

# Blackadder Installation

---

## Introduction

This manual page describes the steps to get, compile and run Blackadder, and all complementary components, like the Topology Manager (TM), in a single machine as well as in a testbed. This manual assumes a clean installation of the latest Debian distribution although it should be accurate for other Debian-like distributions. For the purpose of this manual, we have used Click version 2.0.x and Blackadder release version 0.2.

## Compiling and Running Core Components

Note that some of the commands need to be run with root privileges by using, e.g., **sudo**!

### Installing Click

Get Click from GitHub: *git clone https://github.com/kohler/click*

Note that although Blackadder v0.2 should work also with the Click 2.0.1 codebase from September 2011, you might experience package installation problems if you choose to use Click 2.0.1. Therefore it's recommended to install a more recent Click version (e.g., from January 2012 or later) instead.

Configure Click with kernel support. We assume that a compatible (or a patched kernel) is already installed. All header files are installed in `/usr/src/linux-headers`.

`./configure --with-linux=/usr/src/linux-headers --prefix=/path/to/binaries --with-linux-map=/boot/System.map-version`

Use additional flags to alter the configuration (e.g. `--enable-multithread`, `--enable-user-multithread`).

Compile and install Click: *(sudo) make install*

By default many useless (for Blackadder) Click packages will be compiled and linked with Click resulting to a large library (and Click module if supported). To avoid that use the `mkminidriver` tool (<http://read.cs.ucla.edu/click/docs/click-mkminidriver>) or manually delete the elements that are not required before compiling Click.

### Installing Blackadder

*apt-get install autoconf*

If kernel support is required add the following Netlink protocol in `/path/to/binaries/include/click-linuxmodule/include1/linux/netlink.h`

```
#define NETLINK_BADDER    20
```

In `~/src` folder:

*--run autoconf*

*--run ./configure --with-click=/path/to/binaries --prefix=/path/to/binaries*

*--run (sudo) make install*

HTML-based documentation will be created by running `doxygen` in the root Blackadder folder.

## File Structure

*/path/to/binaries/bin/*: all user-space Click-related tools as well as Click executable.

*/path/to/binaries/sbin/*: all kernel-space tools for starting (*click-install*) and stopping (*click-uninstall*) Click.

*/path/to/binaries/lib/*: all Click-related libraries and all user (*.uo*) and kernel (*.ko*) objects for the installed packages, like Blackadder.

## Running Blackadder

To run Blackadder just run Click with a configuration file describing a valid Blackadder node architecture. Check the *sample.conf* in *~/src* for an example Blackadder configuration. Running this instance will result in a Blackadder node that is capable of Inter-Process Information-centric communication (if you use the *NODE\_LOCAL* strategy in the API). *LINK\_LOCAL* strategy requires at least two nodes and *DOMAIN\_LOCAL* strategy requires the Topology Manager to run.

For user-space run: */path/to/binaries/bin/click configuration.conf*

For kernel-space run: */path/to/binaries/sbin/click-install -u configuration.conf*

## Installing Blackadder user library

First install *libtool* if you don't already have it:

```
$ sudo apt-get install libtool
```

Then go to the *lib* directory.

If needed, you can regenerate the *configure* file and some other files, such as *Makefile.ins* in each subdirectory, by first running *autoreconf*. Note that this requires that *autoconf* and *automake* (and *m4*) have been installed on your system.

```
$ sudo apt-get install autoconf
```

```
$ sudo apt-get install automake
```

```
$ autoreconf -i
```

After the optional *autoreconf* step, run *./configure*. The default installation locations are */usr/local/include* and */usr/local/lib*, but the */usr/local* prefix can be changed by giving a different path with the *--prefix* parameter (e.g.: *--prefix=/path/to*).

```
$ ./configure
```

Then compile the library and install the library and header files:

```
$ make
```

```
$ sudo make install
```

Both a shared (*libblackadder.la* and *libblackadder.so.\**) and a static library (*libblackadder.a*) are generated. The library is linked with applications by specifying *-lblackadder* as a linker option to (e.g.) *g++*. Also *-lpthread* is normally needed. In case you need to do static linking, add *-static* as an

option (and if you want to completely skip installing the library in this case, you can give the *lib/.libs* directory as an *-L* option when you compile the program).

The header files that are normally used in C++ programs are *blackadder.hpp* (that implements blocking event handling) and *nb\_blackadder.hpp* (for non-blocking event handling).

## Installing SWIG Bindings for Python, Ruby and Java

By default, only the C++ API is included when the library is compiled and installed (see above). Bindings for “higher level” languages, such as Python, Ruby and Java, can be generated with SWIG (*Simplified Wrapper and Interface Generator*) and installed along with the library. Support for Ruby and Java should be considered experimental.

