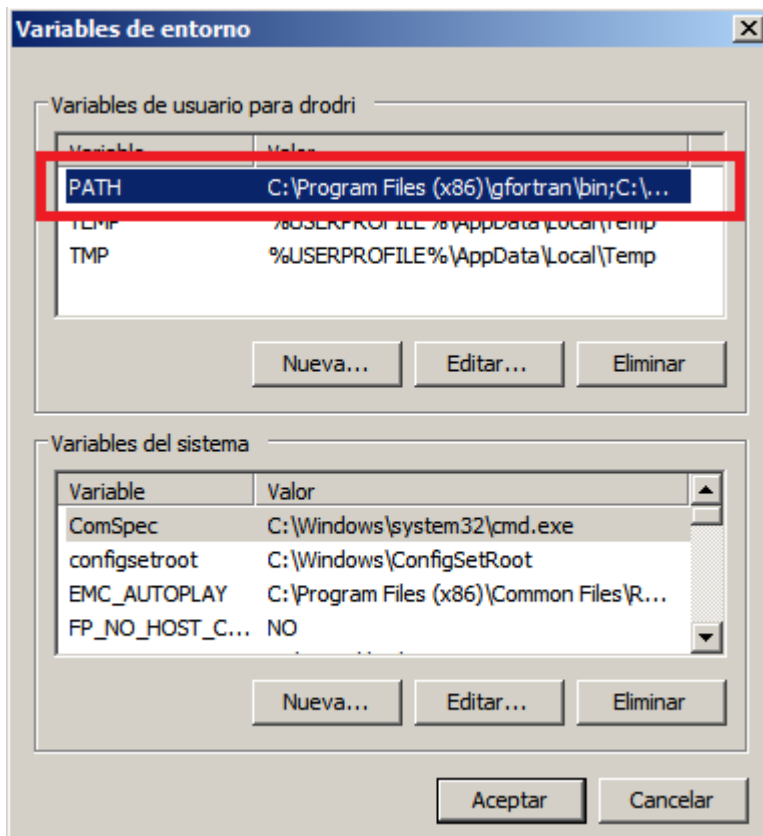
Windows

To develop C/C++ programs in Windows you need:

- **CMake**. Open Source tool that manages the software building process in a compiler-independent manner.
- Compilers and build system. This could be one of the following (among other alternatives):
 - **MinGW** (make sure to include gcc, g++, and mingw32-make with your installation)
 - Visual Studio C++

These are the **steps for manual installation** of our recommended tools:

1. Download and install CMake. You can [download the latest version of CMake here](#).
2. Download and install “base, g++” packages of MinGW. Follow [this link](#) to get the installer, and choose while installing two additional packages, GCC and G++ package.
3. Add to your user PATH environment variable the paths to these tools. We recommend [Rapid Environment Editor](#) for editing environment variables. Otherwise, go to **My Computer**, click **Properties**, click **Advanced System Settings** and in the System Properties window click the **Environment Variables** button. then you will see a new window and in **User Variables** you’ll find the variable PATH:



Add your tools binaries folders (i.e. C:\MinGW\bin for MiGW, and C:\Program Files (x86)\CMake\bin for CMake).

You might need to close and open again any cmd windows in order to load the new value for the PATH variable.

Verify your installation

To check your automatic installation open the Terminal and type **bii setup:cpp**. To check your manual installation, run the following commands. If the output messages look similar to these, the tools are successfully installed.

```
$ cmake --version
cmake version [version]
```

```
$ gcc --version
gcc (GCC) [version]
...
```

```
$ g++ --version
g++ (GCC) [version]
...
```

```
$ mingw32-make --version
GNU Make [version]
...
```

1.1.7 Connect through a proxy server

Set an environment variable “HTTPS_PROXY” with your proxy server address.

Linux/OSx:

```
$ export HTTPS_PROXY="http://user:pass@proxy_ip:port"
```

You need to export this variable whenever you open a new shell. Append previous line on ~/.bashrc file and it will be executed when a shell is opened.

Windows:

1. From the Desktop, right-click the very bottom left corner of the screen to get the Power User Task Menu.
2. From the Power User Task Menu, click System.
3. Click the Advanced System Settings link in the left column.
4. In the System Properties window, click on the Advanced tab, and then click the Environment Variables button near the bottom of that tab.
5. In the Environment Variables window, click “New” and add variable name HTTPS_PROXY and value “http://user:pass@proxy_ip:port”

Example: If my proxy is on localhost (port 7775) and my user is “lasote” with password “mypp”:

```
$ export HTTPS_PROXY="http://lasote:mypp@localhost:7775"
```

If you have any questions, we are available at [. You can also for suggestions and feedback.](#)

1.2 Getting started

Welcome to biicode.

Biicode manages your project's dependencies so you can use the libs you need (Curl, Catch, Fann, OpenSSL, OpenCV, Poco, Boost, Libuv, GTest ...) as you wish within your project. Biicode uses CMake to configure and build your projects and it is compatible with many IDEs and version control systems.

This guide will help you get your first biicode block off the ground. Let's run a block with a unit test using the in biicode. There's no need to install GTest, biicode downloads and configures it for you, is already in biicode!

1.2.1 Basics

Make sure *biicode is installed*. Open the terminal and execute **bii setup:cpp** to ensure you've got all tools required.

1.2.2 Create your first project

bii init -L a folder to create a new project. The commands below create *unit_test/* folder as a biicode project containing a block and a *main.cpp* file.

```
~$ bii init unit_test -L
~$ cd unit_test
~/unit_test$ echo "//main.cpp code goes here" >> main.cpp
```

Place this code into the *main.cpp* file:

```
#include "google/gtest/include/gtest/gtest.h"

int sum(int a, int b) {return a+b;}

TEST(Sum, Normal) {
    EXPECT_EQ(5, sum(2, 3));
}

int main(int argc, char **argv) {
    testing::InitGoogleTest(&argc, argv);
```

```
return RUN_ALL_TESTS();
}
```

It is just a sum function and a test using Google Test framework.

The `#include` is composed as `#include <biicode_user>/<block_name>/path/to/file`. In this case, it *#includes* the `include/gtest/gtest.h` header from block and user **google**.

In the web, you see its latest version is **10 tagged STABLE**:

The screenshot shows the biicode web interface. On the left, a 'Code Browser' sidebar displays a file tree with 'include/gtest/gtest.h' selected. The main area shows the 'google/gtest' block details, including its version (10), stability (STABLE), and description ('gtest 1.7.0 Google C++ Testing Framework'). Below this, the 'Block requirements', 'Block dependencies', and 'Dependencies graph' tabs are visible. The 'Raw' tab is selected, displaying the source code of 'google/gtest/include/gtest/gtest.h'. A blue box highlights the file path in the sidebar and the file name in the 'Raw' tab's header.

Retrieve the dependency:

```
~/unit_test$ bii find
...

INFO: Analyzing compatibility for found dependencies...
INFO: All dependencies resolved
Find resolved new dependencies:
  google/gtest: 10
INFO: Saving files from: google/gtest
```

bii find creates a *biicode.conf* file and downloads GoogleTest block into your *bii/deps* folder:

```
unit_test/
  -- bii/
```

```
|   -- deps/  
|   |   -- google/  
|   |       -- gtest/  
-- biicode.conf  
-- main.cpp
```

[optional] Keeping #includes short

Keep reading to see how to keep your #includes as usual. You can also *skip this section*.

```
#include "gtest/gtest.h"
```

Instead of using long *#includes*, you can write the specs to retrieve this dependency in your *biicode.conf*.

- Split the long `#include "google/gtest/include/gtest/gtest.h"` in two halves:

```
[requirements]  
  google/gtest: 10  
  
[includes]  
  gtest/gtest.h: google/gtest/include
```

You can also use patterns:

```
[includes]  
  gtest/*.h: google/gtest/include
```

1.2.3 Using an IDE

biicode configures your default settings to no IDE and MinGW (Windows) or UNIX Makefiles (MacOS and Linux). You can change these values executing **bii configure**:

```
~/unit_test$ bii configure -G "Visual Studio 10"
```

Here's more about *configuring your IDE*.

1.2.4 Build and run

Build and run your Unit Test, check it works:

```
~/unit_test$ bii build  
...  
~/unit_test$ bin/user_unit_test_main
```

```
[=====] Running 1 test from 1 test case.
...
[-----] Global test environment tear-down
[=====] 1 test from 1 test case ran. (15 ms total)
[ PASSED ] 1 test.
```

Linux/Mac users might run as:

```
~/unit_test$ ./bin/user_unit_test_main
```

That's it, that output means Google Test was downloaded, configured and built in your project!

```
unit_test/
  -- bii/
  -- biicode.conf
  -- bin
  |   -- user_unit_test_main
  -- CMakeLists.txt
  -- main.cpp
```

Congrats! You have just used GoogleTest within your project. You know that we are available at for any problems. You can also for suggestions and feedback.

1.2.5 Publishing

Publish to make your libs available in biicode.

- Execute **\$ bii user your_username**.

```
~/unit_test$ bii publish

INFO: *****
INFO: ***** Publishing public *****
INFO: *****
INFO: Successfully published your_username/unit_test: 0
```

Go to your profile at **www.biicode.com/your_username** to check what you've just uploaded.


```
~$ mkdir deps_example
~$ cd deps_example
~/deps_example$ bii init -L
~/deps_example$ echo "//main.cpp code goes here" >> main.cpp
```

- Write your source code as usual in your *main.cpp*:

```
#include <stdio.h>
#include <string.h>
#include "openssl/md5.h"

int main()
{
    unsigned char digest[MD5_DIGEST_LENGTH];
    char string[] = "happy";

    MD5((unsigned char*)&string, strlen(string), (unsigned char*)&digest);

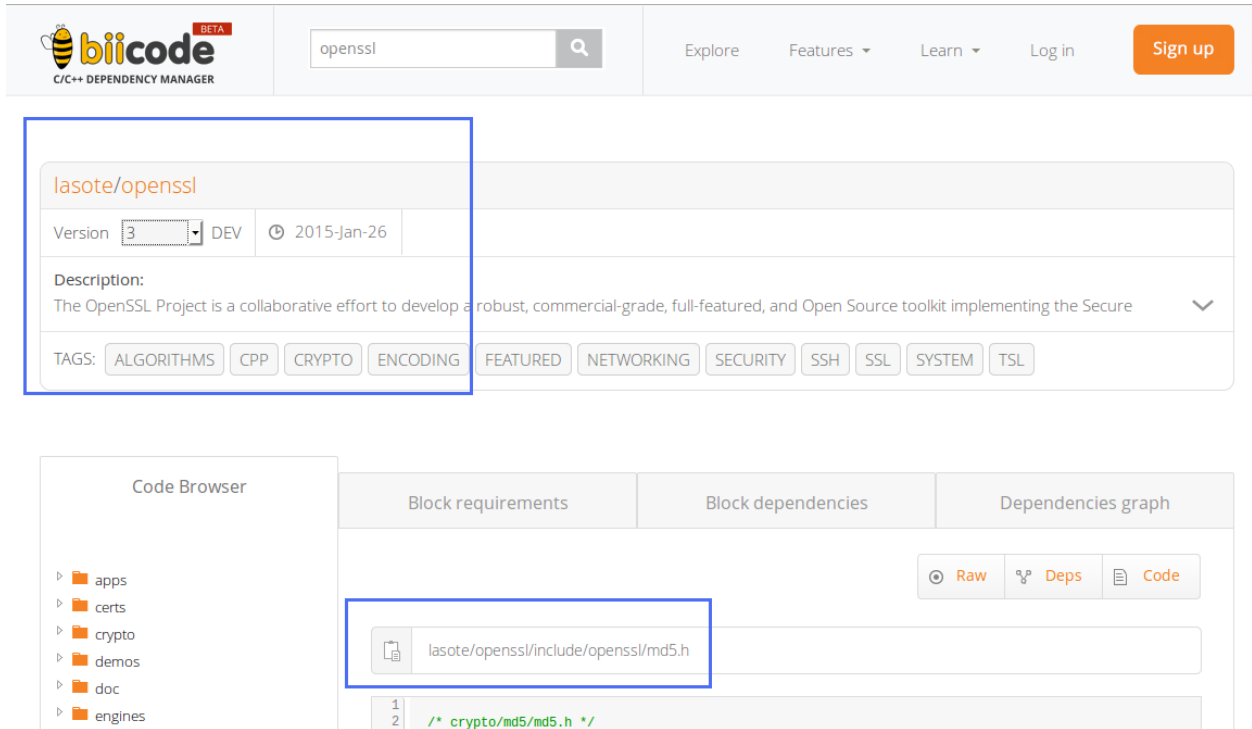
    char mdString[33];

    for(int i = 0; i < 16; i++)
        sprintf(&mdString[i*2], "%02x", (unsigned int)digest[i]);

    printf("md5 digest: %s\n", mdString);

    return 0;
}
```

- Search the library you want in biicode, and specify which lib you're using in your *biicode.conf* file.



Let's first use OpenSSL 1.0.1 available in *lasote/openssl* version 0:

- Write in your *biicode.conf* file:

```
[requirements]
    lasote/openssl: 0

[includes]
    openssl/md5.h: lasote/openssl/include
```

- **bii build** and you're done.

Here's more on this [OpenSSL example project](#).

1.3.2 Modifying the version you depend on

Manually edit your **biicode.conf** file to depend on any version you want.

To depend on *lasote/openssl* version 2, write in your *biicode.conf*:

```
[requirements]
    lasote/openssl: 2

[includes]
    openssl/md5.h: lasote/openssl/include
```

Update your *biicode.conf* file to depend on *lasote/OpenSSL* tagged version 1.0.1l:

```
[requirements]
  lasote/openssl: @1.0.11

[includes]
  openssl/md5.h: lasote/openssl/include
```

Run **bii build** and you'll see the new dependencies in your *bii/deps* folder.

For OpenSSL, there are two tracks available:

- OpenSSL 1.0.1 is available at *lasote/openssl* versions 0,1,2 and 3.
- OpenSSL 1.0.2 is available at *lasote/openssl(v1.0.2)* versions 0 and 1.

Update to release 1.0.2, just write it in your *biicode.conf*:

```
[requirements]
  lasote/openssl(v1.0.2): 0

[includes]
  openssl/md5.h: lasote/openssl/include
```

Execute **bii build** and you'll see the new dependencies in your *bii/deps* folder.

Depending on a block track

Currently, **libuv** keeps four maintained versions or **block tracks**:

- (Stable, used by Nodejs)
- (Non stable, but commonly used)
-
- (Latest)

Depend on one or another to fit your needs:

- Write this *#include line* in your source code:

```
#include "uv.h"
```

- And depend on , write in your *biicode.conf* file `[requirements]` :

```
[requirements]
  lasote/libuv(v0.11): 1

[includes]
  uv.h : lasote/libuv/include
```

- Execute **bii build** and you're ready to go.

Let's switch to :

- Modify [requirements] section in your *biicode.conf* :

```
[requirements]
    lasote/libuv(v1.0): 0

[includes]
    uv.h : lasote/libuv/include
```

- Execute **bii build** and it's switched.

And now, switch to :

- Modify [requirements] section in your *biicode.conf* :

```
[requirements]
    lasote/libuv(v0.10): 1

[includes]
    uv.h : lasote/libuv/include
```

- **bii build** and it's switched.

At last, switch to :

- Modify [requirements] section in your *biicode.conf* :

```
[requirements]
    lasote/libuv(v1.x): 8

[includes]
    uv.h : lasote/libuv/include
```

- **bii build** and it's switched.

Got any doubts? or .

1.4 Publishing

bii publish command publishes your code to biicode.

```
$ bii publish
```

Tag's default value is DEV.

```
$ bii publish --tag STABLE
```

STABLE versions are frozen after publication and DEV versions are overridden by a new version. Each time you publish to biicode your local [parent] value updates to the latest one you just published.

Let's understand this behavior with an example:

- Publish your first **DEV** version.

Its [parent] section should be **empty** or with its first value like this:

```
[parent]
  user_name/block_name: -1
```

Once published, your [parent] updates to version 0 and that's also number 0 in biicode.

- Publish a DEV version. That's still number 0.
- Publish a STABLE version. That's still number 0.
- Publish another STABLE version. That's number 1 in biicode.

1.4.1 Tag a version

Put a semantic name to your block versions. Once tagged, you can *depend on a version just knowing its semantic tag*. Just execute:

```
$ bii publish --tag STABLE --version tag=1.2rc3
```

DEV versions can not be tagged.

1.4.2 Private blocks

Upgrade your account to Premium, , to use Private blocks. Store your code in private, choose who can see or edit your blocks.

Create private blocks in our web page. Just press **Add block button** and choose private.

Got any doubts? or .

1.5 Custom build configuration

Biicode knows how the source code files connect to each other, with this information, biicode constructs a base **CMake** layout to build your project automatically.

However, this automatic process is just a feature, **you can have full control over the building process**.

Here, you'll learn how to define your **CMakeLists.txt** to delve into full functionality.

1.5.1 CMake basics

If you're a **CMake** newbie, these are the tips you need to know before understanding how biicode uses CMake.

So CMake:

- Basically requires one thing, a *CMakeLists.txt* file.
- Allows defining targets which are each **executable** or **library** you want to build for your project.
- Is **multi-platform** and automatically generates the files your O.S./compiler needs to build your project.
- *CMakeLists.txt* basic example (without biicode):

```
# Specify the CMake version used (biicode makes this automatically)
cmake_minimum_required(VERSION 2.6)
# Name your project here (biicode names it automatically)
project(fibonacci)

# This tells CMake to build a library with other.c and file.c and name it mylib
# (biicode automatically creates a library for each block)
add_library(mylib other.c file.c)

# Sends the -std=c99 flag to the gcc compiler
add_definitions(-std=c99)

# This tells CMake to build an executable with fib.c and name it fibonacci
# (biicode automatically adds detected targets in your source code)
add_executable(fibonacci fib.c)

# Links mylib to the fibonacci executable
# (biicode automatically links block's library to each executable)
target_link_libraries(fibonacci PUBLIC mylib)
```

Got any doubts? or .

1.5.2 Where is biicode's “magic”?

bii configure or **bii build** commands generate a *CMakeLists.txt* file in each block.

CMakeLists.txt has 1 line by default (stripping out comments):

```
ADD_BII_TARGETS ()
```

So biicode:

- Detects how source code files are connected and builds a dependency graph (following `#includes` and searching for implementations).
- Generates, for each block, a *CMakeLists.txt* defining some variables according to the dependency graph detected. These variables allow Biicode to “plug” your library to its cloud so it is easily reused.
- Builds a **library** for each block you have in your project (including each dependency).
- Builds an **executable** for each file with a `main()` function.
- Link the block’s **library** to all **executables** within the block.
- Builds the block’s library **only** with the source code files needed, according to the dependencies detected (how files are connected).

1.5.3 Define and prepare targets

In the *CMakeLists.txt* file, before `ADD_BII_TARGETS()` you can adjust:

Which source code files are part of the block’s library

`BII_LIB_SRC` contains all the source code biicode adds to the library (through *CMake add_library*)

EXAMPLE: Exclude *my_file.cpp* from being compiled in the block’s library.

```
# Remove my_file.cpp from being compiled in library
LIST(REMOVE_ITEM BII_LIB_SRC my_file.cpp)

ADD_BII_TARGETS()
```

EXAMPLE: Include *other_file.cpp* to be compiled in the block library.

```
# Include my_file.cpp to be compiled in library
LIST(APPEND BII_LIB_SRC other_file.cpp)

ADD_BII_TARGETS()
```

ESSENTIAL TIP: If biicode did not add a file needed to your block’s library, you could specify this dependency in the `[dependencies]` section of the *biicode.conf* file. *Why?* This way you wire the dependency (fixing the dependency graph). If someone depends on your library, biicode will also download the missing file and will add it to `BII_LIB_SRC` automatically. Otherwise file won’t be downloaded and their build will fail.

Choose **STATIC** or **SHARED** library

BII_LIB_TYPE value is empty by default (turns out it is **STATIC** in most cases). It can be either **STATIC** or **SHARED**.

EXAMPLE: Making a *shared* library (.dll, .so):

```
SET (BII_LIB_TYPE SHARED)

ADD_BII_TARGETS ()
```

Modify which executable targets are made

BII_BLOCK_EXES List of executable targets (mains). Each “exe” matches this pattern: *path_to_mainfile*.

For example, if the block *lasote/game* has a *main.cpp* in a folder named “src”, the variable has an element: “src_main”

EXAMPLE: Prevent biicode from creating an EXE target for (*examples/src/my_program.cpp*):

```
LIST (REMOVE_ITEM BII_BLOCK_EXES examples_src_my_program)

ADD_BII_TARGETS ()
```

You can also do this by adding `!examples_src_my_program` line to `[mains]` section of *biicode.conf*.

Which source code files are part of each executable

BII_exe_name_SRC contains all source code that will be added to the exe. “exe_name” matches this pattern: *path_to_mainfile*.

For example, if the block *lasote/game* has a *main.cpp* in a folder named “src” the variable is: “BII_src_main_SRC”

EXAMPLE: Exclude *my_file.cpp* from being compiled with *examples/main.cpp* executable.

```
LIST (REMOVE_ITEM BII_examples_main_SRC my_file.cpp)

ADD_BII_TARGETS ()
```


Modify which test targets are made

BII_BLOCK_TESTS is a subset of BII_BLOCK_EXES and contains the executables specified in [tests] section of *biicode.conf* file.

EXAMPLE: Exclude *tests/one.cpp* from tests.

```
LIST(REMOVE_ITEM BII_BLOCK_TESTS tests_one)

ADD_BII_TARGETS()
```

1.5.4 Configure targets

Once we have selected which files belong to each target and the targets we want, we are ready to call `ADD_BII_TARGETS()`.

`ADD_BII_TARGETS()` generates the block's **library** target and a target for each **executable**.

Configure library target

BII_LIB_TARGET contains the name of the block's library target. This target may be an INTERFACE target (no source files) if BII_LIB_SRC is empty before `ADD_BIICODE_TARGETS` call. For this reason we recommend you to always use BII_BLOCK_TARGET.

BII_BLOCK_TARGET: Use this better, instead of BII_LIB_TARGET. Created to ease target configuration. It always exists and it is always a CMake **Interface**. Represents the whole block, and it is applied to BII_LIB_TARGET and *each target executable*.

EXAMPLE: Linking with pthread.

```
# Link against the always existing BII_BLOCK_TARGET
TARGET_LINK_LIBRARIES(${BII_BLOCK_TARGET} INTERFACE pthread)
# or link against the library (if it's not an interface we specify PUBLIC attr
TARGET_LINK_LIBRARIES(${BII_LIB_TARGET} PUBLIC pthread)
```

You can also do this by adding pthread to `${BII_LIB_DEPS}` before calling `ADD_BII_TARGETS()`

EXAMPLE: Adding include directories to all targets of this block.

```
TARGET_INCLUDE_DIRECTORIES(${BII_BLOCK_TARGET} INTERFACE myincludedir)

# You can also add private include directories to the Lib (if existing)
TARGET_INCLUDE_DIRECTORIES(${BII_LIB_TARGET} PRIVATE myincludedir)
```

You can also do this by adding `myincludedir` line to `[paths]` *section of biicode.conf*.

EXAMPLE: How to activate C++11 for all targets (including lib target).

```
IF (APPLE)
    TARGET_COMPILE_OPTIONS (${BII_BLOCK_TARGET} INTERFACE "-std=c++11 -stdlib=li
ELSEIF (WIN32 OR UNIX)
    TARGET_COMPILE_OPTIONS (${BII_BLOCK_TARGET} INTERFACE "-std=c++11")
ENDIF (APPLE)
```

EXAMPLE: Adding compile definitions to all targets (including lib target).

```
TARGET_COMPILE_DEFINITIONS (${BII_BLOCK_TARGET} PUBLIC "MY_DEFINITION=1")
```

EXAMPLE: Setting properties to lib target.

```
SET_TARGET_PROPERTIES (${BII_LIB_TARGET} PROPERTIES COMPILE_DEFINITIONS "IOV_MA
```

`SET_TARGET_PROPERTIES` only allows setting some white-listed properties to `BII_BLOCK_TARGET`, because it is an interface. Use `BII_LIB_TARGET` to set target properties.

Configure executable target

`BII_exe_name_TARGET` contains the name of the target for each executable target. Each “exe” matches this pattern: *path_to_mainfile*.

EXAMPLE: Linking pthread to an executable target (file: *examples/one.cpp*):

```
TARGET_LINK_LIBRARIES (${BII_examples_one_TARGET} PUBLIC pthread)
```

EXAMPLE: Adding compile definitions to an executable target (file: *my_main.cpp*).

```
TARGET_COMPILE_DEFINITIONS (${BII_my_main_TARGET} PUBLIC "MY_DEFINITION=1")
```

When someone depends on your library, biicode only downloads the required files (according to the dependency graph). So you can not assume that `${BII_my_main_TARGET}` target will exist. It may seem obvious, but if you reference a target that doesn’t exist build fails. When possible it’s better to not act upon EXE targets. Remember that `BII_BLOCK_TARGET` will be applied to each target in your block.

It is best to act upon `BII_BLOCK_TARGET`.

1.5.5 Select build type: Debug or Release

You can set the build type with *-D option* in **bii configure** command:

```
$ bii configure -DCMAKE_BUILD_TYPE=DEBUG
$ bii build
```

Possible values are: **DEBUG**, **RELEASE**, **RELWITHDEBINFO**, **MINSIZEREL**

Check official docs from .

If you are using *Visual Studio* or any other IDE with a select list box for build type use:

```
$ bii build --config=DEBUG
```

Use **bii clean** command to restore most of your project's meta-information. Here's more about *bii clean command*.

1.5.6 Complete variable reference

Sorted according to their specific use **before** or **after** `ADD_BII_TARGETS ()` variable:

BII_LIB_SRC List of files belonging to the library.

BII_LIB_TYPE Empty by default, (STATIC in most cases) STATIC or SHARED.

BII_LIB_DEPS Dependencies to other libraries (user2_block2, user3_blockX).

BII_LIB_SYSTEM_HEADERS System linking requirements as windows.h, pthread.h, etc.

BII_LIB_INCLUDE_PATHS List of directories that the library target will include through a call to `TARGET_INCLUDE_DIRECTORIES`

BII_BLOCK_EXES List of targets that represent the executables (mains) defined in this block. If you want to prevent biicode from creating an EXE target, first remove it from this list.

BII_exe_name_SRC List of files belonging to an "exe". "exe_name" matches this pattern: *path_to_mainfile*. For example, if the block *lasote/game* has a *main.cpp* in a folder named "src" the variable will be: `BII_src_main_SRC`

BII_exe_name_DEPS Dependencies of this "exe" target to other libraries, including its own block library if any (user2_block2, user3_blockX).

BII_BLOCK_TESTS List of executables specified in `[tests]` section of *biicode.conf* file. Will be excluded from **bii build** compilation and compiled with **bii test** command. `add_test`

```
ADD_BII_TARGETS ()
```

BII_LIB_TARGET Target library name, usually in the form “user_block”. It may not exist if **BII_LIB_SRC** is empty, so better use `${BII_BLOCK_TARGET}` as a general rule.

BII_BLOCK_TARGET CMake **Interface** that represents the whole block. It always exists and it’s applied both library and executables (each target). You can use it to configure a block’s building configuration: Link libraries, compile flags...etc

BII_BLOCK_TARGETS List of all targets defined in the block

BII_exe_name_TARGET Executable target (listed in `${BII_BLOCK_EXES}`). e.g. `${BII_main_TARGET}`. You can also use directly the name of the executable target (e.g. user_block_main)

Got any doubts? or .

1.6 Adapt your library

Transforming your **library** in a full functional biicode **block** can be straightforward or require some work. The bigger or “heavier” a library is, the higher time it takes to adapt it.

We assume you’ve read *Custom build configuration* section and have understood how biicode builds your code.

1.6.1 Concepts to understand

- Place your library’s source code in a biicode block.
- Biicode **analyzes** your code and builds a **dependency graph** with how each file connects to the others. These files are appended to **BII_LIB_SRC** variable in your *CMakeLists.txt* file.
- When you *#include* a header (ex: file.h) from a remote block, biicode only downloads the files that depend on “file.h” (recursively) and builds a **library** with the files needed. The dependency library built is linked to your targets automatically.
- When you’re the one uploading a “reusable” library, it’s really important that the dependency graph for that lib is built correctly.
- A quick way to be sure that your library is fully reusable, is publishing with DEV tag and then depend on it from another project making an example. The example can be a main including a header from your own library. You can check a lot of examples reusing libs in .

Key facts

As biicode may build the libraries with just a few files from the whole library (biicode only downloads and builds the needed files), you shouldn’t assume in your *CMakeLists.txt* that all your library

files will be present.

> **Example:** Make sure an *exe target* exists before executing `TARGET_LINK_LIBRARIES` upon it.

> **Example:** Adding *my_file.cpp* to your library explicitly isn't recommended as you don't know if biicode has downloaded this file.

Biicode needs a library in `BII_LIB_TARGET` variable to make it reusable, as a "plug".

It builds `${BII_LIB_TARGET}` for each block with the source code files in `BII_LIB_SRC` variable (list).

1.6.2 Without a previous CMakeLists.txt

If your current library doesn't have a *CMakeLists.txt* biicode creates it when you execute **bii configure** or **bii build**.

1. Look for unresolved dependencies with bii deps

- If **some of your header** files (*.h) are *unresolved*, biicode has not been able to detect them. You can solve this by filling *[paths]* section in *biicode.conf* with the folders containing the headers to let biicode find them.

You only need to specify your paths when your project has non-file-relative `#include (s)`.

For example:

```
[paths]
# Local directories to look for headers in your block
include
/
```

- If there are references to **external headers**, look for the library you need in biicode. You can use the search engine in <https://www.biicode.com> and search for the file typing `file:my_include.h`
 - Found the library in biicode? Just fill your *[requirements]* of *biicode.conf* as shown in *dependencies* section. Re-run **bii deps** command to ensure the `#includes` are resolved.
 - Didn't find the library in biicode? You could be the first one adding it ;)
- If there are no unresolved dependencies or it seems your unresolved dependencies are **system dependencies**, try to compile it (point 2).

2. Execute bii build

- There are **compilation errors**:
 - Check if some compile definition is needed. You can use `TARGET_COMPILE_DEFINITIONS(${BII_BLOCK_TARGET})` `PUBLIC "MY_DEFINITION=1"` in your `CMakeLists` after `ADD_BII_TARGETS()`.
 - Review the `BII_LIB_SRC` variable in `CMakeLists.txt` (and `BII_exe_name_SRC`) and look for missing files.

If you detect a file is missing, add it to [\[dependencies\]](#) section in *biicode.conf*.

- If you receive **linker errors**, search in the code the missing symbols.
 - If they are in your source code, maybe biicode is not finding some implementation and the dependency graph wasn't built correctly. You can use *bii deps -files* to inspect how the code is connected. Use [\[dependencies\]](#) section in *biicode.conf* to specify the missing source file.
 - Can't find them in your sources? Try to google them. You may need to link a system library. You can use `TARGET_LINK_LIBRARIES(${BII_LIB_TARGET})` `PUBLIC pthread` in your *CMakeLists.txt* after `ADD_BII_TARGETS()`.

3. Test the library's reusability

At this point biicode knows how to build your code. But you are not done yet. You should check that your library can be included and works fine.

- **bii publish** to publish a DEV version of your code.
- Open a new terminal and create a new biicode project with an example including your library. You can check a lot of reuse examples in . Create a new folder and execute **bii init -l** and **bii new -hello cpp**. Replace *main.cpp* code with your example code.
- Run **bii configure** to create *biicode.conf* and *CMakeLists.txt* files.
- Require your original block library in [\[requirements\]](#) section of *biicode.conf*
- Execute **bii deps** to ensure your requirement is wired right.
- Execute **bii build** to build the example
 - If compilation fails because any files are missing, check `bii/deps/` folder to review the files biicode downloaded. If you notice some file are missing you probably need to add them in [\[dependencies\]](#) section in *biicode.conf*. Fix the library and **bii publish** again. Then execute **bii build** in your example folder again, this downloads the updated library automatically. Check again the files downloaded.

- If compilation fails in cause of an error in your library's *CMakeLists.txt* check that you are not presuming that (*key fact 1*) all files are present. Fix *CMakeLists.txt* or wire a dependency (if needed) in [\[dependencies\]](#) section in *biicode.conf*.
- You can build more examples including more headers from your library to ensure it works well.
- Congrats! You have a full functional library in biicode! Execute **bii publish –tag STABLE** to freeze an stable version.

Got any doubts? Ask in or .

1.6.3 With a previous CMakeLists.txt

Option 1: Let biicode do its job in an isolated file

If you already have a *CMakeLists.txt* file there's no need to replace it, just adapt it like this:

```
IF (BIICODE)
  INCLUDE ("biicode.cmake")
  RETURN ()
ENDIF ()
# Your regular project configuration here
```

Now create a file named *biicode.cmake* and add the line **ADD_BII_TARGETS()**. Then read [without a previous CMakeLists.txt](#) section knowing that *biicode.cmake* is now the file where you will write the code needed.

Option 2: Build your own target library and link them to **BII_LIB_TARGET**

Sometimes, when adapting big and complex libraries that already have a *CMakeLists.txt* building its own library, the best approach is to link the resulting library to `${BII_LIB_TARGET}`

- As you want to use your own library targets and these take for granted that all files are always present, it's violating *key fact n°1*. The way to proceed is wiring all your library files together in [\[dependencies\]](#) section in *biicode.conf*.

EXAMPLE: `[dependencies]` section from .

```
[dependencies]
    # Nothing depend on tests, so do not include tests if not needed
    src/* - tests/*
    lib/* - tests/*
    include/* - tests/*

    # Lib doesn't depend on src
    lib/* - src/*
```

```
# Everything depends on libcurl
src/* + lib/* docs/MANUAL docs/curl.1 src/mkhelp.pl
include/* + lib/*
tests/*.h + src/* lib/* include/* tests/*

# Src module goes together
src/*.h + src/*.c
```

- Enable a plug for biicode (*key fact n°2*) at the end of your *CMakeLists.txt* (or before installation steps), assuming `${LIB_NAME}` is the name of the library you’ve built:

```
IF(BIICODE)
  # Clear biicode auto detected files.
  # BII_LIB_TARGET will be an interface target.
  SET(BII_LIB_SRC)

  ADD_BII_TARGETS()

  # If you have configured some file, include the output directory
  # TARGET_INCLUDE_DIRECTORIES(${BII_LIB_TARGET} INTERFACE ${CMAKE_CURRENT_SOURCE_DIR})

  # Apply biicode dependencies to my library
  TARGET_LINK_LIBRARIES(${LIB_NAME} PUBLIC ${BII_LIB_DEPS})
  # Also the interface properties
  TARGET_LINK_LIBRARIES(${LIB_NAME} PUBLIC ${BII_BLOCK_TARGET})
  # Wire your lib to ${BII_LIB_TARGET} so biicode can use it
  TARGET_LINK_LIBRARIES(${BII_LIB_TARGET} INTERFACE ${LIB_NAME})

ENDIF()
```

- Don’t presume that targets are always present (*key fact n°1*):

EXAMPLE: *tests* folder is not present (because tests not depend on any header of your library), so its not downloaded.

```
IF(BIICODE AND (EXISTS "${CMAKE_CURRENT_SOURCE_DIR}/tests"))
  # Your code for generate examples targets
ENDIF()
```

- If your *CMakeLists.txt* uses `find_package` directive and you want to replace these dependencies and depend on biicode blocks:

- Let biicode handle requirements:

EXAMPLE: This library links OpenSSL library of the system. But we want to link openssl from biicode:


```

if(NOT BIICODE) # Biicode uses OpenSSL as a dep, do not find it in syst
  find_package(OpenSSL)
  if(OPENSSL_FOUND)
    set(USE_OPENSSL ON)
    # ...
    # ...
  endif()
else()
  set(USE_OPENSSL ON)
endif()

```

* Require your original block library in *[requirements]* section in *biicode.conf*

* Execute **bii deps** to ensure your requirement is wired right.

There's a complete example of **Option 2** you can check here at [and](#) .

Option 2 is not “ideal” because is downloading, compiling and linking the whole library and some files may be unnecessary. But if your library files are heavily connected and/or there are so many files this is your best option.

Option 3: Adapt your CMakeLists.txt filtering files

There is a third option, a mix of the two previous options:

- Filter the files with the set of files detected by biicode `${BII_LIB_SRC}`, not forcing all source code to interconnect.

key fact I said not to presume all files exist in our CMakeLists.txt, but we know which files has downloaded looking the `BII_LIB_SRC` variable, so you can always compose your library with the intersection of your list of sources and `BII_LIB_SRC`

EXAMPLE:

```

MACRO(INTERSECTION var_name list1 list2)
  # Store the intersection between the two given lists in var_name.
  SET(intersect_tmp "")
  FOREACH(l ${list1})
    IF("${list2}" MATCHES "^(;){1}(;|$)")
      SET(intersect_tmp ${intersect_tmp} ${l})
    ENDIF("${list2}" MATCHES "^(;){1}(;|$)")
  ENDFOREACH(l)
  SET(${var_name} ${intersect_tmp})
ENDMACRO(INTERSECTION)

```

```
# Biicode detects that file2.cpp is not a dependency of the block that inc
# So in BII_LIB_SRC there are only file1.cpp and file3.cpp
# If we try to add_library using file2.cpp will fail, so lets filter it.

set(my_library_files file1.cpp file2.cpp file3.cpp)
IF(BIICODE)
  INTERSECTION(filtered_files "${my_library_files}" "${BII_LIB_SRC}")
ELSE()
  set(filtered_files ${my_library_files})
END()
add_library(my_library ${filtered_files})
```

You can include from *biicode/cmake* block and reuse the macro `INTERSECTION`. Check *Publish, share and reuse CMake scripts* section for more information.

- Keep the way you build the library:

Following *key fact 2*, you can build your library and *link to* ``${BII_LIB_TARGET}``, or even change the value of `BII_LIB_TARGET` variable to match your library name. The only thing important is that the variable `BII_LIB_TARGET` contains a cmake library.

```
SET(BII_LIB_TARGET my_library)
```

As you know we're available at for questions and answers. You can also .

1.7 Advanced Usage

Here's a few usage cases for understanding biicode in depth:

1.7.1 Custom Layouts

Blocks live in Biicode projects, each biicode project can have in it as many blocks as you want.

A **project** is a combination of meta-data and folders containing your blocks, dependencies and files like *policies.bii* to apply when finding or updating your dependencies.

Use **bii init -L** or **bii init --layout** command to use a different folder structure.

Simple Layout

Place your repo's code directly in your project's folder. Use **bii init -L** .

```
~$ bii init myproject -L
```

Creates a simple folder structure in which *deps/*, *build/* and *cmake/* folders - all auxiliary folders but *bin/* - are inside *bii/* folder:

```
+-- myproject/
|   +-- bii/
|       +-- layout.bii
|       +-- policies.bii
|       +-- settings.bii
```

in which *layout.bii* content is:

```
# Minimal layout, with all auxiliary folders inside "bii" and
# The binary "bin" folder as is, and enabled code edition in the project root
cmake: bii/cmake
lib: bii/lib
build: bii/build

deps: bii/deps
# Setting this to True enables directly editing in the project root
# instead of blocks/youruser/yourblock
# the block will be named as your project folder
auto-root-block: True
```

For example, this is a **project with simple layout** :

```
+-- myproject/
|   +-- bii/
|       +-- layout.bii
|       +-- policies.bii
|       +-- settings.bii
|       +-- build/
|       +-- cmake/
|       +-- deps/
|   +-- bin/
|   +-- src/
|   +-- biicode.conf
|   +-- CMakeLists.txt
```

A project's layout is fully customizable via *layout.bii* file, you can place the auxiliary folders wherever you want, just specify the relative routes to the folders you want to use instead.

TMP Layout

Looking for an even cleaner layout? Use **bii init -l tmp**.

This layout option redirects *deps/*, *build/* and *cmake/* folders to the temporal folder of your system *tmp/myproject/* -all aux folders but *bin/*- and places your repo's code directly in your project's folder.

Creates a folder structure in which *deps/*, *build/* and *cmake/* folders (all auxiliary folders but *bin/* are inside *bii/* folder:

```
+-- myproject/
|   +-- bii/
|       +-- layout.bii
|       +-- policies.bii
|       +-- settings.bii
```

in which *layout.bii* content is:

```
# Layout that redirect aux folders to your tmp/project folder
cmake: $TMP/cmake
lib: $TMP/lib
build: $TMP/build
deps: $TMP/deps
auto-root-block: True
```

For example, this is a **project with TMP layout** :

```
+-- myproject/
|   +-- bii/
|       +-- layout.bii
|       +-- policies.bii
|       +-- settings.bii
|   +-- bin/
|   +-- src/
|   +-- biicode.conf
|   +-- CMakeLists.txt
```

Classic Layout

bii init myproject creates a simple folder structure:

```
+-- myproject/
|   +-- bii/
|       +-- policies.bii
|       +-- settings.bii
```

And executing:

```
~$ cd myproject
~/myproject$ bii new username/blockname --hello=cpp
```

creates this structure into *myproject*:

```
+-- myproject/
|   +-- bii/
|   |   +-- policies.bii
|   |   +-- settings.bii
|   +-- blocks/
|   |   +-- username/
|   |   |   +-- blockname/
|   |   |   |   +-- main.cpp
```

Each project follows the same standard structure, for example:

```
+-- myproject/
|   +-- bii/
|   +-- blocks/
|   |   +-- owner1/
|   |   |   +-- blockA/
|   |   |   |   +-- src/
|   |   |   |   +-- include/
|   |   |   |   +-- test/
|   |   |   |   +-- biicode.conf
|   |   |   |   +-- CMakeLists.txt
|   +-- deps/
```

This structure empowers consistency between the blocks published in biicode, it also enables working with *different owner/blocks* at the same time:

```
+-- myproject/
|   +-- bii/
|   |   +-- policies.bii
|   |   +-- settings.bii
|   +-- blocks/
|   |   +-- owner1/
|   |   |   +-- blockA/
|   |   |   |   +-- src/
|   |   |   |   +-- include/
|   |   |   |   +-- test/
|   |   |   |   +-- biicode.conf
|   |   |   |   +-- CMakeLists.txt
|   |   |   +-- blockB/
|   |   |   |   +-- main.cpp
|   |   |   |   +-- biicode.conf
|   |   |   |   +-- CMakeLists.txt
|   |   +-- owner2/
|   |   |   +-- blockC/
|   |   |   |   +-- tool.h
|   |   |   |   +-- tool.cpp
```

```
|      |      |      |      +-- biicode.conf
|      |      |      |      +-- CMakeLists.txt
|      +-- deps/
```

CLion Layout

Use with biicode, just like the regular biicode layout but with a *bii/layout.bii* to integrate biicode with the C/C++ IDE.

This layout places your repo's code as usual, in your *project_name/blocks/owner/blockname* directory:

This working project looks like this:

```
+-- myproject/
|   +-- bii/
|       +-- policies.bii
|       +-- settings.bii
|       +-- layout.bii
|   +-- blocks/
|       +-- owner1/
|           +-- blockA/
|               +-- src/
|               +-- include/
|               +-- test/
|               +-- biicode.conf
|               +-- CMakeLists.txt
|   +-- deps/
```

in which *layout.bii* content is:

```
# Layout for CLion IDE with root CMakeLists at project root
# This layout DOES NOT allow root-block, as it will overwrite the project CMakeList
cmake: /
```

Here's more info about *working with CLion*.

Check our and/or for questions and answers. You can also for suggestions and feedback.

1.7.2 Tests

Sometimes your library includes some tests to check your its functionality. Your *biicode.conf* [tests] section is here to cover these tests.

Just write the test files specifically or the path to the folder that contains them like this:

```
[tests]

projects/SelfTest/*
tests/unit_test.cpp
```

This way, these files are not compiled executing **bii build** command.

Run **bii test** and you're ready to go. Check [bii test options here](#).

You can specify in your `[mains]` section that which tests aren't mains. [Here's more on \[mains\]](#) and [\[tests\]](#) sections.

1.7.3 Open multiple blocks

Working with your own blocks

In the [Getting Started guide](#) we built a program and reused the **sum function**. Now it's time to add new functionality to your published **myuser/math** block, like a **subtract function**, and use it in your block **myuser/calc**.

The layout is:

```
+-- mycalc
|   +-- blocks
|   |   +-- myuser
|   |   |   +-- calc
|   |   |   |   +-- main.cpp
|   +-- deps
|   |   +-- myuser
|   |   |   +-- math
|   |   |   |   +-- operations.cpp
|   |   |   |   +-- operations.h
```

Opening your block

Open the block **myuser/math** for editing on the same project, execute:

```
~/mycalc$ bii open myuser/math
```

bii open command retrieves the complete block to your *blocks* folder, and deletes it from your *deps* folder. In this case, it will open the specific version you depend on.

The resulting layout is:

```
+-- mycalc
|   +-- blocks
```

```
|      |      +-- myuser
|      |      |      +-- calc
|      |      |      |      +-- main.cpp
|      |      |      |      +-- math
|      |      |      |      +-- main.cpp
|      |      |      |      +-- operations.cpp
|      |      |      |      +-- operations.h
|      +-- deps
```

Now, add the new function, **subtract** and use it on your main.cpp

operations.h

```
#pragma once
int sum(int a, int b);
int subtract(int a, int b);
```

operations.cpp

```
#include "operations.h"
int sum(int a, int b) {return a+b;}
int subtract(int a, int b) {return a-b;}
```

main.cpp

```
#include "google/gtest/gtest.h"
#include "operations.h"

TEST(Sum, Normal) {
    EXPECT_EQ(5, sum(2, 3));
}

TEST(Subtract, Normal) {
    EXPECT_EQ(-1, subtract(2, 3));
}

int main(int argc, char **argv) {
    testing::InitGoogleTest(&argc, argv);
    return RUN_ALL_TESTS();
}
```

Build, **bii build** and run your tests `myuser_math_main` to check everything is working.

Publishing updated code

Publish the math block again. As you now have 2 blocks opened (calc, math), specify the name of the block you want to publish:

```
~/mycalc$ bii publish myuser/math
```


By default, **bii publish** uses the DEV tag. Check on your online biicode profile it's been published.

Using DEV tag, the latest DEV version is overridden, so `[parents]` section of your *biicode.conf* remains unmodified:

```
[parent]
  myuser/math: 0
```

Closing edited block

You can now close **myuser/math** block, it and it will return, with the code already updated, to your *deps* folder:

```
~/mycalc$ bii close myuser/math
```

Then you can modify the content of your **myuser/calc**:

main.cpp

```
#include <iostream>
#include "myuser/math/operations.h"

using namespace std;
int main() {
    cout<<"2 + 3 = "<< sum(2, 3)<<endl;
    cout<<"2 - 3 = "<< subtract(2, 3)<<endl;
}
```

and build it, reusing also the new function:

```
~/mycalc$ bii build
~/mycalc$ bin\myuser_calc_main
2 + 3 = 5
2 - 3 = -1
```

Congrats! You just edited your dependencies and updated the changes. You know that we are available at for any problems. You can also for suggestions and feedback, they are always welcomed.

Working with any published block

To **edit a published block**, follow the steps below:

Open a block

Open a block locally to modify and publish a new version of a block.

```
~/ $ bii init myproject
~/ $ cd myproject
~/myproject $ bii open username/blockname:VERSION
```

Example

Let's open (version=latest by default) to edit it:

```
$ bii open lasote/json11
```

Then you can code on it as if it was yours and changes will be reflected in your code, at build time.

Suppose that you want to open version 1 instead of the latest `lasote/json11` version, you should execute:

```
$ bii open lasote/json:1
```

Publish the changes

Once your changes build, publish your own version of the block.

If the block in edition isn't yours:

- Rename **lasote** folder with your **username**.
- Delete the `[parents]` section content in your *biicode.conf* file.

Remember to **bii build** before publishing!

And publish:

```
$ bii publish
```

Check *bii publish command* to know more.

Close the block

Then you can close the block to remove it from your blocks folder:

```
$ bii close user_name/block_name
```

If you're following the **Example**, execute:

```
$ bii close user_name/json11
```

Depend on the block you've just published

Now, if you want to depend on the block you've just published:

- Update your `#include` (s) to the ones referring to your new published version
- If you didn't publish it as STABLE, do it or update your *policies.bii* file to accept DEV versions.
- Execute **bii find** and you're ready to build as usual. Here's *bii find command documentation*.

If you need more information about publish or close command:

- *Publish command*
- *Close command*

Here's how to *create a biicode block from a git repo*.

Check our and/or for questions and answers. You can also for suggestions and feedback.

Got any doubts? Do not hesitate to [contact us](#) visit our [forum](#) and feel free to ask any questions.

1.7.4 Toolchains

When you build your projects, biicode automatically generates a default tool-chain to build it. To **use a custom tool-chain** you need to **place it in the bii folder** of your project *with the name* `<your_toolchain_name>-toolchain.cmake`.

To use it, just pass it as argument of **bii configure -t your_toolchain_name**.

For example, I want to write a program to my armv7 and I have a toolchain named armv7-toolchain.cmake. First, copy my toolchain with the name armv7-toolchain.cmake into the bii folder. Then, execute **bii configure** with **-t** or **-toolchain** flag with the name **armv7**:

```
$ bii init my_armv7_machine
$ cd my_armv7_machine
$ #copy armv7-toolchain.cmake into init my_armv7_machine/bii
$ bii configure -t armv7
```

To change the toolchain you are using, just execute **bii configure -t my_new_toolchain_name**

If you want to use the native environment, execute **bii configure -t** without any toochain name or use *None* as name.

There are two toolchains you can use by default, the *arduino-toolchain.cmake* and the *rpi-toolchain.cmake*. If you want to use one of it, just use **bii arduino:settings** and **bii configure -t arduino** or **bii rpi:settings** and **bii configure -t rpi**.

More information about toolchains is available in [CMake's docs](#) .

We are available at for any issues. You can also for suggestions and feedback.

1.7.5 Override a dependency

Let's say you depend on:

- `erincatto/box2d:10` that depends on `diego/glfw:0`.

And you'd rather depend on:

- `erincatto/box2d:10` and `diego/glfw:1`.

Write your preferred versions in your *biicode.conf* and biicode will use those versions in your project:

biicode.conf

```
[requirements]
# required blocks (with version)
erincatto/box2d: 10
diego/glfw:1
```

Execute **bii build** and it's updated.

Override a dependency with block tracks

Create a block track when you need a personalized fix over the original library.

Let's create a block track from **diego/glfw** block:

- Open the block:

```
~$ bii init myproject
~$ cd myproject
~/myproject$ bii open diego/glfw
```

- Code, adjust it to your needs.
- Write the track name between brackets in the `[parent]` section of the **biicode.conf** file. Specify `version -1` because we want create a new block.

biicode.conf

```
[parent]
diego/glfw(myuser/glfw): -1
```

- Execute **bii publish** and enter your profile *www.biicode.com/myuser* to check the new track.

Depend on that new block track:

- Write in your *biicode.conf* file [requirements] :

biicode.conf

```
[requirements]
# required blocks (with version)
diego/glfw(myuser/glfw): 1
```

- Execute **bii build** and it's updated.

What if you want to get back again to the original library?

- Write in your *biicode.conf* file [requirements] :

```
[requirements]
# required blocks (with version)
diego/glfw: 0
```

1.7.6 Advanced build configuration

Apart from all the building with CMake tips available in *custom build configuration guide*, there's also a few things that may come in handy while using biicode with CMake.

Publish, share and reuse CMake scripts

Now, biicode lets you publish, share and reuse CMake scripts. You can reuse other user's CMake macros/functions and apply any content in your *CMakeLists.txt*.

Reusing CMake code is as simple as *#including* libraries in C++ with biicode.

Edit your *CMakeLists.txt* file and include the CMake file from the block that you want:

```
INCLUDE(user/block/path_to_macros_file) # Without .cmake extension
MACRO_NAME_TO_USE() # Macro defined in My_macros.cmake

# Actually create targets: EXEcutable and libraries.
ADD_BII_TARGETS()
```

And run **bii find** command:

```
$ bii find
```

All CMake dependencies will be downloaded into your project/deps/user/block folder

EXAMPLE: How to activate C++11 with a macro already programmed?

biicode featured user has a block named where you can find useful macros from the *tools.cmake* file, like one to activate C++11 flags for any OS, or to link a OSX framework to a target, etc.

Just edit your *CMakeLists.txt* file, include `INCLUDE(biicode/cmake/tools)` and use the Macros.

CMakeLists.txt

```
# Including tools.cmake from biicode/cmake user block
# see https://www.biicode.com/biicode/cmake
INCLUDE(biicode/cmake/tools)

ADD_BII_TARGETS()

# Calling specific macro to activate c++11 flags
ACTIVATE_CPP11(INTERFACE ${BII_BLOCK_TARGET})
```

Remember to run **bii find** to download the dependency.

```
$ bii find
```

Overriding dependencies build options and configuration

Sometimes you need to override some configuration of how your dependency libraries are built.

This is the project layout when you have dependencies:

```
|-- my_project
|   |-- blocks
|   |   |-- my_user
|   |   |   |-- my_block
|   |   |   |   |-- biicode.conf
|   |   |   |   |-- CMakeLists.txt
|   |   |   |   |-- main.cpp
|   |-- deps
|   |   |-- lasote
|   |   |   |-- superlibrary
|   |   |   |   |-- biicode.conf
|   |   |   |   |-- CMakeLists.txt
|   |   |   |   |-- library.h
```

```
|      |      |      |      +-- library.cpp
|      |      +-- sara
|      |      |      +-- coollibrary
|      |      |      |      +-- biicode.conf
|      |      |      |      +-- CMakeLists.txt
|      |      |      |      +-- tool.h
|      |      |      |      +-- tool.cpp
```

You should not edit the source code in `deps` directory because it will be overwritten by biicode. As can't change the `CMakeLists.txt` files of our dependencies directly, here's a way to override their build configuration.

How does it work?

Create a file named *bii_deps_config.cmake* in your block (*my_user/my_block/*) and write into it the CMake code you need.

You can act upon dependency target following this naming rule:

```
[USER]_[BLOCK]_interface
```

For example, if we have *lasote/superlibrary* block as a dependency, we can refer to it using this interface name:

```
lasote_superlibrary_interface
```

- **EXAMPLE:** Activate C++ 11 in the dependency *lasote/superlibrary* block:

```
target_compile_options(lasote_superlibrary_interface INTERFACE -std=c++11)
```

- **EXAMPLE:** Change a compilation option:

```
SET(MY_OPTION OFF CACHE BOOL "MyCoolOption" FORCE)
```

We are available at for any issues. You can also for suggestions and feedback.

1.7.7 Publish a block track

Each block has an *owner*, *name*, *version* and *tag*. For example, the block has an **owner** (*lasote*) and latest version is **4 DEV**.

Use **Block Tracks** to publish different development *versions* of a block using the same block name-space. This way, dependent blocks can keep the same *#includes* in their source code.

Publish a new block Track

Write the track name between brackets in the `[parent]` section of the *biicode.conf* file. Specify version `-1` because we want create a new block.

```
[parent]
  myuser/myblock(track1) : -1
```

Now you have configured a track of your block.

In case you need a personalized fix over the original library from other user, just indicate it in the `[parent]` section like this:

```
[parent]
  lasote/libuv(myuser/track1) : -1
```

This way, you have configured a track of other user whitout changing *includes*.

Execute **bii publish** and enter your profile www.biicode.com/myuser to check the new track.

Read a bit more about how *tracks* work, visit our post in the blog about .

1.7.8 Private blocks

Upgrade your account to Premium, , to use Private blocks. Store your code in private, choose who can see or edit your blocks.

Create private blocks in our web page. Just press **Add block button** and choose private.

Got any doubts? or .

1.8 Examples

This doesn't aim to be a comprehensive list of all the contents of biicode, just some examples of already existing code that is ready for use.

1.8.1 Basic Compression Library

Basic Compression Library by Marcus Geelnardis a set of open source implementations of several well known lossless compression algorithms, such as Huffman and RLE, written in portable ANSI C. Currently, RLE (Run Length Encoding), Huffman, Rice, Lempel-Ziv (LZ77) and Shannon-Fano compression algorithms are implemented.

The Basic Compression Library is completely independent of system functions, such as file I/O or memory allocation routines. As such it can be used in almost any system, ranging from Windows, Mac OS X and Linux-systems to embedded systems.

You can check the [BCL documentation](#) for more information.

BCL library is stored at [marcus256/bcl](#) and it is generated from this [Github repository](#).

Simple Huffman Compression - Uncompression

This example demonstrates how to get started using **BCL**. You'll learn to compress and uncompress a text file. It is simple to run.

Creating a new project

Create a new project and a *main.cpp* inside like this:

```
$ bii init bcl_example -L
$ cd bcl_example
$ # Create main.cpp
```

The code of the example is this one, it simply creates a new file *myfile.txt* and then it compresses and uncompresses the file created. It also calculates the time it takes to compress it.

Copy the code in the *main.cpp*:

main.cpp

```
/* Standard libraries */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* Basic Compression Library */
#include "huffman.h"

/* Timing */
#include "systemer.h"

/*****
 * GetFileSize()
 *****/

long GetFileSize( FILE *f )
{
    long pos, size;
    pos = ftell( f );
    fseek( f, 0, SEEK_END );
    size = ftell( f );
    fseek( f, pos, SEEK_SET );
```

```
    return size;
}

/*****
 * CompressFile()
 *****/

int TestFile( )
{
    unsigned int    insize, outsize, bufsize, *work, k, err_count;
    unsigned char *in, *out, *buf;
    FILE           *f;
    double          t0, t1, t_comp, t_uncomp;

    printf( "Compressing MyFile, " );

    /* Open input file */
    f = fopen( "myfile.txt", "rb" );
    if( !f )
    {
        printf( "unable to open!\n" );
        return 0;
    }

    /* Get input size */
    insize = GetFileSize( f );
    printf( "File Size:", f );
    if( insize < 1 )
    {
        printf( "empty file!\n" );
        fclose( f );
        return 0;
    }

    /* Worst case output buffer size */
    bufsize = (insize*104+50)/100 + 384;

    /* Allocate memory */
    in = (unsigned char *) malloc( insize + 2*bufsize );
    if( !in )
    {
        printf( "out of memory!\n" );
        fclose( f );
        return 0;
    }
}
```

```

/* Pointers to compression buffer and output memory */
buf = &in[ insize ];
out = &buf[ bufsize ];

/* Read and close input file */
fread( in, 1, insize, f );
fclose( f );

/* Compress and decompress */

t0 = GetTime();
outsize = Huffman_Compress( in, buf, insize );
t_comp = GetTime() - t0;
t1 = GetTime();
Huffman_Uncompress( buf, out, outsize, insize );
t_uncomp = GetTime() - t1;

err_count = 0;
if(outsize > 0)
{
    /* Show compression result */
    printf( "\n Compression: %d/%d bytes (%.1f%%)", outsize, insize,
        100*(float)outsize/(float)insize );

    /* Compare input / output data */
    for( k = 0; k < insize; ++ k )
    {
        if( in[ k ] != out[ k ] )
        {
            if( err_count == 0 ) printf( "\n" );
            if( err_count == 30 ) printf( "... \n" );
            else if( err_count < 30 )
            {
                printf( " %d: %d != %d\n", k, out[ k ], in[ k ] );
            }
            ++ err_count;
        }
    }

    /* Did we have success? */
    if( err_count == 0 )
    {
        printf( " - OK!\n" );
        printf( " Compression speed: %.1f KB/s (%.2f ms)\n",
            (double) insize / (1024.0 * t_comp), 1000.0 * t_comp );
        printf( " Uncompression speed: %.1f KB/s (%.2f ms)\n",
            (double) insize / (1024.0 * t_uncomp), 1000.0 * t_uncomp );
    }
}

```

```
    }
    else
    {
        printf( "      *****\n" );
        printf( "      ERROR: %d faulty bytes\n", err_count );
        printf( "      *****\n" );
    }
}

/* Free all memory */
free( in );

return (outsize > 0) && (err_count == 0);
}

int main()
{
    FILE * pFile;
    char buffer [100];

    pFile = fopen ( "myfile.txt" , "w+" );
    fprintf(pFile, "%s %s %s %d", "We", "are", "in", 2014);
    if (pFile == NULL) perror ( "Error opening file" );
    else
    {
        while ( ! feof (pFile) )
        {
            if ( fgets (buffer , 100 , pFile) == NULL ) break;
            fputs (buffer , stdout);
        }
        fclose (pFile);
    }
    TestFile();
}
```

Manage your dependencies

Check the dependencies of the project with **bii deps**:

```
$ bii deps
your_user/bcl_example depends on:
  system:
    stdio.h
    stdlib.h
    string.h
  unresolved:
```

```
huffman.h
sys timer.h
```

Now, edit the *biicode.conf* file generated in the project folder. Add your [requirements] depending on the version you want and map your [includes]:

```
[requirements]
    marcus256/bcl: 2

[includes]
    *.h: marcus256/bcl/src
```

Check again with **bii deps** to show all resolved dependencies.

Build the project

Now, build and run the huffman compression-uncompression example.

```
$ bii build
$ cd bin
$ examples_bcl_main
```

Once you execute you should see an output like this one, it may vary depending on your computer:

```
Compressing MyFile, File Size:
  Compression: 20/14 bytes (142.9%) - OK!
  Compression speed: 1246.6 KB/s (0.01 ms)
  Uncompression speed: 4778.7 KB/s (0.00 ms)
```

That's all! You can try it with other files too.

Open and build

This example is already in biicode: [examples/bcl](#).

To give it a try, create a new project and open the block:

```
$ bii init bcl_example
$ cd bcl_example
$ bii open examples/bcl
```

Build the example and execute it:

```
$ bii build
$ cd bin
$ # Execute it
  Compressing MyFile, File Size:
```

```
Compression: 20/14 bytes (142.9%) - OK!
Compression speed: 1246.6 KB/s (0.01 ms)
Uncompression speed: 4778.7 KB/s (0.00 ms)
```

Any doubts? Do not hesitate to [contact us](#) visit our [forum](#) and feel free to ask any questions.

1.8.2 Boost Libraries

Boost is a set of libraries for the C++ programming language that provide support for tasks and structures such as linear algebra, multithreading, image processing, unit testing...

The examples below demonstrate how to use biicode to set up Boost-based projects.

First let's configure the examples project:

```
$ bii init boost-examples
$ cd boost-examples
```

We will be using that project across all the examples.

Boost.Lambda

This is an example about the Boost.Lambda library, extracted from the [Boost getting started](#) header-only section.

First create a block for the example:

```
$ bii new your-account/boost-lambda
```

Create a source file:

blocks/your-account/boost-lambda/example.cpp:

```
#include <boost/lambda/lambda.hpp>
#include <iostream>
#include <iterator>
#include <algorithm>

int main()
{
    using namespace boost::lambda;
    typedef std::istream_iterator<int> in;

    std::for_each(
        in(std::cin), in(), std::cout << (_1 * 3) << " " );
}
```

biicode sets up a Boost installation inside the biicode environment. This allows to support and switch between multiple Boost versions easily.

Setting-up Boost

To enable Boost in your biicode project, just go to the block's `CMakeLists.txt` and add the following lines:

`blocks/your-account/boost-lambda/CMakeLists.txt`

```
#Include the biicode Boost setup script
include(biicode/boost/setup)

ADD_BII_TARGETS()

#Setup Boost and build (if needed) the required Boost components
#Since lambda is header-only, there are no components to build and find
bii_find_boost()

#Add Boost headers to the block include directories
target_include_directories(${BII_BLOCK_TARGET} INTERFACE ${Boost_INCLUDE_DIRS})
```

`bii_find_boost()` is intended to wrap `find_package(Boost ...)`, with almost the same input and output variables:

```
bii_find_boost([COMPONENTS component1 component2 ...] [REQUIRED])
```

It takes the Boost components you need, exactly as `find_package(Boost)`, configures the biicode boost installation, builds that components, and then calls `find_package(Boost COMPONENTS ...)`.

biicode uses the current cmake C++ compiler as Boost toolset, multiple toolsets are supported inside the same Boost version installation. Also note that, except you delete the libraries at the biicode environment, each Boost component will be built only once for each Boost version and toolset.

Build and run project

To configure and build your project, run `bii find` to get the dependencies (The boost setup block), and then run `bii configure` and `bii build` as usual:

```
$ bii find
$ bii configure
      block your-account/boost-lambda
-----
-- Setting up biicode Boost configuration...
```

```
-- Boost version: 1.57.0
...

$ bii build
```

Boost version

You may notice that the example above uses Boost 1.57.0. This is the default Boost version, shipped by the `biicode/boost` master track. If you want other version, say Boost 1.56.0, go to the block's `biicode.conf` file and edit the requirements entry, explicitly asking for the `biicode/boost` track you want:

```
blocks/your-account/boost-lambda/biicode.conf
```

```
[requirements]
    biicode/boost(1.56.0)
```

Tip

You may want to support multiple tracks for your block depending on the Boost version it uses. One way could be set different tracks for your block, matching the Boost version requested:

```
[requirements]
    biicode/boost(1.56.0)

[parent]
    your-account/boost-lambda(1.56.0): -1, 0, whatever version is available
```

Boost.Coroutine

Boost.Coroutine implements [coroutines](#) which can be useful to implement cooperative multitasking, iterators, etc, in a more natural way.

Open the [Boost.Coroutine example](#)

```
$ bii open examples/boost-coroutine
```

The example source code:

```
#include <boost/coroutine/all.hpp>
#include <iostream>

using namespace boost::coroutines;

void cooperative(coroutine<void>::push_type &sink)
{
    std::cout << "Hello";
```



```

    sink();
    std::cout << "world";
}

int main()
{
    coroutine<void>::pull_type source{cooperative};
    std::cout << ", ";
    source();
    std::cout << "!\n";
}

```

In the code above, the string “Hello, world!” is written by writing “Hello”, going back to `main()` which writes the comma, then calling the coroutine again to continue printing “world”, and finally printing “!” on `main()`.

You can visualize it as:

```

main() | cooperative()
-----+-----
      | "Hello"
", " |
      | "world"
"! " |

```

This is the `CMakeLists.txt` from the example:

```

include(biicode/boost/setup)

ADD_BII_TARGETS()

set(Boost_USE_STATIC_LIBS ON)
bii_find_boost(COMPONENTS system coroutine context thread REQUIRED)
target_include_directories(${BII_BLOCK_TARGET} INTERFACE ${Boost_INCLUDE_DIRS})
target_link_libraries(${BII_BLOCK_TARGET} INTERFACE ${Boost_LIBRARIES})

if(MSVC)
    set(CMAKE_EXE_LINKER_FLAGS "${CMAKE_EXE_LINKER_FLAGS} /SAFESEH:NO")
else()
    target_compile_options(${BII_BLOCK_TARGET} INTERFACE -std=c++11)
endif()

```

Besides Visual Studio specific configuration ([See](#)), the configuration is pretty straightforward:

1. Set the way you want to link against Boost libraries with `Boost_USE_STATIC_LIBS` as usually when using Boost with CMake. biicode assumes static linking by default.
2. Set up and find the required Boost components with `bii_find_boost()`. Note the order matters, since it's the order the libraries are linked together.

3. Add Boost to your target include directories. Again as usual when using Boost with CMake, via `target_include_directories()` and `Boost_INCLUDE_DIRS` variable.
4. Link your target against Boost libraries using the `Boost_LIBRARIES` variable.

Now just run `bii configure`, wait until the Boost components are built (If those were not used previously), and then build your project with `bii build`.

Configure Generator

Generators recommended for this example:

- **Windows:** Visual Studio
- **Linux & MacOS:** Unix Makefiles

```
$ bii find
$ bii configure -G "Unix Makefiles"

examples/boost-coroutine
-----
-- Setting up Biicode Boost...
-- Building Boost 1.57.0 components with toolset gcc-4.9.2...
-- Building system library...
-- Building coroutine library...
-- Building context library...
-- Building thread library...
-- Boost 1.57.0

Found the following Boost libraries
  system
  coroutine
  context
  thread
...
$ bii build
```

Alternative setup call

`bii_find_boost()` sets up Boost and then calls `find_package(Boost)` with the components passed to the former. While this setup works in 90% cases, there are some situations when the Boost components you need do not correspond to Boost libraries directly.

If that's the case, you can call `bii_setup_boost()` function passing the Boost libraries to build, and then do the classic `find_package(Boost COMPONENTS ...)` with the components you need.

Take `Boost.Log` as an example:

`examples/boost-log/CMakeLists.txt`

```
include(biicode/boost/setup)

ADD_BII_TARGETS()

set(Boost_USE_STATIC_LIBS OFF)
set(Boost_USE_MULTITHREADED ON)

bii_setup_boost(COMPONENTS system thread filesystem log REQUIRED)
find_package(Boost COMPONENTS system thread filesystem date_time log log_setup REQUIRED)

target_compile_options(${BII_BLOCK_TARGET} INTERFACE -DBOOST_LOG_DYN_LINK)
target_include_directories(${BII_BLOCK_TARGET} INTERFACE ${Boost_INCLUDE_DIRS})
target_link_libraries(${BII_BLOCK_TARGET} INTERFACE ${Boost_LIBRARIES})
```

Note how the components required by Log are not exactly the same libraries that should be built.

Extra configuration variables

The Boost setup scripts have some extra variables to debug and or customize Boost a bit:

- `BII_BOOST_VERBOSE`: When is set to on, the setup scripts print some internal info about the current setup being run, and the different steps.
- `BII_BOOST_LIBCXX`: When using the Clang compiler, enables Boost build using LLVM's libc++ instead of the default GNU's stdlibc++ standard library implementation.
- `BII_BOOST_BUILD_J`: Specifies the number of threads used when building Boost libraries. May be useful to speed up Boost builds in setups where the libraries are always built, like continuous integration builds without cache. For example: `bii configure -DBII_BOOST_BUILD_J=16`
- `BII_BOOST_GLOBAL_USE_STATIC_LIBS`: Overrides the `Boost_USE_STATIC_LIBS` values specified in each `CMakeLists.txt` and sets a value globally. Useful when you depend on many Boost-related blocks and you may experience issues related to different linkages against Boost in that blocks. Use this variable carefully.

Contribute to the setup scripts

The setup scripts are maintained as an [open source project on GitHub](#), you may want to ask about new features, report bugs, etc.

1.8.3 Box2D

is an open source C++ engine to simulate rigid bodies in 2D, it is also, AngryBirds' motor engine. You can check .

Box2D library is stored at [erincatto/box2d](https://github.com/erincatto/box2d), which is generated from this .

Bounces of a circle falling

In this example you will calculate whenever a circle falls from a certain height and bounces at a defined lower limit in the created world.

Creating a new project

Create a new project and a *main.cpp* file:

```
$ bii init box2d_example -L
$ cd box2d_example
$ # Create main.cpp
```

Now place the following code inside *main.cpp*:

main.cpp

```
#include "Box2D/Box2D.h"
#include <iostream>

using namespace std;

int main(int argc, char** argv)
{
    B2_NOT_USED(argc);
    B2_NOT_USED(argv);

    //*****//
    //                      Creating a World                      //
    //*****//

    // Define the gravity vector.
    b2Vec2 gravity(0.0f, -10.0f);

    // Construct a world object, which will hold and simulate the rigid bodies.
    b2World world(gravity);

    //*****//
```

```

//                                     Creating a Ground Box                                     //
//*****

// Define the ground body.
b2BodyDef groundBodyDef;
groundBodyDef.position.Set(0.0f, -10.0f);

// Call the body factory which allocates memory for the ground body
// from a pool and creates the ground box shape (also from a pool).
// The body is also added to the world.
b2Body* groundBody = world.CreateBody(&groundBodyDef);

// Define the ground box shape.
b2PolygonShape groundBox;

// The extents are the half-widths of the box.
groundBox.SetAsBox(50.0f, 10.0f);

// Add the ground fixture to the ground body.
groundBody->CreateFixture(&groundBox, 0.0f);

//*****
//                                     Creating a Circle Shape                                     //
//*****

b2BodyDef BodyDef;
BodyDef.type = b2_dynamicBody;
BodyDef.position = b2Vec2(0.0f, 4.0f);
BodyDef.userData = (void *) "Circle";
b2Body* body = world.CreateBody(&BodyDef);

b2CircleShape circle;
circle.m_radius = 1.0f;

b2FixtureDef fixtureDef;
fixtureDef.density = 1.0f;
fixtureDef.friction = 2.0f;
fixtureDef.restitution = 0.5f;

fixtureDef.shape = &circle;

body->CreateFixture(&fixtureDef);

//*****
//                                     Simulating the World (of Box2D)                                     //
//*****

```

```
float32 timeStep = 1.0f / 60.0f;
int32 velocityIterations = 6;
int32 positionIterations = 2;

// This is our little game loop.
for (int32 i = 0; i < 100; ++i)
{
    // Instruct the world to perform a single step of simulation.
    // It is generally best to keep the time step and iterations fixed.
    world.Step(timeStep, velocityIterations, positionIterations);

    // Now print the position and angle of the body.
    b2Vec2 position = body->GetPosition();
    float32 angle = body->GetAngle();

    if (position.y - 1.00 <= 0.001)
        cout<< "Ball hits the ground!!" << endl;
    else
        cout<< "X = " << position.x << " Y = " << position.y << endl;
}
}
```

Manage your dependencies

Check the dependencies of the project with **bii deps**:

```
$ bii deps
your_user/box2d_example depends on:
  system:
    iostream
  unresolved:
    Box2D/Box2D.h
```

Now, edit the *biicode.conf* file generated in the project folder. Add your [requirements] depending on the version you want and map your [includes]:

```
[requirements]
  erincatto/box2d: 10

[includes]
  Box2D/Box2d.h: erincatto/box2d
```

Check again with **bii deps** and now all dependencies are resolved.

Build the project

Next, the only thing left is building the project:

```
$ bii build
```

Execute the binary placed in bin directory and this is how output looks like:

```
~/box2d$ bin/myuser_box2d_example_main
X = 0 Y = 3.99722
X = 0 Y = 3.99167
X = 0 Y = 3.98333
X = 0 Y = 3.97222
X = 0 Y = 3.95833
X = 0 Y = 3.94167
X = 0 Y = 3.92222
Ball hits the ground!!
```

That's it!

Open and build

This example is already in biicode: [examples/box2d](#).

To give it a try, create a new project and open the block:

```
$ bii init box2d_example
$ cd box2d_example
$ bii open examples/box2d
```

Build the example and execute it:

```
$ bii build
$ cd bin
$ # Execute it
...
X = 0 Y = 3.95833
X = 0 Y = 3.94167
X = 0 Y = 3.92222
Ball hits the ground!!
```

Got any doubts? Do not hesitate to [contact us](#), visit our [forum](#) and feel free to ask any questions.

1.8.4 C++ challenge

We have proposed a [C++ challenge](#) for showing the benefits of our technology. The goal is to build a wrapper around two linear systems solver; one dense, and the other one sparse.

The [source files](#) have dependencies to 3 well known open source libraries:

- [Eigen Library](#) is headers-only.
- [CSparse](#) is very simple, just a *.h and some *.c files
- [Google GTest](#) is also very portable, independent and prepared to be integrated in other projects.

The challenge consists in building and running the code, in three different platforms: Windows, Linux and Mac. Try solving it by your own means, and then solve the problem with biicode. You'll see how much easier and natural it is!

In this section we explain, step by step, how to solve this challenge with the help of biicode.

Create a new project

Open your console and create a new project named “challenge”:

```
$ bii init challenge
$ cd challenge
```

Copy the code

Download the [challenge sources from github](#) or in [zipped format from this link](#). Copy all the files into challenge/blocks/your_user_name/challenge. If you got the sources from github, you need to replace the #include directives as follows: (#include “sparse/cs.h” => #include “tdavis/sparse/cs.h”, #include <eigen/Dense> => #include <eigen/eigen/Dense>, #include “gtest/include/gtest/gtest.h” => #include “google/gtest/include/gtest/gtest.h”)

Now you can check for all the code dependencies of the current project using the `bii deps` command. Please, note that while most of the dependencies are correctly solved, three of them are not. These **unresolved dependencies** have been highlighted in the following figure:

```
$ bii deps
Detected 5 files created, 0 updated
Processing project
  Cell your_user_name/challenge/systemsolver.h is implemented by set(['your_user_r
Find resources with include to gtest ['your_user_name/challenge/test1.cpp']

Adding resources to your_user_name/challenge/gtest_main.cc
Saving files on disk
DepTable:
Declarations:
  Resolved
  map
```



```

vector
iostream
your_user_name/challenge/systemsolver.h
fstream
Unresolved
eigen/eigen/Dense
google/gtest/include/gtest/gtest.h
tdavis/csparse/include/cs.h
Files deps:
System
  map
  vector
  iostream
  fstream
Explicit
  your_user_name/challenge/systemsolver.h
Implicit
  your_user_name/challenge/test1.cpp
  your_user_name/challenge/systemsolver.cpp

```

Find and retrieve dependencies

Now that we know that our project has some missing dependencies, we'll show you how easily biicode helps you to automatically retrieve all of them. You only need to write the `bii find` command. You'll be asked to provide your biicode password, and the client will find and retrieve from our servers any missing dependencies:

```

$ bii find
Finding missing dependencies in server
Password for your_user_name:
Looking for eigen/eigen...
  >> Block candidate: eigen/eigen(eigen/master)
  >> Version eigen/eigen(eigen/master): 0 (STABLE) valid due your policy!
  Found blocks: eigen/eigen(eigen/master): 0
Looking for tdavis/csparse...
  >> Block candidate: tdavis/csparse(tdavis/master)
  >> Version tdavis/csparse(tdavis/master): 0 (STABLE) valid due your policy!
  Found blocks: tdavis/csparse(tdavis/master): 0
Looking for google/gtest...
  >> Block candidate: google/gtest(google/master)
  >> Version google/gtest(google/master): 2 (STABLE) valid due your policy!
  >> Version google/gtest(google/master): 1 (STABLE) valid due your policy!
  >> Version google/gtest(google/master): 0 (DEV) discarded due your policy!
  Found blocks: google/gtest(google/master): 2
  Found blocks: google/gtest(google/master): 1
Analyzing compatibility for found dependencies...

```

```
Resolved block!
Resolved block!
Resolved block!
Dependencies resolved in server:
Find resolved new dependencies:
    eigen/eigen(eigen/master): 0
    google/gtest(google/master): 2
    tdavis/csparse(tdavis/master): 0
All dependencies resolved
Saving files on disk
Computing dependencies
Retrieving resources from server
Retrieving resources from server
Retrieving resources from server
Retrieving resources from server
Saving dependences on disk
```

At this point, you'll find some new folders and files in your `challenge/dep` folder. These are the blocks that biicode considers as needed for compiling the project.

Build and run

The final step is to actually compile and run the app. You can accomplish this task with the `bii build` command and run the executable inside the `bin` folder:

```
$ bii build

...

$ cd bin
$ #run solver executable
***** SPARSE *****
0: 1
1: 0.5
2: 0.333333
3: 0.25
4: 0.2
5: 0.166667
6: 0.142857
7: 0.125
8: 0.111111
9: 0.1
***** DENSE *****
0: 1
1: 0.5
2: 0.333333
```

```

3: 0.25
4: 0.2
5: 0.166667
6: 0.142857
7: 0.125
8: 0.111111
9: 0.1

```

Running the tests is really easy too. Note that in this case one of the tests fails due to sparse solver accuracy, but the execution of the test itself works just fine!.

```

$ cd bin
$ #run test executable
[=====] Running 2 tests from 1 test case.
[-----] Global test environment set-up.
[-----] 2 tests from Solver
[ RUN      ] Solver.BasicDiagonalSparse
challenge/blocks/your_user_name/challenge/test1.cpp:21: Failure
Value of: 1./(i+1)
  Actual: 0.5
Expected: sol[i]
Which is: 0.5
challenge/blocks/your_user_name/challenge/test1.cpp:21: Failure
Value of: 1./(i+1)
  Actual: 0.333333
Expected: sol[i]
Which is: 0.333333
[  FAILED  ] Solver.BasicDiagonalSparse (0 ms)
[ RUN      ] Solver.BasicDiagonalDense
[          OK ] Solver.BasicDiagonalDense (1 ms)
[-----] 2 tests from Solver (1 ms total)

[-----] Global test environment tear-down
[=====] 2 tests from 1 test case ran. (1 ms total)
[ PASSED   ] 1 test.
[  FAILED  ] 1 test, listed below:
[  FAILED  ] Solver.BasicDiagonalSparse

1 FAILED TEST

```

1.8.5 CImg

The CImg Library is an open-source C++ toolkit for image processing. It consists of a single header file “CImg.h” providing a minimal set of C++ classes and methods that can be used in your own sources, to load/save, process and display images. Very portable (Unix/X11, Windows, MacOS X, FreeBSD, ..), efficient, easy to use, it’s a pleasant library for developping image processing

algorithms in C++.

The main CImg block is at [tschumperle/cimg](https://github.com/tschumperle/cimg) and contains several examples. Its generated from this [github repo](#).

Tron game

This is a classic Tron game that shows how to load and manipulate images with Cimg library in a simple way.

Let's try it out!

Create a new project

Init a new project and a new *tron.cpp* file inside and copy the code below:

```
$ bii init tron_example -L
$ cd tron_example
$ # Create tron.cpp and copy the code
```

tron.cpp

```
1  /*
2   #
3   #  File      : tron.cpp
4   #              ( C++ source file )
5   #
6   #  Description : A clone of the famous (and very simple) Tron game.
7   #                  This file is a part of the CImg Library project.
8   #                  ( http://cimg.sourceforge.net )
9   #
10  #  Copyright  : David Tschumperle
11  #                  ( http://tschumperle.users.greyc.fr/ )
12  #
13  #  License     : CeCILL v2.0
14  #                  ( http://www.cecill.info/licences/Licence_CeCILL_V2-en.html )
15  #
16  #  This software is governed by the CeCILL license under French law and
17  #  abiding by the rules of distribution of free software. You can use,
18  #  modify and/ or redistribute the software under the terms of the CeCILL
19  #  license as circulated by CEA, CNRS and INRIA at the following URL
20  #  "http://www.cecill.info".
21  #
22  #  As a counterpart to the access to the source code and rights to copy,
23  #  modify and redistribute granted by the license, users are provided only
24  #  with a limited warranty and the software's author, the holder of the
25  #  economic rights, and the successive licensors have only limited
```

```

26 # liability.
27 #
28 # In this respect, the user's attention is drawn to the risks associated
29 # with loading, using, modifying and/or developing or reproducing the
30 # software by the user in light of its specific status of free software,
31 # that may mean that it is complicated to manipulate, and that also
32 # therefore means that it is reserved for developers and experienced
33 # professionals having in-depth computer knowledge. Users are therefore
34 # encouraged to load and test the software's suitability as regards their
35 # requirements in conditions enabling the security of their systems and/or
36 # data to be ensured and, more generally, to use and operate it in the
37 # same conditions as regards security.
38 #
39 # The fact that you are presently reading this means that you have had
40 # knowledge of the CeCILL license and that you accept its terms.
41 #
42 */
43
44 #include "CImg.h"
45
46 using namespace cimg_library;
47
48 // Main procedure
49 //-----
50 int main(int argc, char **argv) {
51
52     // Print usage, help and retrieve command line options
53     //-----
54     cimg_usage("A very simple Tron game, using the CImg Library");
55     cimg_help("--- Quick help -----\n"
56              " Player 1 (blue) :\n"
57              "   Use keys 'W' (up), 'S' (down), 'A' (left)\n"
58              "       and 'D' (right) to control your player.\n"
59              "       Right 'CONTROL' key enables turbospeed\n"
60              " Player 2 (red) : \n"
61              "       Use arrow keys to control your player.\n"
62              "       'TAB' key enables turbospeed.\n"
63              "-----");
64
65     const char *geom      = cimg_option("-g", "300x300", "Size of the game board");
66     const int  delay      = cimg_option("-s", 10, "Game speed (lower value means faster)");
67     const bool twoplayers = !cimg_option("-l", false, "One player only");
68     const int  zoom       = cimg_option("-z", 1, "Zoom factor");
69     const bool full       = cimg_option("-f", false, "Fullscreen mode");
70     unsigned int W = 400, H = 400;
71     std::sscanf(geom, "%u%*c%u", &W, &H);
72

```

```

73 // Define game colors and variables
74 //-----
75 const unsigned char blue[] = { 128,200,255}, red[] = { 255,0,0 }, white[] = { 255,255,255};
76 int score1=0, score2=0, round_over=0, ix1=-1, iy1=-1, x1=0, y1=0, u1=0, v1=0, ix2=-1, iy2=-1, x2=0, y2=0, u2=0, v2=0;
77 bool start_round = true, turbo1 = false, turbo2 = false;
78
79 // Create background image
80 //-----
81 CImg<unsigned char> background, img;
82 background.assign(64,64,1,3,0).noise(60).draw_plasma().resize(W,H).blur(2).normalize();
83
84 // Open display window
85 //-----
86 CImgDisplay disp(background, "* CImg-Tron *");
87 if (zoom>1) disp.resize(-100*zoom,-100*zoom);
88 if (full) disp.toggle_fullscreen().display(background);
89
90 // Start main game loop
91 //-----
92 while (!disp.is_closed() && !disp.is_keyESC()) {
93
94     // Init new game round if necessary
95     //-----
96     if (start_round) {
97
98         // Init game variables
99         round_over = 0;
100         ix1=-1; iy1=-1; x1 = 10; y1 = 10; u1 = 1; v1 = 0; turbo1 = false;
101         ix2=-1; iy2=-1; x2 = W-11; y2 = H-11; u2 = -1; v2 = 0; turbo2 = false;
102         img = background;
103         start_round = false;
104
105         // Display a simple pre-round page
106         CImg<unsigned char> logo, pressakey;
107         logo.draw_text(0,0, " CImg-Tron ",white,0,1,33).resize(-100,-100,1,3);
108         CImg<unsigned char> tmp = (+background).draw_image((W-logo.width())/2,(H-logo.height())/2);
109         draw_text(W/2-60,H/2+10,"Blue ( %u )",blue,0,1,13,score1);
110         draw_text(W/2+10,H/2+10,"Red ( %u )",red,0,1,13,score2);
111         pressakey.draw_text(0,0,"* Press a key to start round *",white);
112         for (float i = 0; i<1; i+=0.05f) ((+tmp)*=i).display(disp.wait(20));
113         disp.flush();
114         for (unsigned long t = 0; !disp.key() && !disp.is_closed(); ++t) {
115             if (!(t%10)) { if (t%20) disp.display(tmp); else disp.display((+tmp).draw_image((W-logo.width())/2,(H-logo.height())/2)); }
116         if (disp.wait(20).is_resized()) disp.resize(disp);
117         }
118         if (disp.is_keyESC()) disp.flush();
119     }

```

```

120
121 // Test collision between players and borders
122 if (x1<0 || x1>=img.width() || y1<0 || y1>=img.height() ||
123     img(x1,y1,0)!=background(x1,y1,0) ||
124     img(x1,y1,1)!=background(x1,y1,1) ||
125     img(x1,y1,2)!=background(x1,y1,2) ||
126     ((ix1>=0 || iy1>=0) && (img(ix1,iy1,0)!=background(ix1,iy1,0) || // Collis
127                             img(ix1,iy1,1)!=background(ix1,iy1,1) ||
128                             img(ix1,iy1,2)!=background(ix1,iy1,2)))) { round_ov
129
130 if (twoplayers) {
131     if (x2<0 || x2>=img.width() || y2<0 || y2>=img.height() ||
132         img(x2,y2,0)!=background(x2,y2,0) ||
133         img(x2,y2,1)!=background(x2,y2,1) ||
134         img(x2,y2,2)!=background(x2,y2,2) ||
135         ((ix2>=0 || iy2>=0) && (img(ix2,iy2,0)!=background(ix2,iy2,0) || // Coll
136                             img(ix2,iy2,1)!=background(ix2,iy2,1) ||
137                             img(ix2,iy2,2)!=background(ix2,iy2,2)))) { round_
138
139 // Draw new players positions
140 img.draw_point(x1,y1,blue);
141 if (ix1>=0 && iy1>=0) img.draw_point(ix1,iy1,blue);
142 if (twoplayers) {
143     img.draw_point(x2,y2,red);
144     if (ix2>=0 && iy2>=0) img.draw_point(ix2,iy2,red);
145 }
146 if (disp.is_resized()) disp.resize(disp);
147 img.display(disp);
148
149 // Update players positions
150 x1+=u1; y1+=v1;
151 if (turbo1) { ix1 = x1; iy1 = y1; x1+=u1; y1+=v1; } else { ix1 = iy1 = -1; }
152 if (twoplayers) {
153     x2+=u2; y2+=v2;
154     if (turbo2) { ix2 = x2; iy2 = y2; x2+=u2; y2+=v2; } else { ix2 = iy2 = -1; }
155 }
156
157 // Test keyboard events
158 int nul = u1, nv1 = v1, nu2 = u2, nv2 = v2;
159 if (disp.is_keyARROWLEFT()) { nul = -1; nv1 = 0; }
160 if (disp.is_keyARROWRIGHT()) { nul = 1; nv1 = 0; }
161 if (disp.is_keyARROWUP()) { nul = 0; nv1 = -1; }
162 if (disp.is_keyARROWDOWN()) { nul = 0; nv1 = 1; }
163 turbo1 = disp.is_keyCTRLRIGHT();
164 if (twoplayers) {
165     if (disp.is_keyA()) { nu2 = -1; nv2 = 0; }
166     if (disp.is_keyD()) { nu2 = 1; nv2 = 0; }

```

```
167     if (disp.is_keyW()) { nu2 = 0; nv2 = -1; }
168     if (disp.is_keyS()) { nu2 = 0; nv2 = 1; }
169     turbo2 = disp.is_keyTAB();
170 }
171 if (nul!==-u1 && nv1!==-v1) { u1 = nul; v1 = nv1; }
172 if (nu2!==-u2 && nv2!==-v2) { u2 = nu2; v2 = nv2; }
173
174 // Check if round is over.
175 if (round_over) {
176     const int xc = round_over==1?x1:x2, yc = round_over==1?y1:y2;
177     for (int r=0; r<50; r+=3) img.draw_circle(xc,yc,r,round_over==1?blue:red,r/30);
178     for (int rr=0; rr<50; rr+=3)
179         ((+img)*=(50-rr)/50.0f).draw_circle(xc,yc,(50+rr),round_over==1?blue:red,1/50);
180     start_round = true;
181 }
182
183 // Wait a small amount of time
184 disp.wait(delay);
185 }
186 return 0;
187 }
```

Manage your dependencies

Check the dependencies of the project with **bii deps**:

```
$ bii deps
INFO: Processing changes...
your_user/cimg_example depends on:
  unresolved:
    CImg.h
```

Now, edit the *biicode.conf* file generated in the project folder. Add your [requirements] depending on the version you want and map your [includes]:

```
[requirements]
  tschumperle/cimg: 4

[includes]
  CImg.h: tschumperle/cimg
```

Now, checking again **bii deps** shows all resolved dependencies.

If you're not using Windows OS, you might need the external X11 library dependency, check it!

Linux OS:


```
$ sudo apt-get install libx11-dev
```

Mac OS:

Go to [XQuartz home page](#), download the package and install it.

You might need to modify the CMakeLists.txt of your block in order to include the external X11 dependency in the linking process:

```
include(${CMAKE_HOME_DIRECTORY}/biicode.cmake)

ADD_BII_TARGETS ()

IF (APPLE)
    FIND_PACKAGE (X11)
    TARGET_LINK_LIBRARIES (${BII_BLOCK_TARGETS} PUBLIC ${X11_LIBRARIES})
    INCLUDE_DIRECTORIES (/opt/X11/include)
ENDIF ()

IF (UNIX)
    IF (NOT APPLE)
        TARGET_LINK_LIBRARIES (${BII_BLOCK_TARGETS} PUBLIC X11)
    ENDIF ()
ENDIF ()
```

Build the project

Now, build and run your Tron game!

```
$ bii build
$ cd bin
$ # Run the executable
```

Open and build

This example is already in biicode: [examples/cimg](#). Give it a try, just open a new project and open the block.

```
$ bii init cimg_example
$ cd cimg_example
$ bii open examples/cimg
$ bii build
$ cd bin
$ # Run the generated executable
$ ./examples_cimg_tron # on linux
$ examples_cimg_tron.exe # on windows
```

Note that this example block already includes all required modifications above in both *biicode.conf* and *CMakeLists.txt*.

Any doubts? Do not hesitate to [contact us](#) visit our [forum](#) and feel free to ask any questions.

1.8.6 Crypto++

Crypto++ is a C++ class library of cryptographic algorithms and schemes written by Wei Dai. For more information about this library, visit their [official website](#) or [wiki](#).

Crypto++ library is allocated and ready to use at [cryptopp/cryptopp](#).

Encrypt a message

The following example shows how to encrypt a message with SHA1 code.

Create a new project

Start a new project and copy the code below:

```
$ bii init cryptopp_example -L
$ cd cryptopp_example
$ # Create main_cryptopp.cpp
$ # Copy the code
```

main_crypto.cpp

```
1 #include "sha.h"
2 #include "filters.h"
3 #include "hex.h"
4 #include <iostream>
5 #include <string>
6
7 int main() {
8     CryptoPP::SHA1 sha1;
9     std::string source = "Hello"; //This will be randomly generated somehow
10    std::string hash = "";
11    CryptoPP::StringSource(source, true, new CryptoPP::HashFilter(sha1, new Cry
12    std::cout << hash;
13 }
```

Manage your dependencies

Check the dependencies of the project with **bii deps**:

```
$ bii deps
INFO: Processing changes...
your_user/cryptopp depends on:
  system:
    iostream
    string
  unresolved:
    filters.h
    hex.h
    sha.h
```

Edit the *biicode.conf* file generated in the project folder. Add your `[requirements]` depending on the version you want and map your `[includes]`:

```
[requirements]
  cryptopp/cryptopp: 8

[includes]
  *.h: cryptopp/cryptopp
```

Check again with **bii deps** to show all dependencies are now resolved.

Build the project

Now, build and run the encryption code.

```
$ bii build
$ cd bin
$ # run executable
F7FF9E8B7BB2E09B70935A5D785E0CC5D9D0ABF0
```

Open and build

You can find this example in [the biicode crypto samples block](#). See how it works in a few steps [here](#).

Create a project:

```
$ bii init cryptopp_example
$ cd cryptopp_example
$ bii open examples/cryptopp
```

You will see next console output after executing the command:

```
$ ./bin/examples_cryptopp_main_crypto
F7FF9E8B7BB2E09B70935A5D785E0CC5D9D0ABF0
```

1.8.7 CSparse

CSparse is a C library which implements a number of direct methods for sparse linear systems, by Timothy Davis.

Csparse library can be used form , this block is generated from this .

Read a matrix and solve a linear system

With this example you can read a matrix saved in a file and solve a linear system.

You need *cs_demo.h* and *cs_demo.c* to encapsule some functions to use in the example and a matrix file *t1*. Then, *cs_demo2.c* implements the main function. Let's do it!

Create a new project

```
$ bii init csparse_example -L
$ cd csparse_example
$ # copy the files below
```

cs_demo.h

```
1 #include "cs.h"
2
3 typedef struct problem_struct
4 {
5     cs *A ;
6     cs *C ;
7     csi sym ;
8     double *x ;
9     double *b ;
10    double *resid ;
11 } problem ;
12
13 problem *get_problem (FILE *f, double tol) ;
14 csi demo2 (problem *Prob) ;
15 csi demo3 (problem *Prob) ;
16 problem *free_problem (problem *Prob) ;
```

cs_demo.c

```
1 #include "cs_demo.h"
2 #include <time.h>
3 /* 1 if A is square & upper tri., -1 if square & lower tri., 0 otherwise */
4 static csi is_sym (cs *A)
5 {
6     csi is_upper, is_lower, j, p, n = A->n, m = A->m, *Ap = A->p, *Ai = A->i ;
```

```

7   if (m != n) return (0) ;
8   is_upper = 1 ;
9   is_lower = 1 ;
10  for (j = 0 ; j < n ; j++)
11  {
12      for (p = Ap [j] ; p < Ap [j+1] ; p++)
13      {
14          if (Ai [p] > j) is_upper = 0 ;
15          if (Ai [p] < j) is_lower = 0 ;
16      }
17  }
18  return (is_upper ? 1 : (is_lower ? -1 : 0)) ;
19 }
20
21 /* true for off-diagonal entries */
22 static csi dropdiag (csi i, csi j, double aij, void *other) { return (i != j) ;}
23
24 /* C = A + triu(A,1)' */
25 static cs *make_sym (cs *A)
26 {
27     cs *AT, *C ;
28     AT = cs_transpose (A, 1) ;          /* AT = A' */
29     cs_fkeep (AT, &dropdiag, NULL) ;    /* drop diagonal entries from AT */
30     C = cs_add (A, AT, 1, 1) ;          /* C = A+AT */
31     cs_spfree (AT) ;
32     return (C) ;
33 }
34
35 /* create a right-hand side */
36 static void rhs (double *x, double *b, csi m)
37 {
38     csi i ;
39     for (i = 0 ; i < m ; i++) b [i] = 1 + ((double) i) / m ;
40     for (i = 0 ; i < m ; i++) x [i] = b [i] ;
41 }
42
43 /* infinity-norm of x */
44 static double norm (double *x, csi n)
45 {
46     csi i ;
47     double normx = 0 ;
48     for (i = 0 ; i < n ; i++) normx = CS_MAX (normx, fabs (x [i])) ;
49     return (normx) ;
50 }
51
52 /* compute residual, norm(A*x-b,inf) / (norm(A,1)*norm(x,inf) + norm(b,inf)) */
53 static void print_resid (csi ok, cs *A, double *x, double *b, double *resid)

```

```

54 {
55     csi i, m, n ;
56     if (!ok) { printf ("      (failed)\n") ; return ; }
57     m = A->m ; n = A->n ;
58     for (i = 0 ; i < m ; i++) resid [i] = -b [i] ; /* resid = -b */
59     cs_gaxpy (A, x, resid) ; /* resid = resid + A*x */
60     printf ("resid: %8.2e\n", norm (resid,m) / ((n == 0) ? 1 :
61         (cs_norm (A) * norm (x,n) + norm (b,m)))) ;
62 }
63
64 static double tic (void) { return (clock () / (double) CLOCKS_PER_SEC) ; }
65 static double toc (double t) { double s = tic () ; return (CS_MAX (0, s-t)) ; }
66
67 static void print_order (csi order)
68 {
69     switch (order)
70     {
71         case 0: printf ("natural      ") ; break ;
72         case 1: printf ("amd(A+A')   ") ; break ;
73         case 2: printf ("amd(S'*S)   ") ; break ;
74         case 3: printf ("amd(A'*A)   ") ; break ;
75     }
76 }
77
78 /* read a problem from a file; use %g for integers to avoid csi conflicts */
79 problem *get_problem (FILE *f, double tol)
80 {
81     cs *T, *A, *C ;
82     csi sym, m, n, mn, nz1, nz2 ;
83     problem *Prob ;
84     Prob = cs_calloc (1, sizeof (problem)) ;
85     if (!Prob) return (NULL) ;
86     T = cs_load (f) ; /* load triplet matrix T from a file */
87     Prob->A = A = cs_compress (T) ; /* A = compressed-column form of T */
88     cs_spfree (T) ; /* clear T */
89     if (!cs_dupl (A)) return (free_problem (Prob)) ; /* sum up duplicates */
90     Prob->sym = sym = is_sym (A) ; /* determine if A is symmetric */
91     m = A->m ; n = A->n ;
92     mn = CS_MAX (m,n) ;
93     nz1 = A->p [n] ;
94     cs_dropzeros (A) ; /* drop zero entries */
95     nz2 = A->p [n] ;
96     if (tol > 0) cs_droptol (A, tol) ; /* drop tiny entries (just to test) */
97     Prob->C = C = sym ? make_sym (A) : A ; /* C = A + triu(A,1)', or C=A */
98     if (!C) return (free_problem (Prob)) ;
99     printf ("\n--- Matrix: %g-by-%g, nnz: %g (sym: %g: nnz %g), norm: %8.2e\n",
100         (double) m, (double) n, (double) (A->p [n]), (double) sym,

```

```

101         (double) (sym ? C->p [n] : 0), cs_norm (C)) ;
102     if (nz1 != nz2) printf ("zero entries dropped: %g\n", (double) (nz1 - nz2));
103     if (nz2 != A->p [n]) printf ("tiny entries dropped: %g\n",
104         (double) (nz2 - A->p [n])) ;
105     Prob->b = cs_malloc (mn, sizeof (double)) ;
106     Prob->x = cs_malloc (mn, sizeof (double)) ;
107     Prob->resid = cs_malloc (mn, sizeof (double)) ;
108     return ((!Prob->b || !Prob->x || !Prob->resid) ? free_problem (Prob) : Prob) ;
109 }
110
111 /* free a problem */
112 problem *free_problem (problem *Prob)
113 {
114     if (!Prob) return (NULL) ;
115     cs_spfree (Prob->A) ;
116     if (Prob->sym) cs_spfree (Prob->C) ;
117     cs_free (Prob->b) ;
118     cs_free (Prob->x) ;
119     cs_free (Prob->resid) ;
120     return (cs_free (Prob)) ;
121 }
122
123 /* solve a linear system using Cholesky, LU, and QR, with various orderings */
124 csi demo2 (problem *Prob)
125 {
126     cs *A, *C ;
127     double *b, *x, *resid, t, tol ;
128     csi k, m, n, ok, order, nb, ns, *r, *s, *rr, sprank ;
129     csd *D ;
130     if (!Prob) return (0) ;
131     A = Prob->A ; C = Prob->C ; b = Prob->b ; x = Prob->x ; resid = Prob->resid;
132     m = A->m ; n = A->n ;
133     tol = Prob->sym ? 0.001 : 1 ; /* partial pivoting tolerance */
134     D = cs_dmperm (C, 1) ; /* randomized dmperm analysis */
135     if (!D) return (0) ;
136     nb = D->nb ; r = D->r ; s = D->s ; rr = D->rr ;
137     sprank = rr [3] ;
138     for (ns = 0, k = 0 ; k < nb ; k++)
139     {
140         ns += ((r [k+1] == r [k]+1) && (s [k+1] == s [k]+1)) ;
141     }
142     printf ("blocks: %g singletons: %g structural rank: %g\n",
143         (double) nb, (double) ns, (double) sprank) ;
144     cs_dfree (D) ;
145     for (order = 0 ; order <= 3 ; order += 3) /* natural and amd(A'*A) */
146     {
147         if (!order && m > 1000) continue ;

```

```

148     printf ("QR  ") ;
149     print_order (order) ;
150     rhs (x, b, m) ;                                /* compute right-hand side */
151     t = tic () ;
152     ok = cs_qrsol (order, C, x) ;                    /* min norm(Ax=b) with QR */
153     printf ("time: %8.2f ", toc (t)) ;
154     print_resid (ok, C, x, b, resid) ;                /* print residual */
155 }
156 if (m != n || sprank < n) return (1) ;              /* return if rect. or singular*/
157 for (order = 0 ; order <= 3 ; order++)              /* try all orderings */
158 {
159     if (!order && m > 1000) continue ;
160     printf ("LU  ") ;
161     print_order (order) ;
162     rhs (x, b, m) ;                                /* compute right-hand side */
163     t = tic () ;
164     ok = cs_lusol (order, C, x, tol) ;                /* solve Ax=b with LU */
165     printf ("time: %8.2f ", toc (t)) ;
166     print_resid (ok, C, x, b, resid) ;                /* print residual */
167 }
168 if (!Prob->sym) return (1) ;
169 for (order = 0 ; order <= 1 ; order++)              /* natural and amd(A+A') */
170 {
171     if (!order && m > 1000) continue ;
172     printf ("Chol ") ;
173     print_order (order) ;
174     rhs (x, b, m) ;                                /* compute right-hand side */
175     t = tic () ;
176     ok = cs_cholsol (order, C, x) ;                  /* solve Ax=b with Cholesky */
177     printf ("time: %8.2f ", toc (t)) ;
178     print_resid (ok, C, x, b, resid) ;                /* print residual */
179 }
180 return (1) ;
181 }
182
183 /* free workspace for demo3 */
184 static csi done3 (csi ok, css *S, csn *N, double *y, cs *W, cs *E, csi *p)
185 {
186     cs_sfree (S) ;
187     cs_nfree (N) ;
188     cs_free (y) ;
189     cs_spfree (W) ;
190     cs_spfree (E) ;
191     cs_free (p) ;
192     return (ok) ;
193 }
194

```



```

195 /* Cholesky update/downdate */
196 csi demo3 (problem *Prob)
197 {
198     cs *A, *C, *W = NULL, *WW, *WT, *E = NULL, *W2 ;
199     csi n, k, *Li, *Lp, *Wi, *Wp, p1, p2, *p = NULL, ok ;
200     double *b, *x, *resid, *y = NULL, *Lx, *Wx, s, t, t1 ;
201     css *S = NULL ;
202     csn *N = NULL ;
203     if (!Prob || !Prob->sym || Prob->A->n == 0) return (0) ;
204     A = Prob->A ; C = Prob->C ; b = Prob->b ; x = Prob->x ; resid = Prob->resid;
205     n = A->n ;
206     if (!Prob->sym || n == 0) return (1) ;
207     rhs (x, b, n) ; /* compute right-hand side */
208     printf ("\nchol then update/downdate ") ;
209     print_order (1) ;
210     y = cs_malloc (n, sizeof (double)) ;
211     t = tic () ;
212     S = cs_schol (1, C) ; /* symbolic Chol, amd(A+A') */
213     printf ("\nsymbolic chol time %8.2f\n", toc (t)) ;
214     t = tic () ;
215     N = cs_chol (C, S) ; /* numeric Cholesky */
216     printf ("numeric chol time %8.2f\n", toc (t)) ;
217     if (!S || !N || !y) return (done3 (0, S, N, y, W, E, p)) ;
218     t = tic () ;
219     cs_ipvec (S->pinv, b, y, n) ; /* y = P*b */
220     cs_lsolve (N->L, y) ; /* y = L\y */
221     cs_ltsolve (N->L, y) ; /* y = L'\y */
222     cs_pvec (S->pinv, y, x, n) ; /* x = P'*y */
223     printf ("solve chol time %8.2f\n", toc (t)) ;
224     printf ("original: ") ;
225     print_resid (1, C, x, b, resid) ; /* print residual */
226     k = n/2 ; /* construct W */
227     W = cs_spalloc (n, 1, n, 1, 0) ;
228     if (!W) return (done3 (0, S, N, y, W, E, p)) ;
229     Lp = N->L->p ; Li = N->L->i ; Lx = N->L->x ;
230     Wp = W->p ; Wi = W->i ; Wx = W->x ;
231     Wp [0] = 0 ;
232     p1 = Lp [k] ;
233     Wp [1] = Lp [k+1] - p1 ;
234     s = Lx [p1] ;
235     srand (1) ;
236     for ( ; p1 < Lp [k+1] ; p1++)
237     {
238         p2 = p1 - Lp [k] ;
239         Wi [p2] = Li [p1] ;
240         Wx [p2] = s * rand () / ((double) RAND_MAX) ;
241     }

```

```

242     t = tic () ;
243     ok = cs_updown (N->L, +1, W, S->parent) ;    /* update: L*L'+W*W' */
244     t1 = toc (t) ;
245     printf ("update:   time: %8.2f\n", t1) ;
246     if (!ok) return (done3 (0, S, N, y, W, E, p)) ;
247     t = tic () ;
248     cs_ipvec (S->pinv, b, y, n) ;                /* y = P*b */
249     cs_lsolve (N->L, y) ;                        /* y = L\y */
250     cs_ltsolve (N->L, y) ;                      /* y = L'\y */
251     cs_pvec (S->pinv, y, x, n) ;                /* x = P'*y */
252     t = toc (t) ;
253     p = cs_pinv (S->pinv, n) ;
254     W2 = cs_permute (W, p, NULL, 1) ;           /* E = C + (P'W)*(P'W)' */
255     WT = cs_transpose (W2, 1) ;
256     WW = cs_multiply (W2, WT) ;
257     cs_spfree (WT) ;
258     cs_spfree (W2) ;
259     E = cs_add (C, WW, 1, 1) ;
260     cs_spfree (WW) ;
261     if (!E || !p) return (done3 (0, S, N, y, W, E, p)) ;
262     printf ("update:   time: %8.2f (incl solve) ", t1+t) ;
263     print_resid (1, E, x, b, resid) ;           /* print residual */
264     cs_nfree (N) ;                             /* clear N */
265     t = tic () ;
266     N = cs_chol (E, S) ;                       /* numeric Cholesky */
267     if (!N) return (done3 (0, S, N, y, W, E, p)) ;
268     cs_ipvec (S->pinv, b, y, n) ;                /* y = P*b */
269     cs_lsolve (N->L, y) ;                        /* y = L\y */
270     cs_ltsolve (N->L, y) ;                      /* y = L'\y */
271     cs_pvec (S->pinv, y, x, n) ;                /* x = P'*y */
272     t = toc (t) ;
273     printf ("rechol:   time: %8.2f (incl solve) ", t) ;
274     print_resid (1, E, x, b, resid) ;           /* print residual */
275     t = tic () ;
276     ok = cs_updown (N->L, -1, W, S->parent) ;    /* downdate: L*L'-W*W' */
277     t1 = toc (t) ;
278     if (!ok) return (done3 (0, S, N, y, W, E, p)) ;
279     printf ("downdate: time: %8.2f\n", t1) ;
280     t = tic () ;
281     cs_ipvec (S->pinv, b, y, n) ;                /* y = P*b */
282     cs_lsolve (N->L, y) ;                        /* y = L\y */
283     cs_ltsolve (N->L, y) ;                      /* y = L'\y */
284     cs_pvec (S->pinv, y, x, n) ;                /* x = P'*y */
285     t = toc (t) ;
286     printf ("downdate: time: %8.2f (incl solve) ", t1+t) ;
287     print_resid (1, C, x, b, resid) ;           /* print residual */
288     return (done3 (1, S, N, y, W, E, p)) ;

```

289

}

cs_demo2.c

```

1 #include "cs_demo.h"
2 /* cs_demo2: read a matrix and solve a linear system */
3 int main (void)
4 {
5     problem *Prob = get_problem (stdin, 1e-14) ;
6     demo2 (Prob) ;
7     free_problem (Prob) ;
8     return (0) ;
9 }

```

Place *t1* file in a new folder called *Matrix*:

Matrix/t1

```

1 2 2 3.0
2 1 0 3.1
3 3 3 1.0
4 0 2 3.2
5 1 1 2.9
6 3 0 3.5
7 3 1 0.4
8 1 3 0.9
9 0 0 4.5
10 2 1 1.7

```

Manage your dependencies

Create a *biicode.conf* file in the project folder. Add your [requirements] depending on tdavis/csparse and map your [includes]:

```

[requirements]
    tdavis/csparse: 1

[includes]
    *.h: tdavis/csparse/Include

```

Check with **bii deps** to show all dependencies are resolved.

Build the project

Now, build and run the example

```
$ bii build
$ cd bin
$ your_user_csparse_example_cs_demo2 < ../Matrix/t1
$ # NOTE "your_user" should be your user's name

--- Matrix: 4-by-4, nnz: 10 (sym: 0: nnz 0), norm: 1.11e+001
blocks: 1 singletons: 0 structural rank: 4
QR  natural      time:      0.00 resid: 1.15e-017
QR  amd(A'*A)    time:      0.00 resid: 1.53e-017
LU  natural      time:      0.00 resid: 1.04e-017
LU  amd(A+A')    time:      0.00 resid: 4.94e-018
LU  amd(S'*S)    time:      0.00 resid: 4.94e-018
LU  amd(A'*A)    time:      0.00 resid: 4.94e-018
```

Open and build

You can check all the csparse examples in .

Give it a quick try following the next steps.

Create a new project and open the examples.

```
~$ bii init csparse_example
~$ cd csparse_example
~$ bii open examples/csparse
~$ bii build
```

Execute any you want, for example, read a matrix saved in a file and solve a linear system:

```
$ cd bin
$ examples_csparse_cs_demo2 < ../blocks/examples/csparse/Matrix/t1

--- Matrix: 4-by-4, nnz: 10 (sym: 0: nnz 0), norm: 1.11e+001
blocks: 1 singletons: 0 structural rank: 4
QR  natural      time:      0.00 resid: 1.15e-017
QR  amd(A'*A)    time:      0.00 resid: 1.53e-017
LU  natural      time:      0.00 resid: 1.04e-017
LU  amd(A+A')    time:      0.00 resid: 4.94e-018
LU  amd(S'*S)    time:      0.00 resid: 4.94e-018
LU  amd(A'*A)    time:      0.00 resid: 4.94e-018
```

1.8.8 cURL

cURL is a command line tool and library for transferring data with URL syntax, supporting DICT, FILE, FTP, FTPS, Gopher, HTTP, HTTPS, IMAP, IMAPS, POP3, POP3S, SCP, SMTP, SMTPS,

Telnet, TFTP... and supports SSL certificates, HTTP POST, HTTP PUT, FTP uploading, HTTP form based upload...

You can check [cURL documentation](#) for more information.

cURL library is stored at [lasote/curl](#) generated from this [github repo](#).

HTML page gatherer

This example demonstrates some basics using **cURL**. You'll learn to open a website URL, show and copy to a file its html's code.

Creating a new project

Create a new project and a *html-page.cpp* inside like this:

```
$ bii init curl_example -L
$ cd curl_example
$ # Create html-page.cpp
```

The code of the example is this one, it goes to [biicode's docs website](#) and copies the html code and streams it (copies it to a file, copies it into a string and shows it in the terminal).

Copy the code in *html-page.cpp*:

html-page.cpp

```
#include <curl/curl.h>
#include <fstream>
#include <sstream>
#include <iostream>

using namespace std;

// callback function writes data to a std::ostream
static size_t data_write(void* buf, size_t size, size_t nmemb, void* userp)
{
    if(userp)
    {
        std::ostream& os = *static_cast<std::ostream*>(userp);
        std::streamsize len = size * nmemb;
        if(os.write(static_cast<char*>(buf), len))
            return len;
    }

    return 0;
}
```

```
//timeout is in seconds
CURLcode curl_read(const std::string& url, std::ostream& os, long timeout = 30)
{
    CURLcode code(CURLE_FAILED_INIT);
    CURL* curl = curl_easy_init();

    if(curl)
    {
        if(CURLE_OK == (code = curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, &data_writ
        && CURLE_OK == (code = curl_easy_setopt(curl, CURLOPT_NOPROGRESS, 1L))
        && CURLE_OK == (code = curl_easy_setopt(curl, CURLOPT_FOLLOWLOCATION, 1L))
        && CURLE_OK == (code = curl_easy_setopt(curl, CURLOPT_FILE, &os))
        && CURLE_OK == (code = curl_easy_setopt(curl, CURLOPT_TIMEOUT, timeout))
        && CURLE_OK == (code = curl_easy_setopt(curl, CURLOPT_URL, url.c_str()))
        {
            code = curl_easy_perform(curl);
        }
        curl_easy_cleanup(curl);
    }
    return code;
}

int main()
{
    curl_global_init(CURL_GLOBAL_ALL);

    std::ofstream ofs("html-web-output.html");
    if(CURLE_OK == curl_read("http://docs.biicode.com/", ofs))
    {
        cout<<"Web page successfully written to file!!"<<endl;
    }

    std::ostringstream oss;
    if(CURLE_OK == curl_read("http://docs.biicode.com/", oss))
    {
        cout<<"Web page successfully written to string!!"<<endl;
        std::string html = oss.str();
    }

    if(CURLE_OK == curl_read("http://docs.biicode.com/", std::cout))
    {
        cout<<endl<<"Web page successfully written to standard output!!"<<endl;
    }

    curl_global_cleanup();
}
```

Manage your dependencies

Check the dependencies of the project with **bii deps**:

```
$ bii deps
your_user/curl_example depends on:
  system:
    fstream
    iostream
    sstream
  unresolved:
    curl/curl.h
```

Now, edit the *biicode.conf* file generated in the project folder. Add your [requirements] depending on the version you want and map your [includes]:

```
[requirements]
  lasote/curl: 2

[includes]
  curl/*.h: lasote/curl/include
```

Check again with **bii deps** to show all resolved dependencies.

Build the project

Now, build and run the HTML page gatherer.

```
$ bii build
$ cd bin
$ # execute it!
Web page successfully written to file!!
Web page successfully written to string!!
<!DOCTYPE html>
<!--[if IE 8]><html class="no-js lt-ie9" lang="en" > <![endif]-->
...
Web page successfully written to standard output!!
```

Once you execute you should see an output like that one. Go to your bin folder and open *html-web-view.html* in your browser to see **biicode's docs web page**!

Open and build

This and other examples with cURL are already in biicode: [examples/curl](#).

To give this one a try, create a new project and open the block:

```
$ bii init curl_example
$ cd bcl_example
$ bii open examples/curl
```

Build the example and execute it:

```
$ bii build
$ cd bin
$ examples_curl_html-page
Web page successfully written to file!!
Web page successfully written to string!!
<!DOCTYPE html>
<!--[if IE 8]><html class="no-js lt-ie9" lang="en" > <![endif]-->
...
Web page successfully written to standard output!!
```

Any doubts? Do not hesitate to [contact us](#) visit our [forum](#) and feel free to ask any questions.

1.8.9 Eigen

is a high-level C++ library of template headers for linear algebra, matrix and vector operations, numerical solvers and related algorithms.

Eigen library is at [http://eigen.tuxfamily.org](#), which is generated from this [repository](#).

Middle rows from a matrix

This example shows the way to generate a random matrix, print it and then print only the middle rows.

Create a project

Create a simple project and the following file inside it.

```
$ bii init eigen_example -L
$ cd eigen_example
$ # copy DenseBase_middleRows_int.cpp
```

DenseBase_middleRows_int.cpp

```
1 #include <Eigen/Core>
2 #include <iostream>
3
4 using namespace Eigen;
5 using namespace std;
```



```

6
7 int main(void)
8 {
9     int const N = 5;
10    MatrixXi A(N,N);
11    A.setRandom();
12    cout << "A =\n" << A << '\n' << endl;
13    cout << "A(2..3, :) =\n" << A.middleRows(2,2) << endl;
14    return 0;
15 }

```

Manage your dependencies

Check the dependencies of the project with **bii deps**:

```

$ bii deps
INFO: Processing changes...
your_user/eigen_example depends on:
    system:
        iostream
    unresolved:
        Eigen/Core

```

Edit the *biicode.conf* file generated in the project folder. Add your [requirements] depending on the version you want and map your [includes]:

```

[requirements]
    eigen/eigen: 6

[includes]
    Eigen/*: eigen/eigen

```

Check again with **bii deps** to show all dependencies are now resolved.

Build the project

Now, build and run the code.

```

$ bii build
$ cd bin
$ # run executable
A =
-16343   -660 -10679 -15893   16007
  2083   -4906  11761 -13389   -1780
-10050  12974   6897  -4442  -12482
 10116  10578    443 -11557  -16231

```

```
    2785    8080   -6423  -10948  -16092

A(2..3,:) =
-10050  12974   6897   -4442  -12482
 10116  10578    443  -11557  -16231
```

Open and build

You can check all the Eigen examples uploaded in biicode and execute any of them, just have to open .

Create a project and open the examples:

```
~$ bii init eigen_example
~$ cd eigen_example
~$ bii open examples/eigen
~$ bii build
```

Execute any you want, for example, show the matrix's middle rows:

```
~/eigen_sample$ bin/examples_eigen_DenseBase_middleRows_int
A =
-16343   -660  -10679  -15893   16007
   2083  -4906   11761  -13389   -1780
-10050  12974   6897   -4442  -12482
 10116  10578    443  -11557  -16231
   2785    8080   -6423  -10948  -16092

A(2..3,:) =
-10050  12974   6897   -4442  -12482
 10116  10578    443  -11557  -16231
```

1.8.10 Expression Parser

Expression Parser is a C++ library to parse a character sequence as an expression using Dijkstra's [Shunting-yard algorithm](#) which modifies [Jesse Brown's code](#).

The main block is at [amalulla/cpp-expression-parser](#), which is generated from this [github repo](#).

Simple form of mathematical expression parsing

In this example we'll show a simple code to develop a mathematical expression parser, with expressions like "pi" or "gravity".

Create a new project

Create a simple project and the code inside it:

```
$ bii init exp-parser_example -L
$ cd exp-parser_example
$ # copy the code below
```

main.cpp

```
#include <iostream>
#include <limits>
#include <map>
#include <string>
#include "shunting-yard.h"

void calculation(const char* expr,
    std::map<std::string, double>* vars = 0) {
    double result = calculator::calculate(expr, vars);
    std::cout << " " << expr << " calculated result is: " <<
        result << "." << std::endl;
}

int main() {
    std::map<std::string, double> vars;
    vars["gravity"] = 9.78;
    vars["pi"] = 3.14;
    calculation("-pi+1", &vars);
    calculation("10*gravity", &vars);
    return 0;
}
```

Manage your dependencies

Check the dependencies of the project with **bii deps**:

```
$ bii deps
INFO: Processing changes...
your_user/expression_parser depends on:
  system:
    iostream
    limits
    map
    string
  unresolved:
    shunting-yard.h
```

Now, edit the *biicode.conf* file generated in the project folder. Add your `[requirements]` depending on the version you want and map your `[includes]`:

```
[requirements]
    amalulla/cpp-expression-parser: 2

[includes]
    shunting-yard.h: amalulla/cpp-expression-parser
```

Now, checking again **bii deps** shows all resolved dependencies.

Build the project

Just build the project and run this example!

```
$ bii build
$ cd bin
$ # run executable
```

You can see the results of the parsed expressions in the output:

```
$ '-pi+1' calculated result is -2.14.
$ '10*gravity' calculated result is 97.8.
```

Open and build

This example is already in biicode at [examples/expression_parser](#).

To give it a simple run, just open the block and build it like this:

```
$ bii init exp-parser_example
$ cd exp-parser_example
$ bii open examples/expression_parser
$ bii build
$ cd bin
$ # run executable
```

Here is the output:

```
$ '-pi+1' calculated result is -2.14.
$ '10*gravity' calculated result is 97.8.
```

Any doubts? Do not hesitate to [contact us](#) visit our [forum](#) and feel free to ask any questions.

1.8.11 fit

is a C++11 header-only library that provides utilities for functions and function objects. Fit is divided into three components:

1. Function Adapters: These take functions and return a new function that provides an additional capability to the previous function.
2. Functions: These return functions that achieve a specific purpose.
3. Utilities: These are general utilities that are useful when defining or using functions.

The main block is , which is generated from this .

Tests

Calculate giving the coordinates, the angles and sides length of a polygon between 3 and 10 vertex. To start, create a project, open the example and execute:

Note: If you are using CLANG you may need to use `clang --version` to check it's higher than 3.5.

```
~$ bii init fit
~$ cd fit
~/fit$ bii open examples/fit
~/fit$ bii build
~/fit$ #execute the tests on bin folder
```

Now let's check the code, **open one of the test files (test/always.cpp)** :

```
#include <fit/always.h>
#include <memory>
#include "test.h"

FIT_TEST_CASE()
{
    static const int ten = 10;
    FIT_STATIC_TEST_CHECK(fit::always(ten)(1,2,3,4,5) == 10);
    FIT_TEST_CHECK( fit::always(ten)(1,2,3,4,5) == 10 );
}
```

Execute the binary and this is how the output looks like:

```
~/fit$ bin/examples_fit_test_always
**CORRECT** fit::always(ten)(1,2,3,4,5) == 10
```

The output is what always.cpp file is testing, if successes it writes **CORRECT**, if it doesn't it writes **FAILED**.

Didn't work? No problem, read or contact us in .

Any suggestion or feedback? It is very welcomed :)

1.8.12 Flatbuffers

is an efficient cross platform serialization library for C++, with support for Java and Go. It was created at Google specifically for game development and other performance-critical applications.

The main block is , which is generated from this .

Charge a *.fbs file and generate a C++ header

You can check all the flatbuffers examples which are uploaded in biicode and execute any of them. Then, create a new project and open the .

```
~$ bii init flatb_sample
~$ cd flatb_sample
~/flatb_sample$ bii open examples/flatbuffers
~/flatb_sample$ bii build
```

MinGW compiler bug

MinGW users may need to edit *io.h* to avoid building errors. Look for **MinGW/include/io.h** and replace lines 301 and 302:

```
__CRT_INLINE off64_t lseek64 (int, off64_t, int);
__CRT_INLINE off64_t lseek64 (int fd, off64_t offset, int whence)
```

with

```
__CRT_INLINE _off64_t lseek64 (int, _off64_t, int);
__CRT_INLINE _off64_t lseek64 (int fd, _off64_t offset, int whence)
```

Now, you can charge the file “monster.fbs” and generate a C++ header for tables/structs:

```
~/flatb_sample$ cd bin
~/flatb_sample/bin$ examples_flatbuffers_flatc -c ../blocks/examples/flatbuffers/mo
```

Now, you’ll see one single file “monster_generated.h” in your current folder that has been created correctly.

1.8.13 Freeglut

This example demonstrates how to get started using **OpenGL** with biicode. **Freeglut** is an [open source alternative to the OpenGL Utility Toolkit \(GLUT\) library](#). It allows you to create and manage windows containing OpenGL contexts on a wide range of platforms, and dealing with

user input from mouse, keyboard and joystick devices. You can visit the following pages to learn more about OpenGL and GLUT (and hence freeglut):

1. [The Official Guide to Learning OpenGL v1.1](#)
2. [List of OpenGL methods](#)
3. [The Free OpenGL Utility Toolkit](#)
4. [List of GLUT methods](#)

1. Create a new project

First, create a new project as described in the *hello world example*, using the following options:

```
~$ cd cpp_freeglut_project
~/cpp_freeglut_project$ bii new anonymous/cpp_freeglut --hello=cpp
```

2. Creating reusable code

This is some example code that makes use of GLUT functions. Note that **you must include a reference to a biicode GLUT wrapper library**. This is the only information biicode needs to fetch the required files when needed, taking into account your actual development platform. We'll see how this happens in the next step.

Now, simply put the following code into your `cpp_freeglut` block folder (you can also download these files here: `sphere-glut.zip`, unzip and copy them into your block folder):

sphere.h

```
#pragma once
#include "glui/glutwrapper/glut.h"

class Sphere{
public:
    Sphere(float _radio=1.0, int _slices=20, int _stacks=20, bool _solid=true);
    virtual ~Sphere(); //default virtual destructor
    void draw();
    void setPosition(float _x=0.0, float _y=0.0, float _z=0.0){
        x = _x; y = _y; z = _z;
    }
    void setColor(unsigned char r=255, unsigned char g=255, unsigned char b=255){
        red = r; green = g; blue = b;
    }

private:
    float radius;
    int slices;
```

```
        int stacks;
        unsigned char red, green, blue;
        float x,y,z;
        bool solid;
};
```

sphere.cpp

```
#include "sphere.h"

Sphere::Sphere(float _radius, int _slices, int _stacks, bool _solid):
radius(_radius),slices(_slices),stacks(_stacks),solid(_solid){
    setPosition();
    setColor();
}

Sphere::~Sphere(){}

void Sphere::draw(){
    glColor3ub(red,green,blue);
    glTranslatef(x,y,z);
    if (solid)
        glutSolidSphere(radius, slices, stacks);
    else
        glutWireSphere(radius, slices, stacks);
    glTranslatef(-x,-y,-z);
}
```

main.cpp

```
#include "sphere.h"

//Functions declarations
void Init();
void OnDraw();

void Init(){
    //Initialize GLUT windows manager
    //Create the window
    int argc=1;
    char* argv[1]={"Glut Application"};
    glutInit(&argc, argv);
    glutInitWindowSize(800, 600);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutCreateWindow("My World");

    //enable lights and define perspective
    glEnable(GL_LIGHT0);
```



```

    glEnable(GL_LIGHTING);
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_COLOR_MATERIAL);
    glMatrixMode(GL_PROJECTION);
    gluPerspective( 40.0, 800/600.0f, 0.1, 150);
}

void OnDraw(void) {
    //cleaning the screen
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    //Define view
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    // eye position -> (0.0, 10, 20)
    // target -> (0,0,0)
    // define positive Y axis -> (0.0, 1.0, 0.0)
    gluLookAt(0.0, 10, 20,
              0.0, 0, 0.0,
              0.0, 1.0, 0.0);

    //Put your code here to draw objects
    Sphere sphere1;
    sphere1.draw();

    //no delete this line
    glutSwapBuffers();
}

int main(int argc, char** argv) {
    Init();
    //Enter the callbacks
    glutDisplayFunc(OnDraw);

    glutMainLoop(); // begin the loop
    return 0;
}

```

3. Find dependencies

Now is when the biicode magic takes place. The previous code needs to link with some library implementing the drawing functions being used. You can resolve all missing dependencies for your platform using the `bii find` command. **From your project's folder location** execute:

```
$ bii find
```

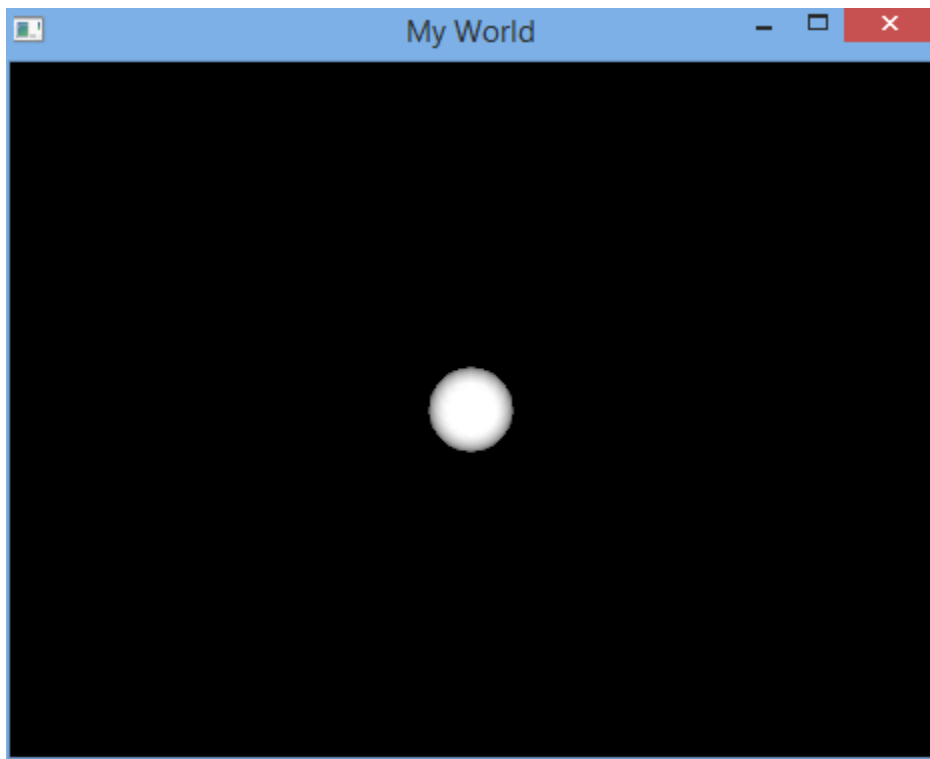
This will find and retrieve all missing files from the biicode servers to your local filesystem, under the `deps` folder of your project. You should see a success message confirming that all dependencies have been resolved.

4. Build and run

Now it is time to see the result of the previous steps. You can build your main file with the `bii cpp:run` command and and run the executable inside the `bin` folder.

```
$ bii build
$ cd bin
$ #run your executable
```

That's it! You should see a new window named “My World”, containing a single white sphere in its center. Just like this!:



If you experience any difficulties during the coding process, or get any errors during the program execution, visit [our forum](#) and feel free to ask any questions.

1.8.14 GLFW

GLFW is a free, Open Source, multi-platform library for OpenGL and OpenGL ES application

development. It provides a simple, platform-independent API for creating windows and contexts, reading input, handling events, etc.

It is a great library, if you want to build OpenGL applications it is highly recommended.

You can find GLFW library at [diego/glfw](#).

Running the examples

It is very simple. Create a **new project**, open the examples, build and run them:

```
$ bii init glfwexamples
$ cd glfwexamples
$ bii open examples/glfw
$ bii build
$ ./bin/examples_glfw_particles
```

Linux users may need to install some additional software

If you are using Ubuntu Linux, you may need to install some additional packages. To do so, execute:

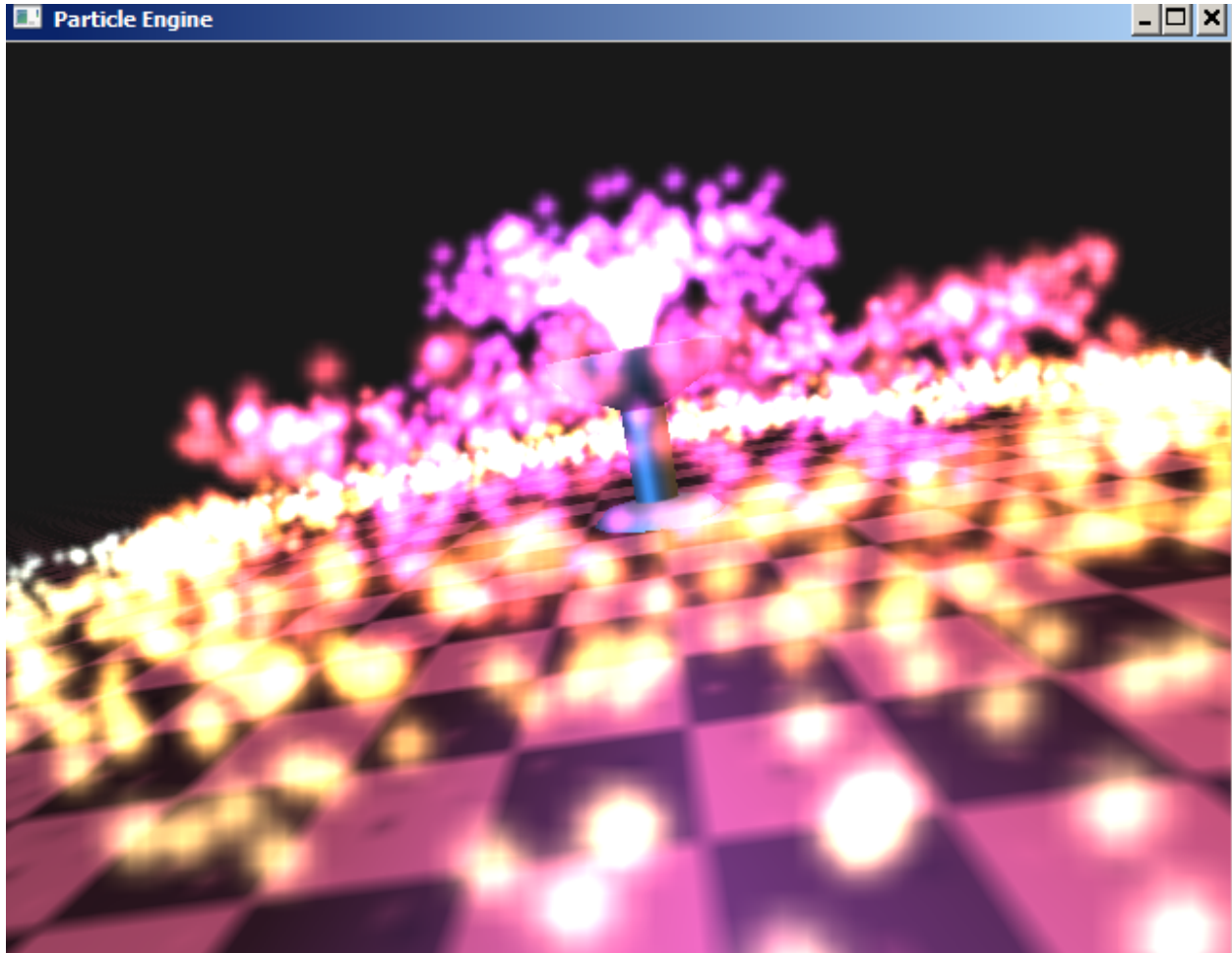
```
$ sudo apt-get install mesa-common-dev libglu1-mesa-dev libxi-dev
```

Depending on your linux setup you might need to install also libxinerama-dev lib libxrandr-dev libxcursor-dev libxxf86vm-dev

If you are in RH - Fedora, you may need to install:

```
$ sudo yum install libGLU-devel libXrandr-devel libXinerama-devel libXcursor-devel
```

You should see in your screen:



There are also other examples that have been built! Enjoy them and GLFW!

Of course, you can also do your own GLFW programs from your own block:

```
$ bii init myproject
$ cd myproject
$ bii new yourusername/yourblock
```

Then copy your source files to yourusername/yourblock folder, change your includes to:

```
#include <diego/glfw/deps/glad/glad.h>
#include <diego/glfw/include/GLFW/glfw3.h>
```

And then, find and retrieve dependencies, build and run your program:

```
$ bii find
$ bii build
```

1.8.15 GLUI User Interface Library

GLUI is a **GLUT-based C++ user interface library** which provides controls such as buttons, checkboxes, radio buttons, and spinners to OpenGL applications. It is window-system independent, relying on **GLUT** to handle all system-dependent issues, such as window and mouse management.

For more information about this library, visit their [official website](#).

This is the [biicode library site](#).

Following, there is an example using this library with biicode technology.

GLUI Window Template

This example is a small modification of the code originally programmed by [Ali Bader Eddin](#), available from [the Code Project](#). In words of the author:

“To avoid having to write the same code every time you want to create an OpenGL graphical application with GUI components, this program code can be used as a template to get you directly started”.

You can explore the **source code block** for this example [following this link](#). In order to try this example, you only need to follow these steps:

Ubuntu users may need to install some additional software

If you are using Ubuntu Linux, you may need to install some additional packages. To do so, execute:

```
sudo apt-get install mesa-common-dev libglu1-mesa-dev libxi-dev
```

1. Create a **new project**:

```
$ bii init <project_name>
$ cd <project_name>
```

2. **Open “examples/glui_example” block.** Download the `examples/glui_example` block to your project’s `blocks` folder. Inside your project folder, execute the command:

```
<project_name>$ bii open examples/glui_example
```

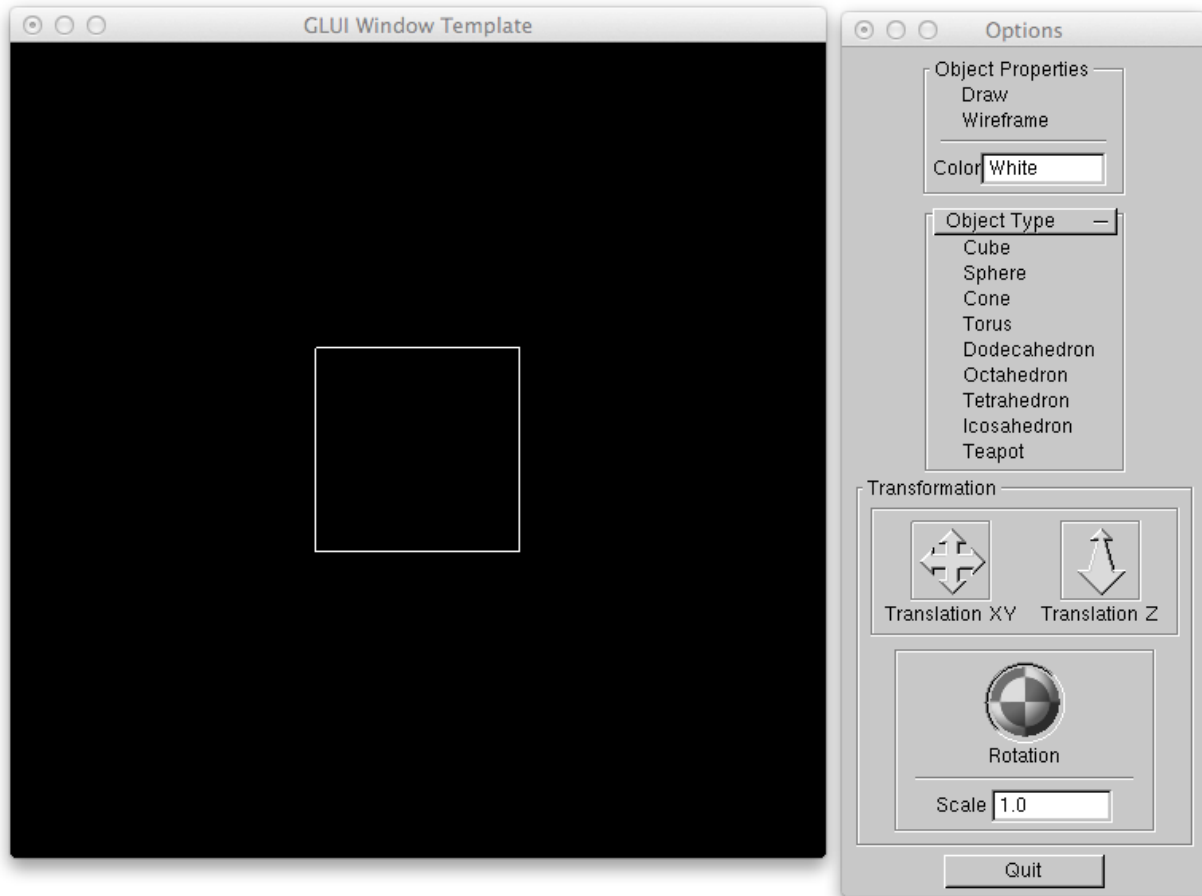
3. **Retrieve all missing dependencies** using the `bii find` command. This way all missing dependencies will be downloaded into the `deps` folder of your project.

```
<project_name>$ bii find
```

4. Finally, **compile your program** using the `bii build` command:

```
<project_name>$ bii build
```

If there were no errors during compilation, you'll find a new executable file inside your project's `bin` folder. If you run this program, you should see something similar to this screen capture. If you found any problems, [please contact us at our forum](#).



Check more GLUI examples at [this block](#) and enjoy using [GLUI library](#)!

1.8.16 Google Mock (GMock)

is a C++ library for writing and using C++ mock classes.

The main block is , which is generated from this .

GMock Examples

All GMock samples are available in biicode: [GMock samples block](#) .

You can execute all of them locally just creating a new project and opening the block:

```

~$ bii init gmock_sample
~$ cd gmock_sample
~/gmock_sample$ bii open google/gmocksamples
~/gmock_sample$ bii build
~/gmock_sample$ #execute any example

```

Mocking a simple function

Let's run an example based in [Google C++ Mocking Framework for Dummies](#) sample. You can execute it locally just creating a new project and opening the block: [GMock example block](#).

```

~$ bii init samples
~$ cd samples
~/samples$ bii open examples/gmock
~/samples$ bii build
~/samples$ bin/examples_gmock_test_mock_turtle_test
    [=====] Running 1 test from 1 test case.
    [-----] Global test environment set-up.
    [-----] 1 test from PainterTest
    [ RUN      ] PainterTest.CanDrawSomething
    [          OK ] PainterTest.CanDrawSomething (0 ms)
    [-----] 1 test from PainterTest (0 ms total)

    [-----] Global test environment tear-down
    [=====] 1 test from 1 test case ran. (2 ms total)
    [ PASSED   ] 1 test.

```

Here is the main code used in this example, you can navigate it on-line here: [GMock example block](#).

turtle.h

```

#pragma once

class Turtle {

public:

    virtual ~Turtle() {}
    virtual void PenUp() = 0;
    virtual void PenDown() = 0;
    virtual void Forward(int distance) = 0;
    virtual void Turn(int degrees) = 0;
    virtual void GoTo(int x, int y) = 0;
    virtual int GetX() const = 0;
    virtual int GetY() const = 0;

```

```
};
```

painter.h

```
#pragma once
#include "turtle.h"

class Painter
{
    Turtle* turtle;
public:
    Painter( Turtle* turtle )
        :      turtle(turtle){}

    bool DrawCircle(int, int, int){
        turtle->PenDown();
        return true;
    }
};
```

mock_turtle.h

```
#pragma once

#include "turtle.h"
#include "google/gmock/gmock.h" // Brings in Google Mock

class MockTurtle : public Turtle {
public:

    MOCK_METHOD0(PenUp, void());
    MOCK_METHOD0(PenDown, void());
    MOCK_METHOD1(Forward, void(int distance));
    MOCK_METHOD1(Turn, void(int degrees));
    MOCK_METHOD2(GoTo, void(int x, int y));
    MOCK_CONST_METHOD0(GetX, int());
    MOCK_CONST_METHOD0(GetY, int());
};
```

mock_turtle_test.cc

```
#include "../mock_turtle.h"
#include "../painter.h"

#include "google/gtest/gtest.h"

using ::testing::AtLeast;
TEST(PainterTest, CanDrawSomething) {
```



```

MockTurtle turtle;
EXPECT_CALL(turtle, PenDown())
    .Times(AtLeast(1));

Painter painter(&turtle);

EXPECT_TRUE(painter.DrawCircle(0, 0, 10));
}

int main(int argc, char** argv) {
    // The following line must be executed to initialize Google Mock
    // (and Google Test) before running the tests.
    ::testing::InitGoogleMock(&argc, argv);
    return RUN_ALL_TESTS();
}

```

You can aggregate as many tests as you want and verify all the methods actions, returns, calls, etc.

1.8.17 Google Test (GTest)

is a C++ library for testing your projects.

The main block is at and it is generated from this .

You can check all the [gtest examples](#) which are uploaded in biicode and execute any of them.

Testing a factorial function: Simple test

There are two examples to build and execute, a simple test and a test suites. In this part, you'll run a simple test for a simple “math example code”.

Creating a new project

Create a new project and *math_ext.h*, *math_ext.cpp* files:

```

$ bii init gtest_example -L
$ cd gtest_example
$ # Create files manually

```

Our math example is a simple function to **calculate the factorial of an integer number**, which returns

- -1 if the number is negative,
- 1 if the number is zero, or

- The factorial if the number is positive: $(num)*(num-1)*(num-2)*\dots*1$.

Copy the code for each file:

math_ext.h

```
#pragma once
//Function returns the factorial of an integer number
int Factorial (int num);
```

math_ext.cpp

```
#include "math_ext.h"
int Factorial (int num) {
    if (!num) return 1;
    if (num<0) return -1;
    return num*Factorial(num-1);
}
```

Okay, now create a new directory *simple_test* with *simple_test.cpp* file and copy its content:

simple_test/simple_test.cpp

```
#include "../math_ext.h"
#include "gtest/gtest.h"
#include "stdio.h"
// Tests Factorial()
// Tests factorial of negative numbers.
TEST(FactorialTest, Negative) {
    // This test is named "Negative", and belongs to the "FactorialTest"
    // test case
    EXPECT_EQ(-1, Factorial(-5));
    EXPECT_EQ(-1, Factorial(-1));
    EXPECT_LT(Factorial(-10), 0);
}
// Tests factorial of 0.
TEST(FactorialTest, Zero) {
    EXPECT_EQ(1, Factorial(0));
}
// Tests factorial of positive numbers.
TEST(FactorialTest, Positive) {
    EXPECT_EQ(1, Factorial(1));
    EXPECT_EQ(2, Factorial(2));
    EXPECT_EQ(6, Factorial(3));
    EXPECT_EQ(40320, Factorial(8));
}
```

```

}
int main(int argc, char **argv) {
    printf("Running main() from sample1\n");
    testing::InitGoogleTest(&argc, argv);
    return RUN_ALL_TESTS();
}

```

Configure biicode.conf

Check the dependencies of the project with **bii deps**:

```

$ bii deps
INFO: Processing changes...
your_user/gtest_example depends on:
    system:
        stdio.h
    unresolved:
        gtest/gtest.h

```

Now, edit the *biicode.conf* file generated in the project folder. Add your [requirements] depending on the version you want and map your [includes]:

biicode.conf

```

[requirements]
    google/gtest: 10

[includes]
    gtest/*.h: google/gtest/include

```

Type **bii deps** again to check all dependencies are resolved.

To configure the *simple_test.cpp* you have to include it in [tests] section like this:

biicode.conf

```

[tests]
    simple_test/simple_test.cpp

```

Build and run the test

Compile and execute the test, the convenient command for that is **bii test**:

```

$ bii test
INFO: Processing changes...
INFO: Saving files from: google/gtest
...

```

```
test 1
  Start 1: examples_gtest_example_simple_test_simple_test
1: Running main() from sample1
1: [=====] Running 3 tests from 1 test case.
1: [-----] Global test environment set-up.
1: [-----] 3 tests from FactorialTest
1: [ RUN      ] FactorialTest.Negative
1: [          OK ] FactorialTest.Negative (0 ms)
1: [ RUN      ] FactorialTest.Zero
1: [          OK ] FactorialTest.Zero (0 ms)
1: [ RUN      ] FactorialTest.Positive
1: [          OK ] FactorialTest.Positive (0 ms)
1: [-----] 3 tests from FactorialTest (0 ms total)
1:
1: [-----] Global test environment tear-down
1: [=====] 3 tests from 1 test case ran. (1 ms total)
1: [ PASSED   ] 3 tests.
1/1 Test #1: examples_gtest_example_simple_test_simple_test ... Passed 0.04 se

100% tests passed, 0 tests failed out of 1

Total Test time (real) = 0.06 sec
[100%] Built target check
```

Great! Your math code passed the simple test.

Testing a factorial function: Test suites

To have multiple test classes and run them all together you don't need to create multiple main methods, you just have to indicate which tests you want to include in your suite.

For example, you can split the *simple_test.cpp* in the following three files.

Place these three files in a new directory called *test_suites* inside your project.

test_suites/test_factorial1.cpp

```
#include "../math_ext.h"
#include "gtest/gtest.h"
// Tests Factorial()
// Tests factorial of negative numbers.
TEST(FactorialTest, Negative) {
    // This test is named "Negative", and belongs to the "FactorialTest"
    // test case
    EXPECT_EQ(-1, Factorial(-5));
    EXPECT_EQ(-1, Factorial(-1));
    EXPECT_LT(Factorial(-10), 0);
}
```

```

}
// Tests factorial of 0.
TEST(FactorialTest, Zero) {
    EXPECT_EQ(1, Factorial(0));
}

```

test_suites/test_factorial2.cpp

```

#include "../math_ext.h"
#include "gtest/gtest.h"
#include "stdio.h"
// Tests Factorial()
// Tests factorial of positive numbers.
TEST(FactorialTest, Positive) {
    EXPECT_EQ(1, Factorial(1));
    EXPECT_EQ(2, Factorial(2));
    EXPECT_EQ(6, Factorial(3));
    EXPECT_EQ(40320, Factorial(8));
}

```

test_suites/test_suites.cpp

```

#include "gtest/gtest.h"
#include "stdio.h"
// Tests Factorial()
int main(int argc, char **argv) {
    printf("Running main() from sample1\n");
    testing::InitGoogleTest(&argc, argv);
    return RUN_ALL_TESTS();
}

```

Configure biicode.conf

You have to indicate in the [dependencies] section in your **biicode.conf** file that the main file *test_suites.cpp* **depends on those test files:** *test_factorial1.cpp* and *test_factorial2.cpp*.

Read more about [dependencies] section [here](#).

And you have to indicate the new test in the [tests] section.

Edit your *biicode.conf* like this:

biicode.conf

```

[dependencies]
    test_suites/test_suites.cpp + test_suites/test_factorial1.cpp
    test_suites/test_suites.cpp + test_suites/test_factorial2.cpp

```

```
[tests]
  simple_test/simple_test.cpp
  test_suites/test_suites.cpp
```

Build and run the tests

Now execute your tests and you'll obtain the following output:

```
$ bii test
INFO: Processing changes...
INFO: Saving files from: google/gtest
...
test 1
  Start 1: examples_gtest_example_simple_test_simple_test
1: Running main() from sample1
1: [=====] Running 3 tests from 1 test case.
1: [-----] Global test environment set-up.
1: [-----] 3 tests from FactorialTest
1: [ RUN      ] FactorialTest.Negative
1: [          OK ] FactorialTest.Negative (0 ms)
1: [ RUN      ] FactorialTest.Zero
1: [          OK ] FactorialTest.Zero (0 ms)
1: [ RUN      ] FactorialTest.Positive
1: [          OK ] FactorialTest.Positive (0 ms)
1: [-----] 3 tests from FactorialTest (0 ms total)
1:
1: [-----] Global test environment tear-down
1: [=====] 3 tests from 1 test case ran. (0 ms total)
1: [ PASSED   ] 3 tests.
1/2 Test #1: examples_gtest_example_simple_test_simple_test ... Passed 0.04 se

test 2
  Start 2: examples_gtest_example_test_suites_test_suites
2: Running main() from sample1
2: [=====] Running 3 tests from 1 test case.
2: [-----] Global test environment set-up.
2: [-----] 3 tests from FactorialTest
2: [ RUN      ] FactorialTest.Positive
2: [          OK ] FactorialTest.Positive (0 ms)
2: [ RUN      ] FactorialTest.Negative
2: [          OK ] FactorialTest.Negative (0 ms)
2: [ RUN      ] FactorialTest.Zero
2: [          OK ] FactorialTest.Zero (0 ms)
2: [-----] 3 tests from FactorialTest (0 ms total)
2:
2: [-----] Global test environment tear-down
```

```

2: [=====] 3 tests from 1 test case ran. (0 ms total)
2: [ PASSED ] 3 tests.
2/2 Test #2: examples_gtest_example_test_suites_test_suites ... Passed 0.03 se

100% tests passed, 0 tests failed out of 2

Total Test time (real) = 0.09 sec
[100%] Built target check

```

Congrats! Your math code passed both “simple_test” and “test_suites”.

You can aggregate as many tests as you want to a suite so you can organize your tests to fit your needs.

Note: You can find more google test samples in the [google/gtestsamples](#) block.

Open and build

This example is already in biicode: [examples/gtest](#).

To give it a try and see how it’s configured, create a new project and open the block:

```

$ bii init gtest_example
$ cd gtest_example
$ bii open examples/gtest

```

Have a look at the *biicode.conf* file:

biicode.conf

```

[requirements]
    google/gtest: 10

[parent]
    examples/gtest: 1

[paths]

[dependencies]
    test_suites/test_suites.cpp + test_suites/test_factorial1.cpp
    test_suites/test_suites.cpp + test_suites/test_factorial2.cpp

[mains]

[hooks]

[includes]
    gtest/*.h: google/gtest/include

```

```
[data]

[tests]
    simple_test/simple_test.cpp
    test_suites/test_suites.cpp
```

Read more about [biicode.conf](https://biicode.com/docs/biicode.conf).

Now execute the tests:

```
$ bii test
...
100% tests passed, 0 tests failed out of 2

    Total Test time (real) = 0.09 sec
    [100%] Built target check
```

Got any doubts? Do not hesitate to [contact us](#), visit our [forum](#) and feel free to ask any questions.

1.8.18 HTTP Server

You can write and extend your own multi-platform http server with the [lasote/httpserver](#) block.

How does it work?

It allows you to implement just a subclass of `httpserver::HttpMiddleware` to attend browsers or other http client requests.

A method `call` from your subclass object will be called with a `Request*` object and a `Response*` object.

Just modify the body and/or headers of `Response` object in your method and the server will do all the work.

How can I use it?

- Just copy the files contained in the following section to a new block.
- Find the dependencies and execute your code:

```
$ bii find
$ cd bin
$ #run server executable
```

- Open your web browser and go to `http://localhost:9000`

The code

These are the files you will need in your block to have your HTTP Server up and running:

main_server.cpp

This file just instantiates the server and runs it with simple configuration parameters.

```

1 #include "lasote/httpserver/http_server.h"
2 #include "my_http_middle_ware.h"
3
4 using namespace httpserver;
5 using namespace gip;
6
7 int main() {
8
9     MyHttpMiddleware my_middleware;
10    HttpServerConf conf(9000, 300, 60, 5);
11
12    HttpServer http_server;
13
14    http_server.run(&my_middleware, &conf);
15
16    return 0;
17
18 }
```

my_http_middle_ware.h

Defines your HttpMiddleware subclass.

```

1 #pragma once
2
3 #include "lasote/httpserver/http_middleware.h"
4
5 using namespace httpserver;
6
7 class MyHttpMiddleware : public httpserver::HttpMiddleware {
8     public:
9         MyHttpMiddleware() : HttpMiddleware(NULL){}
10        MyHttpMiddleware(HttpMiddleware* other_middleware) : HttpMiddleware(other_middleware){}
11
12        virtual ~MyHttpMiddleware();
13        virtual void call(Request&, Response&);
14 };
```

my_http_middle_ware.cpp

Implements HttpMiddleware subclass. You should implement the `call` method, reading the header variables from the request, and modifying the response to return the

output.

```
1 #include "my_http_middle_ware.h"
2 #include "sstream"
3 #include "iostream"
4
5
6 MyHttpMiddleware::~MyHttpMiddleware() {
7
8 }
9
10 void MyHttpMiddleware::call(Request& request, Response& response) {
11     ostreamstream html;
12
13     html << "<!DOCTYPE html>\n<html>\n<body>\n";
14
15     if(request.get("name") != "") {
16         html << "Hello " << request.get("name") << "<br><br>";
17     }
18
19     //Build the html form
20     string form;
21     form = "\
22         <form name='form' action='/' method='POST'>\n\
23             Name: <br>\n\
24             <input type='text' name='name'> <br>\n\
25             <input type='submit' />\n\
26         </form>\n\
27     ";
28
29     html << form << "</body>\n</html>\n";
30
31     // Set content type we are printing
32     response.content_type("text/html");
33     // Set the body
34     response.body = html.str();
35 }
```

Download: [httpserver.zip](#)

Supported Operating Systems

The previous code has been tested on:

- Linux with GCC
- Windows with Mingw

- Windows with Visual Studio
- MacOS with Clang

More information

You can find more information in the **readme.txt** file of [lasote/httpserver](#)

List of dependencies

- [melikyan/ptypes](#): PTypes (C++ Portable Types Library) is a simple alternative to the STL that includes multithreading and networking. It defines dynamic strings, variants, character sets, lists and other basic data types along with portable thread and synchronization objects, IP sockets and named pipes. Its main ‘target audience’ is developers of complex network daemons, robots or non-visual client/server applications of any kind.
- [lasote/genericipserver](#): Generic and extensible IP server.
- [lasote/thread_jobs](#): Execute your tasks in threads managed by a pool.

1.8.19 json11

json11 is a tiny JSON library for C++11, providing JSON parsing and serialization.

The main block is [here](#), which is generated from this [github repo](#).

Simple convert data to json and vice versa

This example is [already in biicode](#), it is very simple to build it, by just opening the block and building it.

```
$ bii init json11
$ cd json11
$ bii open examples/json11
$ bii build
```

The code of the example is like this:

```
#include <string.h>
#include <cstdio>
#include <iostream>
#include <sstream>
#include "lasote/json11/json11.hpp"
#include <cassert>
#include <list>
```

```
#include <set>
#include <unordered_map>

using namespace json11;
using std::string;

int main(int argc, char **argv) {

    // STRING TO JSON
    const string simple_test =
        R"({ "k1": "v1", "k2": 42, "k3": ["a", 123, true, false, null] })";

    string err;
    auto json = Json::parse(simple_test, err);
    std::cout << "k1: " << json["k1"].string_value() << "\n";
    std::cout << "k3: " << json["k3"].dump() << "\n";

    // JSON FROM LITERAL
    Json obj = Json::object({
        { "k1", "v1" },
        { "k2", 42.0 },
        { "k3", Json::array({ "a", 123.0, true, false, nullptr }) },
    });

    std::cout << "obj: " << obj.dump() << "\n";

    // CUSTOM CLASS JSON ENCODE
    class Point {
    public:
        int x;
        int y;
        Point (int x, int y) : x(x), y(y) {}
        Json to_json() const { return Json::array { x, y }; }
    };

    std::vector<Point> points = { { 1, 2 }, { 10, 20 }, { 100, 200 } };
    std::string points_json = Json(points).dump();
    printf("%s\n", points_json.c_str());
}
```

Now, run the example.

```
$ bin/examples_json11_test
```

As you can see, with this library you can create json objects from string literals, standard collec-

tions and even custom classes in an easy way!

```
k1: v1
k3: ["a", 123, true, false, null]
obj: {"k1": "v1", "k2": 42, "k3": ["a", 123, true, false, null]}
[[1, 2], [10, 20], [100, 200]]
```

1.8.20 json++

JSON++ is a light-weight JSON parser, writer and reader written in C++. JSON++ can also convert JSON documents into lossless XML documents.

The main block is [here](#), which is generated from this [github repo](#).

Simple parser and converter from JSON to XML

This example is [already in biicode](#), it is very simple to build it, by just opening the block and building it.

```
~$ bii init jsonxx
~$ cd jsonxx
~/jsonxx$ bii open examples/jsonxx
~/jsonxx$ bii build
```

The code of the example is like this:

```
#include "hjiang/jsonxx/jsonxx.h"
using namespace jsonxx;
using namespace std;

int main() {
    #define QUOTE(...) #__VA_ARGS__
    string input = QUOTE(
        {
            "name/surname": "John Smith",
            'alias': 'Joe',
            "address": {
                "streetAddress": "21 2nd Street",
                "city": "New York",
                "state": "NY",
                "postal-code": 10021,
            }
        }
    );

    Object o;
```

```

    if( o.parse(input) ) {
        cout << o.xml(JSONx) << endl;           // XML output, JSONx flavor
        cout << o.xml(JXML) << endl;           // XML output, JXML flavor
        cout << o.xml(JXMLex) << endl;         // XML output, JXMLex flavor
        cout << o.xml(TaggedXML) << endl;       // XML output, tagged XML flavor
    }
    return 0;
}

```

Now, run the hello example.

```
~/jsonxx$ bin/examples_jsonxx_json_to_xml
```

You can see four diferent XML at the output:

```

<?xml version="1.0" encoding="UTF-8"?><!-- generated by jsonxx 0.22-a -->
<json:object xsi:schemaLocation="http://www.datapower.com/schemas/json jsonxx.xsd" >
  <json:object name="address">
    <json:string name="city">New York</json:string>
    <json:number name="postal-code">10021</json:number>
    <json:string name="state">NY</json:string>
    <json:string name="streetAddress">21 2nd Street</json:string>
  </json:object>
  <json:string name="alias">Joe</json:string>
  <json:string name="name\surname">John Smith</json:string>
</json:object>

```

```

<?xml version="1.0" encoding="UTF-8"?><!-- generated by jsonxx 0.22-a -->
<j son="o">
  <j son="o:address">
    <j son="s:city">New York</j>
    <j son="n:postal-code">10021</j>
    <j son="s:state">NY</j>
    <j son="s:streetAddress">21 2nd Street</j>
  </j>
  <j son="s:alias">Joe</j>
  <j son="s:name\surname">John Smith</j>
</j>

```

```

<?xml version="1.0" encoding="UTF-8"?><!-- generated by jsonxx 0.22-a -->
<j son="o">
  <j son="o:address" address="">
    <j son="s:city" city="New York">New York</j>
    <j son="n:postal-code" postal_code="10021">10021</j>
    <j son="s:state" state="NY">NY</j>
    <j son="s:streetAddress" streetAddress="21 2nd Street">21 2nd Street</j>
  </j>
  <j son="s:alias" alias="Joe">Joe</j>

```

```
<j son="s:name\surname" name_surname="John Smith">John Smith</j>
</j>
```

```
<?xml version="1.0" encoding="UTF-8"?><!-- generated by jsonxx 0.22-a -->
<JsonItem type="json:object">
  <address type="json:object" name="address">
    <city type="json:string" name="city">New York</city>
    <postal_code type="json:number" name="postal-code">10021</postal_code>
    <state type="json:string" name="state">NY</state>
    <streetAddress type="json:string" name="streetAddress">21 2nd Street</streetAddress>
  </address>
  <alias type="json:string" name="alias">Joe</alias>
  <name_surname type="json:string" name="name\surname">John Smith</name_surname>
</JsonItem>
```

1.8.21 Miniutf

miniutf is a C++ implementation of several basic Unicode manipulation functions developed by Dropbox.

The following example shows how to use miniutf to do some format conversions. You can find this example in the [biicode miniutf samples block](#).

miniutf.cpp

```
1 #include <cstdio>
2 #include <fstream>
3 #include <sstream>
4 #include <random>
5
6 #include <hithwen/miniutf/miniutf.hpp>
7 #include <hithwen/miniutf/miniutf_collation.hpp>
8
9
10 using namespace std;
11
12 void dump(const string & str) {
13     for (size_t i = 0; i < str.length(); )
14         printf(i ? "%04X" : " %04X", miniutf::utf8_decode(str, i));
15 }
16
17 int check_eq(const char *test, const string & expected, const string & got) {
18     if (expected == got)
19         return 1;
20
21     printf("%s: expected \"%", test);
```

```
22     dump(expected);
23     printf("\", got \"");
24     dump(got);
25     printf("\n");
26     return 0;
27 }
28
29 string match_key_as_hex(const vector<uint32_t> & key) {
30     string out;
31     for (uint32_t c : key) {
32         char outc[10];
33         snprintf(outc, 10, "%08X ", (unsigned int)c);
34         out.append(outc);
35     }
36     return out.substr(0, out.size() - 1);
37 }
38
39 template <class T> string string_as_hex(const T & s) {
40     string out;
41     for (size_t i = 0; i < s.size(); i++) {
42         char outc[10];
43         snprintf(outc, 10, "%02X ", (unsigned int)s[i]);
44         out.append(outc);
45     }
46     return out.substr(0, out.size() - 1);
47 }
48
49 bool check_match_key(const string & s1, const string & s2) {
50     vector<uint32_t> k1 = miniutf::match_key(s1);
51     vector<uint32_t> k2 = miniutf::match_key(s2);
52     if (k1 != k2) {
53         printf("match_key(%s,%s) test failed\n", string_as_hex(s1).c_str(), string_
54         printf(" got %s, expected %s\n", match_key_as_hex(k1).c_str(), match_key_a
55         return false;
56     }
57     return true;
58 }
59
60
61 int main(void) {
62
63     string utf8_test = { '\x61', '\x00', '\xF0', '\x9F', '\x92', '\xA9' };
64     ul6string utf16_test = { 0x61, 0, 0xD83D, 0xDCA9 };
65
66     // We also have some tests of UTF-8 to UTF-16 conversion
67     string utf8 = miniutf::to_utf8(utf16_test);
68     if (!check_eq("16-to-8", utf8_test, utf8))
```



```

69         return 1;
70
71     ul6string utf16 = min(utf8::to_utf16(utf8_test));
72     if (utf16 != utf16_test) {
73         printf("utf8-to-utf16 test failed: got ");
74         for (size_t i = 0; i < utf16.length(); i++) printf("%04x ", (uint16_t)utf16[i]);
75         printf("\n");
76         return 1;
77     }
78
79     // Test match_key function
80     if (!check_match_key(u8"Øç",
81                         u8"oaec")) { return 1; }
82     if (!check_match_key(u8"ääääèèëëüöñ",
83                         u8"aaaaeeeuon")) { return 1; }
84
85     printf("OK\n");
86     return 0;
87 }

```

Create a block:

```

$ bii init my_project
$ cd my_project
$ bii open examples/miniutf

```

Now you can compile it:

```

$ bii build

```

You will see next console output after executing the command:

```

$ ./bin/examples_miniutf_miniutf
OK

```

Any doubts? Do not hesitate to [contact us](#) visit our [forum](#) and feel free to ask any question.

1.8.22 Multivariate Splines

Multivariate Splines is a function approximation library implementing various multivariate splines in C++.

You can find here the [Multivariate splines original github page](#).

This example does not compile in Windows. Please, use Linux or MacOS instead.

The following example demonstrates the use of Multivariate Splines. Note that there are no restrictions to the dimension of x (except that it has to be ≥ 1 , of course).

main_muparser.cpp

```
1  #include <iostream>
2  #include "lasote/splines/include/datatable.h"
3  #include "lasote/splines/include/bspline.h"
4  #include "lasote/splines/include/pspline.h"
5  #include "lasote/splines/include/rbfspline.h"
6
7  using std::cout;
8  using std::endl;
9
10 using namespace MultivariateSplines;
11
12 // Six-hump camelback function
13 double f(DenseVector x)
14 {
15     assert(x.rows() == 2);
16     return (4 - 2.1*x(0)*x(0)
17 + (1/3.)*x(0)*x(0)*x(0)*x(0))*x(0)*x(0)
18 + x(0)*x(1)
19 + (-4 + 4*x(1)*x(1))*x(1)*x(1);
20 }
21
22 int main(int argc, char *argv[])
23 {
24     // Create new DataTable to manage samples
25     DataTable samples;
26
27     // Sample function
28     DenseVector x(2);
29     double y;
30     for(int i = 0; i < 20; i++)
31     {
32         for(int j = 0; j < 20; j++)
33         {
34             // Sample function at x
35             x(0) = i*0.1;
36             x(1) = j*0.1;
37             y = f(x);
38
39             // Store sample
40             samples.addSample(x,y);
41         }
42     }
43
44     // Build B-splines that interpolate the samples
45     BSpline bspline1(samples, BSplineType::LINEAR);
46     BSpline bspline3(samples, BSplineType::CUBIC_FREE);
```

```

47 // Build penalized B-spline (P-spline) that smooths the samples
48 PSpline pspline(samples, 0.03);
49
50 // Build radial basis function spline that interpolate the samples
51 RBFspline rbf spline(samples, RadialBasisFunctionType::THIN_PLATE_SPLINE);
52
53 // Evaluate the splines at x = (1,1)
54 x(0) = 1; x(1) = 1;
55
56 cout << "-----" << endl;
57 cout << "Function at x: \t\t\t" << f(x) << endl;
58 cout << "Linear B-spline at x: \t\t" << bspline1.eval(x) << endl;
59 cout << "Cubic B-spline at x: \t\t" << bspline3.eval(x) << endl;
60 cout << "P-spline at x: \t\t\t" << pspline.eval(x) << endl;
61 cout << "Thin-plate spline at x:\t\t" << rbf spline.eval(x) << endl;
62 cout << "-----" << endl;
63
64 return 0;
65 }

```

Project should be compiled with C++11 so you also need the CMakeLists which specify it:

CMakeLists.txt

```

1 # This CMakeLists.txt file helps defining your block building and compiling
2 # Include the main biicode macros and functions
3 # To learn more about the CMake use with biicode, visit http://docs.biicode.com/c++
4 # Or check the examples below
5
6
7 include(${CMAKE_HOME_DIRECTORY}/biicode.cmake)
8
9 # ACTIVATING C++11 FLAG
10 IF (APPLE)
11     SET(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++11 -stdlib=libc++)
12 ELSEIF (WIN32 OR UNIX)
13     SET(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++11")
14 ENDIF (APPLE)
15 ADD_BII_TARGETS()

```

- This example is **already in biicode**, it's simple to run, just open the block and build it like this:

```

$ bii init my_project
$ cd my_project
$ bii open examples/multivariate_splines
$ bii build

```

- **Only if you want to do it manually**, create a *cpp file* in your block, copy the code above

and resolve all the dependencies of the `main.cpp`. Create a `CMakeLists.txt` file with the content above too. Then you can compile it:

```
$ bii init my_project
$ cd my_project
$ bii new my_user/my_block
$ # Create a main.cpp file in my_user/my_block and copy the code
$ # Create a CMakeLists.txt file in my_user/my_block an copy the code
$ bii find
$ bii build
```

You will see next console output after executing the command:

```
$ cd bin
$ bin $ ././examples_multivariate_splines_splines_main
Computing B-spline control points using dense solver.
Computing B-spline control points using dense solver.
Computing B-spline control points using dense solver.
Computing RBF weights using dense solver.
Error: 6.33755e-14
-----
Function at x:                3.23333
Linear B-spline at x:         3.23333
Cubic B-spline at x:          3.23333
P-spline at x:                3.23285
Thin-plate spline at x:       3.23333
-----
```

1.8.23 libuv

libuv i a multi-platform library for asynchronous, event-driven style of programming. It offers core utilities like timers, non-blocking networking support, asynchronous file system access, child processes and more.

libuv libraries available at biicode are:

- libuv v0.10 at [lasote/libuv \(v0.10\)](#).
- libuv v0.11 at [lasote/libuv \(v0.11\)](#).
- libuv v1.x at [lasote/libuv \(v1.x\)](#).

Blocks are generated for this [github repo](#).

Http client/server application

In this example we'll show a **Http client/server** application.

Creating a new project

This example is already in biicode at [examples/libuv\(v0.11\)](#). So just open the block:

```
$ bii init libuvproject
$ cd libuvproject
$ bii open examples/libuv(v.11)
```

Check dependencies

Let's check the dependencies of this example:

```
$ bii deps
INFO: Processing changes...
examples/libuv depends on:
    lasote/libuv(v0.11): 2
        include/uv.h
    system:
        stdio.h
        stdlib.h
        string.h
```

Now take a look at *biicode.conf* file:

```
[requirements]
    lasote/libuv (v0.11): 2

[parent]
    examples/libuv(v0.11): 1

[includes]
    uv.h: lasote/libuv/include
```

Note that the original `[includes]` are mapped, so you just have to `#include "uv.h"` to add libuv to the project.

Build the project

Now execute **bii build** to build the project.

```
$ bii build
```

Go to bin directory and execute:

```
$ cd bin
$ examples_libuv_network_server.exe
```

And in other terminal/console:

```
$ cd bin
$ examples_libuv_network_client.exe
```

And you'll see the next output:

Server	Client
Connection ok! Received: Ping... Sending: Ping... Pong! Closed ok!	Writing: Ping... Write ok! Received: Ping... Pong!

Hey! That's your ping-pong client/server application!

Change libuv's version

You can change among libuv's versions with just a single mod in *biicode.conf* file:

- To depend on **libuv v0.10**:

```
[requirements]
  lasote/libuv (v0.10): 3
```

- To depend on **libuv v0.11**:

```
[requirements]
  lasote/libuv (v0.11): 2
```

- To depend on **libuv v1.x**:

```
[requirements]
  lasote/libuv (v1.x): 7
```

Try the example above switching versions!

Got any doubts? Do not hesitate to [contact us](#), visit our [forum](#) and feel free to ask any questions.

1.8.24 Little CMS

Little CMS is an Open Source Color Management Engine. This example demonstrates how to get started using **LittleCMS** by Marti Maria with biicode.

Little CMS intends to be an OPEN SOURCE **small-footprint color management engine**, with special focus on accuracy and performance. It uses the International Color Consortium standard (ICC), which is the modern standard when regarding to color management. This examples have been tested in Windows, OS X and Linux-systems. Check the Sources:

1. [Original Little CMS Library](#).

2. Biicode Little CMS block at [martimaria/littlecms](#).
3. [Github repository](#).
4. [LittleCMS Documentation](#).

ICC Profile Examples

In this example we'll create a devicelink that decodes TIFF8 Lab files using Little CMS. You only have to write `#include "lcms2.h"` at the top of your code.

Creating a new project

Create a new project and a *lcms-main.c* file:

```
$ bii init lcms -L
$ cd lcms
$ # Create lcms-main.c and copy the code
```

lcms-main.c

```
// Little cms
// Copyright (C) 1998-2010 Marti Maria
//
// Permission is hereby granted, free of charge, to any person obtaining
// a copy of this software and associated documentation files (the "Software"),
// to deal in the Software without restriction, including without limitation
// the rights to use, copy, modify, merge, publish, distribute, sublicense,
// and/or sell copies of the Software, and to permit persons to whom the Software
// is furnished to do so, subject to the following conditions:
//
// The above copyright notice and this permission notice shall be included in
// all copies or substantial portions of the Software.
//
// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
// EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO
// THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
// NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE
// LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
// OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION
// WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

// Creates a devicelink that decodes TIFF8 Lab files

#include "lcms2.h"
#include <stdlib.h>
#include <math.h>
```

```
static
double DecodeAbTIFF(double ab)
{
    if (ab <= 128.)
        ab += 127.;
    else
        ab -= 127.;

    return ab;
}

static
cmsToneCurve* CreateStep(void)
{
    cmsToneCurve* Gamma;
    cmsUInt16Number* Table;
    int i;
    double a;

    Table = calloc(4096, sizeof(cmsUInt16Number));
    if (Table == NULL) return NULL;

    for (i=0; i < 4096; i++) {
        a = (double) i * 255. / 4095.;
        a = DecodeAbTIFF(a);
        Table[i] = (cmsUInt16Number) floor(a * 257. + 0.5);
    }
    Gamma = cmsBuildTabulatedToneCurve16(0, 4096, Table);
    free(Table);
    return Gamma;
}

static
cmsToneCurve* CreateLinear(void)
{
    cmsUInt16Number Linear[2] = { 0, 0xffff };
    return cmsBuildTabulatedToneCurve16(0, 2, Linear);
}

// Set the copyright and description
static
cmsBool SetTextTags(cmsHPROFILE hProfile)
{

```



```

    cmsMLU *DescriptionMLU, *CopyrightMLU;
    cmsBool rc = FALSE;
    DescriptionMLU = cmsMLUalloc(0, 1);
    CopyrightMLU = cmsMLUalloc(0, 1);
    if (DescriptionMLU == NULL || CopyrightMLU == NULL) goto Error;
    if (!cmsMLUsetASCII(DescriptionMLU, "en", "US", "Little cms Tiff8 CIELab")) goto Error;
    if (!cmsMLUsetASCII(CopyrightMLU, "en", "US", "Copyright (c) Marti Maria, 2003")) goto Error;
    if (!cmsWriteTag(hProfile, cmsSigProfileDescriptionTag, DescriptionMLU)) goto Error;
    if (!cmsWriteTag(hProfile, cmsSigCopyrightTag, CopyrightMLU)) goto Error;
    rc = TRUE;

Error:
    if (DescriptionMLU)
        cmsMLUfree(DescriptionMLU);
    if (CopyrightMLU)
        cmsMLUfree(CopyrightMLU);
    return rc;
}

int main(int argc, char *argv[])
{
    cmsHPROFILE hProfile;
    cmsPipeline *AToB0;
    cmsToneCurve* PreLinear[3];
    cmsToneCurve *Lin, *Step;

    fprintf(stderr, "Creating lcmstiff8.icm...");

    remove("lcmstiff8.icm");
    hProfile = cmsOpenProfileFromFile("lcmstiff8.icm", "w");

    // Create linearization
    Lin = CreateLinear();
    Step = CreateStep();

    PreLinear[0] = Lin;
    PreLinear[1] = Step;
    PreLinear[2] = Step;

    AToB0 = cmsPipelineAlloc(0, 3, 3);
    cmsPipelineInsertStage(AToB0,
        cmsAT_BEGIN, cmsStageAllocToneCurves(0, 3, PreLinear));
    cmsSetColorSpace(hProfile, cmsSigLabData);
    cmsSetPCS(hProfile, cmsSigLabData);
    cmsSetDeviceClass(hProfile, cmsSigLinkClass);
    cmsSetProfileVersion(hProfile, 4.2);

```

```
cmsWriteTag(hProfile, cmsSigAToB0Tag, AToB0);
SetTextTags(hProfile);
cmsCloseProfile(hProfile);
cmsFreeToneCurve(Lin);
cmsFreeToneCurve(Step);
cmsPipelineFree(AToB0);

fprintf(stderr, "Done.\n");
return 0;
}
```

Manage your dependencies

Check the dependencies of the project with **bii deps**:

```
INFO: Processing changes...
youruser/lcms depends on:
  system:
    math.h
    stdlib.h
  unresolved:
    lcms2.h
```

Now, edit the *biicode.conf* file generated in the project folder. Map your `[includes]` to point to `martimaria/littlecms`:

```
[includes]
  lcms2.h: martimaria/littlecms/include/
```

Now do **bii find** and biicode will find the most recent version available of Little CMS library:

```
$ bii find
INFO: Processing changes...
INFO: Finding missing dependencies in server
INFO: Looking for martimaria/littlecms...
INFO: Block candidate: martimaria/martimaria/littlecms/master
INFO:   Version martimaria/littlecms: 1 (STABLE) valid
INFO:   Version martimaria/littlecms: 0 (STABLE) valid
INFO: Analyzing compatibility for found dependencies...
INFO: All dependencies resolved
Find resolved new dependencies:
  martimaria/littlecms: 1
INFO: Saving files from: martimaria/littlecms
```

Have a look at your *biicode.conf* again to ensure Little CMS library was added to your project:

```
[requirements]
  martimaria/littlecms: 1

[includes]
  lcms2.h: martimaria/littlecms/include/
```

Check again with **bii deps** and now all dependencies are resolved.

Build the project

Next, the only thing left is building the project:

```
$ bii build
```

Execute the binary placed in bin directory:

```
$ bin/youruser_lcms_lcms-main
```

Once you execute you should see an output like this one, and a the `lcmstiff8.icm` file created into your bin folder:

```
Creating lcmstiff8.icm...Done
```

You can find more examples at [examples/littlecms](#), give them a try!

Open and build

This example is already in biicode at [examples/littlecms](#).

It is simple to run, just open the block and build it like this:

```
$ bii init lcms
$ cd lcms
$ bii open examples/littlecms
```

There are three different files in the project, note that all of them use Little CMS , simply by including the library.

Build the block and execute any of them!

```
$ bii build
$ cd bin
$ # Execute!
```

Any doubts? Do not hesitate to [contact us](#) visit our [forum](#) and feel free to ask any questions.

1.8.25 Log4z

is an open source C++ lightweight & cross platform log library.

It provides in a C++ application log and trace debug function for 7*24h service program.

Support: 64/32 bits of debian, redhat, centos, suse, windows.

The main block is , which is generated from this .

Fast stream log strings test

Now, you can check all the log4z examples which are uploaded in biicode and execute any of them. Then, create a new project and open the .

```
~$ bii init log4z_sample
~$ cd log4z_sample
~/log4z_sample$ bii open examples/log4z
~/log4z_sample$ bii build
```

Windows Users

It is necessary to specify *Visual Studio* generator before building (it doesn't work with MinGW), otherwise, you can skip this step.

```
bii configure -G "Visual Studio 12"
```

When all the examples are built, execute for fast stream log strings testing:

```
~/log4z_sample$ bin/examples_log4z_fast_test
2014-09-19 12:15:08.223 LOG_ALARM ----- log4z thread started! -----
2014-09-19 12:15:08.223 LOG_ALARM logger id=0 path=./log/ name=examples_log4z_fast_
2014-09-19 12:15:08.223 LOG_INFO begin test stream log utf-16 string input.... ( C
...

```

Any doubts? Do not hesitate to [contact us](#) visit our [forum](#) and feel free to ask any questions.

1.8.26 Lwan Web Server

Lwan is a high-performance & scalable web server for glibc/Linux platforms. For more information about this project, visit their [official website](#).

Hello World Example

This example creates a very simple web service which greets you by your name.

How to replicate

This example is [already in biicode](#), it's simple to run, just open the block and build it like this:

```
$ bii init lwan
$ cd lwan
$ bii open examples/lwan
$ bii build
```

This example needs three files, first one that defines the server configuration:

lwan.conf

```
1 listener *:8080 {
2     prefix /hello {
3         handler = hello_world
4     }
5 }
```

The actual source code:

lwan_example.c

```
1 #include "hithwen/lwan/common/lwan.h"
2
3 lwan_http_status_t
4 hello_world(lwan_request_t *request,
5             lwan_response_t *response,
6             void *data __attribute__((unused)))
7 {
8     static lwan_key_value_t headers[] = {
9         { .key = "X-The-Answer-To-The-Universal-Question", .value = "42" },
10        { NULL, NULL }
11    };
12    response->headers = headers;
13    response->mime_type = "text/plain";
14
15    const char *name = lwan_request_get_query_param(request, "name");
16    if (name)
17        strbuf_printf(response->buffer, "Hello, %s!", name);
18    else
19        strbuf_set_static(response->buffer, "Hello, world!", sizeof("Hello, world!"));
20
21    return HTTP_OK;
22 }
23
24 int
25 main(void)
26 {
```

```
27     lwan_t l;  
28  
29     lwan_init(&l);  
30     lwan_main_loop(&l);  
31     lwan_shutdown(&l);  
32  
33     return 0;  
34 }
```

And the Cmake lists that copies config file to bin folder:

CMakeLists.txt

```
1 include(${CMAKE_HOME_DIRECTORY}/biicode.cmake)  
2  
3 ADD_BII_TARGETS()  
4  
5 MESSAGE("Copying lwan.conf to ${CMAKE_RUNTIME_OUTPUT_DIRECTORY}")  
6 FILE(COPY ${CMAKE_CURRENT_SOURCE_DIR}/lwan.conf DESTINATION ${CMAKE_RUNTIME_OUTPUT_DIRECTORY})  
7 FILE(RENAME ${CMAKE_RUNTIME_OUTPUT_DIRECTORY}/lwan.conf ${CMAKE_RUNTIME_OUTPUT_DIRECTORY}/lwan.conf)
```

Now, run the hello example.

```
$ cd bin  
$ ./examples_lwan_lwan_example
```

You can see the server running:

```
62528 lwan-job.c:76 lwan_job_thread_init() Initializing low priority job thread.  
62528 lwan-response.c:75 lwan_response_init() Initializing default response.  
62528 lwan-tables.c:44 lwan_tables_init() Uncompressing MIME type table.  
62528 lwan.c:58 lwan_module_init() Initializing module registry.  
62528 lwan.c:73 lwan_module_register() Registering module "serve_files".  
62528 lwan.c:73 lwan_module_register() Registering module "redirect".  
62528 lwan.c:360 setup_from_config() Loading configuration file: examples_lwan_lwan.conf  
62528 lwan.c:476 lwan_init() Initializing lwan web server.  
62528 lwan.c:487 lwan_init() Using 2 threads, maximum 2048 sockets per thread.  
62528 lwan-thread.c:393 lwan_thread_init() Initializing threads.  
62528 lwan-socket.c:212 lwan_socket_init() Initializing sockets.  
62528 lwan-socket.c:197 _setup_socket_normally() Listening on http://0.0.0.0:8080.  
62528 lwan.c:561 lwan_main_loop() Ready to serve.  
62531 lwan-thread.c:290 _thread_io_loop() Starting IO loop on thread #2.  
62530 lwan-thread.c:290 _thread_io_loop() Starting IO loop on thread #1.
```

Now you can go to any browser and navigate to <http://localhost:8080?name=fred>

Any doubts? Do not hesitate to [contact us](#) visit our [forum](#) and feel free to ask any questions.

1.8.27 MiLi

MiLi is a collection of useful C++ libraries, composed only by headers. For more information about this library, visit their [official website](#) or [wiki](#).

You can find here the [MiLi biicode library site](#).

The following example shows a simple use for doing type-safe bitwise operations. You can find this and other examples in [the biicode MiLi samples block](#).

main_mili.cpp

```

1  /*bitwise_enums: A minimal library for doing type-safe bitwise operations.
2  This file is part of the MiLi Minimalistic Library.
3
4  Copyright (C) Daniel Gutson, FuDePAN 2008-2009
5                      Adrian Remonda FuDePAN 2011
6  Distributed under the Boost Software License, Version 1.0.
7  (See accompanying file LICENSE_1_0.txt in the root directory or
8  copy at http://www.boost.org/LICENSE_1_0.txt)
9
10 MiLi IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
11 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
12 FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT
13 SHALL THE COPYRIGHT HOLDERS OR ANYONE DISTRIBUTING THE SOFTWARE BE LIABLE
14 FOR ANY DAMAGES OR OTHER LIABILITY, WHETHER IN CONTRACT, TORT OR OTHERWISE,
15 ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
16 DEALINGS IN THE SOFTWARE.
17
18 This is an example file.*/
19
20 #include <iostream>
21 #include "danielgutson/mili/mili.h"
22
23 using namespace mili;
24 using namespace std;
25
26 enum MasksSet1
27 {
28     kZero    = 0,
29     kOne     = 1,
30     kTwo     = 2,
31     kThree   = 4,
32     kFour    = 8
33 };
34 BITWISE_ENUM_ENABLE(MasksSet1)
35
36 enum MasksSet2

```

```
37 {
38     kEight    = 8,
39     kSixteen  = 16
40 };
41
42 typedef bitwise_enum<MasksSet1> M1;
43
44 void show_bits(M1 b)
45 {
46     if (b.has_bits(kZero))  cout << "kZero  turned on\n";
47     if (b.has_bits(kOne))   cout << "kOne   turned on\n";
48     if (b.has_bits(kTwo))   cout << "kTwo   turned on\n";
49     if (b.has_bits(kThree)) cout << "kThree turned on\n";
50     if (b.has_bits(kFour))  cout << "kFour  turned on\n";
51     cout << endl;
52 }
53
54 int main()
55 {
56     //with bitwise nums
57     bitwise_enum<MasksSet1> myEnum(kOne | kTwo | kFour);
58
59     show_bits(myEnum);
60
61     //enum with bitwise enum
62     myEnum = kOne & myEnum;
63     show_bits(myEnum);
64
65     // 2 enums
66     show_bits(kOne | kThree);
67
68     // << operator
69     cout << "<< Operator test: 0x" << hex << (myEnum << 2) << endl;
70
71     //without bitwise nums (built-in types)
72     int normalEnum = kEight | kSixteen;
73     cout << "Normal Enum: 0x" << hex << normalEnum << endl;
74
75     normalEnum = kEight << 2;
76     cout << "Normal Enum: 0x" << hex << normalEnum << endl;
77
78     return 0;
79 }
```

Create a new project and open the example block:


```
$ biibuild
$ cd bin
$ # run executable
```

```
kOne    turned on
kTwo    turned on
kFour   turned on

kOne    turned on

kOne    turned on
kThree  turned on

<< Operator test: 0x4
Normal Enum: 0x18
Normal Enum: 0x20
```

1.8.28 MuParser

main_muparser.cpp

[illegible]

```

8  //      \/\      \/\      \/\
9  //  (C) 2013 Ingo Berg
10 //
11 //  example1.cpp - using the parser as a static library
12 //
13 //-----
14
15 #include "ingoberg/muparser/include/muParserTest.h"
16
17 #if defined(_WIN32) && defined(_DEBUG)
18     #define _CRTDBG_MAP_ALLOC
19     #include <stdlib.h>
20     #include <crtDBG.h>
21     #define CREATE_LEAKAGE_REPORT
22 #endif
23
24 #if defined( USINGDLL ) && defined( _WIN32 )
25 #error This sample can be used only with STATIC builds of muParser (on win32)
26 #endif
27
28 /** \brief This macro will enable mathematical constants like M_PI. */
29 #define _USE_MATH_DEFINES
30
31 #include <cstdlib>
32 #include <cstring>
33 #include <cmath>
34 #include <string>
35 #include <iostream>
36 #include <locale>
37 #include <limits>
38 #include <ios>
39 #include <iomanip>
40 #include <numeric>
41
42 #include "ingoberg/muparser/include/muParser.h"
43
44 using namespace std;
45 using namespace mu;
46
47
48 #if defined(CREATE_LEAKAGE_REPORT)
49
50 // Dumping memory leaks in the destructor of the static guard
51 // guarantees i won't get false positives from the ParserErrorMsg
52 // class wich is a singleton with a static instance.
53 struct DumpLeaks
54 {

```

```

55 ~DumpLeaks()
56 {
57     _CrtDumpMemoryLeaks();
58 }
59 } static LeakDumper;
60
61 #endif
62
63 // Forward declarations
64 void CalcBulk();
65
66 // Operator callback functions
67 value_type Mega(value_type a_fVal) { return a_fVal * 1e6; }
68 value_type Milli(value_type a_fVal) { return a_fVal / (value_type)1e3; }
69 value_type Rnd(value_type v) { return v*std::rand()/(value_type)(RAND_MAX+1.0); }
70 value_type Not(value_type v) { return v==0; }
71 value_type Add(value_type v1, value_type v2) { return v1+v2; }
72 value_type Mul(value_type v1, value_type v2) { return v1*v2; }
73
74 //-----
75 value_type ThrowAnException(value_type)
76 {
77     throw std::runtime_error("This function does throw an exception.");
78 }
79
80 //-----
81 value_type BulkFun1(int nBulkIdx, int nThreadIdx, value_type v1)
82 {
83     // Note: I'm just doing something with all three parameters to shut
84     // compiler warnings up!
85     return nBulkIdx + nThreadIdx + v1;
86 }
87
88 //-----
89 value_type Ping()
90 {
91     mu::console() << "ping\n";
92     return 0;
93 }
94
95 //-----
96 value_type StrFun0(const char_type *szMsg)
97 {
98     if (szMsg)
99         mu::console() << szMsg << std::endl;
100
101     return 999;

```

```

102 }
103
104 //-----
105 value_type StrFun2(const char_type *v1, value_type v2,value_type v3)
106 {
107     mu::console() << v1 << std::endl;
108     return v2+v3;
109 }
110
111 //-----
112 value_type Debug(mu::value_type v1, mu::value_type v2)
113 {
114     ParserBase::EnableDebugDump(v1!=0, v2!=0);
115     mu::console() << _T("Bytecode dumping ") << ((v1!=0) ? _T("active") : _T("inactive"));
116     return 1;
117 }
118
119 //-----
120 // Factory function for creating new parser variables
121 // This could as well be a function performing database queries.
122 value_type* AddVariable(const char_type *a_szName, void *a_pUserData)
123 {
124     // I don't want dynamic allocation here, so i used this static buffer
125     // If you want dynamic allocation you must allocate all variables dynamically
126     // in order to delete them later on. Or you find other ways to keep track of
127     // variables that have been created implicitly.
128     static value_type afValBuf[100];
129     static int iVal = -1;
130
131     ++iVal;
132
133     mu::console() << _T("Generating new variable \"")
134                 << a_szName << std::dec << _T("\" (slots left: ")
135                 << 99-iVal << _T(")")
136                 << _T(" User data pointer is:")
137                 << std::hex << a_pUserData << endl;
138     afValBuf[iVal] = 0;
139
140     if (iVal>=99)
141         throw mu::ParserError( _T("Variable buffer overflow.") );
142     else
143         return &afValBuf[iVal];
144 }
145
146 int IsHexValue(const char_type *a_szExpr, int *a_iPos, value_type *a_fVal)
147 {
148     if (a_szExpr[1]==0 || (a_szExpr[0]!='0' || a_szExpr[1]!='x') )

```

1.8. Examples 137

```

196     }
197
198     return 0;
199 }
200
201 //-----
202 value_type Help()
203 {
204     mu::console() << _T( "-----\n\n");
205     mu::console() << _T( "Commands:\n\n");
206     mu::console() << _T( "  list var      - list parser variables\n");
207     mu::console() << _T( "  list exprvar - list expression variables\n");
208     mu::console() << _T( "  list const   - list all numeric parser constants\n");
209     mu::console() << _T( "  opt on      - enable optimizer (default)\n");
210     mu::console() << _T( "  opt off     - disable optimizer\n");
211     mu::console() << _T( "  locale de   - switch to german locale\n");
212     mu::console() << _T( "  locale en   - switch to english locale\n");
213     mu::console() << _T( "  locale reset - reset locale\n");
214     mu::console() << _T( "  test bulk   - test bulk mode\n");
215     mu::console() << _T( "  quit        - exits the parser\n");
216     mu::console() << _T( "\nConstants:\n\n");
217     mu::console() << _T( "  \"_e\"      2.718281828459045235360287\n");
218     mu::console() << _T( "  \"_pi\"     3.141592653589793238462643\n");
219     mu::console() << _T( "-----\n\n");
220     return 0;
221 }
222
223 //-----
224 /*
225 void CheckLocale()
226 {
227     // Local names:
228     // "C" - the classic C locale
229     // "de_DE" - not for Windows?
230     // "en_US" - not for Windows?
231     // "German_germany" - For MSVC8
232     try
233     {
234         std::locale loc("German_germany");
235         console() << _T("Locale settings:\n");
236         console() << _T("  Decimal point:  ") << std::use_facet<numput<char_type>> >
237         console() << _T("  Thousands sep:  ") << std::use_facet<numput<char_type>> >
238         console() << _T("  Grouping:       ") << std::use_facet<numput<char_type>> >
239         console() << _T("  True is named:  ") << std::use_facet<numput<char_type>> >
240         console() << _T("  False is named: ") << std::use_facet<numput<char_type>> >
241         console() << _T("-----\n\n");
242     }

```

```

243 catch(...)
244 {
245     console() << _T("Locale settings:\n");
246     console() << _T("  invalid locale name\n");
247     console() << _T("-----\n");
248 }
249 }
250
251 //-----
252 void CheckDiff()
253 {
254     mu::Parser parser;
255     value_type x = 1,
256                 v1,
257                 v2,
258                 v3,
259                 eps(pow(std::numeric_limits<value_type>::epsilon(), 0.2));
260     parser.DefineVar(_T("x"), &x);
261     parser.SetExpr(_T("_e^-x*sin(x)"));
262
263     v1 = parser.Diff(&x, 1),
264     v2 = parser.Diff(&x, 1, eps);
265     v3 = cos((value_type)1.0)/exp((value_type)1) - sin((value_type)1.0)/exp((value_type)1);
266     mu::console() << parser.GetExpr() << _T("\n");
267     mu::console() << _T("v1 = ") << v1 << _T("; v1-v3 = ") << v1-v3 << _T("\n");
268     mu::console() << _T("v2 = ") << v2 << _T("; v2-v3 = ") << v2-v3 << _T("\n");
269 }
270 */
271
272 //-----
273 void ListVar(const mu::ParserBase &parser)
274 {
275     // Query the used variables (must be done after calc)
276     mu::varmap_type variables = parser.GetVar();
277     if (!variables.size())
278         return;
279
280     cout << "\nParser variables:\n";
281     cout << "-----\n";
282     cout << "Number: " << (int)variables.size() << "\n";
283     varmap_type::const_iterator item = variables.begin();
284     for (; item!=variables.end(); ++item)
285         mu::console() << _T("Name: ") << item->first << _T("  Address: [0x") << item->second << _T("]\n");
286 }
287
288 //-----
289 void ListConst(const mu::ParserBase &parser)

```

```

290 {
291     mu::console() << _T("\nParser constants:\n");
292     mu::console() << _T("-----\n");
293
294     mu::valmap_type cmap = parser.GetConst();
295     if (!cmap.size())
296     {
297         mu::console() << _T("Expression does not contain constants\n");
298     }
299     else
300     {
301         valmap_type::const_iterator item = cmap.begin();
302         for (; item!=cmap.end(); ++item)
303             mu::console() << _T(" ") << item->first << _T(" = ") << item->second << _T(" ");
304     }
305 }
306
307 //-----
308 void ListExprVar(const mu::ParserBase &parser)
309 {
310     string_type sExpr = parser.GetExpr();
311     if (sExpr.length()==0)
312     {
313         cout << _T("Expression string is empty\n");
314         return;
315     }
316
317     // Query the used variables (must be done after calc)
318     mu::console() << _T("\nExpression variables:\n");
319     mu::console() << _T("-----\n");
320     mu::console() << _T("Expression: ") << parser.GetExpr() << _T("\n");
321
322     varmap_type variables = parser.GetUsedVar();
323     if (!variables.size())
324     {
325         mu::console() << _T("Expression does not contain variables\n");
326     }
327     else
328     {
329         mu::console() << _T("Number: ") << (int)variables.size() << _T("\n");
330         mu::varmap_type::const_iterator item = variables.begin();
331         for (; item!=variables.end(); ++item)
332             mu::console() << _T("Name: ") << item->first << _T(" Address: [0x") << item->second << _T("]\n");
333     }
334 }
335
336 //-----

```



```

337 /** \brief Check for external keywords.
338 */
339 int CheckKeywords(const mu::char_type *a_szLine, mu::Parser &a_Parser)
340 {
341     string_type sLine(a_szLine);
342
343     if ( sLine == _T("quit") )
344     {
345         return -1;
346     }
347     else if ( sLine == _T("list var") )
348     {
349         ListVar(a_Parser);
350         return 1;
351     }
352     else if ( sLine == _T("opt on") )
353     {
354         a_Parser.EnableOptimizer(true);
355         mu::console() << _T("Optimizer enabled\n");
356         return 1;
357     }
358     else if ( sLine == _T("opt off") )
359     {
360         a_Parser.EnableOptimizer(false);
361         mu::console() << _T("Optimizer disabled\n");
362         return 1;
363     }
364     else if ( sLine == _T("list const") )
365     {
366         ListConst(a_Parser);
367         return 1;
368     }
369     else if ( sLine == _T("list exprvar") )
370     {
371         ListExprVar(a_Parser);
372         return 1;
373     }
374     else if ( sLine == _T("list const") )
375     {
376         ListConst(a_Parser);
377         return 1;
378     }
379     else if ( sLine == _T("locale de") )
380     {
381         mu::console() << _T("Setting german locale: ArgSep=';' DecSep=',' ThousandsSep=
382         a_Parser.SetArgSep(';');
383         a_Parser.SetDecSep(',');

```

```
384     a_Parser.SetThousandsSep('.');
385     return 1;
386 }
387 else if ( sLine == _T("locale en") )
388 {
389     mu::console() << _T("Setting english locale: ArgSep=', ' DecSep='.' ThousandsSep=");
390     a_Parser.SetArgSep(',');
391     a_Parser.SetDecSep('.');
392     a_Parser.SetThousandsSep();
393     return 1;
394 }
395 else if ( sLine == _T("locale reset") )
396 {
397     mu::console() << _T("Resetting locale\n");
398     a_Parser.ResetLocale();
399     return 1;
400 }
401 else if ( sLine == _T("test bulk") )
402 {
403     mu::console() << _T("Testing bulk mode\n");
404     CalcBulk();
405     return 1;
406 }
407
408 return 0;
409 }
410
411 //-----
412 void CalcBulk()
413 {
414     const int nBulkSize = 200;
415     value_type *x = new value_type[nBulkSize];
416     value_type *y = new value_type[nBulkSize];
417     value_type *result = new value_type[nBulkSize];
418
419     try
420     {
421         for (int i=0; i<nBulkSize; ++i)
422         {
423             x[i] = i;
424             y[i] = (value_type)i/10;
425         }
426         mu::Parser parser;
427         parser.DefineVar(_T("x"), x);
428         parser.DefineVar(_T("y"), y);
429         parser.DefineFun(_T("fun1"), BulkFun1);
430         parser.SetExpr(_T("fun1(0)+x+y"));
```

```

431     parser.Eval(result, nBulkSize);
432
433     for (int i=0; i<nBulkSize; ++i)
434     {
435         mu::console() << _T("Eqn. ") << i << _T(": x=") << x[i] << _T("; y=") << y[i]
436     }
437 }
438 catch(...)
439 {
440     delete [] x;
441     delete [] y;
442     delete [] result;
443     throw;
444 }
445
446 delete [] x;
447 delete [] y;
448 delete [] result;
449 }
450
451 //-----
452 void Calc()
453 {
454     mu::Parser  parser;
455
456     // Change locale settings if necessary
457     // function argument separator:  sum(2;3;4)  vs.  sum(2,3,4)
458     // decimal separator:            3,14        vs.  3.14
459     // thousands separator:         1000000      vs  1.000.000
460 // #define USE_GERMAN_LOCALE
461 #ifdef USE_GERMAN_LOCALE
462     parser.SetArgSep(';');
463     parser.SetDecSep(',');
464     parser.SetThousandsSep('.');
465 #else
466     // this is the default, so i it's commented:
467     // parser.SetArgSep(',');
468     // parser.SetDecSep('.');
469     // parser.SetThousandsSep('');
470 #endif
471
472     // Add some variables
473     value_type  vVarVal[] = { 1, 2 }; // Values of the parser variables
474     parser.DefineVar(_T("a"), &vVarVal[0]); // Assign Variable names and bind them to
475     parser.DefineVar(_T("b"), &vVarVal[1]);
476     parser.DefineStrConst(_T("strBuf"), _T("hello world"));
477     parser.AddValIdent(IsHexValue);

```

```
478
479 // Add user defined unary operators
480 parser.DefinePostfixOprt(_T("M"), Mega);
481 parser.DefinePostfixOprt(_T("m"), Milli);
482 parser.DefineInfixOprt(_T("!"), Not);
483 parser.DefineFun(_T("strfun0"), StrFun0);
484 parser.DefineFun(_T("strfun2"), StrFun2);
485 parser.DefineFun(_T("ping"), Ping);
486 parser.DefineFun(_T("rnd"), Rnd); // Add an unoptimizeable function
487 parser.DefineFun(_T("throw"), ThrowAnException);
488
489
490 parser.DefineOprt(_T("add"), Add, 0);
491 parser.DefineOprt(_T("mul"), Mul, 1);
492
493 // These are service and debug functions
494 parser.DefineFun(_T("debug"), Debug);
495 parser.DefineFun(_T("selftest"), SelfTest);
496 parser.DefineFun(_T("help"), Help);
497
498 #ifdef _DEBUG
499 // parser.EnableDebugDump(1, 0);
500 #endif
501
502 // Define the variable factory
503 parser.SetVarFactory(AddVariable, &parser);
504
505 for(;;)
506 {
507     try
508     {
509         string_type sLine;
510         std::getline(mu::console_in(), sLine);
511
512         switch (CheckKeywords(sLine.c_str(), parser))
513         {
514             case 0: break;
515             case 1: continue;
516             case -1: return;
517         }
518
519         if (!sLine.length())
520             continue;
521
522         parser.SetExpr(sLine);
523         mu::console() << std::setprecision(12);
524
```

```

525 // There are multiple ways to retrieve the result...
526 // 1.) If you know there is only a single return value or in case you only ne
527 //      result of an expression consisting of comma separated subexpressions y
528 //      simply use:
529 mu::console() << _T("ans=") << parser.Eval() << _T("\n");
530
531 // 2.) As an alternative you can also retrieve multiple return values using t
532 int nNum = parser.GetNumResults();
533 if (nNum>1)
534 {
535     mu::console() << _T("Multiple return values detected! Complete list:\n");
536
537     // this is the hard way if you need to retrieve multiple subexpression
538     // results
539     value_type *v = parser.Eval(nNum);
540     mu::console() << std::setprecision(12);
541     for (int i=0; i<nNum; ++i)
542     {
543         mu::console() << v[i] << _T("\n");
544     }
545 }
546 }
547 catch(mu::Parser::exception_type &e)
548 {
549     mu::console() << _T("\nError:\n");
550     mu::console() << _T("-----\n");
551     mu::console() << _T("Message:      ") << e.GetMsg() << _T("\n");
552     mu::console() << _T("Expression:  \") << e.GetExpr() << _T("\n");
553     mu::console() << _T("Token:      \") << e.GetToken() << _T("\n");
554     mu::console() << _T("Position:   ") << (int)e.GetPos() << _T("\n");
555     mu::console() << _T("Errc:      ") << std::dec << e.GetCode() << _T("\n");
556 }
557 } // while running
558 }
559
560 //-----
561 int main(int, char**)
562 {
563     Splash();
564     SelfTest();
565     Help();
566
567     // CheckLocale();
568     // CheckDiff();
569
570     mu::console() << _T("Enter an expression or a command:\n");
571

```

```

572 try
573 {
574     Calc();
575 }
576 catch(Parser::exception_type &e)
577 {
578     // Only erros raised during the initialization will end up here
579     // formula related errors are treated in Calc()
580     console() << _T("Initialization error: ") << e.GetMsg() << endl;
581     console() << _T("aborting...") << endl;
582     string_type sBuf;
583     console_in() >> sBuf;
584 }
585 catch(std::exception & /*exc*/)
586 {
587     // there is no unicode compliant way to query exc.what()
588     // so i'll leave it for this example.
589     console() << _T("aborting...\n");
590 }
591
592 return 0;
593 }

```

Create a block and open the example block:

```
$ bii init my_project
$ cd my_project
$ bii open examples/muparser
```

The `main.cpp` file is now in your block. Now you just have to build it and run the executable:

```
$ bii build
$ cd bin
$ # run executable
```

You will see next console output after executing the command:

```

/      \      \      \      \      \      \      \      \      \      \
|  Y Y  \  |  |  \  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|_|_|_|  /|_|_|/  |_|_|_|  (  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
      \      \      \      \      \      \      \      \      \      \
Version 2.2.3 (20121222; SF; 64BIT; RELEASE; ASCII)
(C) 2013 Ingo Berg
-----
Running test suite:

testing name restriction enforcement...passed

```

```

testing syntax engine...passed
testing postfix operators...
  fail: 1000{m} (Unexpected token "1000" found at position 0.)
  fail: 1000 {m} (Unexpected token "1000" found at position 0.)
  fail: f1of1(1000){m} (Unexpected token "1000" found at position 6.)
  fail: -f1of1(1000){m} (Unexpected token "1000" found at position 7.)
  fail: -f1of1(-1000){m} (Unexpected token "1000" found at position 8.)
  fail: f4of4(0,0,0,1000){m} (Unexpected token "1000" found at position 12.)
  fail: 2+(a*1000){m} (Unexpected token "1000" found at position 5.)
  fail: 2*3000meg+2 (Unexpected token "3000meg" found at position 2.)
  failed with 8 errors
testing infix operators...passed
testing variable/constant detection...passed
testing multiarg functions...passed
testing expression samples...passed
testing if-then-else operator...passed
testing member functions...passed
testing binary operators...passed
testing error codes...passed
testing string arguments...passed
Test failed with 8 errors (527 expressions)
-----
Commands:

  list var      - list parser variables
  list exprvar  - list expression variables
  list const    - list all numeric parser constants
  opt on        - enable optimizer (default)
  opt off       - disable optimizer
  locale de     - switch to german locale
  locale en     - switch to english locale
  locale reset  - reset locale
  test bulk     - test bulk mode
  quit         - exits the parser

Constants:

  "_e"  2.718281828459045235360287
  "_pi" 3.141592653589793238462643
-----
Enter an expression or a command:

```

You can now start typing mathematical expressions in the console.

Any doubts? Do not hesitate to [contact us](#) visit our [forum](#) and feel free to ask any questions.

1.8.29 OpenCV

OpenCV (Open Source Computer Vision Library) is a great open source computer vision and machine learning software library. OpenCV was built to provide a *common infrastructure for computer vision* applications and to accelerate the use of machine perception in the commercial products.

The versions of OpenCV library at biicode are:

- OpenCV 2.4.10 at [diego/opencv](#).
- OpenCV 3.0 beta at [diego/opencv \(beta\)](#).

Both versions are available at biicode using the *hooks feature* generated from this [github repo](#).

In this example we'll show how to get started with OpenCV using two features: showing an image and detecting faces with the [object detection module](#).

Showing an image and detecting faces

In this example we'll show how to get started with OpenCV using two features: showing an image and detecting faces with the [object detection module](#).

Creating a new project

Create a project and place the source files and images inside:

```
$ bii init opencv_sample -L
$ cd opencv_sample
$ # copy all following files inside
```

- Showing an image:

main.cpp

```
1 #include "opencv/cv.h"           // include it to used Main OpenCV functions.
2 #include "opencv/highgui.h"    //include it to use GUI functions.
3
4 int main(int argc, char** argv)
5 {
6     IplImage* img = cvLoadImage( "examples/opencv_sample/bii.png" ); //NOTE "e
7     cvNamedWindow( "Example1", CV_WINDOW_AUTOSIZE );
8
9     cvShowImage( "Example1", img );
10    cvMoveWindow( "Example1", 0, 0 );
11    cvWaitKey( 0 );
12    cvReleaseImage( &img );
13    cvDestroyWindow( "Example1" );
```



```

14     return 0;
15 }

```

Download file: [bii.png](#).

- Object detection example uses a CascadeClassifier class and a xml file with the algorithm to detect faces.

mainface.cpp

```

1  #include "opencv2/objdetect/objdetect.hpp"
2  #include "opencv2/highgui/highgui.hpp"
3  #include "opencv2/imgproc/imgproc.hpp"
4
5  #include <iostream>
6  #include <stdio.h>
7
8  using namespace std;
9  using namespace cv;
10
11  /** Function Headers */
12  void detectAndDisplay( Mat frame );
13
14  /** Global variables */
15  String face_cascade_name = "examples/opencv_sample/haarcascade_frontalface_al
16  String eyes_cascade_name = "examples/opencv_sample/haarcascade_eye_tree_eyegl
17  CascadeClassifier face_cascade;
18  CascadeClassifier eyes_cascade;
19  string window_name = "Capture - Face detection";
20  RNG rng(12345);
21
22  /** @function main */
23  int main( int argc, const char** argv )
24  {
25      CvCapture* capture;
26
27      //-- 1. Load the cascades
28      if( !face_cascade.load( face_cascade_name ) ){ printf("--(!)Error loading\n");
29      if( !eyes_cascade.load( eyes_cascade_name ) ){ printf("--(!)Error loading\n");
30
31      // Read the image file
32      Mat frame = imread("examples/opencv_sample/hugh.png"); //NOTE "examples" s
33      // Apply the classifier to the frame
34      if ( !frame.empty() )
35          detectAndDisplay(frame);
36      else{
37          printf(" --(!) No captured frame -- Break!");
38      }

```

```
39
40     int c = waitKey();
41
42     return 0;
43 }
44
45 /** @function detectAndDisplay */
46 void detectAndDisplay( Mat frame )
47 {
48     std::vector<Rect> faces;
49     Mat frame_gray;
50
51     cvtColor(frame, frame_gray, COLOR_BGR2GRAY);
52     equalizeHist( frame_gray, frame_gray );
53
54     //-- Detect faces
55     face_cascade.detectMultiScale(frame_gray, faces, 1.1, 2, 0 | CASCADE_SCALE_I
56
57     for( size_t i = 0; i < faces.size(); i++ )
58     {
59         Point center( faces[i].x + faces[i].width*0.5, faces[i].y + faces[i].height*0.5 );
60         ellipse( frame, center, Size( faces[i].width*0.5, faces[i].height*0.5 ), 0,
61
62         Mat faceROI = frame_gray( faces[i] );
63         std::vector<Rect> eyes;
64
65         //-- In each face, detect eyes
66         eyes_cascade.detectMultiScale(faceROI, eyes, 1.1, 2, 0 | CASCADE_SCALE
67
68         for( size_t j = 0; j < eyes.size(); j++ )
69         {
70             Point center( faces[i].x + eyes[j].x + eyes[j].width*0.5, faces[i].y +
71                 int radius = cvRound( (eyes[j].width + eyes[j].height)*0.25 );
72             circle( frame, center, radius, Scalar( 255, 0, 0 ), 4, 8, 0 );
73         }
74     }
75     //-- Show what you got
76     imshow( window_name, frame );
77 }
```

Download files: [hugh.png](#), [haarcascade_eye_tree_eyeglasses.xml](#),
[haarcascade_frontalface_alt.xml](#).

Manage your dependencies

Check the dependencies of the project with **bii deps**:

```
$ bii deps
INFO: Processing changes...
examples/mycvproject depends on:
  system:
    iostream
    stdio.h
  unresolved:
    opencv/cv.h
    opencv/highgui.h
    opencv2/highgui/highgui.hpp
    opencv2/imgproc/imgproc.hpp
    opencv2/objdetect/objdetect.hpp
```

Edit *biicode.conf* file generated in the project folder. Add your [requirements] depending on the version you want and map your [includes] and your [data]:

```
[requirements]
  diego/opencv: 2

[includes]
  opencv/*: diego/opencv
  opencv2/*: diego/opencv

[data]
  main.cpp + bii.png
  mainface.cpp + haarcascade_frontalface_alt.xml haarcascade_eye_tree_eyeglasses.xml
```

Now, checking **bii deps**, your dependencies are resolved.

Build the project

Now execute **bii build** to build the project.

Windows users need Visual Studio. Execute:

```
$ bii configure -G "Visual Studio 12"
```

```
$ bii build
```

Go to bin directory and execute the binaries:

```
$ cd bin
$ ./youruser_opencv_sample_main
```



```
$/youruser_opencv_sample_mainface
```



Open and build

This example is already in biicode at [examples/opencv_sample](#), so you can give it a try.

Create a new project and open the block:

```
$ bii init mycvproject
$ cd mycvproject
$ bii open examples/opencv_sample
```

Just build and run both examples:

```
$ bii build
$ ./youruser_opencv_sample_main
$ ./youruser_opencv_sample_mainface
```

Got any doubts? Do not hesitate to [contact us](#) visit our [forum](#) and feel free to ask any questions.

1.8.30 OpenSSL

OpenSSL is a robust, commercial-grade, full-featured, and Open Source toolkit implementing the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS) protocols as well as a full-strength general purpose cryptography library.

The versions of OpenSSL library at biicode are:

- OpenSSL 1.0.1 at [lasote/openssl](#).
- OpenSSL 1.0.2 at [lasote/openssl \(v1.0.2\)](#).

Both versions are generated from this [github repo](#).

Encrypting with MD5 and SHA1

This example shows how to use the **cryptography feature** of OpenSSL using a MD5 and SHA1 algorithm to encrypt a string.

Creating a new project

Create a new simple layout project with the name of your block and copy the code below:

```
$ bii init mysslproject -L
$ cd mysslproject
$ # Copy both files inside
```

The MD5 example encrypts a string “happy”:

md5.cpp

```
#include "openssl/md5.h"
#include <stdio.h>
#include <string.h>

int main()
{
    unsigned char digest[MD5_DIGEST_LENGTH];
    char string[] = "happy";

    MD5((unsigned char*)&string, strlen(string), (unsigned char*)&digest);

    char mdString[33];

    for(int i = 0; i < 16; i++)
        sprintf(&mdString[i*2], "%02x", (unsigned int)digest[i]);

    printf("md5 digest: %s\n", mdString);

    return 0;
}
```

Next, we'll use a similar code to encrypt a "hello world!" string with SHA1:

sha1.cpp

```
#include "openssl/sha.h"
#include <string.h>
#include <stdio.h>

int main() {
    // The data to be hashed
    const unsigned char data[] = "Hello, world!";
    size_t length = sizeof(data);

    unsigned char hash[SHA_DIGEST_LENGTH];
    SHA1(data, length, hash);

    char mdString[SHA_DIGEST_LENGTH];

    for(int i = 0; i < (SHA_DIGEST_LENGTH/2) - 1; i++)
        sprintf(&mdString[i*2], "%02x", (unsigned int)hash[i]);

    printf("sha1 digest: %s\n", mdString);

    return 0;
}
```

Manage your dependencies

Take a look at the `#include` pointing to OpenSSL. Type **bii deps** to see unresolved dependencies:

```
$ bii deps
INFO: Processing changes...
your_user/mysslproject depends on:
  system:
    stdio.h
    string.h
  unresolved:
    openssl/md5.h
    openssl/sha.h
```

Let's edit now the *biicode.conf* file generated in the project folder. Add your `[requirements]` depending on the version you want and map your `[includes]`:

```
[requirements]
  lasote/openssl: 3

[includes]
  openssl/*.h: lasote/openssl/include
```

Retrieve your unresolved dependencies:

```
$ bii find
INFO: Processing changes...
INFO: Downloading files from: lasote/openssl
INFO: Downloading files from: biicode/cmake
INFO: Saving files from: lasote/openssl
...
```

Build the project

Now execute **bii build**.

```
$ bii build
```

Go to `/bin` directory and execute the binaries:

```
$ cd bin
$ ./examples_openssl_md5
md5 digest: 56ab24c15b72a457069c5ea42fcfc640
```

That output is the MD5 hash value of your encrypted string.

```
$/examples_openssl_sha1
sha1 digest: 2d5ec68b0d061c75db
```

And that is the SHA1 hash value for the “hello world!” string.

Develop your project

You can develop your own project with OpenSSL. Choose your version or switch between them using the *block track feature*.

Just modify the `[requirements]` section in the *biicode.conf* file of your block’s folder:

- To depend on **OpenSSL 1.0.1**:

```
[requirements]
    lasote/openssl: 2
```

- To depend on **OpenSSL 1.0.2**:

```
[requirements]
    lasote/openssl(v1.0.2): 1
```

Open and build

This example is already in biicode: [examples/openssl](#).

Just open and build it to give it a quick try.

Create a new project and open the block:

```
$ bii init mysslproject
$ cd mysslproject
$ bii open examples/openssl
```

Now build and run the examples:

```
$ cd bin
$/examples_openssl_md5
md5 digest: 56ab24c15b72a457069c5ea42fcfc640

$/examples_openssl_sha1
sha1 digest: 2d5ec68b0d061c75db
```

Got any doubts? Do not hesitate to [contact us](#), visit our [forum](#) and feel free to ask any questions.

1.8.31 POCO

POCO POCO is an open-source third-party library, which contains a collection of C++ class libraries that simplify and accelerate the development of network-centric, portable applications in C++. With POCO you can develop network- and internet-based applications that run on desktop, server, mobile and embedded systems.

POCO C++ libraries available at biicode are:

- Poco (develop) at [fenix/poco \(develop\)](#).
- Poco v1.6.0 at [fenix/poco \(v1.6.0\)](#).
- Poco v1.5.4 at [fenix/poco \(v1.5.4\)](#).
- Poco v1.4.7p1 at [fenix/poco \(v1.4.7p1\)](#).

There are so many things you can try with POCO. This example shows a **PDF conversion** example from strings.

PDF example

This example is [already in biicode](#). So just open the block:

```
$ bii init pocoproject
$ cd pocoproject
$ bii open examples/poco_pdf
```

On Windows, configure Visual Studio.

```
$ bii configure -G "Visual Studio 12"
```

Choose your PDF output name, the font and size, the page of the document, the content of the string and the margins from the pdf page you want to generate:

pdf.cpp

```
1 #include "Poco/PDF/Document.h"
2 #include "Poco/Path.h"
3 #include "Poco/File.h"
4
5 const std::string fileName = "Text.pdf";
6
7 using Poco::PDF::Document;
8 using Poco::PDF::Font;
9 using Poco::PDF::Page;
10 using Poco::Path;
11 using Poco::File;
12 using namespace std;
```

```
13
14
15 int main(int argc, char** argv)
16 {
17     File file(Path::expand(fileName));
18     if (file.exists()) file.remove();
19
20     Document document(file.path());
21
22     Font helv = document.font("Helvetica");
23     Page page = document[0];
24     page.setFont(helv, 24);
25     std::string mystring = "PDF generated using POCO C++ Libraries";
26     float tw = page.textWidth(mystring);
27     page.writeOnce((page.getWidth() - tw) / 2, page.getHeight() - 50, mystring);
28
29     document.save();
30     cout<<fileName<<" saved correctly"<<endl;
31     return 0;
32 }
```

Note that the original [includes] are mapped in the *biicode.conf* file of the block.

```
# Biicode configuration file

[requirements]
    fenix/poco(develop): 0

[includes]
    Poco/PDF/*.h: fenix/poco/PDF/include
    Poco/*.h: fenix/poco/Foundation/include
```

Generate the PDF

Now execute **bii build** to build the project.

```
$ bii build
```

Go to bin directory and execute:

```
$ cd bin
$ ./examples_poco_pdf_pdf
Text.pdf saved correctly
```

Your pdf file is ready! Look for it in your */bin* folder.

Using NetSSL_OpenSSL or NetSSL_Win library

Making a project using **NetSSL_OpenSSL** and **NetSSL_Win** libraries is a special use case of original includes. Both libraries have the same relative *include* headers, so, the only way to resolve successfully your dependencies is writing the full path for them.

For example:

```
#include "Poco/URISStreamOpener.h"
#include "Poco/StreamCopier.h"
#include "Poco/Path.h"
#include "Poco/URI.h"
#include "Poco/SharedPtr.h"
#include "Poco/Exception.h"

/* headers in Net library */
#include "Poco/Net/HTTPStreamFactory.h"
#include "Poco/Net/FTPStreamFactory.h"

/* headers in NetSSL_OpenSSL library */
#include "fenix/poco/NetSSL_OpenSSL/include/Poco/Net/HTTPStreamFactory.h"
//Instead of #include "Poco/Net/HTTPStreamFactory.h" again.
#include "fenix/poco/NetSSL_OpenSSL/include/Poco/Net/SSLManager.h"
//Instead of #include "Poco/Net/SSLManager.h"
#include "fenix/poco/NetSSL_OpenSSL/include/Poco/Net/KeyConsoleHandler.h"
//Instead of #include "Poco/Net/KeyConsoleHandler.h"
#include "fenix/poco/NetSSL_OpenSSL/include/Poco/Net/ConsoleCertificateHandler.h"
//Instead of #include "Poco/Net/ConsoleCertificateHandler.h"

#include <memory>
#include <iostream>

/* Main code */
```

The *biicode.conf* would be:

```
[includes]
  Poco/Net/*.h: fenix/poco/Net/include
  Poco/*.h: fenix/poco/Foundation/include
```

Warning: take care with `Poco/*.h: fenix/poco/Foundation/include` because it should always be at the end of `[includes]` section for being a really wide search pattern.

Got any doubts? Do not hesitate to [contact us](#), visit our [forum](#) and feel free to ask any questions.

1.8.32 PTypes

PTypes is a simple alternative to the STL that includes multithreading and networking. It defines dynamic strings, character sets, lists and other basic data types along with threads, synchronization primitives, named pipes and IP sockets.

The following example demonstrates how to use PTypes streams to write to a file .

stream.cpp

```
1  /**
2   * Simple example to write to a file
3   */
4  #include <melikyan/ptypes/include/pstreams.h>
5  #include <melikyan/ptypes/include/pport.h>
6
7  using namespace pt;
8
9  int main() {
10     const char* fname = "stmtest.txt";
11     outfile f(fname, false);
12     f.set_bufsize(1024); // the default value in this version is 8192
13     try {
14         f.open();
15         f.put("This is a test file.\n");
16         f.close();
17     } catch (estream* e) {
18         perr.putf(e->get_message());
19         delete e;
20     }
21     return 0;
22 }
```

Create a block:

```
$ bii init my_project
$ cd my_project
$ bii open examples/ptypes
```

Compile stream.cpp file:

```
$ bii build
```

Now you can run it and check it generates a file:

```
$ cd bin
$ ./examples_ptypes_stream
```

This creates a `stmtest.txt` file with something written inside. Just open it and you'll see its content:

```
This is a test file
```

Any doubts? Do not hesitate to [contact us](#) visit our [forum](#) and feel free to ask any questions.

1.8.33 SDL

SDL: Simple DirectMedia Layer is a cross-platform development library designed to provide low level access to audio, keyboard, mouse, joystick, and graphics hardware via OpenGL and Direct3D.

SDL library is available in biicode at [miguel/sdl2](#).

Graphical window interface

In this example you will open a new graphical window and load a background texture.

Creating a new project

Create a new project and a *main_sdl.cpp* file:

```
$ bii init sdl_example -L
$ cd sdl_example
$ # Create main_sdl.cpp
```

Now place the following code inside *main_sdl.cpp*:

main_sdl.cpp

```
#include "SDL.h"
#include <iostream>

int main(int argc, char *argv[])
{
    if (SDL_Init(SDL_INIT_VIDEO) != 0)
    {
        std::cout << "SDL_Init Error: " << SDL_GetError() << std::endl;
        return 1;
    }

    SDL_Window *win = SDL_CreateWindow("Hello World!", 100, 100, 640, 480, SDL_WINDOW_OPENGL);

    SDL_Renderer *ren = SDL_CreateRenderer(win, -1, SDL_RENDERER_ACCELERATED | SDL_RENDERER_PRESENTVSYNC);

    SDL_Surface *bmp = SDL_LoadBMP("examples/sdl_example/media/sdl.bmp");

    SDL_Texture *tex = SDL_CreateTextureFromSurface(ren, bmp);
```

```
SDL_FreeSurface(bmp);

if (tex == nullptr)
{
    SDL_DestroyRenderer(ren);
    SDL_DestroyWindow(win);
    std::cout << "SDL_CreateTextureFromSurface Error: " << SDL_GetError() << std::endl;
    SDL_Quit();
    return 1;
}

//Wait for 3 seconds and render and present the screen each time
for (int i = 0; i < 3; ++i)
{
    //First clear the renderer
    SDL_RenderClear(ren);
    //Draw the texture
    SDL_RenderCopy(ren, tex, NULL, NULL);
    //Update the screen
    SDL_RenderPresent(ren);
    //Take a quick break
    SDL_Delay(3000);
}

SDL_DestroyTexture(tex);
SDL_DestroyRenderer(ren);
SDL_DestroyWindow(win);
SDL_Quit();
}
```

Now, add a cool background .bmp like [this one](#) to a new folder “**media**” and link it with your *main_sdl.cpp* file in *biicode.conf* [data] section like this:

```
[data]
    main_sdl.cpp + media/sdl.bmp
```

Now, change your path to the file in your code:

```
SDL_Surface *bmp = SDL_LoadBMP("your_user/sdl_example/media/sdl.bmp"); //Replace "y
```

Manage your dependencies

Check the dependencies of the project with **bii deps**:

```
$ bii deps
your_user/sdl_example depends on:
  system:
    iostream
  unresolved:
    SDL.h
```

Now, edit the *biicode.conf* file generated in the project folder. Add your [requirements] depending on the version you want and map your [includes] to point to miguel/sdl2/include/SDL.h:

```
[requirements]
  miguel/sdl2: 3

[includes]
  SDL.h: miguel/sdl2/include
```

Check again with **bii deps** and now all dependencies are solved.

Activating C++11

Building your project like this throws the next error:

```
$ bii build
...
C:\...\sdl_example\main_sdl.cpp:21:13: error: 'nullptr' was not declared in this s
if (tex == nullptr)
            ^
```

So we have to activate C++11 support. With biicode you can reuse cmake macros, so we'll use this one: [biicode/cmake/tools.cmake](#). It is very easy.

- Edit *CMakeLists.txt* and write:

```
# Including tools.cmake from biicode/cmake user block
INCLUDE(biicode/cmake/tools)

ADD_BII_TARGETS()

# Calling specific macro to activate c++11 flags
ACTIVATE_CPP11 (INTERFACE ${BII_BLOCK_TARGET})
```

- Type **bii find** and all is done!

```
$ bii find
INFO: Processing changes...
INFO: Finding missing dependencies in server
INFO: Looking for biicode/cmake...
```

```
...  
INFO: Saving files from: biicode/cmake
```

Build the project

The only thing left is building the project:

```
$ bii build
```

Execute the binary placed in bin directory and this is how output looks like:

```
$ cd bin  
$ ./your_user_sdl_example_main_sdl
```

Here is the result:



That's it!

Open and build

This example is already in biicode: [examples/sdl_example](#).

To give it a try, create a new project and open the block:

```
$ bii init sdl_project
$ cd sdl_project
$ bii open examples/sdl_example
```

Build the example and execute it:

```
$ bii build
$ cd bin
$ # Execute it
```

You will see your the above graphical window with SDL background texture.

Got any doubts? Do not hesitate to [contact us](#), visit our [forum](#) and feel free to ask any questions.

1.8.34 SQLite

SQLite is a software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. SQLite is the most widely deployed SQL database engine in the world.

For more information about this library, visit their [official website](#).

You can find [SRombauts' C++ SQLite3 wrapper library](#) at [sqlite/sqlite](#) and plain SQLite is at [fenix/sqlite](#).

Shopping list database

This example uses SQLiteC to create a database called LIST in which stores information about your shopping list. Take a deep look into the code in order to understand how it works and make your own one soon!

Creating a project

Create a new project with a simple layout and place the code inside.

```
$ bii init sqlite_basic -L
$ cd sqlite_basic
$ # create shopping_db.cpp and copy its content
```

shopping_db.cpp

```
#include <stdlib.h>
#include <sqlite3.h>
#include <string>
#include <stdio.h>

using namespace std;

static int select_callback(void *data, int argc, char **argv, char **azColName)
{
    int i;
    for(i=0; i<argc; i++){
        printf("%s = %s\n", azColName[i], argv[i] ? argv[i] : "NULL");
    }
    printf("\n");
    return 0;
}

void execute_sql(sqlite3 *db, string zSql, sqlite3_callback xCallback, void *pArg)
{
    int rc;
    char *pzErrMsg = 0;
    rc = sqlite3_exec(db, zSql.c_str(), xCallback, pArg, &pzErrMsg);
    if( rc != SQLITE_OK ){
        fprintf(stderr, "SQL error: %s\n", pzErrMsg);
        sqlite3_free(pzErrMsg);
    }
}

void connect(sqlite3 **db) {
    int rc;
    rc = sqlite3_open("test.db", db);
    if( rc ){
        fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(*db));
        exit(0);
    }else{
        fprintf(stdout, "Opened database successfully\n");
    }
}

int main(int argc, char* argv[])
{
    sqlite3 *db;
    sqlite3_callback void_callback;
    string query;

    /* Open database */
    connect(&db);

    query = "CREATE TABLE IF NOT EXISTS LIST (STORE CHAR(50),NAME TEXT NOT NULL)";
```

```

    execute_sql(db, query, void_callback, 0);

    query = "INSERT OR REPLACE INTO LIST (STORE, NAME, NUMBER) VALUES ('Veggies', 1, 1);";
    query = "INSERT OR REPLACE INTO LIST (NUMBER, STORE, NAME) VALUES (7, 'Drinks', 'Coffee');";
    query = "INSERT OR REPLACE INTO LIST (NAME, STORE, NUMBER) VALUES ('Onion', 'Veggies', 1);";
    execute_sql(db, query, void_callback, 0);

    printf("\nSELECT, List Veggies\n\n");
    query = "SELECT * from LIST where STORE='Veggies'";
    execute_sql(db, query, select_callback, 0);

    printf("\nSELECT, List Drinks\n\n");
    query = "SELECT * from LIST where STORE='Drinks'";
    execute_sql(db, query, select_callback, 0);

    query = "UPDATE LIST set NUMBER = 2 where NAME='Coffee'";
    execute_sql(db, query, void_callback, 0);

    query = "DELETE from LIST where NAME='Spinach'";
    execute_sql(db, query, void_callback, 0);

    printf("\nSELECT, Updated Lists:\n\n");
    query = "SELECT * from LIST";
    execute_sql(db, query, select_callback, 0);

    /* Close database */
    sqlite3_close(db);
    fprintf(stdout, "Closed database successfully\n");

    return 0;
}

```

Manage your dependencies

Check the dependencies of the project with **bii deps**:

```

$ bii deps
INFO: Processing changes...
your_user/sqlite_basic depends on:
  system:
    stdio.h
    stdlib.h
    string
  unresolved:

```

```
sqlite3.h
```

Edit the *biicode.conf* file generated in the project folder. Add your `[requirements]` depending on the version you want and map your `[includes]`:

```
[requirements]
    sqlite/sqlite: 8

[includes]
    sqlite3.h: sqlite/sqlite/sqlite3
```

Check again with **bii deps** to show all dependencies are now resolved.

Build the project

Build the *shopping_db.cpp* and execute it.

```
$ bii build
$ cd bin
$ # execute it!
```

You can see the results of the queries at the output:

```
SELECT, List Veggies

STORE = Veggies
NAME = Spinach
NUMBER = 3

STORE = Veggies
NAME = Onion
NUMBER = 1

SELECT, List Drinks

STORE = Drinks
NAME = Coffee
NUMBER = 7

SELECT, Updated Lists:

STORE = Drinks
NAME = Coffee
NUMBER = 2
```

```
STORE = Veggies
NAME = Onion
NUMBER = 1
```

```
Closed database successfully
```

SQLite++ Wrapper

The following example from [SRombauts](#), explains how-to use the SQLite++ wrapper. Following the previous example, we'll develop this in the same project's folder.

Jus place *main.cpp*, *example.db3* and *logo.png* files inside:

sqlite_basic/main.cpp

```
/**
 * @file main.cpp
 * @brief A few short examples in a row.
 *
 * Demonstrate how-to use the SQLite++ wrapper
 *
 * Copyright (c) 2012-2014 Sebastien Rombauts (sebastien.rombauts@gmail.com)
 *
 * Distributed under the MIT License (MIT) (See accompanying file LICENSE.txt
 * or copy at http://opensource.org/licenses/MIT)
 */

#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <cstdlib>
#include <SQLiteCpp/SQLiteCpp.h>

#ifdef SQLITECPP_ENABLE_ASSERT_HANDLER
namespace SQLite
{
    /// definition of the assertion handler enabled when SQLITECPP_ENABLE_ASSERT_HANDLER
    void assertion_failed(const char* apFile, const long apLine, const char* apFunc, const int apLineNo)
    {
        // Print a message to the standard error output stream, and abort the program.
        std::cerr << apFile << ":" << apLine << ":" << " error: assertion failed (" <<
        std::abort();
    }
}
#endif

/// Example Database
```

```
static const char* filename_example_db3 = "examples/sqlite_basic/example.db3"; //NOTE

/// Image
static const char* filename_logo_png    = "examples/sqlite_basic/logo.png"; //NOTE

/// Object Oriented Basic example
class Example
{
public:
    // Constructor
    Example() :
        mDb(filename_example_db3), // Open a database
        mQuery(mDb, "SELECT * FROM test WHERE weight > :min_weight") // Compile a SQL query
    {
    }
    virtual ~Example()
    {
    }

    /// List the rows where the "weight" column is greater than the provided aParamValue
    void ListGreaterThanOrEqual (const int aParamValue)
    {
        std::cout << "ListGreaterThanOrEqual (" << aParamValue << ") \n";

        // Bind the integer value provided to the first parameter of the SQL query
        mQuery.bind(":min_weight", aParamValue); // same as mQuery.bind(1, aParamValue)

        // Loop to execute the query step by step, to get one a row of results at a time
        while (mQuery.executeStep())
        {
            std::cout << "row (" << mQuery.getColumn(0) << ", \" " << mQuery.getColumn(1) << "\n";
        }

        // Reset the query to be able to use it again later
        mQuery.reset();
    }

private:
    SQLite::Database    mDb;      ///< Database connection
    SQLite::Statement    mQuery;  ///< Database prepared SQL query
};

int main ()
{
    // Basic example (1/6) :
```

```

try
{
    // Open a database file in readonly mode
    SQLite::Database db(filename_example_db3); // SQLITE_OPEN_READONLY
    std::cout << "SQLite database file '" << db.getFilename().c_str() << "' opened\n";

    // Test if the 'test' table exists
    bool bExists = db.tableExists("test");
    std::cout << "SQLite table 'test' exists=" << bExists << "\n";

    // Get a single value result with an easy to use shortcut
    std::string value = db.execAndGet("SELECT value FROM test WHERE id=2");
    std::cout << "execAndGet=" << value.c_str() << std::endl;

    // Compile a SQL query, containing one parameter (index 1)
    SQLite::Statement query(db, "SELECT id as test_id, value as test_val, weight as test_weight FROM test WHERE id=?");
    std::cout << "SQLite statement '" << query.getQuery().c_str() << "' compiled\n";
    // Bind the integer value 2 to the first parameter of the SQL query
    query.bind(1, 2);
    std::cout << "binded with integer value '2' :\n";

    // Loop to execute the query step by step, to get one a row of results at a time
    while (query.executeStep())
    {
        // Demonstrate how to get some typed column value (and the equivalent using the C++ API)
        int id = query.getColumn(0); // = query.getColumn(0).getInt();
        //const char* pvalue = query.getColumn(1); // = query.getColumn(1).getText();
        std::string value2 = query.getColumn(1); // = query.getColumn(1).getText();
        int bytes = query.getColumn(1).getBytes();
        double weight = query.getColumn(2); // = query.getColumn(2).getDouble();

        static bool bFirst = true;
        if (bFirst)
        {
            // Show how to get the aliased names of the result columns.
            std::string name0 = query.getColumn(0).getName();
            std::string name1 = query.getColumn(1).getName();
            std::string name2 = query.getColumn(2).getName();
            std::cout << "aliased result [" << name0.c_str() << ", " << name1.c_str() << ", " << name2.c_str() << "]\n";

#ifdef SQLITE_ENABLE_COLUMN_METADATA
            // Show how to get origin names of the table columns from which the data comes
            // Requires the SQLITE_ENABLE_COLUMN_METADATA preprocessor macro to be defined
            // also defined at compile times of the SQLite library itself.
            name0 = query.getColumn(0).getOriginName();
            name1 = query.getColumn(1).getOriginName();
            name2 = query.getColumn(2).getOriginName();
            std::cout << "origin table 'test' [" << name0.c_str() << ", " << name1.c_str() << ", " << name2.c_str() << "]\n";
#endif
        }
    }
}

```

```
#endif

        bFirst = false;
    }
    std::cout << "row (" << id << ", \"" << value2.c_str() << "\" " << byt
}

// Reset the query to use it again
query.reset();
std::cout << "SQLite statement '" << query.getQuery().c_str() << "' reseted
// Bind the string value "6" to the first parameter of the SQL query
query.bind(1, "6");
std::cout << "binded with string value \"6\" :\\n";

while (query.executeStep())
{
    // Demonstrate that inserting column value in a std::ostream is natural
    std::cout << "row (" << query.getColumn(0) << ", \"" << query.getColumn
}
}
catch (std::exception& e)
{
    std::cout << "SQLite exception: " << e.what() << std::endl;
    return EXIT_FAILURE; // unexpected error : exit the example program
}

////////////////////////////////////
// Object Oriented Basic example (2/6) :
try
{
    // Open the database and compile the query
    Example example;

    // Demonstrate the way to use the same query with different parameter value
    example.ListGreaterThan(8);
    example.ListGreaterThan(6);
    example.ListGreaterThan(2);
}
catch (std::exception& e)
{
    std::cout << "SQLite exception: " << e.what() << std::endl;
    return EXIT_FAILURE; // unexpected error : exit the example program
}

// The execAndGet wrapper example (3/6) :
try
{
    // Open a database file in readonly mode
```



```

SQLite::Database db(filename_example_db3); // SQLITE_OPEN_READONLY
std::cout << "SQLite database file '" << db.getFilename().c_str() << "' opened" << std::endl;

// WARNING: Be very careful with this dangerous method: you have to
// make a COPY OF THE result, else it will be destroyed before the next line
// (when the underlying temporary Statement and Column objects are destroyed)
std::string value = db.execAndGet("SELECT value FROM test WHERE id=2");
std::cout << "execAndGet=" << value.c_str() << std::endl;
}
catch (std::exception& e)
{
    std::cout << "SQLite exception: " << e.what() << std::endl;
    return EXIT_FAILURE; // unexpected error : exit the example program
}

//////////////////////////////////////
// Simple batch queries example (4/6) :
try
{
    // Open a database file in create/write mode
    SQLite::Database db("test.db3", SQLITE_OPEN_READWRITE|SQLITE_OPEN_CREATE);
    std::cout << "SQLite database file '" << db.getFilename().c_str() << "' opened" << std::endl;

    // Create a new table with an explicit "id" column aliasing the underlying
    db.exec("DROP TABLE IF EXISTS test");
    db.exec("CREATE TABLE test (id INTEGER PRIMARY KEY, value TEXT)");

    // first row
    int nb = db.exec("INSERT INTO test VALUES (NULL, 'test')");
    std::cout << "INSERT INTO test VALUES (NULL, 'test')\n", returned " << nb << std::endl;

    // second row
    nb = db.exec("INSERT INTO test VALUES (NULL, 'second')");
    std::cout << "INSERT INTO test VALUES (NULL, 'second')\n", returned " << nb << std::endl;

    // update the second row
    nb = db.exec("UPDATE test SET value='second-updated' WHERE id='2'");
    std::cout << "UPDATE test SET value='second-updated' WHERE id='2', returned " << nb << std::endl;

    // Check the results : expect two rows of result
    SQLite::Statement query(db, "SELECT * FROM test");
    std::cout << "SELECT * FROM test :\n";
    while (query.executeStep())
    {
        std::cout << "row (" << query.getColumn(0) << ", " << query.getColumn(1) << ")\n";
    }
}

```

```

        db.exec("DROP TABLE test");
    }
    catch (std::exception& e)
    {
        std::cout << "SQLite exception: " << e.what() << std::endl;
        return EXIT_FAILURE; // unexpected error : exit the example program
    }
    remove("test.db3");

    ////////////////////////////////////////
    // RAII transaction example (5/6) :
    try
    {
        // Open a database file in create/write mode
        SQLite::Database db("transaction.db3", SQLITE_OPEN_READWRITE|SQLITE_OPEN_CREATE);
        std::cout << "SQLite database file '" << db.getFilename().c_str() << "' opened" << std::endl;

        db.exec("DROP TABLE IF EXISTS test");

        // Exemple of a successful transaction :
        try
        {
            // Begin transaction
            SQLite::Transaction transaction(db);

            db.exec("CREATE TABLE test (id INTEGER PRIMARY KEY, value TEXT)");

            int nb = db.exec("INSERT INTO test VALUES (NULL, \"test\")");
            std::cout << "INSERT INTO test VALUES (NULL, \"test\")\n", returned " << nb << std::endl;

            // Commit transaction
            transaction.commit();
        }
        catch (std::exception& e)
        {
            std::cout << "SQLite exception: " << e.what() << std::endl;
            return EXIT_FAILURE; // unexpected error : exit the example program
        }

        // Exemple of a rollbacked transaction :
        try
        {
            // Begin transaction
            SQLite::Transaction transaction(db);

            int nb = db.exec("INSERT INTO test VALUES (NULL, \"second\")");
            std::cout << "INSERT INTO test VALUES (NULL, \"second\")\n", returned " << nb << std::endl;

            // Rollback transaction
            transaction.rollback();
        }
        catch (std::exception& e)
        {
            std::cout << "SQLite exception: " << e.what() << std::endl;
            return EXIT_FAILURE; // unexpected error : exit the example program
        }
    }
}

```

```

        nb = db.exec("INSERT INTO test ObviousError");
        std::cout << "INSERT INTO test \"error\", returned " << nb << std::endl;

        return EXIT_FAILURE; // unexpected success : exit the example program

    // Commit transaction
    transaction.commit();
}
catch (std::exception& e)
{
    std::cout << "SQLite exception: " << e.what() << std::endl;
    // expected error, see above
}

// Check the results (expect only one row of result, as the second one has
SQLite::Statement query(db, "SELECT * FROM test");
std::cout << "SELECT * FROM test :\n";
while (query.executeStep())
{
    std::cout << "row (" << query.getColumn(0) << ", \" " << query.getColumn(1) << "\")\n";
}
}
catch (std::exception& e)
{
    std::cout << "SQLite exception: " << e.what() << std::endl;
    return EXIT_FAILURE; // unexpected error : exit the example program
}
remove("transaction.db3");

////////////////////////////////////
// Binary blob and in-memory database example (6/6) :
try
{
    // Open a database file in create/write mode
    SQLite::Database db(":memory:", SQLITE_OPEN_READWRITE|SQLITE_OPEN_CREATE);
    std::cout << "SQLite database file '" << db.getFilename().c_str() << "' opened\n";

    db.exec("DROP TABLE IF EXISTS test");
    db.exec("CREATE TABLE test (id INTEGER PRIMARY KEY, value BLOB)");

    FILE* fp = fopen(filename_logo_png, "rb");
    if (NULL != fp)
    {
        char buffer[16*1024];
        void* blob = &buffer;
        int size = static_cast<int>(fread(blob, 1, 16*1024, fp));
    }
}

```

```

        buffer[size] = '\0';
        fclose (fp);
        std::cout << "blob size=" << size << " :\n";

        // Insert query
        SQLite::Statement query(db, "INSERT INTO test VALUES (NULL, ?)");
        // Bind the blob value to the first parameter of the SQL query
        query.bind(1, blob, size);
        std::cout << "blob binded successfully\n";

        // Execute the one-step query to insert the blob
        int nb = query.exec ();
        std::cout << "INSERT INTO test VALUES (NULL, ?)\", returned " << nb <<
    }
    else
    {
        std::cout << "file " << filename_logo_png << " not found !\n";
        return EXIT_FAILURE; // unexpected error : exit the example program
    }

    fp = fopen("out.png", "wb");
    if (NULL != fp)
    {
        const void* blob = NULL;
        size_t size;

        SQLite::Statement query(db, "SELECT * FROM test");
        std::cout << "SELECT * FROM test :\n";
        if (query.executeStep())
        {
            SQLite::Column colBlob = query.getColumn(1);
            blob = colBlob.getBlob ();
            size = colBlob.getBytes ();
            std::cout << "row (" << query.getColumn(0) << ", size=" << size <<
                size_t sizew = fwrite(blob, 1, size, fp);
            SQLITECPP_ASSERT(sizew == size, "fwrite failed"); // See SQLITECPP
            fclose (fp);
        }
    }
    else
    {
        std::cout << "file out.png not created !\n";
        return EXIT_FAILURE; // unexpected error : exit the example program
    }
}
catch (std::exception& e)
{

```

```

        std::cout << "SQLite exception: " << e.what() << std::endl;
        return EXIT_FAILURE; // unexpected error : exit the example program
    }
    remove("out.png");

    std::cout << "everything ok, quitting\n";

    return EXIT_SUCCESS;
}

```

Download: `example.db3`, `logo.png`.

Manage your dependencies

Check again with **bii deps** and edit the *biicode.conf* file.

```

[requirements]
    sqlite/sqlite: 8

[includes]
    sqlite3.h: sqlite/sqlite/sqlite3
    SQLiteCpp/*: sqlite/sqlite/include

[data]
    main.cpp + examle.db3 logo.png

```

Build the example

Compile it and run the executable by doing:

```

$ bii build
$ cd bin
$ # run executable

```

You will see next console output:

```

SQLite database file 'examples/sqlite/example.db3' opened successfully
SQLite table 'test' exists=1
execAndGet=second line
SQLite statement 'SELECT id as test_id, value as test_val, weight as test_weight FROM test'
bound with integer value '2' :
aliased result ["test_id", "test_val", "test_weight"]
row (1, "first word" 10 bytes, 2.3)
row (2, "second line" 11 bytes, 6.7)
row (3, "and a last one" 14 bytes, 9.5)

```

```
row (4, "" 0 bytes, 18)
...
```

Open and build

This examples are already in biicode at [examples/sqlite_basic](#) and [examples/sqlite](#).

This is a way to give them a quick look and check how it works.

Both examples are simple to run, just open the blocks and build them like this:

```
$ bii init sqlite_project
$ cd sqlite_project
$ bii open examples/sqlite_basic
$ bii open examples/sqlite
$ bii build
```

Any doubts? Do not hesitate to [contact us](#) visit our [forum](#) and feel free to ask any questions.

1.8.35 TinyThread++

TinyThread++ is a minimalist, portable, threading library for C++, intended to make it easy to create multi threaded C++ applications. The library is closesly modeled after the C++11 standard, but only a subset is implemented at the moment. Need portable threads for your C++ app? Use TinyThread++!

The main block is [here](#), which is generated from this [github repo](#).

Open and build the examples

This example is [already in biicode](#). It is very simple to build it, by just opening the block.

```
$ bii init tinythread
$ cd tinythread
$ bii open examples/tinythread
$ bii build
```

Simple Hello World with a thread

Now, run the hello example.

```
$ bin/examples_tinythread_hello
```

You should see how a thread say “Hello world!”

This Hello World just defines the thread method

```
void HelloThread(void * aArg)
{
    cout << "Hello world!" << endl;
}
```

and then, launch the thread and wait for it.

```
thread t(HelloThread, 0);
t.join();
```

You can see all the code [here](#).

Draw a fractal

If you run the fractal example, a set of threads will be launched to generate a fractal image. You can see how to launch diferent threads to do a distributed task.

```
// Start calculation threads (we run one thread on each processor core)
cout << "Running " << numThreads << " calculation thread(s)..." << flush;
list<thread*> threadList;
for(int i = 0; i < numThreads; ++ i)
{
    thread * t = new thread(CalcThread, (void *) &dispatcher);
    threadList.push_back(t);
}

// Wait for the threads to complete...
for(list<thread*>::iterator i = threadList.begin(); i != threadList.end(); ++ i)
{
    thread * t = *i;
    t->join();
    delete t;
}
cout << "done!" << endl;

// Write the final image to a file
cout << "Writing final image..." << flush;
img.WriteToTGAFFile("fractal.tga");
cout << "done!" << endl;
```

You can see [all the code here](#).

```
$ bin/examples_tinythread_fractal
```

1.8.36 Zlib

Zlib is a software library written in C language used for data compression. For more information about this library, visit its [official website](#) or [Documentation](#).

This is the [biicode library site](#) and this is the [biicode library examples](#).

Usage example (difficulty: medium)

To check this library, we're using an [example](#) to compress and decompress a single file. We will use a *.h and a *.cpp file named **infdef.h** (inflate and deflate) and a **zpipe.cpp**.

The following code would be in each one.

infdef.h

```
1  #pragma once
2
3  #include <stdio.h>
4  #include <assert.h>
5  #include "zlib/zlib/zlib.h"
6
7  /***** Methods declarations *****/
8
9  /* Compress from file source to file dest until EOF on source.
10 def() returns Z_OK on success, Z_MEM_ERROR if memory could not be
11 allocated for processing, Z_STREAM_ERROR if an invalid compression
12 level is supplied, Z_VERSION_ERROR if the version of zlib.h and the
13 version of the library linked do not match, or Z_ERRNO if there is
14 an error reading or writing the files. */
15
16 int def(FILE *source, FILE *dest, int level);
17
18 /* Decompress from file source to file dest until stream ends or EOF.
19 inf() returns Z_OK on success, Z_MEM_ERROR if memory could not be
20 allocated for processing, Z_DATA_ERROR if the deflate data is
21 invalid or incomplete, Z_VERSION_ERROR if the version of zlib.h and
22 the version of the library linked do not match, or Z_ERRNO if there
23 is an error reading or writing the files. */
24
25 int inf(FILE *source, FILE* dest);
26
27
28 /* report a zlib or i/o error */
29
30 void zerr(int ret);
```

infdef.cpp


```

1  #include "infdef.h"
2
3  #define CHUNK 20000
4
5  int def(FILE *source, FILE *dest, int level)
6  {
7      int ret, flush;
8      unsigned have;
9      z_stream strm;
10     unsigned char in[CHUNK];
11     unsigned char out[CHUNK];
12
13     /* allocate deflate state */
14     strm.zalloc = Z_NULL;
15     strm.zfree = Z_NULL;
16     strm.opaque = Z_NULL;
17     ret = deflateInit(&strm, level);
18     if (ret != Z_OK)
19         return ret;
20
21     /* compress until end of file */
22     do {
23         strm.avail_in = fread(in, 1, CHUNK, source);
24         if (ferror(source)) {
25             (void)deflateEnd(&strm);
26             return Z_ERRNO;
27         }
28         flush = feof(source) ? Z_FINISH : Z_NO_FLUSH;
29         strm.next_in = in;
30
31         /* run deflate() on input until output buffer not full, finish
32            compression if all of source has been read in */
33         do {
34             strm.avail_out = CHUNK;
35             strm.next_out = out;
36             ret = deflate(&strm, flush); /* no bad return value */
37             assert(ret != Z_STREAM_ERROR); /* state not clobbered */
38             have = CHUNK - strm.avail_out;
39
40             if (fwrite(out, 1, have, dest) != have || ferror(dest)) {
41                 (void)deflateEnd(&strm);
42                 return Z_ERRNO;
43             }
44         } while (strm.avail_out == 0);
45         assert(strm.avail_in == 0); /* all input will be used */
46
47         /* done when last data in file processed */

```

```
48     } while (flush != Z_FINISH);
49     assert(ret == Z_STREAM_END); /* stream will be complete */
50
51     /* clean up and return */
52     (void) deflateEnd(&strm);
53     return Z_OK;
54 }
55
56 int inf(FILE *source, FILE* dest)
57 {
58     int ret;
59     unsigned have;
60     z_stream strm;
61     unsigned char in[CHUNK];
62     unsigned char out[CHUNK];
63
64     /* allocate inflate state */
65     strm.zalloc = Z_NULL;
66     strm.zfree = Z_NULL;
67     strm.opaque = Z_NULL;
68     strm.avail_in = 0;
69     strm.next_in = Z_NULL;
70     ret = inflateInit(&strm);
71     if (ret != Z_OK)
72         return ret;
73
74     /* decompress until deflate stream ends or end of file */
75     do {
76         strm.avail_in = fread(in, 1, CHUNK, source);
77         if (ferror(source)) {
78             (void) inflateEnd(&strm);
79             return Z_ERRNO;
80         }
81         if (strm.avail_in == 0)
82             break;
83         strm.next_in = in;
84
85         /* run inflate() on input until output buffer not full */
86         do {
87             strm.avail_out = CHUNK;
88             strm.next_out = out;
89             ret = inflate(&strm, Z_NO_FLUSH);
90             assert(ret != Z_STREAM_ERROR); /* state not clobbered */
91             switch (ret) {
92                 case Z_NEED_DICT:
93                     ret = Z_DATA_ERROR; /* and fall through */
94                 case Z_DATA_ERROR:
```

```

95         case Z_MEM_ERROR:
96             (void)inflateEnd(&strm);
97             return ret;
98     }
99     have = CHUNK - strm.avail_out;
100     if (fwrite(out, 1, have, dest) != have || ferror(dest)) {
101         (void)inflateEnd(&strm);
102         return Z_ERRNO;
103     }
104     } while (strm.avail_out == 0);
105
106     /* done when inflate() says it's done */
107     } while (ret != Z_STREAM_END);
108
109     /* clean up and return */
110     (void)inflateEnd(&strm);
111     return ret == Z_STREAM_END ? Z_OK : Z_DATA_ERROR;
112 }
113
114 /* report a zlib or i/o error */
115 void zerr(int ret)
116 {
117     fputs("zpipe: ", stderr);
118     switch (ret) {
119         case Z_ERRNO:
120             if (ferror(stdin))
121                 fputs("error reading stdin\n", stderr);
122             if (ferror(stdout))
123                 fputs("error writing stdout\n", stderr);
124             break;
125         case Z_STREAM_ERROR:
126             fputs("invalid compression level\n", stderr);
127             break;
128         case Z_DATA_ERROR:
129             fputs("invalid or incomplete deflate data\n", stderr);
130             break;
131         case Z_MEM_ERROR:
132             fputs("out of memory\n", stderr);
133             break;
134         case Z_VERSION_ERROR:
135             fputs("zlib version mismatch!\n", stderr);
136     }
137 }

```

zpipe.cpp

```
1 #include "infdef.h"
2 #include <iostream>
3
4 using namespace std;
5
6 /* compress or decompress */
7 int main(void)
8 {
9     int ret;
10    string request="";
11
12    while (true)
13    {
14        cout<<"Do you want to compress or decompress a file (to exit 'CTRL+C')? <
15        cin>>request;
16        if (request=="comp" || request=="decomp")
17            break;
18    }
19
20    string name_file_source = "";
21    string name_file_dest = "";
22    float size_src=0.0, size_dest=0.0;
23
24    /*
25     Enter or path to file, for example:
26     C:/Users/User/file.txt
27     Or
28     If you save the file in the bin folder, where
29     you will have the *.exe file, you just need write
30     name and extension file:
31         file.txt
32    */
33
34    cout<<"Enter file source name: "<<endl;
35    cin>>name_file_source;
36    cout<<"Enter file destination name: "<<endl;
37    cin>>name_file_dest;
38
39    FILE *source=NULL;//original file
40    FILE* dest=NULL; //file to compress or decompress
41
42    /* do compression if "comp" specified */
43    if (request == "comp") {
44        try
45        {
46            source = fopen(name_file_source.c_str(),"r");
47            if(!source)
```

```

48         throw 0;
49     }
50     catch (int n)
51     {
52         cout << "File does not exist, method return with " << n << endl;
53         return 0;
54     }
55
56     dest = fopen(name_file_dest.c_str(), "wb"); //wb because write to binary fo
57     ret = def(source, dest, Z_DEFAULT_COMPRESSION);
58     if (ret != Z_OK)
59         zerr(ret);
60     else
61     {
62         /* Calculate size file to see the difference */
63         size_src = ftell(source);
64         size_dest = ftell(dest);
65
66         cout<<"The original file size is: "<<size_src<<endl;
67         cout<<"The file size after being compressed is: "<<size_dest<<endl;
68     }
69     return ret;
70 }
71
72 /* do decompression if "decomp" specified */
73 else {
74     try
75     {
76         source = fopen(name_file_source.c_str(), "rb"); //rb because read fr
77         if(!source)
78             throw 0;
79     }
80     catch (int n)
81     {
82         cout << "File does not exist, method return with " << n << endl;
83         return 0;
84     }
85
86     dest = fopen(name_file_dest.c_str(), "w");
87     ret = inf(source, dest);
88     if (ret != Z_OK)
89         zerr(ret);
90     else
91     {
92         /* Calculate size file to see the difference */
93         size_src = ftell(source);
94         size_dest = ftell(dest);

```

```

95
96         cout<<"The original file size is: "<<size_src<<endl;
97         cout<<"The file size after being decompressed is: "<<size_dest<<endl;
98     }
99     return ret;
100 }
101 return 1;
102 }
```

Create a new project and open the example block:

```

$ bii init zlib_example
$ cd zlib_example
$ bii open examples/zlib
```

Next, you can build and run the code:

```

$ bii build
$ cd bin
$ #run executable
```

Then you'll be requested to select compression or decompression any file, and the file source name and file destination name.

This will be the output if you would want to compress a file.txt which is in your desktop directory and the compressed name file would be file.gz. The last one will be created in your ~/project_directory/bin/ directory

```

Do you want to compress or decompress a file (to exit 'CTRL+C')? <comp|decomp>
comp
Enter file source name:
C:\Users\Usuario\Desktop\file.txt
Enter file destination name:
file.gz
The original file size is: 16944
The file size after being compressed is: 5152
```

In this case file.txt had a size of 16944 bytes and file.gz 5152 bytes.

Now, if you want to decompress the file.gz to check all is correct, run the code again and the output will be the next

```

Do you want to compress or decompress a file (to exit 'CTRL+C')? <comp|decomp>
decomp
Enter file source name:
file.gz
Enter file destination name:
myfile.txt
The original file size is: 5152
```

```
The file size after being decompressed is: 16944
```

Finally, you can see that the new file, myfile.txt, has been created in the ~/project_directory/bin/ directory and it's exactly like the file.txt in terms of size and content.

1.8.37 ZMQ

[ZMQ](#) is a multiplatform high level socket library (sockets with steroids), that implements many paradigms as multicast, broadcast, client-server, etc. in a new and brilliant way. It is very actively developed, with a great community and used in many real projects. Are you thinking in building your own distributed application? Don't think more, use ZMQ.

The main block is [here](#), which is generated from this [github repo](#).

You can read a post about this example and the ZMQ project in [our blog](#).

Simple client-server with C++ binding

This example uses the C++ binding, published in [this block](#) which in turn depends on the main libzmq block explained above.

Of course it is possible to just copy the source code files as explained in [the blog post](#), if you want to check that running them is as simple as copying the code inside a biicode project.

But as this example is [already in biicode](#), it is very simple to build it, by just opening the block.

```
$ bii init clientserver
$ cd clientserver
$ bii open examples/zmq_cpp
```

If in windows, it is necessary to specify VS (it doesnt work with MinGW), otherwise, you can skip this step.

```
$ bii configure -G "Visual Studio 12"
```

Then build and run, first the server, then the client (you need to open another console, in the same folder).

```
$ bii build
$ bin/examples_zmq_cpp_hwserver
// another console
$ bin/examples_zmq_cpp_hwclient
```

You should see how the client send "Hello"s to the server and the server respond back "World".

1.8.38 ZMQ with Google Protocol Buffers Serialization

See previous example for more information about ZMQ.

An aspect that ZMQ does not cover is the serialization of messages, how to convert from classes to a flat string (possibly binary for efficiency) and viceversa.

Protocol Buffers from Google is a serialization framework, that can be used for example for storing and retrieving back information from disk. In this example we will use it for sending information over a ZMQ socket.

Protocol buffers can be found in [this block](#) which was very easily published from this [github repo](#).

We will first create a project and open the protocol buffers block:

```
$ bii init zmq_protobuf
$ cd zmq_protobuf
$ bii open google/protobuf
$ bii build
```

This will build the Protocol Buffers compiler (protoc or protoc.exe). Note that this is an optional step, you can of course download manually this binary from the web.

Lets open now the examples block. As usual you can also create an empty block and copy-paste your files there. We can also close the google/protobuf block, we have already compiled the “protoc” application, and we no longer need it (it will be partly used as a dependency).

```
$ bii open examples/zmq_protobuf
$ bii close google/protobuf
```

In protocol-buffers, messages are defined in “.proto” files, that are converted into source code files by the `protoc` generator. In this example we use the “message.proto” file used in the original tutorial:

```
package tutorial;

message Person {
  required string name = 1;
  required int32 id = 2;
  optional string email = 3;

  enum PhoneType {
    MOBILE = 0;
    HOME = 1;
    WORK = 2;
  }

  message PhoneNumber {
    required string number = 1;
    optional PhoneType type = 2 [default = HOME];
  }
}
```



```

    }

    repeated PhoneNumber phone = 4;
}

```

Creating the code is simple, move to the folder where this file is located and run the generator:

```

$ cd blocks/examples/zmq_protobuf
$ ../../../../bin/protoc message.proto --cpp_out="."

```

This will generate 2 files: “message.pb.h” and “message.pb.cc”. These two files can be used from the client and server in the following way:

hwclient.cpp

```

#include "diego/zmqcpp/zmq.hpp"
#include "message.pb.h"
#include <string>
#include <iostream>

int main ()
{
    GOOGLE_PROTOBUF_VERIFY_VERSION;

    tutorial::Person person; //fill a person data
    person.set_id(1234);
    person.set_name("john");
    person.set_email("john@mycompany.com");
    tutorial::Person::PhoneNumber* phone_number = person.add_phone();
    phone_number->set_number("1234567");
    phone_number->set_type(tutorial::Person::MOBILE);
    phone_number = person.add_phone();
    phone_number->set_number("7654321");
    phone_number->set_type(tutorial::Person::HOME);

    // Prepare our context and socket
    zmq::context_t context (1);
    // Note we use here a PAIR socket, only 1 way message
    zmq::socket_t socket (context, ZMQ_PAIR);

    std::cout << "Connecting to server" << std::endl;
    socket.connect ("tcp://localhost:5555");

    std::string msg_str;
    person.SerializeToString(&msg_str);
    // create a zmq message from the serialized string
    zmq::message_t request (msg_str.size());

```

```
memcpy ((void *) request.data (), msg_str.c_str(), msg_str.size());
std::cout << "Sending Person data ..." << std::endl;
socket.send (request);

// Optional: Delete all global objects allocated by libprotobuf.
google::protobuf::ShutdownProtobufLibrary();
return 0;
}
```

hwserver.cpp

```
#include "diego/zmqcpp/zmq.hpp"
#include <string>
#include <iostream>
#ifdef _WIN32
#include <unistd.h>
#else
#include <windows.h>
#endif

#include "message.pb.h"
#include <google/protobuf/text_format.h>

int main () {
    // Prepare our context and socket
    zmq::context_t context (1);
    zmq::socket_t socket (context, ZMQ_PAIR);
    socket.bind ("tcp://*:5555");

    while (true) {
        zmq::message_t request;
        // Wait for next request from client
        socket.recv (&request);
        std::cout << "Received" << std::endl;
        tutorial::Person person;
        std::string msg_str (static_cast<char*>(request.data()), request.size());
        person.ParseFromString (msg_str);
        std::string text_str;
        google::protobuf::TextFormat::PrintToString (person, &text_str);
        std::cout << text_str << std::endl;
    }
    return 0;
}
```

To resolve and retrieve dependencies, we just run the command:

```
$ bii find
```

If in windows, it is necessary to specify VS (it doesnt work with MinGW), otherwise, you can skip this step

```
$ bii configure -G "Visual Studio 12"
```

Then build and run, first the server, then the client (you need to open another console, in the same folder)

```
$ bii build
$ bin/examples_zmq_protobuf_hwserver
// another console
$ bin/examples_zmq_protobuf_hwclient
//in the server console:
Received
name: "john"
id: 1234
email: "john@mycompany.com"
phone {
  number: "1234567"
  type: MOBILE
}
phone {
  number: "7654321"
  type: HOME
}
```

1.9 Integrations

Biicode integrates with other technologies and tools. We're already working with the services below:

1.9.1 Generators and IDEs

Use CMake generators to **create biicode projects for many IDEs and platforms**. By default, biicode configures your projects with no IDE and MinGW (Windows) and UNIX Makefiles (MacOS and Linux).

Execute `cmake --help` to see the full list of CMake generators available in your system:

```
$ cmake --help
```

Choose yours, and tell biicode to configure your project for that IDE:

```
$ bii configure -G "CMake generator name"
```

`bii configure` admits any other CMake directives. For example, to enable the **Debug** build configuration,(e.g. Visual Studio, Eclipse):

```
$ bii configure -G "CMake generator name" -DCMAKE_BUILD_TYPE=Debug
```

Note: When you are working with an IDE (Visual, Eclipse, CLion), and you change your project structure, add/remove files or dependencies, you might need to run **bii configure** again.

Eclipse CDT

To create an Eclipse CDT project, run:

```
$ bii configure -G "Eclipse CDT4 - Unix Makefiles"
```

Windows users might configure it like this:

```
$ bii configure -G "Eclipse CDT4 - MinGW Makefiles"
```

Now, import your project into the Eclipse IDE.

1. From the main Eclipse menu choose: *File > import...*
2. Now, select *general > Existing Projects into Workspace*, and click next.
3. Select the root directory as the **root folder of your project**.
4. You should see a project already selected in the *projects* box. Click *finish*.

If you want to add new files to your block, just right-click on the folder of your block and create a new file.

You can build your application in *Project > Build project*

If you are using Eclipse with **Mac OS X**, you will need some additional setup to execute your binaries within this IDE. [Read this troubleshooting section for more information](#).

Visual Studio

Generate a Microsoft Visual Studio 12 (2013) project:

```
$ bii configure -G "Visual Studio 12"
```

Open your project with Visual Studio. Just **double-click on the .sln file inside the build folder** of your project and a VS project will open.

CLion

Use - the intelligent cross-platform C/C++ IDE - with biicode.

Get into your project's folder and execute:

```
$ bii init myproject -l=clion
```

And configure your project to set the changes:

```
$ bii configure
```

Open the biicode project with CLion (*File -> Open*).

Use biicode's commands from the embedded Terminal in CLion, open it with: **Alt+F12** or *View -> Tool Windows -> Terminal*.

Here's more info about *CLion's project layout*

1.9.2 IDEs and VCS

Eclipse or CLion IDEs need the classic biicode folder layout (can't handle your code directly in the root folder). Next steps cover how to use them when importing a project from a git repo. Just clone/checkout the code at the corresponding folder.

- CLion:

```
$ bii init myproject -l=clion
$ cd myproject
$ git clone https://github.com... blocks/username/blockname
$ bii configure
```

- Eclipse CDT projects:

```
$ bii init myproject
$ cd myproject
$ git clone https://github.com... blocks/username/blockname
$ bii configure -G "Eclipse CDT4 - MinGW Makefiles" (or Unix)
```

If you have any questions, we are available at and/or . You can also for suggestions and feedback.

1.9.3 Git (GitHub, Bitbucket, etc.)

and are notorious Git (Bitbucket works with Mercurial too) repository web-based hosting service which offers the distributed revision control and source code management (SCM) functionality of Git as well as add their own features.

Biicode does not intend to be version control system. We recommend you using a control version system so you can keep your code safe and versioned. You can use also SVN or CVS.

With a new repository

Just `init` the git repository in your block folder. As in the example:

```
$ bii init my_block -L
$ cd my_block
$ git init .
$ git add .
$ git commit -m "my very first commit"
```

You can also add a remote repository:

```
$ git remote add origin https://github.com/user/repo.git
```

Create a block from a git repository

The code

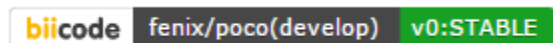
Put your code into a biicode block, as usual:

```
~$ git clone https://Your_Repo_URL.git
~$ cd your_repository
~/your_repository$ bii init -L
```

And follow this [guide to adapt your library to biicode](#).

biicode status badge

The biicode satus badge is a dynamically generated image displaying your block's latest published *version* in biicode.



This badge lets developers know your code is available to reuse at biicode. It is something determinant in the use of a dependency manager and you can place it in the *readme files* of your *VCS repository* and in the biicode block.

Get your badge in your block's **settings**.

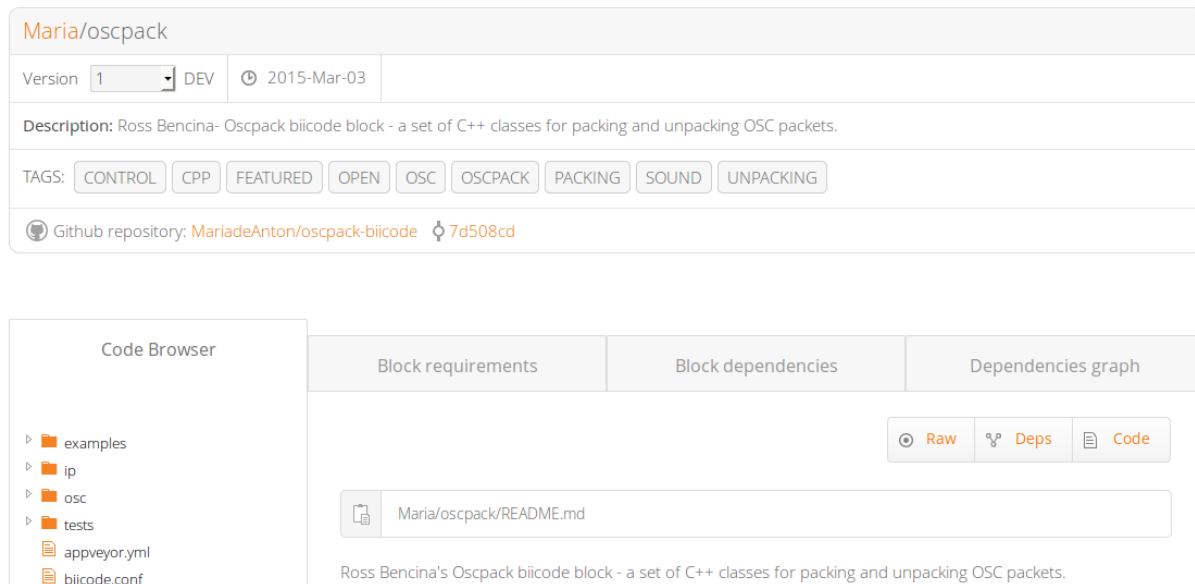
Let people know your code can be reused easily!

Publish from git commit

`bii publish -r` or `bii publish --remote` uses the git info within your block to publish it to biicode along with your block. This way everyone knows “who is” the git repo maintaining the biicode block and the specific commit creating each block version.

```
$ bii publish -r
```

This is how publishing with `bii publish --remote` looks like:



You can mix `bii publish` parameters, for example: `bii publish -r --tag STABLE --versiontag v1.0.2`

Check our [FAQ](#) for questions and answers. You can also [submit suggestions and feedback](#).

1.9.4 Continuous Integration

AppVeyor

provides Continuous Integration and Deploy for Windows and it is compatible with both `msvc` and `gcc`.

Place an `appveyor.yml` file in your repository and each time you push to your Github repository it will kick-off a new build in Windows, executing your tests and publishing it as a biicode block into your biicode user account.

Login AppVeyor and click on `+ NEW PROJECT` and choose the repository you want to deploy with. Create an `appveyor.yml` file in your local project to automatically publish your block to biicode, including your version tags, here's an example file:

```
version: 1.0.{build}

install:
  - cmd: cinst cmake -version 3.0.2 -y
  - cmd: cmake --version
  - cmd: echo "Downloading biicode..."
  - ps: wget http://www.biicode.com/downloads/latest/win -OutFile bii-win.exe
  - cmd: bii-win.exe /VERYSILENT
  - cmd: set PATH=%PATH%;C:\Program Files (x86)\BiiCode\bii
  - cmd: bii -v
  - cmd: del bii-win.exe

before_build:
  - cmd: bii init -L
  - cmd: bii cpp:configure -G "Visual Studio 12"

build_script:
  - cmd: bii build

test_script:
  - cmd: cd bin
  - cmd: amalulla_cpp-expression-parser_test-shunting-yard.exe

deploy_script:
  - cmd: bii user %block_user% -p %secured_passwd%
  - cmd: bii publish -r

environment:
  block_user:
    "amalulla"
  secured_passwd:
    secure: ENCRYPTED_BIIICODE_PASSWORD_HERE
```

Encrypt your biicode password and your access token using `openssl`, copy the values generated and put them in your environment like `secured_password: secure:..`. Use your own `test_script` and environment values to start deploying with it.

Here's an appveyor guide about how to `publish`. Following this Appveyor Guide we're using it as credentials with Git commands. Use this [GitHub guide](#) to create your `ssh-key`. This is a full *appveyor.yml* file to automatically publish to biicode DEV and STABLE versions:

```
version: 1.0.{build}

install:
  - cmd: cinst cmake -version 3.0.2 -y
  - cmd: cmake --version
  - cmd: echo "Downloading biicode..."
```



```

- ps: wget http://www.biicode.com/downloads/latest/win -OutFile bii-win.exe
- cmd: bii-win.exe /VERYSILENT
- cmd: set PATH=%PATH%;C:\Program Files (x86)\BiiCode\bii
- cmd: bii -v
- cmd: del bii-win.exe

before_build:
- cmd: bii init -L
- cmd: bii cpp:configure -G "Visual Studio 12"
- cmd: bii user %block_user%

build_script:
- cmd: bii build

test_script:
- cmd: cd bin
- cmd: amalulla_cpp-expression-parser_test-shunting-yard.exe

deploy_script:
- cmd: bii user %block_user% -p %secured_passwd%
- if defined APPVEYOR_REPO_TAG_NAME set VERSION=%APPVEYOR_REPO_TAG_NAME%
- if defined APPVEYOR_REPO_TAG_NAME bii publish -r --tag=STABLE --versiontag=%VERSION%
- if not defined APPVEYOR_REPO_TAG_NAME bii publish -r

on_success:
- cmd: cd /%project_name%/blocks/%block_user%/%block_name%
- ps: |
    $new_biiconf = get-content biicode.conf
    $orig_biiconf = get-content "$env:APPVEYOR_BUILD_FOLDER\biicode.conf"
    if (diff $new_biiconf $orig_biiconf){
        'different, updating biicode parents'
        git checkout "$env:APPVEYOR_REPO_BRANCH"
        git config --global core.autocrlf true
        git config --global credential.helper store
        Add-Content "$env:USERPROFILE\.git-credentials" "https://$(($env:access_token)):x-oauth-basic@github.com:"
        git remote add neworigin "$env:github_repo"
        git config --global user.email "$env:github_email"
        git config --global user.name "$env:github_user"
        git add biicode.conf
        git commit -m "Updated biicode parents [skip ci]"
        git push neworigin "$env:APPVEYOR_REPO_BRANCH"
    }Write-Host "Updated biicode parents" else {
        'equal, no parents update needed'
    }

environment:
  project_name:

```

```
"myproject"
block_user:
  "amalulla"
block_name:
  "cpp-expression-parser"
secured_passwd:
  secure: ENCRYPTED_BIIICODE_PASSWORD_HERE
access_token:
  secure: ENCRYPTED_GITHUB_PASSWORD_HERE
github_user:
  "MariadeAnton"
github_email:
  "maria.deanton@biicode.com"
github_repo:
  "git@github.com:MariadeAnton/cpp-expression-parser.git"
```

What's going on the `appveyor.yml` file?

- `install`: This part installs all tools required to deploy your biicode projects in AppVeyor.
- `before_build`: Moves your project's files into the biicode project and configures it to use Visual Studio 12 via `bii configure`. Check biicode docs and your project's settings in Appveyor to use other build configurations. **Also note** that there's a commented line here you should also write if your project contains folders.
- `test_script`: `cd bin` and execute your project. Ensure about your project's executable, build and execute it locally with `bii build`.
- `deploy_script`: This script publishes your block to biicode, including your version tag only when it's tagged.
- `on_success`: If your `biicode.conf` file is updated commit its changes to github without launching a new build. Else do nothing.
- `environment`: Replace all environment variables here with your values: `project_name`, `tag`, default version tag value... Also your encrypted variables.

You can see this example live:

- in GitHub
- in AppVeyor
- with its automatically published releases

Learn more about AppVeyor visiting their [docs](#).

- Here's a blog-post about [using Appveyor CI](#) and Deploy for Windows.
- MinGW compiler is already installed in Appveyor, just include it in the Path at `install` section: `set PATH=%PATH%;C:\MinGW\bin` and **bii configure "MinGW Makefiles"** at `before_build` section.

Travis CI

takes care of running your tests and deploying your apps. Like we work with VCS, many of the blocks published in our web have their `.travis.yml` files, that lets us pushing to our repository, and automatically build in Linux, execute and publish this project with your biicode user account thanks to this excellent service.

If you're working with it, the `.travis.yml` file format will help to automatically publish to your biicode account with DEV tag:

```
language: cpp
compiler:
  - clang
  - gcc
before_install:
  - if [ "$CXX" == "g++" ]; then sudo add-apt-repository -y ppa:ubuntu-toolchain-r
  - if [ "$CXX" == "clang++" ]; then sudo add-apt-repository -y ppa:h-rayflood/llvm
  - sudo apt-get update -qq

  - if [ "$CXX" == "g++" ]; then sudo apt-get install -qq g++-${GCC_VERSION}; fi
  - if [ "$CXX" == "g++" ]; then sudo apt-get install -qq gcc-${GCC_VERSION}; fi
  - if [ "$CXX" == "g++" ]; then sudo ln -s -v -f $(which g++-${GCC_VERSION}) /usr/
  - if [ "$CXX" == "g++" ]; then sudo ln -s -v -f $(which gcc-${GCC_VERSION}) /usr/
install:
  - wget http://apt.biicode.com/install.sh && chmod +x install.sh && ./install.sh
  - bii --version
  - wget https://s3.amazonaws.com/biibinaries/thirdparty/cmake-3.0.2-Linux-64.tar.g
  - tar -xzf cmake-3.0.2-Linux-64.tar.gz
  - sudo cp -fR cmake-3.0.2-Linux-64/* /usr
  - rm -rf cmake-3.0.2-Linux-64
  - rm cmake-3.0.2-Linux-64.tar.gz
script:
  - cmake --version
  - bii init -L
  - if [ "$CXX" == "clang++" ]; then export CXX="clang++" && bii build; fi
  - if [ "$CXX" == "g++" ]; then export CXX="g++" && bii build; fi
##### CHANGE WITH YOUR CUSTOM CHECKS OR TEST EXECUTION #####
  - ./bin/amalulla_unit_test_main
#####
after_success:
  - bii user $USER -p $BII_PASSWORD
  - bii publish
env:
  global:
    - GCC_VERSION="4.9"
    - USER=amalulla
    - secure: Z1IrvEQFr/poZNXRWAVPXB7eDIUwQWXMgk jMWNTJuXAs jKHTXazd5SaOc88Fd6ajU8ZyJ
```

Here's a way to automatically publish to your biicode account with DEV tag unless your github repository is tagged, in this case, imports the tag and publishes as STABLE to biicode:

```
language: cpp
compiler:
  - gcc
before_install:
  - export TRAVIS_COMMIT_MSG="$(git log --format=%B --no-merges -n 1)"
  - if [[ "$TRAVIS_COMMIT_MSG" = "$COMMIT_IGNORE_BUILD" ]]; then exit 0 ; fi
  - if [ "$CXX" == "g++" ]; then sudo add-apt-repository -y ppa:ubuntu-toolchain-r/
  - sudo apt-get update -qq
  - git config --global user.email "$USER_EMAIL"
  - git config --global user.name "$USER_NAME"
  - git config --global push.default simple
  - git checkout $TRAVIS_BRANCH
install:
  - if [ "$CXX" == "g++" ]; then sudo apt-get install -qq g++-4.8; fi
  - if [ "$CXX" == "g++" ]; then sudo update-alternatives --install /usr/bin/g++ g
  - wget http://www.biicode.com/downloads/latest/ubuntu64
  - mv ubuntu64 bii-ubuntu64.deb
  - sudo dpkg -i bii-ubuntu64.deb && sudo apt-get -f install
  - rm bii-ubuntu64.deb
  - wget https://s3.amazonaws.com/biibinaries/thirdparty/cmake-3.0.2-Linux-64.tar.g
  - tar -xzf cmake-3.0.2-Linux-64.tar.gz
  - sudo cp -fR cmake-3.0.2-Linux-64/* /usr
  - rm -rf cmake-3.0.2-Linux-64
  - rm cmake-3.0.2-Linux-64.tar.gz
  - export TRAVIS_CXX=$CXX
script:
  - cd /tmp
  - bii init -L
  - shopt -s dotglob && mv $TRAVIS_BUILD_DIR/* ./
  - if [ "$CXX" == "clang++" ]; then export CXX="clang++" && bii build; fi
  - if [ "$CXX" == "g++" ]; then export CXX="g++" && bii build; fi
  - cd /tmp/biicode_project
##### CHANGE WITH YOUR CUSTOM CHECKS OR TEST EXECUTION #####
  - ls ./bin/lasote_docker_client_example_main
#####
after_success:
  - bii user $USER -p $BII_PASSWORD
  - if [[ -n $TRAVIS_TAG ]]; then bii publish -r --tag STABLE --versiontag $TRAVIS_
  - if [[ -z $TRAVIS_TAG ]]; then bii publish -r || echo "Ignored publish output..."
  # If there are changes, commit them
  - cd /tmp/biicode_project/blocks/$USER/$BLOCK_NAME
  - git config credential.helper "store --file=.git/credentials"
  - echo "https://${GH_TOKEN}@github.com" > .git/credentials
  - git add -A .
  - git commit -m "$COMMIT_IGNORE_BUILD"
```

```

- git remote -v
- git remote set-url origin https://github.com/$TRAVIS_REPO_SLUG.git
- git push
env:
  global:
    - USER_EMAIL=lasote@gmail.com
    - USER_NAME="Luis Martinez de Bartolome"
    - COMMIT_IGNORE_BUILD="Promoted version.***travis***"
    - BLOCK_NAME=docker_client
    - USER=lasote
    # BII_PASSWORD: Biicode USER's password. > travis encrypt BII_PASSWORD=XXXXXX -
    - secure: ENCRYPTED_BIICODE_PASSWORD_HERE
    # GH_TOKEN: Github token > travis encrypt GH_TOKEN=XXXXXX --add
    - secure: NCRYPTED_GITHUB_PASSWORD_HERE

```

What's going on the `.travis.yml` file?

- `language` and `compiler` are totally clear (this is where you choose the language and compiler that Travis CI will use).
- `before_installing`, establishes our automatic commit must be ignored and configures git to push later, on the `after_success` part.
- `install` provides the tools necessary to test our code with BIICODE.
- `script`, creates, builds and runs the project and checks if the project successes.
- `after_success` part is to publish your project to biicode as STABLE with VERSION_TAG if tagged in github, otherwise it publishes as DEV. Also, if your biicode.conf file is updated, this commits its changes to github without launching a new build.
- `env`: replace all environment values with your own ones. Don't delete the ***travis*** text, as it is the one needed to specify that commit should skip build, avoiding entering an endless build loop.

To learn more about Travis using C++ language, visit its [documentation](#).

Here's how to [automatically build and publish via Travis CI and Github](#). You can also [deploy directly with biicode](#).

Check our [FAQ](#) for questions and answers. You can also [submit suggestions and feedback](#).

1.9.5 Koding

gives you the necessary environment to start developing your apps, run them, collaborate and share with the world. This amazing development tool helps you to work with a great environment everywhere, without installing or executing difficult commands, you've already all the necessary prepared in your Koding account.

If you're signed here and you wish to use biicode in your VMs, then execute:

```
~$ wget http://apt.biicode.com/install.sh && chmod +x install.sh && ./install.sh
~$ bii -h
```

Then, you'd ready to start using biicode and building all the projects you wish.

1.9.6 Doxygen

is the standard tool for generating documentation from annotated C++ sources. You can download it from its .

- It can generate an on-line documentation browser (in HTML) and/or an off-line reference manual (in LaTeX) from a set of documented source files. There is also support for generating output in RTF (MS-Word), PostScript, hyperlinked PDF, compressed HTML, and Unix man pages. The documentation is extracted directly from the sources, which makes it much easier to keep the documentation consistent with the source code.
- You can configure doxygen to extract the code structure from undocumented source files. This is very useful to quickly find your way in large source distributions. Doxygen can also visualize the relations between the various elements by means of include dependency graphs, inheritance diagrams, and collaboration diagrams, which are all generated automatically.

Create a Doxyfile template

```
~$ cd /blocks/[USER]/[BLOCK]
~/blocks/[USER]/[BLOCK]$ mkdir docs
~/blocks/[USER]/[BLOCK]$ cd docs
~/blocks/[USER]/[BLOCK]/docs$ doxygen -g
```

Edit your Doxyfile

The minimal info that you need to change in your Doxyfile is the following tags:

```
PROJECT_NAME           = "My Project"

OUTPUT_DIRECTORY       = .

INPUT                  = ../

FILE_PATTERNS          = *.c \
                        *.cc \
                        *.cxx \
                        *.cpp \
                        *.c++ \
```

```
*.h \
*.hh \
*.hxx \
*.hpp \
*.h++ \
```

Generate the Documentation

```
~/docs$ doxygen Doxyfile

#Open the /docs/html/index.html with your web browser.
```

All the info of the previous examples have been written taking as reference a `docs` folder inside your block.

However, you can create your Doxyfile where you want changing the `INPUT` tag in your Doxyfile.

For example, to generate the docs folder in you project, you need to specify: `INPUT = ../blocks/[USER]/[BLOCK_NAME]/.`

Read more info about doxygen in the [official documentation](#).

If you want to make your own main page, you can create a `DoxygenMainpage.h` in the docs folder with the following sections:

```
/**
@mainpage  TITLE_OF_YOUR_HOME_PAGE

@author YOUR_USER_NAME and all the info about the author

Description of you block

@section TITLE

Section info
*/
```

A good example is :

libfreenect 0.1-beta

[Main Page](#)[Classes](#)[Files](#)

libfreenect

Author

The OpenKinect Community - <http://www.github.com/openkinect>

Cross-platform driver for the Microsoft Kinect Camera

Website: <http://www.openkinect.org>

Introduction

libfreenect is an open source, cross platform development library for the Microsoft Kinect camera. It provides basic functionality to connect to the camera, set configuration values, retrieve (and in some cases decompress) images, and provides functionality for the LED and Motor.

Design Overview

libfreenect provides access to devices via two structs:

- A context, which manages aspects of thread safety when using multiple devices on multiple threads.
- A device, which talks to the hardware and manages transfers and configuration.

Either or both of these structs are passed to the functions in order to interact with the hardware. The USB access is handled by libusb-1.0, which should work in a mostly non-blocking fashion across all platforms (see function documentation for specifics).

Should You Use libfreenect?

The main design goal of libfreenect is to provide a simple, usable reference implementation of the Kinect USB protocol for access via non-Xbox hardware. With this in mind, the library does not contain any algorithms relevant to computer vision usages of the camera.

If you are looking for machine vision algorithms, we recommend the OpenCV library, available at

<http://www.opencv.org>

If you are looking to use the kinect in a larger framework that may involve other depth sensors, we recommend the OpenNI framework, available at

<http://www.openni.org>

Note that libfreenect can be used as a hardware node in OpenNI.

Generated on Tue Feb 24 2015 15:02:18 for libfreenect by [doxygen](#) 1.8.9.1

```
/**
 * @mainpage libfreenect
 * @author The OpenKinect Community - http://www.github.com/openkinect
 *
 * Cross-platform driver for the Microsoft Kinect Camera
 *
 * Website: http://www.openkinect.org
 *
 * @section libfreenectIntro Introduction
 *
 * libfreenect is an open source, cross platform development library for
 * the Microsoft Kinect camera. It provides basic functionality to
 * connect to the camera, set configuration values, retrieve (and in some
 * cases decompress) images, and provides functionality for the LED and
```



```
Motor.
```

```
@section libfreenectDesignOverview Design Overview
```

```
libfreenect provides access to devices via two structs:
```

- A context, which manages aspects of thread safety when using multiple devices on multiple threads.
- A device, which talks to the hardware and manages transfers and configuration

```
Either or both of these structs are passed to the functions in order to interact with the hardware. The USB access is handled by libusb-1.0, which should work in a mostly non-blocking fashion across all platforms (see function documentation for specifics).
```

```
@section libfreenectShouldIUseIt Should You Use libfreenect?
```

```
The main design goal of libfreenect is to provide a simple, usable reference implementation of the Kinect USB protocol for access via non-Xbox hardware. With this in mind, the library does not contain any algorithms relevant to computer vision usages of the camera.
```

```
If you are looking for machine vision algorithms, we recommend the OpenCV library, available at
```

```
http://www.opencv.org
```

```
If you are looking to use the kinect in a larger framework that may involve other depth sensors, we recommend the OpenNI framework, available at
```

```
http://www.openni.org
```

```
Note that libfreenect can be used as a hardware node in OpenNI.
```

```
*/
```

Check our and/or for questions and answers. You can also for suggestions and feedback.

1.10 Reference

Biicode offers you a set of configuration files across your projects and blocks. These files empower you with full control of how biicode treats your code and dependencies.

1.10.1 biicode.conf: configure your biicode projects

biicode.conf is a configuration file to –wait for it– configure your biicode projects.

Place it into your block, next to your source code:

```
|-- my_project/  
|   +-- bii/  
|   +-- bin/  
|   +-- blocks  
|       +-- myuser  
|           |  
|           |   +-- my_block  
|           |       |  
|           |       |   |-- main.cpp  
|           |       |   |-- biicode.conf
```

biicode.conf has 9 different sections to configure your project.

Here's a *biicode.conf* example:

```
# Biicode configuration file  
[requirements]  
    # Blocks and versions this block depends on e.g.  
    # user/depblock1: 3  
    # user2/depblock2(track) @tag  
  
[parent]  
    # The parent version of this block. Must match folder name. E.g.  
    # user/block # No version number means not published yet  
    # You can change it to publish to a different track, and change version, e.g.  
    # user/block(track): 7  
  
[paths]  
    # Local directories to look for headers (within block)  
    # /  
    # include  
  
[dependencies]  
    # Manual adjust file implicit dependencies, add (+), remove (-), or overwrite  
    # hello.h + hello_imp.cpp hello_imp2.cpp  
    # *.h + *.cpp  
  
[mains]  
    # Manual adjust of files that define an executable  
    # !main.cpp # Do not build executable from this file  
    # main2.cpp # Build it (it doesnt have a main() function, but maybe it includes  
  
[tests]  
    # Manual adjust of files that define a CTest test  
    # test/* pattern to evaluate this test/ folder sources like tests
```

```
[hooks]
# These are defined equal to [dependencies], files names matching bii*stage*hook
# will be launched as python scripts at stage = {post_process, clean}
# CMakeLists.txt + bii/my_post_process1_hook.py bii_clean_hook.py

[includes]
# Mapping of include patterns to external blocks
# hello*.h: user3/depblock # includes will be processed as user3/depblock/hello*.h

[data]
# Manually define data files dependencies, that will be copied to bin for execution
# By default they are copied to bin/user/block/... which should be taken into account
# when loading from disk such data
# image.cpp + image.jpg # code should write open("user/block/image.jpg")
```

[requirements]

[requirements] section is fulfilled after executing **bii find** with the blocks and versions your block depends on.

You can manually specify the block to depend on with its corresponding version or override a dependency just writing the version you want and executing **bii build** after that.

```
[requirements]
# required blocks (with version)
erincatto/box2d: 10
```

Take a look at the [docs about dependencies](#) to know more.

[parent]

[parent] section tells you “*who is your parent version*”. Indicates the version of the remote block being edited and looks like this:

```
[parent]
myuser/myblock: 0
```

This section is fulfilled automatically when publishing or opening a block and comes in handy while [publishing](#) take a look at it.

When publishing a new block this section should be blank or referenced as -1 version:

```
[parent]
# Comments like this are ignored
```

or

```
[parent]
  myuser/my_new_block: -1
```

[paths]

Use [paths] section to tell biicode in which folders it has to look for the local files specified in your *#includes*. You only need to specify this when your project has *non-file-relative #include (s)*.

Common use case example

Libraries usually have a folder structure like this one

```
|-- library
|   |-- include
|   |   |-- tool.h
|   |-- test
|   |   |-- main1.cpp (#include "tool.h")
```

In which main1.cpp includes: `#include "tool.h"` that it is truly located into */include* folder. The proper `#include` would be `#include "../include/tool.h"`

If we execute **bii deps** on this example, we'll see `#include "tool.h"` as unresolved. Why is this happening? Biicode can't find the *tool.h* file unless we specify where they can find it.

Let's fix this write into the [paths] section:

```
[paths]
  # Local directories to look for headers (within block)
  /include
```

Root directory example

Let's imagine now that we have a folder with the following structure into it

```
|-- mylib.h
|-- mylib.cpp
|   |-- examples
|   |   |-- main.cpp (#include "mylib.h")
```

If we execute **bii deps** on this example, we'll see *mylib.h* as unresolved. Why is this happening? Biicode, considers the `#include(s)` relative to their location. So if there isn't a root folder they can refer to, when looking for *mylib.h* they will search it in the *examples* folder and they won't be able to find it.

What should we write on the [paths] section?

```
[paths]
    # Local directories to look for headers (within block)
    /
```

Write `/` in `[paths]` section and biicode will know that it has to include the root directory on its search.

[dependencies]

Biicode knows how the source code files connect to each other. It parses the source code files and deduces some things. But sometimes, this mechanism can detect non existent dependencies or can fail detecting existent dependencies.

Use `[dependencies]` section to manually define rules to adjust file implicit dependencies.

`[dependencies]` rules match the following pattern:

```
#dependent_file_name [operator] NULL|[[!]dependency_file ]
```

The Operator establishes the meaning of each rule:

- `-` operator to **delete** all specified dependencies from their dependent file.
- `+` operator to **add** all specified dependencies to their dependent file.
- `=` operator to **overwrite** all specified dependencies with existing dependencies.

You can declare that a file has no dependencies using the `NULL` keyword.

Mark a dependency with a `!` symbol to declare a dependency, but **excude it from the building process**. This is sometimes used to define **license files** that must be downloaded along with your code, but shouldn't be included in the compilation process.

The `dependent_file_name` may be defined using **Unix filename pattern matching**.

Pattern	Meaning
<code>*</code>	Matches everything
<code>?</code>	Matches a single character
<code>[seq]</code>	Matches any character in seq
<code>[!seq]</code>	Matches any character not in seq

Examples

Let's see a few examples:

- `matrix32.h` is dependency of the `main.cpp` file.

```
[dependencies]
  main.cpp + matrix32.h
```

- Delete *matrix16.h* dependency to *main.cpp*.

```
[dependencies]
  main.cpp - matrix16.h
```

- *test.cpp* depends on both *example.h* and *LICENSE*. And *LICENSE* will be excluded from the compilation process.

```
[dependencies]
  test.cpp + example.h !LICENSE
```

- All files with *.cpp* extension depend on the *README* file, but this dependency won't be compiled.

```
[dependencies]
  *.cpp + !README
```

- *example.h* = *NULL* tells biicode that *example.h* has no dependencies (even if it truly has).

```
[dependencies]
  example.h = NULL
```

- Both *solver.h* and *type.h* are the only dependencies of *calculator.cpp*, overwriting any existing implicit dependencies.

```
[dependencies]
  calculator.cpp = solver.h type.h
```

[mains]

Use [mains] section to define entry points in your code.

Biicode automatically detects entry points to your programs by examining which files contain a **main function** definition. But when that's not enough you can **explicitly tell biicode where are your entry points**.

[mains] has the following structure:

```
[(!)file ]
```

An example:

- Write the **name of the file** you want to be the entry point.
- Exclude an entry point writing an **exclamation mark, !** before the name of the file.

```
[mains]
    funct.cpp
    !no_main.cpp
```

[tests]

Tests section is useful to define specific tests for your code. Adjust files manually that define a CTest test.

Indicate the patten to your test files:

```
[tests]
    test/*
    test/simple_test.cpp
```

Those test are excluded from the normal building and are built and executed only when doing *bii test*.

[hooks]

Use [hooks] section to link to certain python scripts that will be executed, for example, before building your project. They can be used to download and install a package needed.

This scripts have ".py" extension and name matches:

- *bii*post_process*hook.py*: For scripts that will be launched before project building (**bii build** or **bii configure**)
- *bii*clean*hook.py*: For scripts that will be launched before a **bii clean** command.

These are defined like *[dependencies]*.

In the following example we define that *CMakeLists.txt* depends on two hooks:

```
[hooks]
    CMakeLists.txt + bii/my_post_process1_hook.py bii_clean_hook.pyw
```

Use *bii* variable inside hook scripts to:

- Print text:

```
bii.out.debug("error_msg")
bii.out.info("error_msg")
bii.out.warn("error_msg")
bii.out.error("error_msg")
```

- Download files:

```
bii.download(url, tmp_path)
```

- Read your project settings:

```
bii.settings.cpp.cross_build
```

Check an example in this block: .

[includes]

Enables mapping include patterns to external blocks.

- For example you can tell biicode: Whenever you read `uv.h` in my code, it means `lasote/libuv/include/uv.h`:

```
[requirements]
  lasote/libuv(v1.0): 0

[includes]
  uv.h: lasote/libuv/include
```

In the previous example, the `[requirements]` section has a line specifying a dependency to `lasote/libuv(v1.0): 0` version, so, `lasote/libuv #includes` will be matched against these block.

- You can also specify complex patterns. To process `hello*.h #includes` as `user3/depblock/hello*.h`

```
[includes]
  hello*.h: user3/depblock
```

This is pretty useful when using already existing libraries and you don't want to change all the includes.

[data]

Use `[data]` to specify a link with any file (`.h`, `.cpp`, ...) with any data (`.txt`, `.jpg`, ...) in your block.

Once `[data]` section is specified and the code is built (**bii build**), the data files will be saved, by default, in your `project/bin/user/block` folder.

Example:

You have in your main code this line:

main.cpp


```
CImg<unsigned char> image("phil/cimg_example/lena.jpg")
```

Then, add to your configuration file:

```
[data]
    main.cpp + lena.jpg
```

This will copy *lena.jpg* to *project/bin/user/block/* when *main.cpp* is built.

Any doubts? Do not hesitate to [contact us](#) visit our [forum](#) and feel free to ask any questions.

1.10.2 Commands

These are the **commands available in biicode**. You can:

- **manage** your project and blocks locally, on your computer,
- determine their internal and external **dependencies**,
- **retrieve** any missing code dependencies from the biicode servers,
- **publish** your code, and
- **reuse** from other user's code.

bii build: build your project

This command invokes the C++ compiler to build your project.

```
$ bii build
Building: cmake --build .
...
[100%] Built target myuser_myblock_main
```

You can build your projects with the parameters, depending on your OS, that CMake offers us.

```
$ cmake --build
Usage: cmake --build <dir> [options] [-- [native-options]]
Options:
  <dir>                = Project binary directory to be built.
  --target <tgt>       = Build <tgt> instead of default targets.
  --config <cfg>      = For multi-configuration tools, choose <cfg>.
  --clean-first       = Build target 'clean' first, then build.
```

```
(To clean only, use --target 'clean'.)
--use-stderr    = Don't merge stdout/stderr output and pass the
                  original stdout/stderr handles to the native
                  tool so it can use the capabilities of the
                  calling terminal (e.g. colored output).
--              = Pass remaining options to the native tool.
```

Now, using biicode, for example:

```
$ bii build --use-stderr
```

Building in a Linux or Mac OS X system or MinGW (with make). You can probably **speed up your builds using multiple jobs** (threads, cores):

Execute **bii build -jN** to run Makefile with this option and use the N cores of your machine:

```
$ bii build -j4 == cmake --build . -- -j4
```

This also works with Visual Studio compiler:

```
$ bii build -j4 == cmake --build . -- /m:4
```

You can also use the **-jN make** option where N is the number of concurrent jobs desired (please note the - - before -jN):

```
$ bii build -- -j4
```

Building with Visual Studio compiler (even in the command line), the equivalent option is /m:N, so you can:

```
$ bii build -- /m:4
```

biicode buzz: init, find and build

bii buzz is an all-in-one purpose command that combines different actions: initialize project, find dependencies and build the block.

bii buzz automates the following commands:

- **bii init -L**: Initializes biicode project with simple layout.
- **bii find**: Finds external dependencies of your block.
- **bii build**: Builds your project.

It is useful for a quick getting started of a new project or after cloning a git repository.

To use it, go to your normal project directory and **bii buzz**:

```
$ cd my_project
$ bii buzz
```

Write your *biicode.conf* file before doing **bii buzz** to let biicode find your right dependencies.

bii clean: delete meta-information

bii clean command cleans most of biicode internal project meta-information, keeping the minimum required to reconstruct everything in a subsequent command.

It's an all purpose command, especially useful to restore your default project settings or when upgrading to a major release.

bii clean also restores your project's settings to no IDE and MinGW (Windows) or Unix Makefiles with no IDE (MacOS and Linux).

```
$ bii clean
```

You can delete user cache too:

```
$ bii clean --cache
```

Deleting cache is useful to ensure a clean reconfiguration of biicode. It is used when experiencing troubles with temporal and build files in your projects.

Got any doubts? [Ask in our forum](#)

bii close: finish editing published blocks

You can use **bii close** command to close a block under edition in *blocks* directory. Usually, to close a block you've **bii open** to edit.

```
$ bii close USER_NAME/BLOCK_NAME
```

Got any doubts? [Ask in our forum](#)

bii configure: configure your project

This command analyzes your project and process dependencies, then creates or updates the required project CMake files. You can use it to specify which CMake generator you want to use.

This command accepts all CMake directives, you could say this command is basically a **cmake** invocation within the project *build* folder.

- Use **bii configure** to *configure a project for your IDE*.
- You can also *delve into specifics about building with biicode*.

bii clean command restores your project's minimum settings, here's more about *bii clean command*.

Passing variables to cmake

You can define a variable in your *CMakeLists.txt* and pass the value from command line. To do this, just pass the value as a parameter of **bii configure**.

Imagine you have defined a variable name FOO in your *CMakeList.txt* and you want to define it with TRUE or FALSE.

```
IF (FOO)
    message (STATUS "FOO Enabled!!")
ELSE ()
    message (STATUS "FOO Disabled!!")
ENDIF ()
```

Define the value of FOO with the flag `-DFOO=TRUE` or `-DFOO=FALSE`:

```
$ bii configure -DFOO=TRUE
Running: cmake -G "MinGW Makefiles" -Wno-dev -DFOO=TRUE ../cmake
FOO Enabled!!
```

Define a tool-chain

bii configure --toolchain command define the tool-chain to use, here's more about the *custom tool-chains*.

bii deps: show block dependencies

bii deps checks your project's dependencies. It gives a general idea of which are your code's dependencies. You can use several arguments and filters.

```
$ bii deps
```

bii deps

Show the files your blocks depend on (with their corresponding blocks and versions), also shows system and unresolved dependencies:

```
$ bii deps
phil/math depends on:
  phil/math_lib: 4
    algorithm.h
    lib.h
  system:
    iostream
  unresolved:
    fake_lib.h
phil/hello_world depends on:
  system:
    stdio
```

Include the name of an specific block to see only that block's dependencies:

```
$ bii deps phil/hello_world
phil/hello_world depends on:
  system:
    stdio
```

bii deps --detail

Use **bii deps --detail** but to show the origin files where dependencies are used.

```
$ bii deps --detail
phil/math_example depends on:
  phil/math_lib: 4
    algorithm.h
      phil/math/main.cpp
    lib.h
      phil/math/main.cpp
  system:
    iostream
      phil/math/main.cpp
  unresolved:
    fake_lib.h
      phil/math/main.cpp
phil/hello_world depends on:
  system:
    stdio
      phil/hello_world/hello.cpp
```

bii deps --detail [FILTER]

Enter a filter to see something specific. For example, you can see just a file or a dependency with its origins:

```
$ bii deps --detail fake_lib.h
phil/math_example depends on:
  phil/math_lib: 4
  system:
  unresolved:
    fake_lib.h
    phil/math/main.cpp
phil/hello_world depends on:
  system:
```

bii deps --files

Use **bii deps --files** to know all block files (together with their types) and their dependencies.

```
$ bii deps --files
phil/math_example
  CMakeLists.txt [TEXT]
  biicode.conf [TEXT]
  phil/math/main.cpp [CPP] [M]
    phil/math_lib/algorithm.h (E)
    phil/math_lib/adafruit_sensor.h (E)
    iostream (S)
    fake_lib.h (U)
phil/hello_world
  CMakeLists.txt [TEXT]
  phil/hello_world/hello.cpp [CPP] [M]
    stdio (S)
```

Type of dependency information showed in files:

- E: explicit file
- I: implicit file
- D: data file
- S: system file
- U: unresolved file

Got any doubts? [Ask in our forum](#)

bii diff: compare block versions

Compare files and show their differences with **bii diff** command. You can compare your current local project against a previously published version or compare between published versions.

```
$ bii diff [--short] [block_name] [v1] [v2]
```

Usage

fenix user has an armadillo block in a local project and 4 different published versions.

Let's see the different possibilities:

- Compare the local block against the latest published version:

```
$ bii diff
```

- Compare the local block against a specific version:

```
$ bii diff fenix/armadillo 2
```

- Compare two specific published versions (2 and 3) of your local block:

```
$ bii diff fenix/armadillo 2 3
```

- Show just a short diff in any of the previous examples:

```
$ bii diff --short
```

Got any doubts? [Ask in our forum](#)

bii find: find your external dependencies

Retrieve any code dependencies from biicode's servers.

Biicode analyzes your code, and finds missing dependencies that cannot be resolved searching in your project contents. After that, biicode tries to find the code you need in our serves, and retrieves the best matching version according with your *policies.bii*.

```
$ bii find
```

Update your dependencies

Update your dependencies and find new unresolved ones in one step:

```
$ bii find --update
```

Biicode uses *policies.bii* to resolve your dependencies. You can find *policies.bii* in your project's *bii* folder.

Note: *policies.bii* defines the way **bii find** command retrieves dependencies. For example, if you only want STABLE versions and there's a newer DEV version, this version will only be retrieved if you say so in your *policies.bii*.

Find compatible downgrades:

```
$ bii find --downgrade
```

Got any doubts? [Ask in our forum](#)

- Visit [\[requirements\]](#) section.

bii init: creates a new project

Use **bii init** command to **create new project**.

One step- Create a folder *project_name* and init your project inside:

```
$ bii init project_name
```

Also use **bii init** inside a folder to init a biicode project:

```
~$ mkdir math_project
~$ cd math_project
~/math_project$ bii init
```

Got any doubts? [Ask in our forum](#)

bii new: creates new blocks

bii new command **creates a new biicode block inside your project**.

Use it inside a biicode project folder like this:

```
$ bii new USER_NAME/BLOCK_NAME
```

Take a look at [the Getting Started guide](#) to know more.

bii new *user_name/block_name* command creates new folders in your blocks folder:


```
+-- myproject
|   +-- biicode
|   +-- blocks
|       +-- user_name
|       |   +-- block_name
|       +-- deps
```

It's useful to create a block with a default “Hello World” main file into it:

```
$ biicode new USER_NAME/BLOCK_NAME --hello LANGUAGE
```

For example:

```
$ biicode new fenix/first_block --hello cpp

...

Successfully fenix/first_block folder created in your blocks directory!
Successfully main.cpp file created in PROJECT_DIR/blocks/fenix/first_block
```

Resulting layout:

```
+-- myproject
|   +-- biicode
|   +-- blocks
|       +-- fenix
|       |   +-- first_block
|       |   |   +-- main.cpp
|       +-- deps
```

Got any doubts? [Ask in our forum](#)

biicode open: edit published blocks

Use **biicode open** command to use and edit any published block locally.

You can use this command to edit **any block** you've seen on the web and you want to edit. Just create a new project and once in it, open the block:

```
$ biicode init myproject
$ cd myproject
~/myproject$ biicode open USER_NAME/BLOCK_NAME
```

You can find the block you just opened in the `blocks` folder within your project.

Take a look at [Workflows](#) section to learn how to modify its source files and publish a new version.

If you are updating or creating a block from another service, check the [integration guide](#) to know how to proceed.

bii publish: publish your blocks

bii publish publishes your code to biicode.

```
$ bii publish
```

Once you've successfully published your code to our servers, command line prompt will show a info message about your block name `user_name/block_name` and the version ID number:

```
INFO: Publishing block: myuser/myblock
INFO: *****
INFO: ***** Publishing public *****
INFO: *****
INFO: Successfully published myuser/myblock: 1
```

Available publishing options:

Tag's default value is DEV, but all **release life-cycle tags** are available:

- DEV: Code is under development.
- ALPHA: It's usually code ready to shared with close friends or colleagues.
- BETA: Code ready for BETA testers!
- STABLE: Tested and ready to reuse.

These tags define the development state of your code.

policies.bii file holds your policies about what kind of code tags you allow for each the blocks you use.

Publish your blocks with different tags:

```
$ bii publish --tag STABLE
```

Here's a full guide about *publishing* to biicode.

Publish from a git commit

When working with a block cloned from a git repository, use **bii publish -r** or **bii publish --remote** to publish git info along with your block:

```
$ bii publish -r
```

This way your reference to the repo and commit will be saved and shown in your block header.

You can also use tags:

```
$ bii publish -r --tag STABLE
```

Check [Git integration](#) to know more about working with git repositories.

Publish one of your project's blocks

If there's more than one block within your project, specify which one you want to publish:

```
$ bii publish USER_NAME/BLOCK_NAME
```

An Example fenix user is editing two blocks, `fenix/vector` and `fenix/matrix`. He wants to publish **just fenix/vector** using DEV tag:

```
$ bii publish fenix/vector
```

After a while, he's sure about publishing `fenix/vector` as STABLE version:

```
$ bii publish fenix/vector --tag STABLE
```

Got any doubts? [Ask in our forum](#).

Visit the section: [bii update command](#)

bii setup: install necessary tools

This command helps you setting up all the tools necessary to start using biicode.

C++ tools

Install C++ third party tools:

```
$ bii setup:cpp
```

Arduino tools

Install C++ third party tools and Arduino SW:

```
$ bii setup:arduino
```

Raspberry PI tools

Install cross compiler tools for Raspberry Pi (Linux only):

```
$ bii setup:rpi
```

Got any doubts? [Ask in our forum](#)

bii test: test your code

bii test uses the patterns defined in the `[tests]` section of the *biicode.conf* to execute tests your block's tests.

```
$ bii test
```

bii test command wraps **bii build** and **ctest**. Now your tests are fully configurable with CTest CMake options.

```
$ bii test -E _main -R test_
```

Check to know more.

bii test command truly runs:

```
$ bii test -VV (ctest extra verbose option)
```

But when using Visual Studio, **bii test** is truly executing:

```
$ bii test -VV -C Debug
```

Note that **bii test** command supports CMake and CTest options, like:

```
$ bii test -C Release
```

Also enables **-jx** option to launch build and/or tests with this flag (they've got different effects):

```
$ bii test -j2 == $ bii build -j2 + $ bii test -j2
```

If you got any questions left, you can ask them at our [forum](#).

bii update: update a block

Use **bii update** command to get the latest version of a local block you're working on.

Update an outdated block (get into your project's or block's folder):

```
$ cd my_project
$ bii update
```

Got many blocks in a project? Get into your projects folder and update any of the blocks in it:

```
~$ cd myproject
~/myproject$ bii update my_user/my_block
```

Got any doubts? [Ask in our forum](#)

bii user: specify your username

Show or change your current biicode user.

A quick tip: Run `bii user your_username` before getting started, this way biicode knows you're the one winning a badge.

You're now ready to *get started*.

Execute `bii user` to show the current user.

Note: It can be None (anonymous).

```
$ bii user
INFO: Current user: None (anonymous)
```

Make sure you've got an *user name* `<https://www.biicode.com/>` and use it:

```
$ bii user phil
INFO: Change user from None to phil
```

You can enter it together with your password:

```
$ bii user phil -p myp@ssw0rd
```

Got any doubts? [Ask in our forum](#)

1.10.3 Configuration Files

Biicode offers you a set of configuration files across your projects and blocks. These files empower you with full control of how biicode treats your code and dependencies.

layout.bii: define your project layout

Specify your project layout config with **layout.bii** file.

With *layout.bii* you can place the auxiliary folders (*cmake/ build/ deps/ and lib/*) wherever you want, just specify the relative routes to the folders you want to use instead.

bii init -l simple creates a default *layout.bii* content that places all auxiliary folders in your project's *bii/* folder:

```
# Minimal layout, with all auxiliary folders inside "bii" and
# The binary "bin" folder as is, and enabled code edition in the project root
cmake: bii/cmake
lib: bii/lib
build: bii/build
deps: bii/deps
# Setting this to True enables directly editing in the project root
# instead of blocks/youruser/yourblock
# the block will be named as your project folder
auto-root-block: True
# Parent blockname (if exists)
```

But you can customize it anyway you want.

Check our and/or for questions and answers. You can also for suggestions and feedback.

policies.bii: defining the policies for the code you want to reuse

Policies are **rules bii find applies** when finding or updating external dependencies. Configure them in *your_project/bii/policies.bii*

policies.bii default value accepts all your *DEV* versions and other user's *STABLE* versions. It has this format as self-documented::

```
# This file configures your finds of dependencies.
#
# It is an ordered list of rules, which will be evaluated in order, of the form:
#     block_pattern: TAG
#
# For each possible block that could resolve your dependencies,
# only versions with tag >= TAG will be accepted

your_username/* : DEV
* : STABLE
```

Notes:

- *block_pattern* fits the *owner/block_name* pattern.
- TAG can be **STABLE**, **BETA**, **DEV** or **ALPHA**.

Changing your policies

Accept all your *DEV* versions as well as all *DEV* versions from *owner1* and *owner2* write:

```
# policies.bii file

    your_username/* : DEV
    owner1/* : DEV
    owner2/* : DEV
    * : STABLE
```

Unleash the tester in you, accept all latest *DEV* versions:

```
# policies.bii file

    * : DEV
```

Don't forget to update your dependencies according to your new *policies.bii*:

```
$ bii find --update
```

Got any doubts? [Ask in our forum.](#)

ignore.bii: filtering your files

Specify in **ignore.bii** which files you want biicode to ignore when processing and publishing your blocks. It's a similar approach to *.gitignore* files in a [git repository](#), and allows you to define which local files will be excluded from publication to our servers.

Here's the general *ignore.bii* file biicode generates by default, you can find it in your *.biicode* folder which is in your user folder.

```
# You can edit this file to add accepted and ignored file extensions
# The format is similar to gitignore files.
# Rules are evaluated in order.
#
# Format is as follows:
#   <pattern>
# pattern: conforms Unix Filename Pattern Matching, if preceded by ! it is negated,
#           instead of ignoring (previously ignored by a precedent rule)
#
# Compiled source #
*.com
*.class
*.dll
*.exe
*.o
```

```
*.so
*.obj
*.pyc
*.dir

# Editor backups
*~

# Hidden files
.*
*/.*

# OS generated files
Thumbs.db
ehthumbs.db
.DS_STORE
```

Place additional *ignore.bii* files insider any block folder or subfolder to ignore specific files and/or override the general ignore rules.

Any doubts? Do not hesitate to [contact us](#) visit our [forum](#) and feel free to ask any questions.

settings.bii: defining your tools and preferences

The **settings.bii** file defines your preferred tools and specific configurations for one given project. This file is created when you create a new project, inside the *bii/* folder of the project.

A project's *settings.bii* file stores preferences for any given project (language, compiler and other tools and configurations for that particular scenario). It is important to keep the file yaml format for correct interpretation.

You don't usually need to manually edit this file. **Each language or platform provides its own command for settings customization (except C/C++ language).**

- See the section *How to configure your IDE (C/C++)*
- *bii arduino:settings* for Arduino projects
- *bii rpi:settings* for Raspberry Pi projects

Got any doubts? [Ask in our forum](#).

types.bii: configuring non-standard file extensions

This is a special configuration file that **allows you to specify how your files are interpreted and processed by biicode**. In some cases, we find projects containing—for some reason— non-standard extensions for certain file types.

Consider, for instance, a block containing c++ code in files with a non-standard file extension, such as `.mycpp`, or without extension. In this case, you must explicitly indicate biicode to process those files as c++ files. This is as simple as placing a **types.bii** file in the root of your block folder:

```
# my special c++ files extension:
*.mycpp      cpp
NOEXT        cpp
```

As you can see, this file contains rules of the form:

```
<pattern>      <desired_extension>
```

Where `<pattern>` is a wild-card for the file types you desire to apply the rule, and `<desired_extension>` is the corresponding standard file extension for that particular type.

In the previous example, we are telling biicode that all files with `.mycpp` extension contained in the source folder (and that includes the current folder, and all its descendants) must be interpreted as `cpp` files.

Note: The `<desired_extension>` expression contains only the desired extension string, *without the dot symbol*.

These rules **are always applied hierarchically**. So you can override your type settings using additional *types.bii* place in descendant folders (this is, however, a more exceptional scenario, but illustrates the powerful capabilities biicode offers to the programmer).

The default file extensions understood by biicode are:

- CPP: `.h`, `.hh`, `.hpp`, `.c`, `.cc`, `.cpp`, `.cxx`, `.inl`, `.ino`, `.ipp`
- TEXT: `.txt`, `.bii`, `.md`
- XML: `.xml`
- HTML: `.html`, `.htm`
- SOUND: `.wav`
- IMAGE: `.jpeg`, `.jpg`, `.gif`, `.png`, `.bmp`
- JSON: `.json`
- PYTHON: `.py`
- JS: `.js`, `.node`
- JAVA: `.java`

- FORTRAN: `.f90`, `.for`, `.f`

Got any doubts? [Ask in our forum](#).

Check our and/or for questions and answers. You can also for suggestions and feedback.

1.11 Release notes

3.2 (19-May-2015)

- Fixed problem with cmake Eclipse generator overwriting every time the project config
- Dependencies are managed as SYSTEM for `include_directories`, to avoid compilation warnings, both for absolute and relative includes
- Opt-out for handling `include_directories` as system dependencies
- Fixed bug in migration of old `.bii` files
- Fixed bug in publish not upgrading properly user profiles

3.1.1 (29-Apr-2015)

- Optimized complete internal workflow, avoiding slow and unnecessary serializations to project DB
- Files not modified in disk, removed problems with CRLF conversions
- Fixed behavior of “bii test” in XCode: <https://github.com/biicode/client/issues/12>

3.0 (15-Apr-2015)

- Open source release of client and common repositories, dev infrastructure re-made
- Release of premium accounts for private blocks
- Enabled OAuth with Github and Google
- Fixed bug in C++ parser for *using* statements
- Fixed bug in `[tests]` pattern that incorrectly affected dependencies
- New Terms of Service, clarifying source code licenses and accounting for premium accounts and open source contributions
- Fixed bug in user folder `”.biicode”` path, some config files were stored out of it

2.8 (7-Apr-2015)

- Better computation of binary targets, if an executable does not depend on its own block library, it will not be linked
- Removed automatic creation of `cmake_dummy.cpp` to avoid problems with header only libraries

- Removed automatic handling of system deps (math, pthread, winsock), that created problems in new toolchains. Now users directly specify in CMakeLists their libs.
- `ADD_BIICODE_TARGETS()` has been superseded by `ADD_BII_TARGETS()`, which will admit a biicode version number, this is done to achieve backwards compatibility while introducing new build behavior.
- Fixed bug of include paths missing in [tests] targets
- Fixed extra verbose message of warning about full path #includes
- Optimized performance of ignore.bii for large number of patterns, that was incredibly slow
- Now published blocks show publisher's user name
- Security fixes

2.7 (23-Mar-2015)

- New [tests] section improved, working fine in Visual Studio
- “bii test” now allows parameters (-jN for parallel builds/runs, and CTest other parameters -R -E -C Debug, etc)
- Simpler “policies.bii” configuration, old ones will be automatically migrated
- Now “bii find” works also with [includes], so it is not necessary to fill both the [includes] and the [requirements]
- Search in web for file names with the “file:” label inside the search box
- Issue warning when using absolute #include paths within the current block, as this wouldn't work in simple layouts
- Fixed bug with “bii publish -r” in simple layout
- Renamed custom target “check” associated with “bii test” to “biitest”, as the check target might collide with user ones

2.6.1 (11-Mar-2015)

- New [tests] section in biicode.conf. Automatically define your tests, and build and run them with “bii test”
- New simplified alias “bii build”, “bii configure”, “bii test” for “bii cpp:xxx” commands
- New “bii buzz” command. It “bii init -L” + “bii find” + “bii build” for quick getting started for newbies, or after a git clone
- New project layout fixes, the default block at root is named after its biicode.conf [parents] if existing
- Added layout variables to CMakeLists.txt, so cmake can account for variable layouts. Also added CMake variable to indicate a block is a dependency or not.
- Added layout variables to hooks (bii.paths.deps, bii.paths.bin, etc)

- New command options for arduino:settings (`-board=uno`, `-port=auto`, etc), so interactive mode can be avoided. Same for rpi:settings
- Automatic handling of PATH, `sh.exe` is removed from mingw builds, as `cmake-mingw` builds doesn't work well with it in the path
- Updated supported arduino SDKs to 1.6
- New `-hello=c` (`<stdio>`, `printf`) for pure C projects
- Fixed broken Eclipse project generation
- Fixed changing [requirements] when several blocks are simultaneously opened
- Fixed incorrect warning about bad filenames of files which were in `ignore.bii`

2.5.2 (3-Mar-2015)

- New custom project layouts available, with `layouts.bii`.
- Enabled edition in the project root folder
- Fixed problems with paths with spaces
- Changed arduino toolchains, now use `cpp:configure -t=arduino`, and `cpp:build`
- Fixes to windows installer, shortcuts load biicode path even if not in system path
- Free accounts have collaborators now too
- Fix in noderunner script

2.4.1 (18-Feb-2015)

- Improved windows installer, more user options for path, icons, uninstaller, and fixed `readme.txt` linefeeds
- Improved cmake installation, with option `-interactive "$bii setup:cpp -i"`, now not adding it to path, so can coexist with 2.8 installs
- Recovered colored output of cmake builds
- Enabled blocks linked with symlinks in linux, so 2 different projects can link to the same block
- Fixed problem of cached badges
- Fixed biicode.cmake issues for CLion, now CLion can be used with biicode

2.3 (9-Feb-2015)

- Better toolchain system for cross-compiling
- Added `-remote` option to "bii publish" to publish github, bitbucket etc remotes info (remote, branch, tag and commit).
- Web: Block view with visible VCS remote information.

2.2 (27-Jan-2015)

- Fixed errors with local cache and DEV version updates.
- Web: New biicode badges.

2.1.1 (12-Jan-2015)

- Fixed problem with biicode.conf [includes] of the form block/file.h: username
- Arduino automatic reset for certain boards, those using the “catarina” uploader (leonardo, yun)
- bii open better handling of versions and tracks
- upgraded cmake minimum required version to 3.0
- fixed security issue of token not discarded after changing password
- improved error messages when biicode.conf incorrect
- Website: Better block page header, reuses in Block requirements.

2.0.1 (17-Dic-2014)

- Premium accounts to store code privately and share alike.
- New and simpler configuration in a single “biicode.conf” file, instead of several tiny files, and with more flexibility.
- System for reusing CMake scripts. Now, CMakeLists.txt can automatically depend on existing CMake scripts, they will be automatically retrieved, versioned, shared, exactly as other source code files
- Improved and simpler, CMake and build system, with Interface targets. Possibility of setting options to and configuring blocks you depend on.
- Possible to specify #include mappings to blocks, so no modification at all is necessary in code.
- Support for Arduino 1.5 for Yun board
- Python hooks for custom tasks. Examples of hooks that automatically retrieve and configure system-wide some popular and large frameworks as OpenCV or Boost.
- Blocks can use different variants, called “tracks” to choose and switch between lib versions or flavours. For example, can be used to maintain several development branches simultaneously as lasote/libuv(v1.0) and lasote/libuv(v0.10), and changing from one to the other does not require any change to code.
- Web: See which blocks depend on your block in the requirements tab
- Binary support to WXwidgets
- New doc style and contents

1.11 (19-Nov-2014)

- Block tracks.
- Client hooks.
- Improved client output messages.
- Faster processing of large projects.
- Improved setups (arduino).
- Bug of with Publish STABLE after DEV cache invalidating solved.
- Bug of target_compile_options quotes solved.
- Web: Block permissions and admin management for premium accounts (for friends).
- Web: [Improved Search Engine: by tag, user name, block name...](#)
- Web: Block tagging.
- Web: Fixed bug with some passwords patterns.

1.10.1 (3-Nov-2014)

- Create blocks on web interface
- Lowercase on usernames and blocknames restriction removed
- Removed branches functionality
- Fixed RAW code for images
- Fixed ZIP download for images

1.9 (20-Oct-2014)

- Support any Arduino compatible board
- Improved information messages
- CMake now defines BIICODE to support biicode and non biicode includes
- Web: Show achievements in user profile
- Web: Raw file visualization

1.8.5 (10-Oct-2014)

- Fedora and Arch package
- CMake updated to 3.0.2 and improved setup:cpp command
- Fixed errors with ignore.bii behaviour
- Custom tool-chain for CMake
- Web: Download blocks as ZIP

- Web: Delete blocks

1.7.3 (24-Sept-2014)

- Solved bug in merge.
- dependencies.bii now accepts file patterns.
- New apt server for debian based distributions including wheezy.
- Better and cleaner deb packaging for debian based distributions.

1.6 (16-Sept-2014)

- Solved bug in cpp parser
- Added a new filter with patterns in mains.bii file

1.5.4 (09-Sept-2014)

- Deleted “bii status” command
- Largely improved “bii diff” command
- Solved some bugs with CMake
- biicode is now case sensitive

1.4.1 (04-Sept-2014)

- Minimum CMake version updated to 2.8.12 (it was 2.8, but 2.8.12 was indeed required)
- Added -p (–password) option to “bii user” so biicode can be scripted (e.g. travis-ci) without interactivity
- Largely improved “bii deps” command
- New structure and data of “xxx_vars.cmake” files, allowing choosing to build or not in block library (both static and shared), with better embedded comment string docs
- CMake printing of built targets
- CMake path inserted for upgrades to cmake 3.0 in bii setup:cpp
- Files in web user profile ordered alphabetically
- Bug of web navigation back-forth solved

1.3.3 (21-08-2014)

- Bugfix: colored output

1.3.2 (13-08-2014)

- Bugfix: login not required anymore when not really needed
- Web performance improvements

1.2.1 (07-08-2014)

- Bugfix for recursive system dependencies compilation

1.2 (06-08-2014)

- Bugfix Open command computed deps incorrectly
- Bugfix Incorrect find policies for DEV versions
- Bugfix Solved transitivity problems in cmake for complex deps
- Rpi cmake pre-built custom package
- UX Improvements
- Web fixes:
 - Files tree alphabetically ordered
 - Show pictures in blocks
 - Fixed log in and password recovery

1.1.1 (25-07-2014)

- Bugfixes
- UX Improvements
- Web Bugfixes, dependencies and deps graph

1.0.4 (25-07-2014)

- Bugfixes
- UX Improvements

1.0.1 (15-07-2014)

- No sign up required
- No more workspaces, any folder can hold a project
- Plain configuration files
- Simplified project settings
- Relative includes allowed
- Configuration options with CMake (extensible)
- Bugfixes
- Improved web-page

0.17.3 (28-06-2014)

- Bugfixes in arduino build (bad transitive dependencies)
- Bugfixes in Raspberry Pi commands

- Reduced Arduino.cmake and CMakelists.txt for arduino projects
- Bugfixes in deps command

0.16 (24-04-2014)

- Improved project graph visualization
- Bugfixes in publish command

0.15.3 (11-04-14)

- Now work, find and upload can be done from arduino monitor GUI
- Output information improvements
- Auto remove empty dep folders
- Arduino selection improvements, now you can select among different connected devices
- Improved readme.md layout
- Relative imports within the same block allowed

0.14.1 (03-04-14)

- Fixed Ubuntu 64b installation issues
- Arduino serial monitor (GUI) improvements
- Bugfixes
- Node integration improvement
- Improved block deletion support

0.13.1 (28-03-14)

- Bugfixes in arduino build
- Now you can upload to the arduino from the serial monitor
- Better Node.js support
- `bii clean` command now deletes the build folder
- Removed main and class creation wizards
- Removed `bii cpp:exe` command
- Projects and Blocks can now be deleted from your user profile web page

0.12 (21-03-14)

- Allow to define MS Visual version from `cpp:settings`
- Arduino bugfixes
- Git support improvements

0.11.1 (14-03-14)

- New installation wizards for C++, Arduino, and Raspberry Pi
- Arduino port automatic detection. The `bii arduino:usb` command is deprecated
- Removed `environment.bii` config file
- Add direct access icon for Windows biicode client
- Fix find bug
- Fix local cache bug
- Nicer `bii arduino:monitor` in MacOS
- Removed `--default` option in `bii init` and `bii new`. New parameters for `bii new` command.
- Entry points automatic detection in files with `setup` and `loop` functions
- Adding `import` as valid preprocessor directive.

0.10 (21-02-14)

- Removed the workspace `default_settings.bii` file. Now, new projects' settings are obtained from the workspace `environment.bii` file.
- Node.js support
- Debian wheezy support
- Fix a bug that caused `open` to fail if the block was already in edition

0.09 (13-02-14)

- There is a brand new visualization in browser of projects and dependencies with “`$bii deps -graph`”
- minor bugfixes
- improved *open command*, now any block can be open inside a project
- improved performance of finds in server and connections pools
- setup totally new. Only `setup:cpp` working now experimentaly. Also `rpi:setup` moved to `setup:rpi`
- apt-get repository for debian based (ubuntu, raspbian) distributions
- new “`bii info`” command

0.08 (5-Feb-14)

- Merge bugfixes
- Project download bugfixes

- Size and performance optimizations in macos and linux clients

0.07.2 (31-Jan-14)

- Merge bugfixes
- Various bugfixes
- Deps output improved

0.06.2 (28-Jan-14)

- Added *arduino support*
- Created raspbian native client
- Improved python native libraries
- Improved virtual cells management
- *Policies* made easier and now user find their own DEV (in master branches) versions by default
- Bugfixes
- Added new tagging system comments_tags.
- Added cpp:exe command that allows executing an already compiled binary w/o recompiling
- Improved renaming support
- Adding OpenGL ES for RPI project generation
- Improved cpp wizard

0.05 (10-Jan-14)

- Raspberry now using rsync instead of scp
- Wizard rpi:setup for automatic install of cross compilers
- New breadcrumb navigation bar for blocks in browser
- Reduced computation by an order of magnitude, especially noticeable in large projects
- Fixed bugs in parsers, that kept old state even the file was modified
- Improved normalization of endlines, for handling also \r
- Fixed bug of not finding new dependencies of files in already dependents blocks

0.04 (20-Dec-13)

- Improved wizards behavior
- Added cookies announcement in web as dictated by law

0.03.4 (17-Dec-13)

- Init and new configuration wizards
- Improved Eclipse support. You can read about it [here](#)
- Improved Raspberry Pi support.
- Changed project structure. It's now easier.
- User can edit cmakes.

0.02.3 (2-Dec-13)

- Experimental upload-download of projects to biicode, so it is not necessary to publish to keep working in other computer.
- Navigation of uploaded projects in the web
- Updated exe creation to pyinstaller2.1, as 2.0 had some problems in some windows installs.
- Creation (experimental) of dynamic libraries from C code. Integration from python code with cffi.
- Improved use of biicode for C/C++ dev with RaspberryPI (linux only)

0.01.11 (28-Oct-13)

- Fixed bug in Eclipse Cmake generated project with empty targets
- Fixed bug that failed when trying to reuse just a data file from another published block (not reusing sources)
- Fixed bug of virtual cells in fortran, due to the “include” does not require to build source file
- Improved NMake support, launching vcvarsall in a .bat file to include environment variables
- Web loads much improved, loading of files with Ajax, rendering of color syntax highlighting with JS, client side and paginated to handle large files
- Web styles improvements, back and forward buttons
- Solved bug of project with multiple src blocks, that was overwriting references to dep blocks
- WxWidgets binary support improved
- Improved handling of python imports, solved bug that didnt renamed properly to absolute imports
- Ctrl+C when init bug fixed (it created empty, wrong workspace)
- Applied some limits and constraints to block sizes, file sizes, number of files in a block and in a project

0.01.10.1

- Fixed bug of crash when dep folder had connected cpp_rules files

0.01.10

- Setup & install in windows problem with setting PATH of biicode solved
- Defined C++0x as default, with possibility of changing it in settings
- Changed “find” command, now with parameters “update”, “downgrade”, “modify”
- Block referencing in client changed from full “owner/creator/block/branch” to “creator/block (owner/branch)”
- Improved setup tools, mainly setup:cpp and setup:node, they update the Environment.bii
- Improved cpp:wizard to create classes and mains
- “dependencies.bii” now able to add, remove and redefine dependencies manually
- SyntaxHighlighter done in browser instead of server to avoid timeouts while browsing large code files
- Solved some bugs in renaming files
- Transitive propagation of cpp_rules from libraries to executables requiring those libraries.
- Solved bugs for user login camelcase
- checkout –deps –force flow improved
- Use system proxy

0.01.9

- Added check of client version, so clients are informed about new releases and deprecated versions, with a download URL
- bii deps –detail command improved showing data dependencies and type of file
- Solved bugs in virtual resources that didn’t let reuse published virtual resources
- cpp_rules files now can accept multiple statements per rule as well as rules without condition and else clauses
- Improved merge, but still very experimental
- Solved bug that allowed to “find” dependencies with cycles to own project blocks
- bii deps –graph now working, showing project block graph in browser
- Fixed problem with renaming files.
- Solved bug with user login upper-lower case mismatch
- Improved possibility of editing directly in dep folder, but still discouraged practice.
- Improved detection of implicit implementations in CPP with static class variables.
- Added preliminary support for fortran, and improved java and node; still experimental languages

- Changed folders in node, now using `NODE_PATH` variable so they don't have to be named `node_modules`

1.12 FAQs

1.12.1 Is biicode free?

Yes, and it will be free forever for open source projects. As long as you share your code, you will enjoy all the functionality and benefits of biicode totally free. However, if you prefer to keep your code private and only accessible to you and your collaborators, you'll have to upgrade to a **premium account**.

1.12.2 Is biicode an editor in the cloud?

No. In biicode you develop as usual, with your favourite platform and tools. We provide a software tool that, once installed in your system, and using a very simple project layout (`bin`, `build`, `src`), is able to manage your source code in a powerful manner. Biicode allows you to reuse any single source file from any given project in any other project. So you can use your favourite IDE, builder, debugger... Now we have some fixed settings, that allow to define some tools as Visual, Eclipse, Mingw, CMake, but it will be soon generalized to all settings, and in any case, you have always the source code very well structured in your project.

1.12.3 Is biicode a VCS?

No. If you need real version control for your projects, you can still use any available solution (`git`, `svn`, `plasticscm`). However, biicode does track and manage versions of the published code in order to provide a powerful, deterministic and scalable dependency manager that also allows easy collaboration on the platform without relying on other tools.

1.12.4 Can I use biicode with my favourite VCS?

Yes, it supports common Git web-based repositories such as GitHub or Bitbucket. You can use any git repository with biicode just doing `git clone` in your blocks folder and the *using your VCS as usual*.

We recommend to work with Git and develop taking care of your project's version, then you can build your project with biicode, publish your useful code and just let biicode manage your dependencies.

It also supports Continuous Integration with *Appveyor* and *Travis CI*. Moreover you can download zip blocks in the web view.

1.12.5 Which languages are supported?

We have released our beta with a strong focus on C++, which lacks a multiOS dependency manager. We are experimenting with other languages as Python, Node, Java or Fortran. The support for these languages will mature during the next months, and new languages will enter the pipeline too.

1.12.6 How does biicode relate to Maven, NPM, PyPI...?

Biicode manages your code dependencies and retrieves the missing files from a central repository to your local machine. However, there are some differences with these services:

- Biicode **always retrieves source code**. If necessary, artifacts (libs, jars) will be built locally for efficiency, but all the management is done with source code. In this way it is easier to develop, debug and collaborate on that source code if necessary.
- Furthermore, **it only retrieves the strictly necessary files**. There is no concept of package in biicode. If you need just one file of biicode, just one file is retrieved. Obviously if that particular file has its own dependencies to other files, those files are also retrieved. The retrieved files are always managed at the project level, thus, it is very easy to have and develop with different versions of dependencies in different projects, in a similar way to a virtualenv compared with a global dependency (as packages with apt-get or similar).
- Moreover, biicode **makes a strong effort on determinism and compatibility**. As the version of each file is tracked, we know when an upgrade is totally safe because it does not affect the specific files you are depending on. Biicode can even handle different versions of the same block (repo) if the dependency tree reaches such different versions but the affected files are not altered in such versions, i.e. we do not have to opt for the (typically) latest version (that can break code that rely on previous versions), but we can even mix in certain cases contents from different versions. In that way, the code that was published depending on specific versions of other code, is always compiled using exactly those versions, achieving fully determinism. Although upgrading version usually does not break code in case of mature libraries and packages, this is not true for more young and dynamic code. In this way, biicode works more as a CI system, in which developers quickly move forward their dependencies.

All that said, biicode does not try to compete against the power of such established systems nor tries to replace them, but to coexist with them as another source, and we are working hard with that purpose.

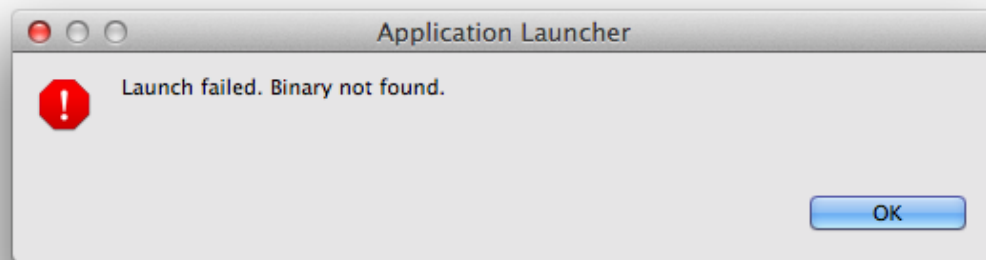
1.13 Troubleshooting

In this section you will find information and helpful resources, in case you encounter any problem while using our service. Don't panic, and try to find a solution for your problem bellow. If you

can't find a solution, please [contact us at our forum](#) and describe your problem.

1.13.1 Eclipse projects: “Launch failed. Binary not found” (OS X)

If you are using **OS X** as development platform, you will need some additional setup for executing your biicode projects within Eclipse IDE. It could happen that after building your project, you cannot execute the binaries from the IDE toolbar, getting the following error:



If you are getting this error, please proceed as follows:

1. Right-click on your project and select *Properties*.
2. Select *C/C++ Make project* and click on the *Binary Parser* subsection tab.
3. Unselect Mach-O Parser (deprecated).
4. Select Mach-O 64 Parser.
5. Click *OK*.

You can [read more in this section about configuring an IDE with your C/C++ projects](#).

1.13.2 g++ doesn't compile simple code, using thread header

If you have a block that **links to pthread library** and you're using **Ubuntu 13.10 or 14.04**, you'll find this bug in g++ compiler:

```
$ ./executable_file
terminate called after throwing an instance of 'std::system_error'
```

Create a **CMakeLists.txt** inside your block that fails and copy the following content:

CMakeLists.txt


```
#####BIICODE MACROS#####
include(${CMAKE_HOME_DIRECTORY}/biicode.cmake)

ADD_BII_TARGETS()
#####

set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -Wl,--no-as-needed")
```

1.13.3 Default Build Configuration with bii build not working

Sometimes the default configuration when installing biicode does not detect the compiler you are using, `bii build` does not work and it is necessary to configure your compiler with biicode.

Windows Users

This can be a problem due to your path configuration. Please, check it out and *ensure everything is right* to use the build configuration by default.

Type inside your project directory:

```
$ bii configure -G "CMake generator name"
```

If you want to check the generators available in your platform:

```
$ cmake --help
```

To configure your project as default just type:

- **Windows:**

```
$ bii configure -G "MinGW Makefiles"
```

- **Linux & MacOS:**

```
$ bii configure -G "Unix Makefiles"
```

Note: You can also type `bii clean` to *restore default configuration*.

Now doing `bii build` should work properly.

Check for more information about **generators** [here](#).

Check our [forum](#) and [Stackoverflow tag](#) for questions and answers.

Arduino

Develop C and C++ projects with Arduino and biicode. Your Arduino with **batteries included**:

1. **Save your project in different folders** for a better organization.
2. **Reuse yours and other user's code**, just `#include` the file you need.
3. **compile and upload** your code in your Arduino.
4. We are hosting [adafruit](#), [sparkfun](#) and [many other libraries](#).

2.1 Installation

Biicode is a file-oriented Dependencies Manager for C and C++ developers. Install both Biicode and the Arduino tools to get started.

2.1.1 Install Biicode

Download [Biicode Installer](#) and double-click the downloaded package to install it.

Open the terminal and **make sure biicode is installed**:

```
~$ bii --version
```

Check alternative installations for:

- *Debian based distrobutions*
- *Arch based distrobutions*

2.1.2 Install Arduino tools

To start, install required tools like CMake or GCC:

```
$ bii setup:arduino
```

Avoid Arduino's SDK installation if you've got it installed already. Re-run `bii setup:arduino` command to verify everything is installed.

Any issues, [contact us at our forum](#), and feel free to ask any questions.

2.1.3 Install Arduino tools manually

Install, set up and verify some **tools to build Arduino projects with biicode**.

Follow these steps if something failed during the automatic installation explained before. If you experience any issues, please [contact us at our forum](#), we'll try to solve your problem as soon as possible. Linux

Install the required development tools as root:

```
$ sudo apt-get install build-essential cmake
```

That's all! MacOS

You need to get installed both XCode Developer Tools and CMake:

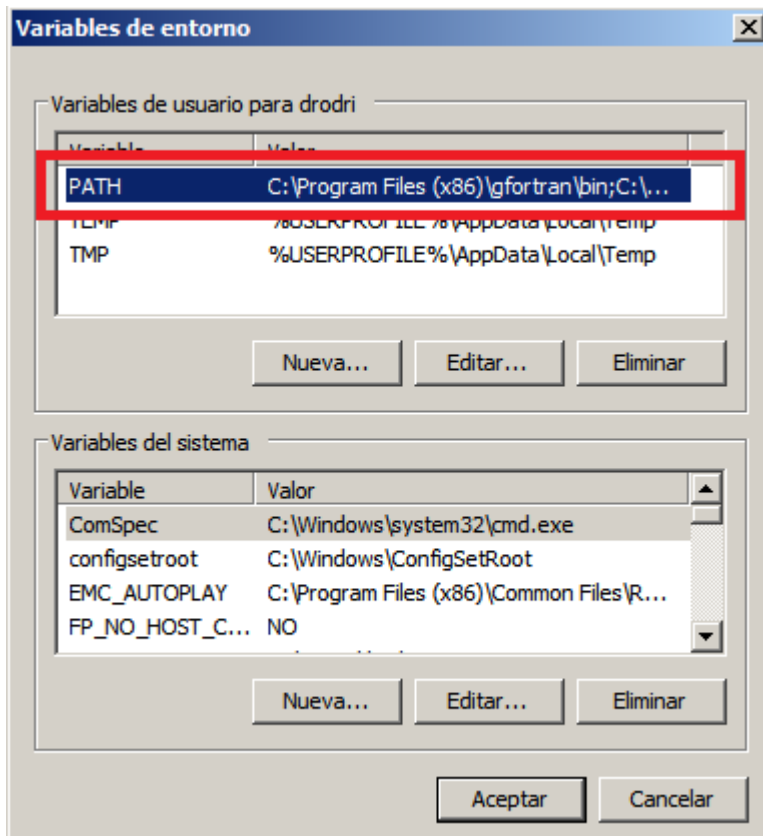
1. The XCode Developer Tools

```
$ xcode-select --install
```

2. Download and install **CMake** for your Mac OSX.

Windows

1. Download and install CMake. You can [download the latest version of CMake here](#).
3. Add to your user `PATH` environment variable the path to CMake. We recommend [Rapid Environment Editor](#) for editing environment variables. Otherwise, go to **My Computer**, click **Properties**, click **Advanced System Settings** and in the System Properties window click the **Environment Variables** button. then you will see a new window and in **User Variables** you'll find the variable `PATH`:



Add CMake binaries folders (i.e. `C:\Program Files (x86)\CMake\bin`).

You might need to close and open again any `cmd` windows in order to load the new value for the `PATH` variable.

Now, you can check CMake is working properly. Open a terminal window and run the following command:

```
$ cmake --version
```

If the output message looks similar to this, CMake is successfully installed.

```
cmake version [version]
```

Install Arduino SDK manually

Download the [Arduino software](#) first, it is important to choose a SDK compatible with your board. If you need more info visit the [official Arduino website](#).

If you have any questions, we are available at . You can also for suggestions and feedback.

2.2 Getting started

This section shows the first steps to use biicode with your Arduino.

Here we will learn:

- *How to install biicode.*
- *Code your first “blink”.*
- *Code a “non blocking blink” reusing from the blink library.*

You don’t need to install the blink library, biicode will download and configure it automatically for you,

2.2.1 Installing biicode and Arduino tools

First and *install biicode*.

Then, open the console and type:

```
~$ bii setup:arduino
```

This will help you to install a group of external tools (Arduino SDK, CMake and MinGW or GCC).

If any problem installing the Arduino tools, check *how to install Arduino tools manually*.

2.2.2 Create your project

First, create a project:

```
~$ bii init myproject
```

Then we can use the convenience `new` command to create some folders and a “Simple Blink” Arduino main file. Of course, you can do it manually too.

```
~$ cd myproject
~/myproject$ bii new myuser/myblock --hello=arduino
```

You can directly type `myuser`, there’s no need to register an account to use biicode, only to upload and share contents. You can use other name too. If you have already registered you might want to replace `myuser` with your real biicode username.

This should be the resulting layout:

```
+-- myproject
|   +-- bii
|   +-- blocks
```

```
|      |      +-- myuser
|      |      |      +-- myblock
|      |      |      |      +-- main.cpp
|      +-- deps
```

2.2.3 Define your board

Just, define your Arduino board using the `arduino:settings` command. In this example we use an Arduino Uno, but you can use another like *Mega2560*.

```
~/myproject$ bii arduino:settings
Enter SDK path (/../biicode_env/arduino-1.0.6): [ENTER]
Enter board (/o list supported options):uno
Using arduino port: COM4
```

2.2.4 Build and upload your program

Let's specify the toolchain to use, build and check that everything is fine by building and uploading the blink application to your Arduino.

```
~/myproject$ bii configure -t arduino
~/myproject$ bii build
...
[100%] Built target myuser_myblock_main

~/myproject$ bii arduino:upload
...
Writing | ##### | 100% 0.00s

avrdude.exe: 0 bytes of eeprom written

avrdude.exe: safemode: Fuses OK

avrdude.exe done. Thank you.

[100%] Built target myuser_myblock_main-upload
Upload finished
```

2.2.5 Depending on Fenix Blink

Now we're going to implement a non blocking blink in arduino. Copy the following code containing the new blink into the *main.cpp* file:

main.cpp

```
#include "Arduino.h"
#include "fenix/blink/blink.h"
Blink my_blink;
void setup() {
    //pin = 13, interval = 1000 ms
    my_blink.setup(13, 1000);
}
void loop() {
    my_blink.loop();
}
```

This code requires the *fenix's* **blink.h** file (You can see it in the `include` section).

If you try to build you will get a **build error**, that's because your project doesn't have the *fenix/blink/blink.h* dependency.

Execute the following command to **find unresolved dependencies** and **retrieve necessary files** from servers:

```
~/myproject$ bii find
```

2.2.6 Build and upload

Now can now build your firmware and upload it to your Arduino:

```
~/arduino_hello_project$ bii build
...
[100%] Built target myuser_myblock_main

~/arduino_hello_project$ bii arduino:upload
...
Writing | ##### | 100% 0.00s

avrdude.exe: 0 bytes of eeprom written

avrdude.exe: safemode: Fuses OK

avrdude.exe done. Thank you.

[100%] Built target myuser_myblock_main-upload
Upload finished
```

That's it! If you see that output it means that *fenix's* `blink.h` was downloaded and uploaded in your project.

Now your Arduino board should be blinking! You have just reused a **non blocking blink!**

You can also check the deps folder, the *blink.h* code is there.

Didn't work? No problem, read or contact us in . Any suggestion or feedback? It is very welcomed :)

2.3 Arduino commands

This section summarizes the **Arduino commands available to be used with the biicode client program**. You can see these tools if you execute:

```
$ bii -h arduino

SYNOPSIS:
    $ bii COMMAND [options]
For help about a command:
    $ bii COMMAND --help
To change verbosity, use options --quiet --verbose

-----Arduino commands-----
arduino:monitor  Open serial monitor
arduino:settings Configure project settings for arduino
arduino:test      Build only the tests declared into your biicode.conf '[tests]' s
arduino:upload    Upload a firmware in Arduino
```

Note: You need to have arduino *correctly set up*.

2.3.1 bii configure -t arduino: configure your project

If you have configured your project as a C/C++ project and you want to develop in arduino language, this command helps you! Enter it and **configure your project like an arduino project**. It invokes arduino cross compiler and you are ready to start with your arduino.

```
$ bii configure -t arduino

invoking cmake -G "MinGW Makefiles" -Wno-dev ../cmake
-- The C compiler identification is GNU 4.3.2
-- The CXX compiler identification is GNU 4.3.2
-- Arduino SDK version 1.0.5: [YOUR_SDK_PATH]
-- Check for working C compiler: [YOUR_PATH]/avr-gcc.exe
-- Check for working C compiler: [YOUR_PATH]/avr-gcc.exe -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working CXX compiler: [YOUR_PATH]/avr-g++.exe
-- Check for working CXX compiler: [YOUR_PATH]/avr-g++.exe -- works
-- Detecting CXX compiler ABI info
```

```
-- Detecting CXX compiler ABI info - done
-- Configuring done
-- Generating done
-- Build files have been written to: [YOUR_BII_WORKSPACE/YOUR_PROJECT]/build
```

Execute `cmake --help` to check all generators available. Here's how to configure *Eclipse for Arduino*.

2.3.2 bii build: build your project

This command uses the cross compiler of Arduino (C compiler -> `avr-gcc` and CXX compiler -> `avr-g++`) to **build and compile the project** via the toolchain you configure it with `bii configure -t arduino`.

```
$ bii build

...

invoking cmake   -G "MinGW Makefiles" -Wno-dev ../cmake
-- The C compiler identification is GNU 4.3.2
-- The CXX compiler identification is GNU 4.3.2
-- Arduino SDK version 1.0.5: [YOUR_SDK_PATH]
-- Check for working C compiler: [YOUR_PATH]/avr-gcc.exe
-- Check for working C compiler: [YOUR_PATH]/avr-gcc.exe -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working CXX compiler: [YOUR_PATH]/avr-g++.exe
-- Check for working CXX compiler: [YOUR_PATH]/avr-g++.exe -- works

...

[100%] Built target your_user_name_block_firmware
```

2.3.3 bii arduino:upload: send your code into the Arduino

When you want to deploy your code into the arduino, this command **sends your previously built firmware** to the arduino.

This command **also builds your code** in case it was not previously built.

```
$ bii arduino:upload

...

[100%] Built target [USER]_my_block_main
```

```
...
Writing | ##### | 100% 0.00s
avrdude.exe: 0 bytes of eeprom written
avrdude.exe: safemode: Fuses OK
avrdude.exe done. Thank you.
[100%] Built target [USER]_my_block_main-upload
```

If you are using an Arduino Yun, you can upload your firmware by ssh with the parameter `--ssh`. To use it, specify the IP of your Arduino.

```
bii arduino:upload --ssh 192.168.0.1
```

Or don't specify anyone and use the default IP (192.168.240.1).

```
bii arduino:upload --ssh
```

2.3.4 bii arduino:settings: configure your Arduino settings

This command **updates your settings with the info about your board and the IDE**, if you want to use one.

```
$ bii arduino:settings
Enter SDK path (/../biicode_env/arduino-1.0.6): [ENTER]
Enter board (/o list supported options): mega2560
Using arduino port: COM13
```

Set your own settings manually. When using non official arduino boards, add the board support to your IDE (if using one), and type the board on `bii arduino:settings`.

2.3.5 bii arduino:monitor: start a serial monitor

This command **opens a serial monitor** to read the serial port of your Arduino board.

```
$ bii arduino:monitor
```

2.3.6 bii configure --toolchain=arduino: enable, disable or change the Arduino cross compilation

Use this command to enable Arduino Cross Compilation.

```
$ bii configure --toolchain=arduino
```

If you need the default arduino-toolchain.cmake, execute `bii arduino:settings` first.

```
$ bii arduino:settings
...
$ bii configure --toolchain=arduino
```

If you want to disable it, use this command.

```
$ bii configure --toolchain=None
```

To use a custom tool-chain you need to **place it in the bii folder** of your project **with the name `<my_toolchain_name>-toolchain.cmake`**.

To use it, just pass it as argument of `bii configure -t my_toolchain_name`.

```
$ bii configure --toolchain=my_toolchain_name
```

If you use a custom toolchain, remember that you need to use the `bii build` to compile your projects.

You can read more info about toolchains in the C++ section

2.4 How to

Here are some useful Arduino use cases:

2.4.1 Eclipse IDE configuration

Biicode offers integration with [Eclipse](#) for Arduino.

First of all, you need to configure your project for arduino and **Eclipse IDE for C/C++**:

```
$ bii arduino:settings
Enter SDK path (/../biicode_env/arduino-1.0.6): [ENTER]
Enter board (/o list supported options): uno
Arduino detected on port COM14
$ bii configure -G "Eclipse CDT4 - Unix Makefiles"
...
A new Eclipse project has been generated for you.
Open eclipse, select "File > Import > General > Existing project into Workspace" and
```

Depending on your OS and desired compiler you can use different Eclipse generators:

- "Eclipse CDT4 - MinGW Makefiles" Generate with MinGW Makefiles.

- "Eclipse CDT4 - Unix Makefiles" Generate with Unix Makefiles.

Now you are ready to import your project into the Eclipse IDE. It is important that you use a version of Eclipse that contains the C/C++ Toolkit. So we recommend using [Eclipse IDE for C/C++ Developers](#).

How to import your project

1. From the main Eclipse menu choose: *File > import...*
2. Now, select *general > Existing Projects into Workspace*, and clic next.
3. Select the root directory as the **root folder of your project**.
4. You should see a project already selected in the *projects* box. Click *finish*.

If you want to add new files to your block, just right-click on the folder of your block and create a new file.

Note: If you add new dependencies to your project you'll need to manually invoke `bii find`.

You can build your application in *Project > Build project* if you don't have automated builds set.

If you are using **Mac** as developing platform, you will need some additional setup:

1. Right-click on your project and select *Properties*.
2. Select *C/C++ Make project* and click on the *Binary Parser* subsection tab.
3. Unselect Mach-O Parser (deprecated).
4. Select Mach-O 64 Parser.
5. Click *OK*.

How to fix “Unresolved inclusion: Arduino.h”

1. Open the project settings and go to C/C++ General -> Paths and Symbols
2. Click “Add external include path” and add:
 - For Arduino IDE installed with biicode:
 - MAC: `~/ .biicode_env/arduino-1.0.6/Arduino.app/Contents/Resources/Java`
 - Linux: `~/ .biicode_env/arduino-1.0.6/hardware/arduino/cores/arduino`
 - Windows: `C:\biicode_env\arduino-1.0.6\hardware\arduino\cores\arduino`
 - For manually installed Arduino IDE, just add the equivalent route.

And this is all you need to work as usual with the Eclipse IDE. **Any doubts?** Do not hesitate to [contact us](#) visit our [forum](#) and feel free to ask any questions.

2.4.2 Configure your SDK, port and board

I changed my Arduino's port, what happens now?

Port detection is automatic, execute `bii arduino:upload` to 're-detect' the board.

If you experience problems changing your port or your arduino board, just execute `bii arduino:settings`

How can I change my Arduino project properties?

Execute `bii arduino:settings` to define your Arduino SDK, board and port:

```
$ bii arduino:settings
Enter SDK path ( ../../biicode_env/arduino-1.0.6 ): [ENTER]
Enter board (/o list supported options): mega2560
Using arduino port: COM13
```

`bii arduino:settings` options

SDK path

Press enter to choose the Arduino SDK path in brackets. To use a different Arduino SDK, write the path where it is located.

Want to install Arduino SDK? execute `bii setup:arduino`.

Boards

Type your Board model, just make sure it's compatible with the Arduino SDK v. 1.0.5 and your IDE supports it. This is a list of the boards supported by default, even though any board is welcomed!

- `uno`: Arduino Uno
- `zum`: [bq ZUM BT-328](#)
- `yun`: Arduino Yun
- `atmega328`: Arduino Duemilanove w/ ATmega328
- `diecimila`: Arduino Diecimila or Duemilanove w/ ATmega168
- `nano328`: Arduino Nano w/ ATmega328
- `nano`: Arduino Nano w/ ATmega168

- `mega2560`: Arduino Mega 2560 or Mega ADK
- `mega`: Arduino Mega (ATmega1280)
- `leonardo`: Arduino Leonardo
- `esplora`: Arduino Esplora
- `micro`: Arduino Micro
- `mini328`: Arduino Mini w/ ATmega328
- `mini`: Arduino Mini w/ ATmega168
- `ethernet`: Arduino Ethernet
- `fio`: Arduino Fio
- `bt328`: Arduino BT w/ ATmega328ad
- `bt`: Arduino BT w/ ATmega168
- `LilyPadUSB`: LilyPad Arduino USB
- `lilypad328`: LilyPad Arduino w/ ATmega328
- `lilypad`: LilyPad Arduino w/ ATmega168
- `pro5v328`: Arduino Pro or Pro Mini (5V, 16 MHz) w/ ATmega328
- `pro5v`: Arduino Pro or Pro Mini (5V, 16 MHz) w/ ATmega168
- `pro328`: Arduino Pro or Pro Mini (3.3V, 8 MHz) w/ ATmega328
- `pro`: Arduino Pro or Pro Mini (3.3V, 8 MHz) w/ ATmega168
- `atmega168`: Arduino NG or older w/ ATmega168
- `atmega8`: Arduino NG or older w/ ATmega8
- `robotControl`: Arduino Robot Control
- `robotMotor`: Arduino Robot Motor

USB port

Biicode automatically detects the USB port your Arduino board is connected. Manual USB Port configuration varies from one operating system to another, so look for your USB port and then type it on the command line after executing `bii arduino:settings`:

Linux/Unix

For Linux/Unix type `ls /dev/ttyUSB*` or `ls /dev/ttyACM*` into a terminal window. You should see a device called like these:

- `/dev/ttyUSBX`

- `/dev/ttyACMX`

Where `/dev/ttyACMX` is for the new Uno and Mega Arduino's and `/dev/ttyUSBX` is for the old ones. **X** is the device number.

MacOS

Under Mac, in the Terminal window, type `ls /dev/cu.*` which should give you a name like this one:

- `/dev/cu.usbserialXXX`

In which **XXX** is a unique ID.

Windows

If using Windows, go to the **Device Manager** and look for an entry under **Ports (COM & LPT)** that says **USB Serial Port (COMX)** specifying the serial port name on Windows, in which **X** is the device number:

- COM1, COM2, etc.

2.4.3 How to adapt your code

Adapt your current Arduino projects and sketches to biicode. Next steps depend on whether your project contains one single `.ino` file, or multiple `.ino` files. Just follow the guide that suits best your current project:

1. Projects with one single `.ino` file

This is the simplest case. Adaptation is almost immediate:

General rules

1. Put your code into a biicode block .
2. **Change** `.ino` file **extensions** to `.cpp`. For example, `sweep.ino` file would be `sweep.cpp`.
3. **Include** the `Arduino.h` **library** in the very beginning of your code:

```
#include "Arduino.h"
```

4. Declare all function prototypes at the beginning of your code. For example: `void servo_loop();`
5. If you're using 3rd party (external) libraries, search for them in biicode and **rewrite** the library includes to the ones available on biicode. Usually, all included files in biicode projects follow this pattern: `<username>/<blockname>/<path_to_file>`.

For example, if you are using [Tiny GPS Library](#) , `tinygps.h` , you should update the include directive like this:

```
#include "mikalhart/tinygps/tinygps.h"
```

Full .ino file adaptation Sample:

Original sweep.ino file

```
#include <Servo.h>

void setup() {
}

void loop() {
  servo_loop(9);
}

void servo_loop(int pin) {
  Servo myservo;
  myservo.attach(pin);
  for (int pos = 0; pos <= 180; pos += 1) {
    myservo.write(pos);
    delay(15);
  }
}
```

Adapted sweep.cpp file

```
#include "Arduino.h"
#include <Servo.h>

void servo_loop();

void setup() {
}

void loop() {
  servo_loop(9);
}

void servo_loop(int pin) {
  Servo myservo;
  myservo.attach(pin);
  for (int pos = 0; pos <= 180; pos += 1) {
    myservo.write(pos);
    delay(15);
  }
}
```

2. Projects with multiple .ino files

General rules

1. Put your code into a biicode block.
2. Change .ino main file extension to .cpp. Also, change the extension of the other .ino files to .h.

For example, if your project has the following layout, being sweep.ino your main file:

```
+++ sweep
|   +-+ sweep.ino
|   +-+ servo_functions.ino
```

Rename sweep.ino to sweep.cpp, and servo_functions.ino to servo_functions.h. Put them into your project/blocks/username/biicode_block_name directory like this:

```
+++ <biicode_block_name>
|   +-+ sweep.cpp
|   +-+ servo_functions.h
```

3. **Include** the Arduino.h **library** in the very beginning of your code:

```
#include "Arduino.h"
```

4. If you're using 3rd party (external) libraries, search for them in biicode and **rewrite** the library includes to the ones available on biicode. Usually, all included files in biicode projects follow this pattern: <username>/<blockname>/<path_to_file>.

For example, if you are using [Tiny GPS Library](#) , tinygps.h , you should update the include directive like this:

```
#include "mikalhart/tinygps/tinygps.h"
```

5. Declare all function prototypes at the beginning of your code. For example void servo_loop() ;.

Find below a full multiple .ino files adaptation sample. Easy adaptation is the one described before and advanced adaptation is the one biicode recommends for complex projects o nice practice:

Original sweep Arduino project

sweep.ino

```
void setup() {
}
```

```
void loop() {
  servo_loop(9);
}
```

servo_functions.ino

```
#include <Servo.h>

void servo_loop(int pin){
  Servo myservo;
  myservo.attach(pin);
  for (int pos = 0; pos <= 180; pos += 1){
    myservo.write(pos);
    delay(15);
  }
}
```

Easy adaptation

sweep.cpp

```
#include "servo_functions.h"

void setup() {
}

void loop() {
  servo_loop(9);
}
```

servo_functions.h

```
#include "Arduino.h"
#include <Servo.h>

void servo_loop(int pin){
  Servo myservo;
  myservo.attach(pin);
  for (int pos = 0; pos <= 180; pos += 1){
    myservo.write(pos);
    delay(15);
  }
}
```

Advanced adaptation

This is an alternative way to adapt your code to biicode. It is recommended as a **best practice** that splits your code into declarations, or interface, and implementations. This separation between **interface** (contained in *header* files, with `.h` extension) and **implementation** (contained in `.cpp`

files) provides many benefits:

- The header declares **what** the code does, while the implementation contains **how** to do it. This is somehow a nice way of structuring your code.
- Compilation times are reduced, and also the need to recompile all your code when only some parts of the implementation have changed.

sweep.cpp

```
#include "servo_functions.h"

void setup() {
}

void loop() {
    servo_loop(9);
}
```

servo_functions.h

```
#include "Arduino.h"

void servo_loop(int pin);
```

servo_functions.cpp

```
#include "servo_functions.h"
#include <Servo.h>

void servo_loop(int pin) {
    Servo myservo;
    myservo.attach(pin);
    for (int pos = 0; pos <= 180; pos += 1) {
        myservo.write(pos);
        delay(15);
    }
}
```

2.4.4 How to use the Arduino Yun

Download Arduino 1.5

Download the **Arduino zip file** from [Arduino download page](#) and unzip it.

Configure your settings

Do **bii arduino:settings** to configure the new SDK path and the board as yun.

In this example we have unzipped our SDK into `c:/biicode_env/arduino-1.5.8`.

```
$ bii arduino:settings
Enter SDK path ( ../../biicode_env/arduino-1.0.6 ): c:/biicode_env/arduino-1.5.8
Enter board (/o list supported options): yun
Using arduino port: COM13
```

And that's all. Now you can use your Yun with biicode.

2.5 Examples

Here you will find some code examples showing the advantages of using biicode with your arduino projects. You can find additional examples on the [Arduino official website](#).

2.5.1 Arduino Serial Monitor

This example shows you how to run **biicode's Arduino Serial Monitor Interface** and turn ON/OFF one LED easily.

If you don't know how to start using Arduino with biicode, check out our [getting started guide for Arduino](#).

C++ code

Put the following file in your `~/project_name/blocks/your_user_name/block_name/` or just *open* the [example block](#).

monitor_led.cpp

```
#include "Arduino.h"

// Pin 13 has an LED connected on most Arduino boards
int led = 13;

void setup() {
    pinMode(led, OUTPUT); // initialize the digital pin as an output
    Serial.begin(9600);
    Serial.print("Send '1' or '0' to turn ON/OFF the led in Pin ");
    Serial.println(led, DEC);
}
```

```
void loop() {  
    //checking data has been sent  
    if (Serial.available() > 0) {  
        char msg = Serial.read(); //read a message  
        if(msg == '1'){  
            digitalWrite(led, HIGH); // turn the LED on  
            Serial.println("LED -> ON");  
        }  
        else if(msg == '0'){  
            digitalWrite(led, LOW); // turn the LED off  
            Serial.println("LED -> OFF");  
        }  
    }  
}
```

Download: `monitor_led.cpp`

Turn ON/OFF one LED

First, configure your arduino settings and toolchain:

```
$ bii arduino:settings  
...  
$ bii configure -toolchain=arduino  
...
```

Sencond, build your project:

```
$ bii build
```

Now **upload this firmware to your Arduino** with the following command:

```
$ bii arduino:upload  
  
...  
  
[100%] Built target [your_user_name]_monitor_led_main-upload
```

You shouldn't get any erros but if you do, please check that your Arduino is correctly connected, and check your settings (using the `bii arduino:settings` command).

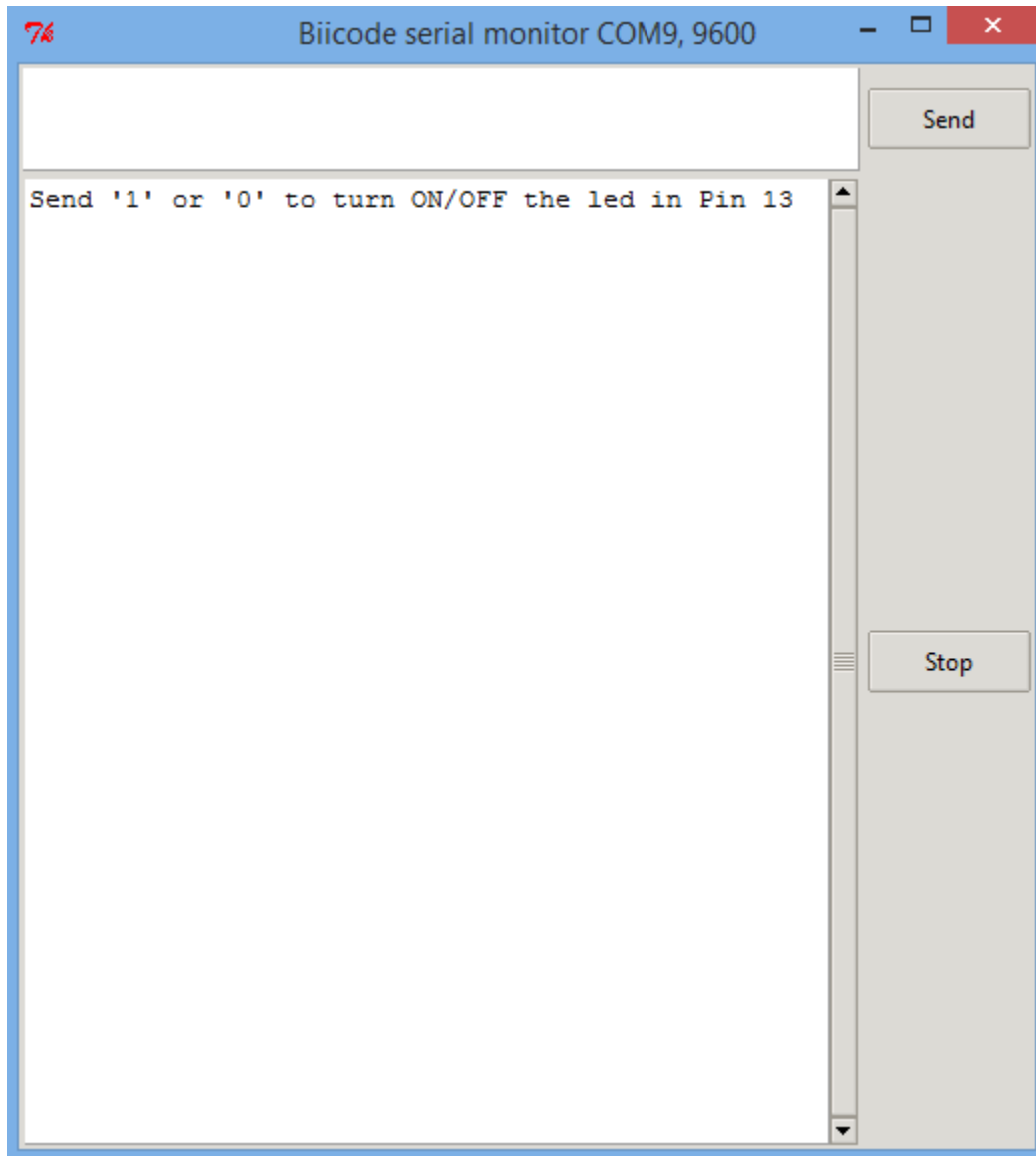
If you still have any issues please contact us at our [forum](#).

Now, **open the Arduino monitor**. You only need to execute the following command:

```
$ bii arduino:monitor  
  
...
```

```
Arduino detected on port [YOUR_PORT]
```

You'll see a window with an interface that allows you to communicate with the Arduino Serial Port. Here you can send 1 or 0 to turn the led ON or OFF, respectively.



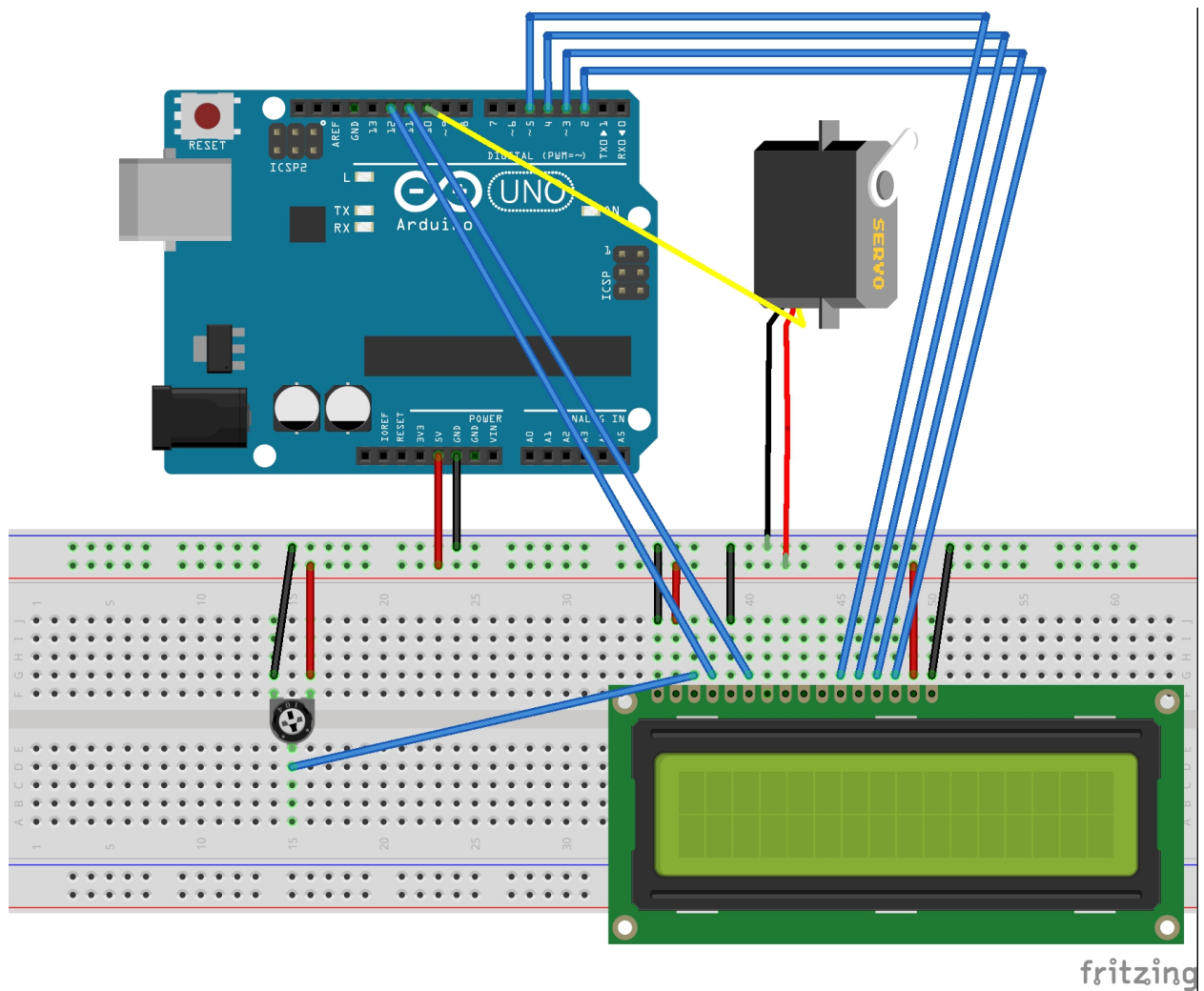
2.5.2 Servo and LCD 2x16

With this example you will display the angle of a servo into a LCD 2x16. It uses the **Servo**, `servo.h`, library to control the servo and the **Liquidcrystal**, `liquidcrystal.h`, library to display the angle on the LCD.

What do we need?

- One standard servo.
- One display lcd 2x16.
- One potentiometer.

Scheme



The code: Display the servo angle into a LCD

This example is [already in biicode](#). You can create your project and open the block or copy the code into a `.cpp` file:


```
$ bii init my_arduino_project
$ cd my_arduino_project
$ bii open examples/servolcd
```

Check the code inside your block's folder:

lcd_and_servo.cpp

```
#include <Arduino.h>
#include <LiquidCrystal.h>
#include <Servo.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
Servo myservo;

void write_angle_lcd(int pos) {
    lcd.setCursor(0, 1);
    lcd.print(pos);
    lcd.print(" ");
}

void setup() {
    myservo.attach(10);
    lcd.begin(16, 2);
    lcd.print("SERVO ANGLE");
    lcd.setCursor(0, 1);
}

void loop() {
    int pos;
    for(pos = 0; pos <= 180; pos += 1) {
        myservo.write(pos);
        write_angle_lcd(pos);
        delay(50);
    }
    for(pos = 180; pos >= 0; pos -= 1) {
        myservo.write(pos);
        write_angle_lcd(pos);
        delay(50);
    }
}
```

Build and upload the code

First, we have to configure the settings and select the arduino toolchain:

```
$ bii arduino:settings
...
```

```
$ bii configure -toolchain=arduino
...
```

Secondly, build this c++ example for arduino:

```
$ bii build
```

And upload it to your board:

```
$ bii arduino:upload
```

You are done! You'll see your servo moving and the angle into the LCD.

Any doubts? Do not hesitate to [contact us](#), visit our [forum](#) and feel free to ask any questions.

2.5.3 Arduino Serial Interface

With this serial interface you'll be able to send commands from a console in your PC to the arduino. You can also define your own commands for the arduino.

In this example we use a **desktop console app** and an **arduino program** to move a servo attached to the Arduino.

Send the “servo” command from our desktop app to the arduino program, and type the desired angle. The arduino board connected via USB will do the rest!

How does it work?

Just need to use the methods `read` and `write` to communicate with the device through serial port. Those functions and the whole API definition are defined in [david/serial_arduino](#) and [david/serial_cpp](#) libraries.

How do I use it?

- You need to create **two projects** `arduino_app` and `cpp_app`, one will contain the code that will be uploaded to arduino and the other one the client application that will run in your PC.

```
$ bii init cpp_app
```

```
$ bii init arduino_app
```

- Open the examples: [C++ app code](#) goes into your pc project and [arduino code](#) goes into the arduino's project.

```
$ cd cpp_app
$ bii open examples/serial_interface_cpp
```

```
$ cd arduino_app
$ bii open examples/serial_interface_arduino
```

C++ code

Change the **Serial Port ID** in the *main_cpp.cpp* file with the one you are using with the arduino.

main_cpp.cpp

```
#include "david/serial_cpp/serial.h"
#include <string>
#include <iostream>

using namespace std;

int main()
{
    string incomingData = "";
    string input = "";
    serial serialport('#', ';', "COM8", 9600); //Change the serial port ID!!

    while(1){
        input = serialport.read(); //read a message
        if (input != "") cout << input << "\n";
        else{
            cout << "Enter: ";
            cin >> incomingData;
            incomingData = "#" + incomingData;
            incomingData += ";";
            serialport.writeString(incomingData); //send a message
        }
    }
    return 0;
}
```

Arduino code

main_arduino.cpp

```
#include <Arduino.h>

#include <Servo.h>
#include "david/serial_arduino/serial.h"
```

```
serial serialport('#', ';', 9600);
String msg = "";
String premsg = "";
Servo myservo;

void setup() {
    myservo.attach(9);
    serialport.init();
}

// bii:#entry_point()
void loop() {

    msg = serialport.read(); //read a message
    if(msg != "")
    {
        serialport.writeOpen();
        serialport.writeString(msg); //send a message
        serialport.writeEnd();

        if(premsg=="servo") {
            int n;
            n = atoi(msg.c_str());
            myservo.write(n);
        }
        premsg = msg;
    }
}
```

Build and run!

Execute following commands in each project:

Arduino App

```
$ bii arduino:settings
...
$ bii configure -t arduino
...
$ bii build
...
$ bii arduino:upload
```

C++ App

```
$ bii build
$ cd bin
$ #run solver executable
Enter: servo
servo
Enter: 180
180
Enter: servo
servo
Enter: 90
...
```

Now you can start hacking your own commands!

2.6 Troubleshooting

2.6.1 Launching Arduino IDE, I get an error ./arduino: 22: ./arduino: java: not found in Ubuntu

You can do it this way:

```
$ sudo apt-get install openjdk-7-jre librtx-java
```

Then you shouldn't have problems executing the IDE.

Check our [forum](#) and [Stackoverflow](#) tag for questions and answers.

Raspberry Pi Cross Compilation

A C and C++ dependency manager with cross compilation integrated for Raspberry Pi.

3.1 Installation

Biicode is a file-oriented Dependencies Manager for C and C++ developers. Install both Biicode and the cross compiling tools for Raspberry PI to get started.

3.1.1 Install Biicode

Download [Biicode Installer](#) and double-click the downloaded package to install it.

Open the terminal and **make sure biicode is installed**:

```
~$ bii --version
```

Check alternative installations for:

- *Debian based distrobutions*
- *Arch based distrobutions*

3.1.2 Install RPI tools

To start, install required tools like CMake or GCC:

```
$ bii setup:rpi
```

Re-run `bii setup:rpi` command to verify everything is installed.

Any issues, [contact us at our forum](#), and feel free to ask any questions.

3.1.3 Install RPI cross-compiling tools manually

Manually install the cross-compiling tools. Go through this section if something failed during the automatic installation explained before. Any issues, please [contact us at our forum](#), and we'll try to solve it as soon as possible.

C++ tools installation

Install the required development tools as root:

```
$ sudo apt-get install build-essential cmake
```

If you are using a **64-bit version of Linux** as development environment, follow the next steps.

Install lib32z1 (only for 64-bit linux versions)

To find if your Linux is 32 or 64 bits, just type:

```
$ uname -m
```

This command may throw one of the following outputs:

- x86_64 ==> 64-bit kernel
- i686 ==> 32-bit kernel

If you are using a 64-bit OS, you need to install the support for 32 bit applications. These libraries are required to use the cross compilers supplied by Raspberry Pi from their Github repository.

```
$ sudo apt-get install lib32z1
```

It is possible that you encounter some of the following errors:

- If it does not find the package, you may need to add a 32-bits architecture to your package list:

```
$ sudo dpkg --add-architecture i386
$ sudo apt-get update
$ sudo apt-get install ia32-libs
```

- If you get:

```
The following packages have unmet dependencies:
ia32-libs : Depends: ia32-libs-multiarch
E: Unable to correct problems, you have held broken packages.
```

execute:

```
$ sudo apt-get install libgl1-mesa-dri:i386
$ sudo apt-get install ia32-libs-multiarch:i386
```



```
$ sudo apt-get install ia32-libs-multiarch
$ sudo apt-get install ia32-libs
```

- If you get:

```
Some packages could not be installed.
This may mean that you have requested an impossible situation the following
ia32-libs : Depends: ia32-libs-multiarch
```

just install the dependencies manually like any other package:

```
$ sudo apt-get install ia32-libs-multiarch
$ sudo apt-get install ia32-libs
```

Raspberry Pi tools installation

One essential step for cross-compiling your programs is downloading [the Raspberry Pi tools from this Github repository](#). You'll need Git installed on your Linux system. You can install an existing package on your Ubuntu platform with the following command:

```
$ sudo apt-get install git
```

Now, you need to clone this Git repo in this folder: `~/.biicode_env/raspberry_cross_compilers`. To do so, execute the following command:

```
$ git clone https://github.com/raspberrypi/tools.git ~/.biicode_env/raspberry_cross_compilers
```

After a while, the RPi cross-compiling tools will be available in your system. To verify that biicode is able to find and use the compilers, run the following command, and check that you receive a success output message:

```
$ bii setup:rpi
INFO: The arm gnu is already downloaded
```

If you have any questions, we are available at . You can also for suggestions and feedback.

3.2 Getting started

This example shows **how to install biicode, code a C++ led blink with , make the cross compilation and send the executable to your Raspberry Pi.**

You don't need to have WiringPi installed in your computer or the rpi. Biicode will download and configure it automatically for you, !

3.2.1 1. Installing biicode and C/C++ cross-building tools

Debian linux distribution required

You need to use a native debian linux or in a virtual machine to use the cross compilation tools.

First, and install biicode.

Then, open the console and type:

```
~$ bii setup:rpi
```

If any problem installing the C/C++ cross-building tools, you can see [how to install RPi cross-compiling tools manually](#)

3.2.2 2. Create your project

First, create a project:

```
~$ bii init myproject
```

Then we can use the convenience `new` command to create some folders and a “Hello World” C++ main file. Of course, you can do it manually too.

```
~$ cd myproject
~/myproject$ bii new myuser/myblock --hello=cpp
```

You can directly type `myuser`, there’s no need to register an account to use biicode, only to upload and share contents. You can use other name too. If you have already registered you might want to replace `myuser` with your real biicode username.

This should be the resulting layout:

```
+-- myproject
|   +-- bii
|   +-- blocks
|   |   +-- myuser
|   |   |   +-- myblock
|   |   |   |   +-- main.cpp
|   +-- deps
```

3.2.3 3. Build and run your program (cross-compiling)

Configure your project to the cross compiling running `bii rpi:settings`:

```
~$ cd myproject
~/myproject$ bii rpi:settings
```

```
Define RPI settings for external C/C++ cross-building
If you are working on board the RPI, you don't need these settings:
RPI username (pi): [ENTER]
RPI IP Address: 192.168.1.44
RPI directory to upload (bin): [ENTER] #This folder must exist into your Raspberry
Creating toolchain for Raspberry PI
Run "$bii cpp:configure --toolchain=rpi" to activate it
Run "$bii cpp:configure --toolchain=None" to disable it
```

Activate the toolchain for Raspberry PI with `bii cpp:configure --toolchain=rpi`:

```
bii configure --toolchain=rpi
```

Lets check that everything is fine by building and trying to run the hello world application. It could fail the execution because it is compiled to Raspberry PI.

```
~/myproject$ bii build
...
~/myproject$ ./bin/user_myblock_main
./bin/user_myblock_main: cannot execute binary file
```

3.2.4 4. Send your executable to your Raspberry Pi

To **send the binary to your Raspberry Pi**, you just need to execute the `bii rpi:send` command and the file will be sent using `rsync` to the address provided in your settings.

```
$ bii rpi:send
Sending with rsync -Pravdtze ssh [PROJECT_DIRECTORY]/bin/* [RPI_USER]@[RPI_IP]:[DIR]
[RPI_USER]@[RPI_IP]'s password:
```

The Raspberry Pi user's password will be asked. If you have not changed your password, for Raspbian the default one is **raspberry**.

Finally, to **execute your program on your Raspberry Pi**, you need to establish a connection. You can use the `rpi:ssh` command if you want remote access. You'll find your program deployed in the path configured in your settings:

```
$ bii rpi:ssh
...
Connecting with ssh <rpi_user>@<rpi_ip>
<rpi_user>@<rpi_ip>'s password:

pi@raspberrypi ~ $ cd hello_rpi
pi@raspberrypi ~/hello_rpi $ ls
myuser_myblock_main
```

```
pi@raspberrypi ~/hello_rpi $ ./user_myblock_main
Hello world!
```

3.2.5 5. Depending on WiringPi

Copy the following code containing a simple sum function and a test into the **main.cpp** file

```
#include "drogon/wiringpi/wiringPi/wiringPi.h"
#define LED 0
int main (void){
    wiringPiSetup () ;
    pinMode (LED, OUTPUT) ;
    digitalWrite (LED, HIGH) ; // On
}
```

Execute the following command to find unresolved dependencies and retrieve necessary files from servers:

```
~/myproject$ bii find
```

Now you are ready to compile and deploy your new application. First, **cross-compile your program** and make sure the binary is generated running `bii build` from your project location:

```
$ bii build
...
Configuring cross compiler for ARM architecture:
...
[100%] Built target myuser_myblock_main
```

The binaries are created in `bin` folder.

Remember that **you cannot run this program locally, as it has been compiled for a different architecture** using the cross-compiling tools. You need to send the binary to your Raspberry Pi before executing it.

Didn't work? No problem, read or contact us in .

Any suggestion or feedback? It is very welcomed :)

Visit the section: *[Upload and reuse code \(C/C++\)](#)*

3.3 RPi commands

This section summarizes the **Raspberry Pi commands** available to be used with the **biicode client program**. You can see these tools if you execute:

```
$ bii -h rpi

SYNOPSIS:
    $ bii COMMAND [options]
For help about a command:
    $ bii COMMAND --help
To change verbosity, use options --quiet --verbose

-----EXPERIMENTAL Raspberry Pi general tools commands-----
rpi:send          Send by scp the bin folder into the specified directory
rpi:settings      Configure Raspberry Pi project settings
rpi:ssh           Connect by ssh with the Raspberry Pi
```

3.3.1 bii rpi:send: send a bin folder

Send your binaries automatically to your Raspberry Pi from your PC.

```
$ bii rpi:send
Sending with rsync -Pravdtze ssh [HIVE_DIRECTORY]/bin/* [RPI_USER]@[RPI_IP]:[DIRECTO
[RPI_USER]@[RPI_IP]'s password:
```

The Raspberry Pi user's password will be asked. If you have not changed your password, for Raspbian the default one is **raspberry**.

3.3.2 bii rpi:settings: configure your Raspberry Pi settings

This command provides an easy way to configure your Raspberry Pi settings.

```
$ bii rpi:settings

Define RPI settings for external C/C++ cross-building
If you are working onboard the RPI, you don't need these settings
RPI username (pi): [RPI_USER]
RPI IP Address: [RPI_IP] #example 192.168.1.44
RPI directory to upload (bin): [RPI_FOLDER] #This folder must exist into your Raspb
```

- **RPI username (pi):** Raspberry Pi user name. Default value is `pi`.
- **RPI IP Address:** Raspberry Pi local IP address. Write here your Raspberry Pi network address, that you can find out executing the `ifconfig` in a console inside the RPi.
- **RPI directory to upload (bin):** Raspberry Pi directory where you want your programs to be saved. Default value is the `bin` user home folder.

Alternatively you can pass your configuration directly into the settings command.

```
$ bii rpi:settings --user USER --ip IP --directory DIRECTORY
```

3.3.3 bii rpi:ssh: connect by ssh with the Raspberry Pi

If you want a remotely access to your Raspberry Pi you can get it with this command.

```
$ bii rpi:ssh
...
Connecting with ssh <rpi_user>@<rpi_ip>
<rpi_user>@<rpi_ip>'s password:

pi@raspberrypi ~ $
```

3.3.4 bii configure --toolchain=rpi: enable, disable or change the Raspberry Pi cross compilation.

Use this command to enable Raspberry Pi Cross Compilation.

```
$ bii configure --toolchain=rpi
```

If you need the default rpi-toolchain.cmake, execute `bii rpi:settings` first.

```
$ bii rpi:settings
...
$ bii configure --toolchain=rpi
```

If you want to disable it, use this command.

```
$ bii configure --toolchain=None
```

To use a custom tool-chain you need to place it in the bii folder of your project with the name `<my_toolchain_name>-toolchain.cmake`.

To use it, just pass it as argument of `bii configure -t my_toolchain_name`.

```
$ bii configure --toolchain=my_toolchain_name
```

You can read more info about toolchains in the C++ section

‘Passwordless secure SSH access is possible by following the instructions on the raspberry pi help section<<https://www.raspberrypi.org/documentation/remote-access/ssh/passwordless.md>>‘_

3.4 How to

3.4.1 Installing the biicode package from downloads page is too slow

You can do it this alternative way:

```
$ sudo dpkg -i bii-ubuntu[ARCH]_[VERSION].deb
```

For example, you want to install the new release biicode package, for example, *0.7.2* in your Ubuntu 12.10 64 bits version, then you have to enter:

```
$ sudo dpkg -i bii-ubuntu64_0_7_2.deb
```

If you would have the 32 bits version:

```
$ sudo dpkg -i bii-ubuntu32_0_7_2.deb
```

3.4.2 Output selection and volume control

You can force the RPi to use a specific interface using the command `amixer cset numid=3 N`, where the `N` parameter can take the following values:

- 0 for **auto** selection
- 1 for **analog** output
- 2 for **hdmi** output

Therefore, to force the Raspberry Pi to use the analog output, you can use the following command:

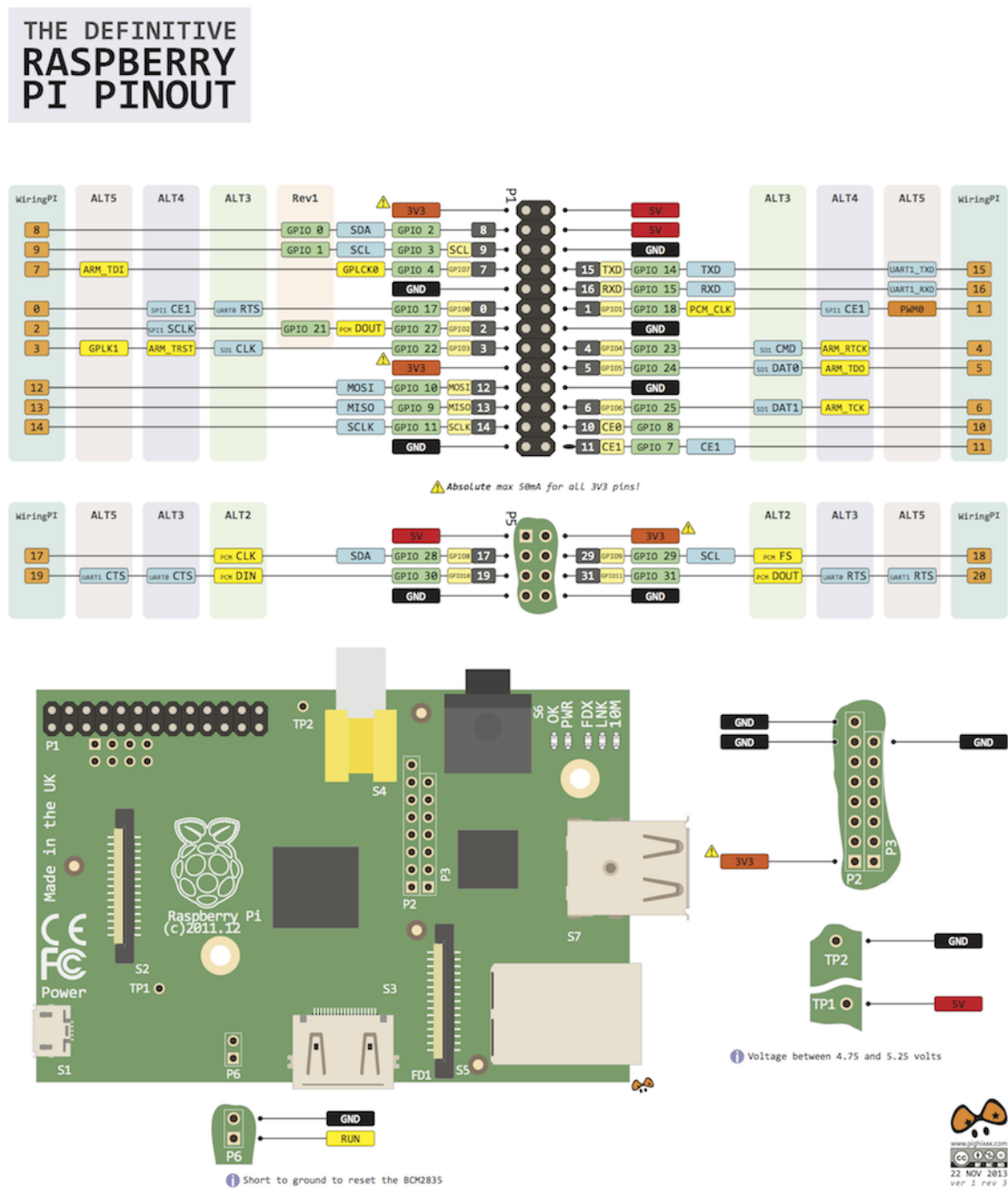
```
$ amixer cset numid=3 1
```

If you want to change the volume level:

```
$ amixer cset numid=1 -- 20%
```

3.4.3 Raspberry Pi GPIO Pin Layout

The definitive Raspberry Pi pinout by pighixx



3.5 Examples

In this section you will find some examples, showing the benefints of **using biicode technology with your Raspberry Pi C++ projects**. All examples make use of well-known libraries, uploaded to the biicode platform and available for everyone to use.

You can compile these projects directly on your Raspberry Pi. However, for performance reasons, we suggest using the cross compilation feature built in biicode. Cross compilation only works on Linux systems at this moment.

3.5.1 WiringPi: C GPIO library

WiringPi is a C library that provides **easy access to the Raspberry Pi GPIO system**. It's designed to provide similar functionality to the Wiring package, core of the Arduino input/output system. The library supports the UART port, SPI, I2C and PWM. In the [project page](#) you will find additional information.

Author: Gordon Henderson
biicode library site
WiringPi library reference page
WiringPi home page

Now we present some examples using this library with biicode projects. The first one is a **simple blinking LED**. The second one shows how to **use the PWM output to control a motor**.

How to make a LED blink with Raspberry Pi

With this example we will have a **LED flashing every half second using the WiringPi pin 0** (or RPi GPIO-17; it's physical location is pin 11 on the GPIO connector). You can learn more about the *Raspberry Pi GPIO pin layout in this documentation*, or reading the [Embedded Linux Wiki](#).

You must create a new project and a new empty block, as explained in the *biicode RPi getting started guide*. Then, place the following source file inside your block, and execute the `bii find` command. All needed source files will be downloaded to the `deps` folder of your project (you can also read the *reference for the bii find command*).

Example: blink.c

```

1 #include <stdio.h>
2 #include <drogon/wiringpi/wiringPi/wiringPi.h>
3
4 // LED Pin - wiringPi pin 0 is BCM_GPIO 17.
5
6 #define          LED          0
7
8 int main (void)
9 {
10     printf ("Raspberry Pi blink\n") ;
11
12     wiringPiSetup () ;
13     pinMode (LED, OUTPUT) ;
14
```

```
15     for (;;)
16     {
17         digitalWrite (LED, HIGH) ;           // On
18         delay (500) ;                         // mS
19         digitalWrite (LED, LOW) ;            // Off
20         delay (500) ;
21     }
22     return 0 ;
23 }
```

Download: `blink.c`

Note that the previous file includes the [main header file of drogon's WiringPi library](#), available from [biicode](#). Now you are ready to build your program using the `biicode build` command, and send it to your board using the `biicode rpi:send` command, as described in the [biicode RPi getting started guide](#).

You must **execute this program on your Raspberry Pi with the `sudo` command**, because it needs to gain access to the board hardware. All needed hardware and wiring setup is explained in [this example](#) available at the [WiringPi creator's projects page](#).

How to use the RPi PWM output to control a motor

With this example we will have a PWM sawtooth function on WiringPi pin 0 (*GPIO-10*). You need to **set pin 1 as PWM whit the `pinMode` function**: `pinMode (1, PWM_OUTPUT)`. Once again, create a new project with an empty block, as described in the [biicode RPi getting started guide](#). Place the following example file inside your block, execute the `biicode find` command to retrieve all dependencies, and build and deploy your program using the `biicode build` and `biicode rpi:send` commands.

Example: `pwm.c`

```
1 #include <drogon/wiringpi/wiringPi/wiringPi.h>
2
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <stdint.h>
6
7 int main (void)
8 {
9     int bright ;
10
11     printf ("Raspberry Pi wiringPi PWM test program\n") ;
12
13     if (wiringPiSetup () == -1)
14         exit (1) ;
15 }
```

```

16     pinMode (1, PWM_OUTPUT) ;
17
18     for (;;)
19     {
20         for (bright = 0 ; bright < 1024 ; ++bright)
21         {
22             pwmWrite (1, bright) ;
23             delay (1) ;
24         }
25
26         for (bright = 1023 ; bright >= 0 ; --bright)
27         {
28             pwmWrite (1, bright) ;
29             delay (1) ;
30         }
31     }
32
33     return 0 ;
34 }

```

Download: [pwm.c](#)

Note that in this example we only need to include the [WiringPi main header file](#). You must execute the binary on your Raspberry Pi using the `sudo` command.

How to use softServo to control a Servo

In this example we make use of the [WiringPi softservo.h](#) header to control a servo. **WiringPi** provides two basic functions to control servos:

- `softServoSetup (int p0, int p1, int p2, int p3, int p4, int p5, int p6, int p7)`. With this function we provide the number of pins to be used as controllers. For more information about the GPIO go to: [Raspberry Pi GPIO Pin Layout](#).
- `softServoWrite (int servoPin, int value)`. With this function we assign to a pin configured previously the value that we want to move. The values that support has a range of **-250 to 1250**. In order to understand this range, we must analyze how this function works: the function adds 1000 to the value that is passed as a parameter, so the final range is from 750 to 2.250 and the average stay in 1500, which is the default value that the library gives servo 90 degrees.

With this example we set a servo in its central position using the WiringPi pin 0 ([GPIO-17](#)) as a control signal. Place the following example source file inside an empty block of a new project, and execute the `bii find` ([command info](#)) to retrieve all dependencies.

The generated binary only works on your Raspberry Pi and must be run as `sudo` because it works on the hardware.

Example: servo.c

```
1 #include <stdio.h>
2 #include <errno.h>
3 #include <string.h>
4
5 #include <drogon/wiringpi/wiringPi/wiringPi.h>
6 #include <drogon/wiringpi/wiringPi/softServo.h>
7
8 int main ()
9 {
10     if (wiringPiSetup () == -1)
11     {
12         fprintf (stdout, "oops: %s\n", strerror (errno)) ;
13         return 1 ;
14     }
15
16     softServoSetup (0, 1, 2, 3, 4, 5, 6, 7) ;
17
18     softServoWrite (0, 500) ;
19
20     /*
21     softServoWrite (1, 1000) ;
22     softServoWrite (2, 1100) ;
23     softServoWrite (3, 1200) ;
24     softServoWrite (4, 1300) ;
25     softServoWrite (5, 1400) ;
26     softServoWrite (6, 1500) ;
27     softServoWrite (7, 2200) ;
28
29     */
30
31     for (;;)
32         delay (10) ;
33 }
```

Download: servo.c

3.5.2 HTTP Server: how to control a led by web

This example use the *HTTP Server* library to control a LED.

How can i use it?

- Just copy the following files to a new block.
- Find the dependencies and execute your code:

Once you have the code, invoke `bii find` to resolve external dependencies. Then, build and run in your Raspberry Pi as usual. Remember, the generated binary only work on your Raspberry Pi and have to run as `sudo` because it works on the hardware:

```
$ bii find
...
$ bii configure --toolchain=rpi
...
$ bii build
...
$ bii rpi:send
...
$ bii rpi:ssh
...
pi@raspberrypi ~ $ cd [project_name]
pi@raspberrypi ~/[project_name] $ ls
[binary_name]
pi@raspberrypi ~/[project_name] $ sudo ./[binary_name]
```

- Open your web browser and go to `http://localhost:9000`

Led Mode (on/off):

☐ On

☐ Off

Enviar consulta

main_server.cpp

This file just instances the server and run it with a simple configuration parameters.

```
1 #include "lasote/httpserver/http_server.h"
2 #include "my_http_middle_ware.h"
3
4 using namespace httpserver;
5 using namespace gip;
6
7 int main() {
8     MyHttpMiddleware my_middleware;
9     HttpServerConf conf(9000, 300, 60, 5);
10
11     HttpServer http_server;
12
13     http_server.run(&my_middleware, &conf);
```

```
14
15     return 0;
16
17 }
```

my_http_middle_ware.h

Defines your HttpMiddleware subclass.

```
1  #pragma once
2
3  #include "lasote/httpserver/http_middleware.h"
4  #include "led.h"
5
6  namespace httpserver {
7
8      class MyHttpMiddleware : public HttpMiddleware {
9      public:
10         MyHttpMiddleware() : HttpMiddleware(NULL) {}
11         MyHttpMiddleware(HttpMiddleware* other_middleware) : HttpM
12
13         }
14         virtual ~MyHttpMiddleware();
15         virtual void call(Request&, Response&);
16     private:
17         Led myLed;
18     };
19
20 } /* namespace httpserver */
```

my_http_middle_ware.cpp

Implements HttpMiddleware subclass. With the call method you can turn on or turn off a LED using the request info of the request.

```
1  #include "my_http_middle_ware.h"
2  #include "lasote/httpserver/exception.h"
3  #include "lasote/httpserver/model/method.h"
4  #include "sstream"
5  #include "iostream"
6
7  namespace httpserver {
8
9      MyHttpMiddleware::~MyHttpMiddleware() {
10
```

```

11     }
12
13     void MyHttpMiddleware::call(Request& request, Response& response){
14         ostreamstream html;
15         string checkedOn = "", checkedOff = "";
16
17         html << "<!DOCTYPE html>\n<html>\n<body>\n";
18
19         if(request.get("mode") == "on"){ myLed.on(); checkedOn = "checked";}
20         if(request.get("mode") == "off"){ myLed.off(); checkedOff = "checked";}
21
22         //Build the html form
23         string form;
24         form = "\
25             <form name='formulary' action='/hello' method='post'\
26                 Led Mode (on/off): <br>\n\
27                 <input type='radio' name='mode' value='on' >on\
28                 <input type='radio' name='mode' value='off' >off\
29                 <input type='submit' value='Submit' />\n\
30             </form>\n\
31             ";
32
33         html << form << "</body>\n</html>\n";
34
35         // Set content type we are printing
36         response.content_type("text/html");
37         // Set the body
38         response.body = html.str();
39     }
40
41
42
43 } /* namespace httpserver */

```

led.h

Defines a Led class for turn on/off the light.

```

1 #pragma once
2
3 #include <drogon/wiringpi/wiringpi/wiringpi.h>
4
5 class Led
6 {
7     public:
8         Led(); //default constructor

```

```
9         virtual ~Led(); //default virtual destructor
10        void on();
11        void off();
12
13    private:
14        int pin;
15    };
```

led.cpp

Implements the Led class

```
1 #include "led.h"
2
3 Led::Led()
4 {
5     pin = 0;
6     wiringPiSetup () ;
7     pinMode (pin, OUTPUT) ;
8 }
9
10 Led::~~Led()
11 {
12
13 }
14
15 void Led::on()
16 {
17     digitalWrite (pin, HIGH);
18 }
19
20 void Led::off()
21 {
22     digitalWrite (pin, LOW);
23 }
```

Download: [httpserver.zip](http://server.zip)

3.5.3 A funny moving doll with Raspberry Pi and biicode

Surprise your friends and family with a **moving doll**, just follow these few simple steps.

You just need paper, scissors, a servo, a Raspberry Pi and biicode!

- Sing up to [biicode](#)
- *Install* biicode in a few easy steps.
- Get [here](#) all the info you need to use your Raspberry Pi with biicode.
- To move the servo, just use the *WiringPi library*, ready to be used at biicode.

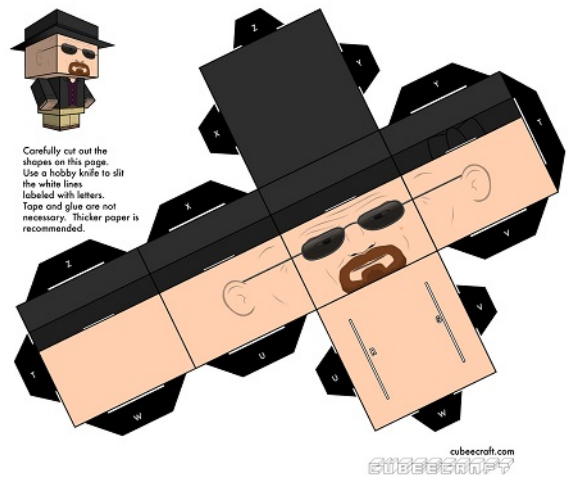
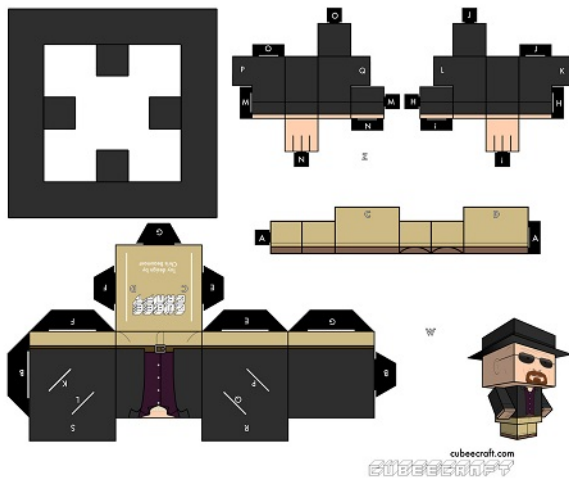
```

1 #include <stdio.h>
2 #include <errno.h>
3 #include <string.h>
4
5 #include <drogon/wiringpi/wiringpi/wiringpi.h>
6 #include <drogon/wiringpi/wiringpi/softservo.h>
7
8 int main ()
9 {
10     if (wiringPiSetup () == -1)
11     {
12         fprintf (stdout, "oops: %s\n", strerror (errno));
13         return 1 ;
14     }
15
16     softServoSetup (0, 1, 2, 3, 4, 5, 6, 7) ;
17
18     softServoWrite (0, 500);
19
20     int range = 500;
21     int vel = 10;
22
23     for (;;) {
24         softServoWrite (0, range);
25         range += vel;
26         if (range > 1250 || range < -250)
27             vel = -vel;
28
29         delay (10);
30     }
31 }
```

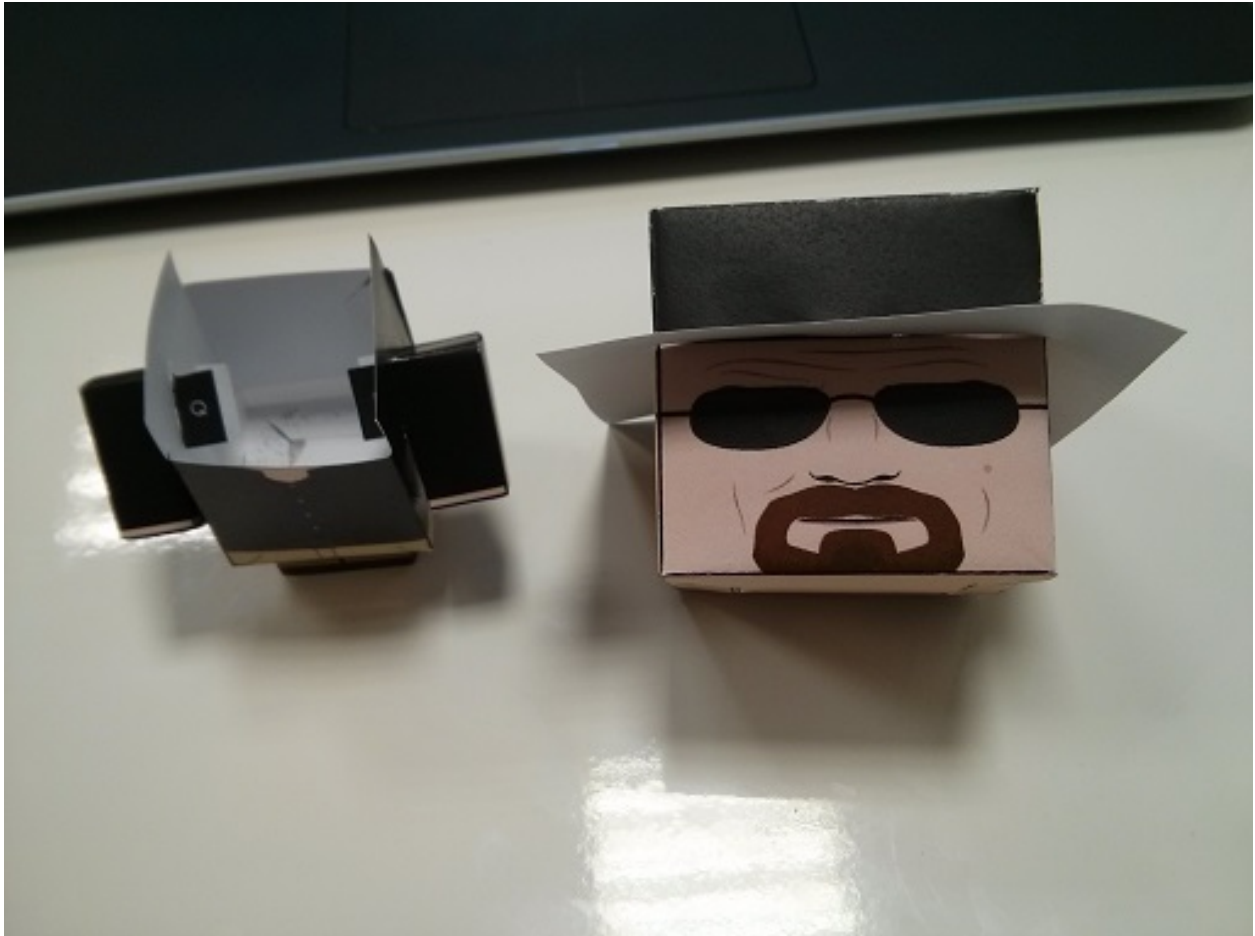
Download: `main.cpp`

Choose the paper doll you like most

As fans of the TV Show we chose to move [Heisenberg](#) paper doll for our little experiment. Feel free to be creative making your own doll:



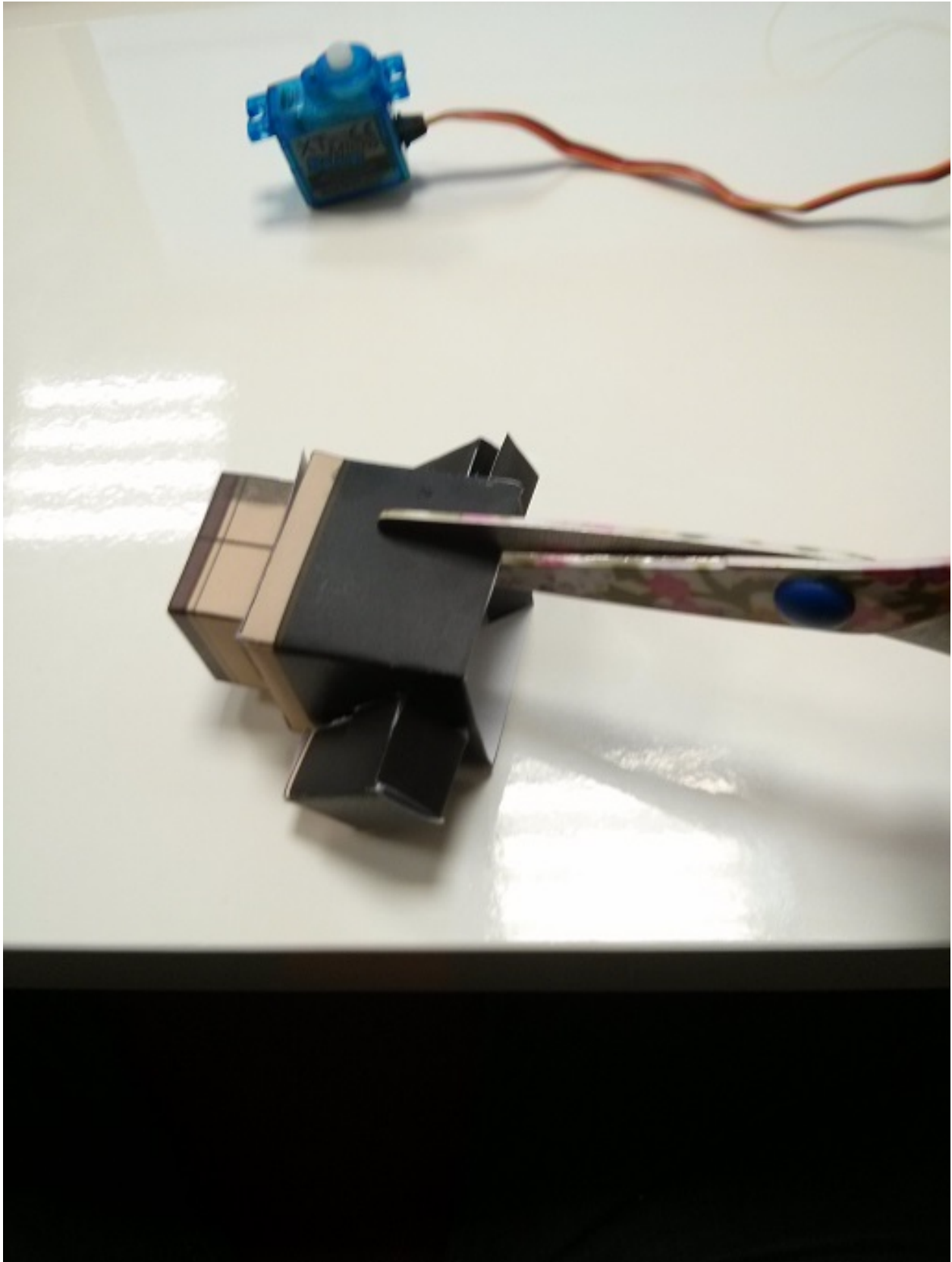
Putting it all together!



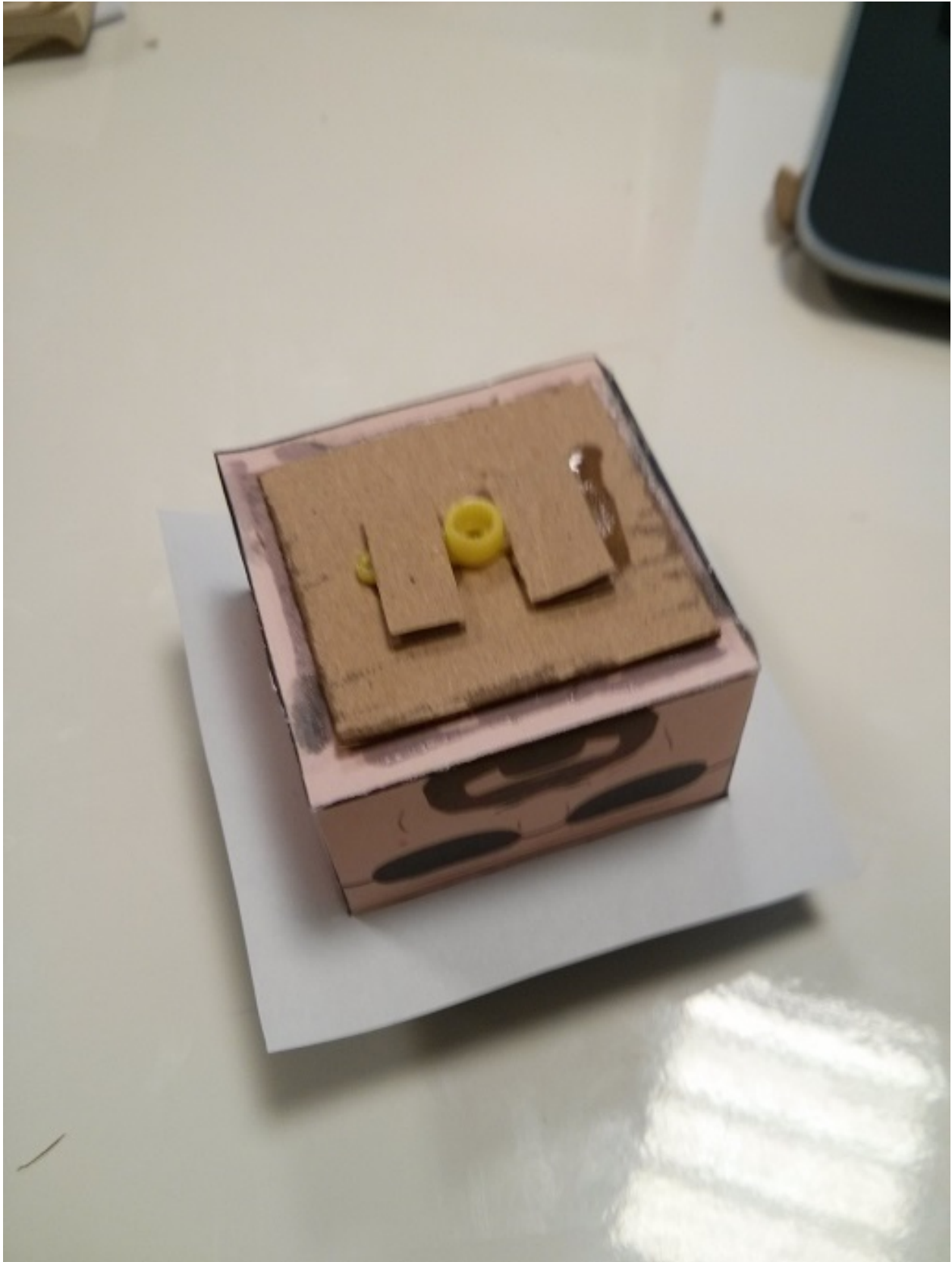
Stick the head to the servo and put the servo in the body

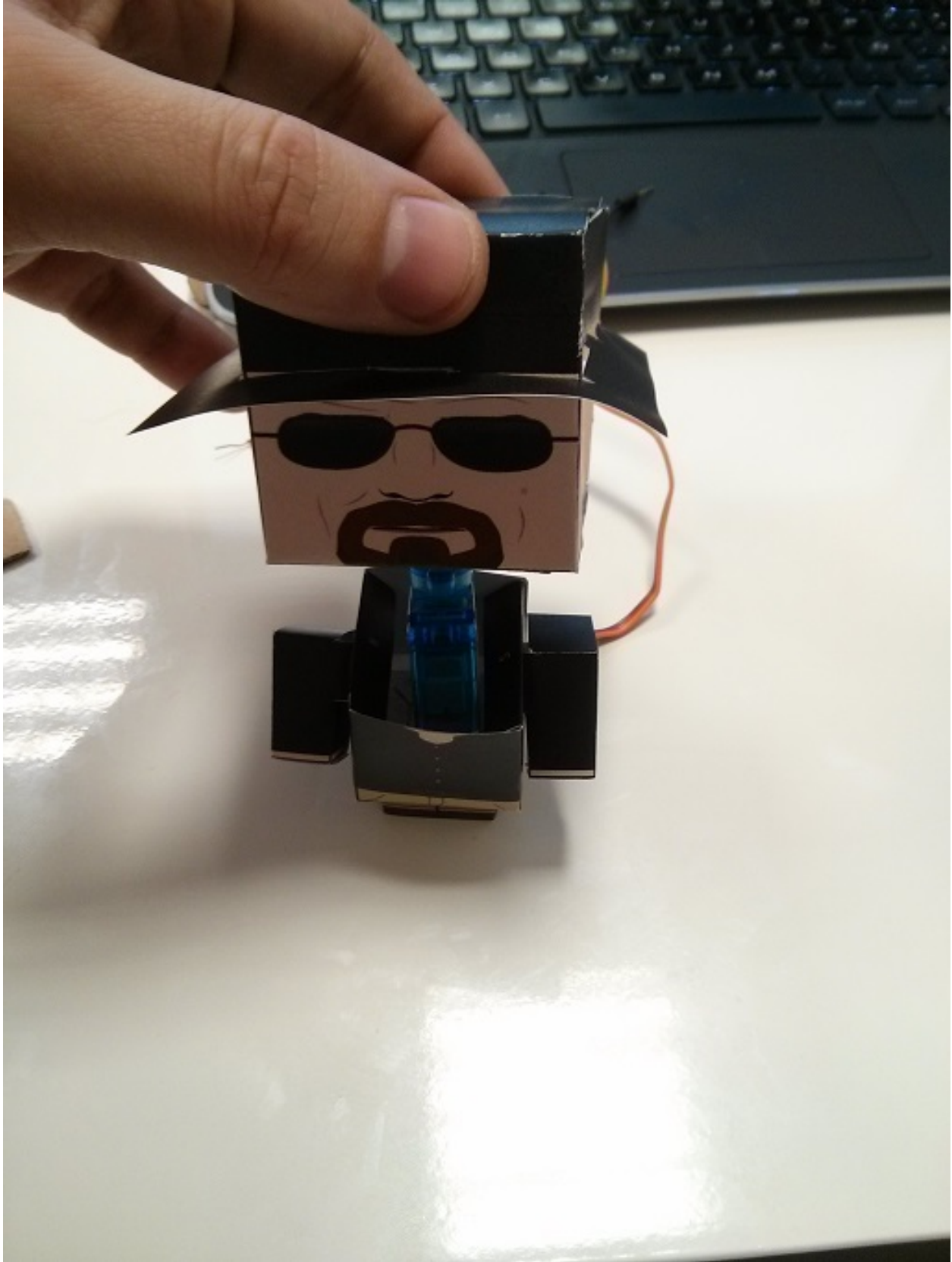
In the following pictures you can see how we built our doll:

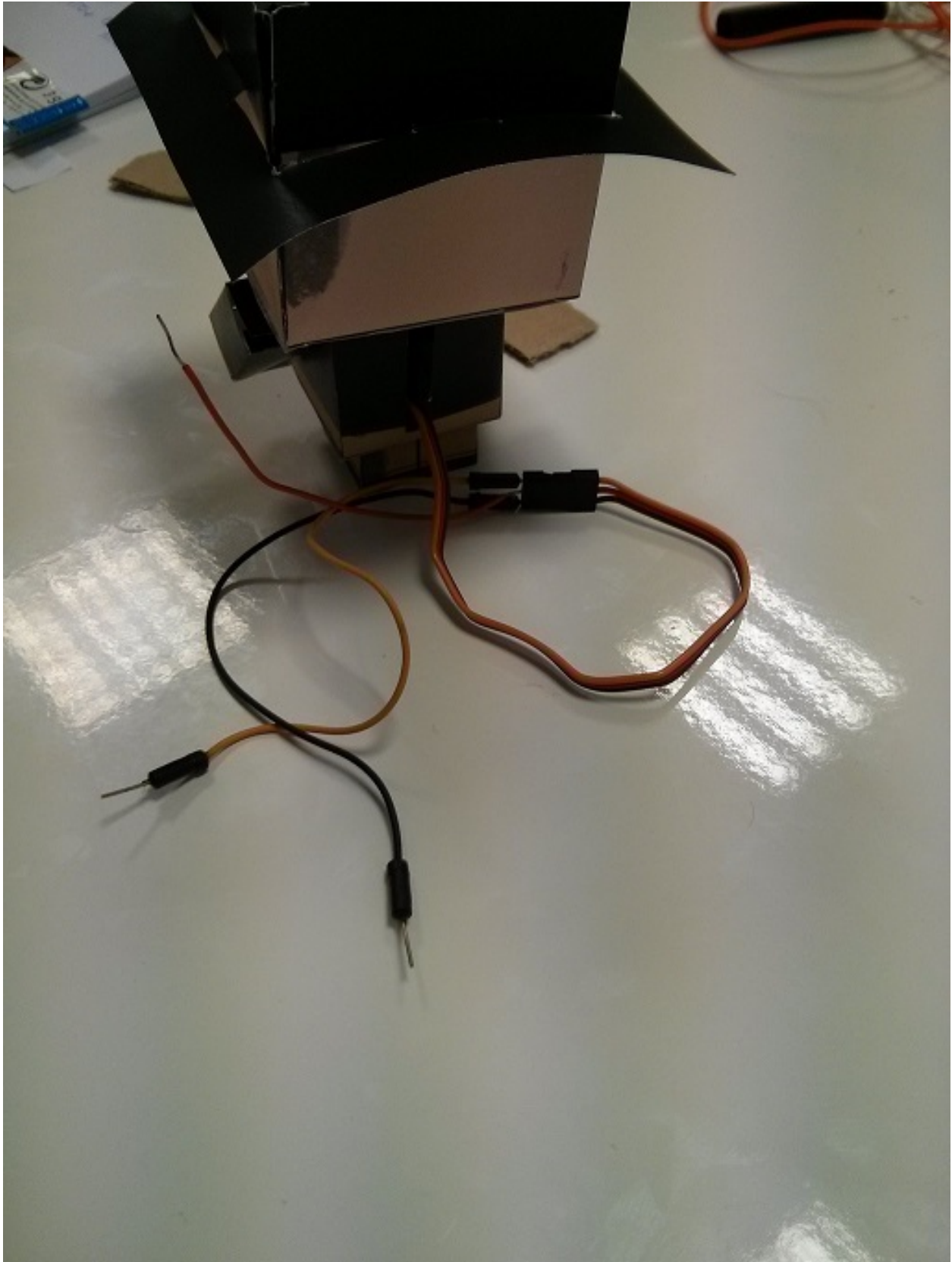








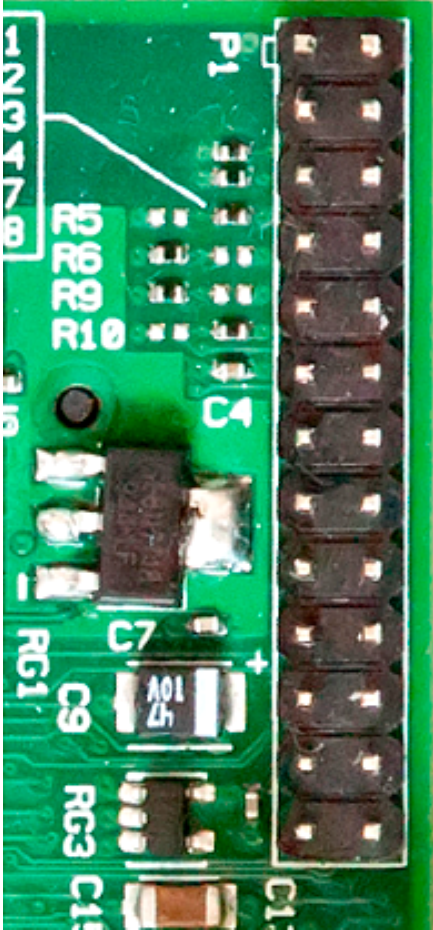

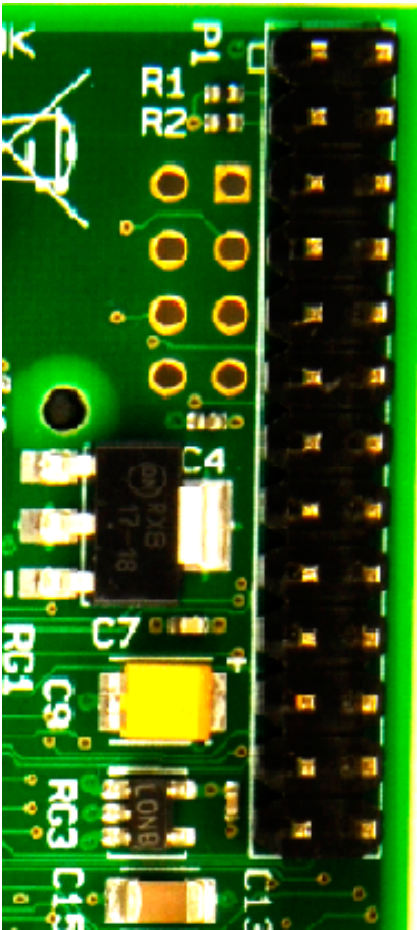


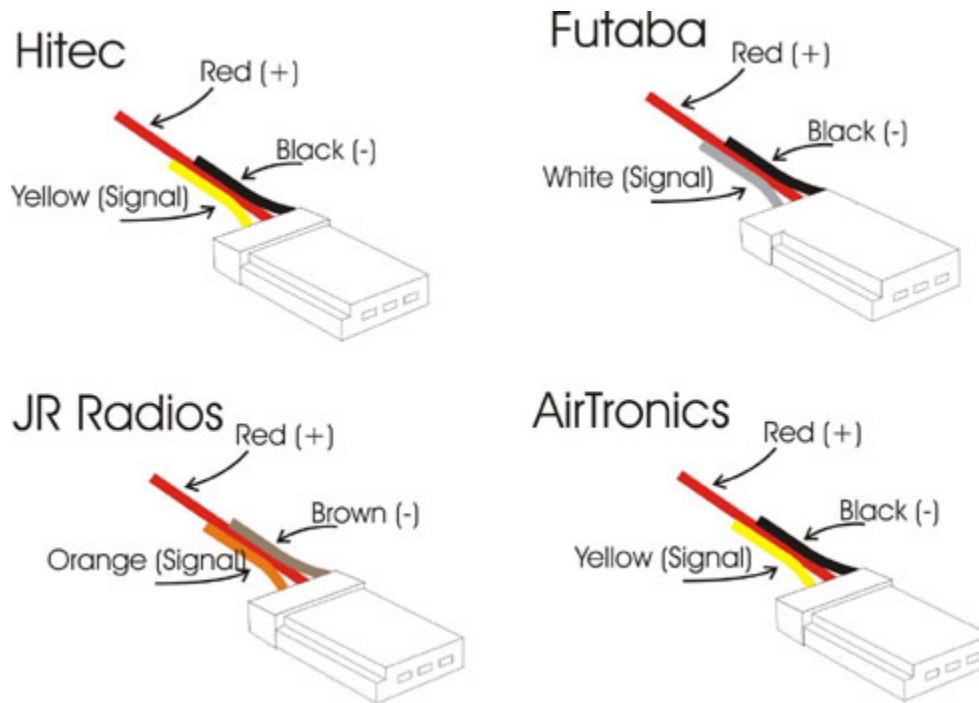


Connect the servo to the 5v, GPIO17 and 0v pins

If you need more information about the GPIO Reference *follow this link*.

Signal	GPIO17
+	5v
-	0v

GPIO Rev.1	GPIO Pin Layout	GPIO Rev.2
		



Have fun with the moving doll!

Now that your doll is moving, share it with your friends and family, make them laugh. We would also be happy to see other videos online. **Feel free to show us your most creative doll :)**

3.6 Troubleshooting

In this section you will find information and helpful resources, in case you encounter any problem while using our service. Don't panic, and try to find a solution for your problem bellow. If you can't find a solution, please [contact us at our forum](#) and describe your problem.

3.6.1 Is it possible to change the version of gcc used for cross-compiling to the Raspberry Pi?

Sure! Check the docs about how to use a [custom toolchain](#).

Check our [forum](#) and [Stackoverflow tag](#) for questions and answers.

Node.js

Node.js support is **experimental**. We're playing and trying to fit our dependency management model developed for C/C++ to the node.js ecosystem. If you're curious about our technology you can easily use it, but for production environments and real development we recommend to use the standard approach with NPM package manager. Also, we're open to suggestions and feedback, so if you have some ideas, please [tell us!](#)

4.1 Getting started

This example shows how to install biicode and code a node.js redis client. You don't need to have installed redis, biicode will download for you,

4.1.1 1. Installing biicode and node.js

First and install biicode

Then, download and install on your system the appropriate [version of Node.js](#) .

4.1.2 2. Create your project

First, create a project:

```
~$ bii init myproject
```

Then we can use the convenience `new` command to create some folders and a "Hello World" Node.js main file. Of course, you can do it manually too.

```
~$ cd myproject  
~/myproject$ bii new myuser/myblock --hello=node
```

You can directly type `myuser`, there's no need to register an account to use biicode, only to upload and share contents. You can use other name too. If you have already registered you might want to replace `myuser` with your real biicode username.

This should be the resulting layout:

```
+-- myproject
|   +-- biicode
|   +-- blocks
|       +-- myuser
|           +-- myblock
|               +-- main.js
|   +-- deps
```

4.1.3 3. Run your program

Lets check that everything is fine by running the hello world application.

```
~/myproject/$ cd blocks
~/myproject/blocks$ noderunner
myuser/myblock/main.js
```

4.1.4 4. Depending on redis

Copy the following code containing a simple redis client into the **main.js** file:

main.js

```
var redis = require("mranney/node_redis");
client = redis.createClient();
client.set("str key", "str val", redis.print);
client.quit(function (err, res) {
  console.log("Exiting from quit command.");
});
```

```
~/myproject$ biicode find
```

Run a redis server. If you need to install redis,

```
~/myproject$ redis server
```

Now, execute your script `main.js`.

```
~/myproject/$ cd blocks
~/myproject/blocks$ noderunner myuser/myblock/main.js
```

```
Reply: OK
Exiting from quit command.
```

That's it, if you see that output redis was downloaded and configured in your project! You can check the `deps` folder, the redis code is there.

Didn't work? No problem, read or contact us in

Any suggestion or feedback? It is very welcomed :)

4.2 How to

4.2.1 Run your node programs

You can run your scripts as usual. However, biicode has a script to execute node.js programs, just execute `noderunner <username/block_name>/<script_name>` inside the `blocks` folder:

```
~/PROJECT_DIR$ cd blocks
~/PROJECT_DIR/blocks$ noderunner <username/block_name>/<script_name>
```