Begin by installing SWIG if you don’t already have it:

```
sudo apt-get install swig
```

Enable additional languages by passing one or more *--enable-LANGUAGE* options to the *configure* script in the *lib* directory. Run *./configure --help* to see which languages are supported. Don’t forget to include any other parameters you need (e.g., *--prefix*). After this, run or re-run *make* and *make install* to install everything according to the configuration. Example:

```
$ ./configure --enable-python --enable-ruby --enable-java
$ make
$ sudo make install
```

Important note: You also need to have additional language-specific development files (headers) installed as a prerequisite to compiling the bindings. You can install these packages with *apt-get* as usual. For Python, the *python-dev* package is needed. For Ruby, *ruby-dev* is required. For Java, we recommend *openjdk-6-jdk*, but the bindings work with *sun-java6-jdk* as well (if *jni.h* for both is found in the system, the one for OpenJDK is used by default).

In order to run applications that use the Blackadder library, you may need to specify the location of the library to the compiler and/or interpreter. For Python, set the *PYTHONPATH* environment variable to point to the installation directory (usually the *site-packages* subdirectory). For Ruby, set the *RUBYLIB* variable. For Java, set the *CLASSPATH* variable when compiling and running the program, and also specify the *-Djava.library.path=...* parameter to *java* when running it (even if you installed the library into */usr/local/lib*).

## Installing the C bindings

Similarly as for the SWIG-based bindings, first run *./configure --enable-c* and then *make install* in the *lib* directory. This installs *libblackadder\_c.\** and *blackadder\_c.h*.

```
$ ./configure --enable-c
$ make
$ sudo make install
```

## Installing the Java Bindings

An alternative to the SWIG-based Java bindings is located in *{blackadder-dir}/lib/bindings/java-binding*. This particular component offers a slight improvement in performance due to reduced

buffer copies between the Blackadder library and JVM when sending and receiving data packets, as well as a few classes that provide an object-oriented abstraction to the network functionalities. The bindings have been tested mainly with OpenJDK (which by default is assumed to be installed on the system), but the native interface should be compatible with Sun's Java VM as well (and can be compiled with Sun's *jni.h*).

To use this binding, first you need to build and install the Blackadder user library, following the instructions above. To build the Java sources, you need to have ant installed. In Debian/Ubuntu distributions, you can install ant via

```
sudo apt-get install ant
```

You'll also need to download the Apache Commons Codec library from <http://archive.apache.org/dist/commons/codec/binaries/commons-codec-1.4-bin.zip>, extract the commons-codec-1.4.jar file and copy it to the *java-binding/lib* directory. (Later versions of the library might also work.)

The Java binding consists of two parts: (i) the native part and (ii) Java code. First, you need to compile and build the native part. Go to the *{blackadder-dir}/lib/bindings/java-binding/jni* directory. Update the *Makefile* and make sure that the *JNI\_LOC* variable points to the location of your JVM header files. If you are unaware of the location of the header files, you may find them by running

```
(sudo) find / -name jni.h
```

Save the *Makefile* and run *make*. This will produce the object file named *eu\_pursuit\_client\_BlackadderWrapper.so*. Remember the location of this file as you will have to give it as input in your java applications. Once you have build the JNI part, go to *{blackadder-dir}/lib/bindings/java-binding/* and run "*ant*" to build the java sources. Classes are copied to the *bin* directory. By running "*ant dist*", you will create the respective .jar file, copied in the *dist* subdirectory. You can import this jar as a library to java applications.

If you want to change/update the Java binding itself, you can directly import it to Eclipse. The java-binding directory is an Eclipse project folder and should be easily imported to the IDE via "File → Import → Existing Projects into Workspace."

## Setting Up and Running the Topology Manager

The Topology Manager is a normal C++ application that accesses the network using the libraries produced by following the aforementioned steps. In order to compile the Topology Manager, iGraph and libconfig++ libraries are required.

Install both by running: *(sudo) apt-get install libigraph0 libigraph0-dev*

Compile the Topology Manager by running *make* in *~/TopologyManager* folder. The executable is called *tm*.

Run the Topology Manager using: *./tm topology.graphml*, *topology.graphml* is a standard format for defining a graph (a network topology in this case). This file is produced automatically by the deployment application, described below.

## Deploying Blackadder

The deployment utility is an external tool that reads a configuration file, the syntax of which is described below, and produces Click configuration files for all nodes in a testbed. It copies these files to the destination nodes and runs Click (in the mode defined in the configuration file). It also creates the .graphml file and runs the Topology Manager in the node declared in the configuration using this file that is copied there.

The deployment utility requires iGraph, libconfig++ and libtclap libraries. Install them by running:

```
(sudo) apt-get install libigraph0 libigraph0-dev libconfig++8 libconfig++8-dev libtclap-dev
```

## Configuration Options

### Mandatory Global Options

BLACKADDER\_ID\_LENGTH: the length of Scope IDs and Information IDs supported by Blackadder. Currently this parameter has to be configured separately in Blackadder at compile time (see *helper.h*).

LIPSIN\_ID\_LENGTH: the length of LinkIDs, internal Link IDs, and LIPSIN identifiers in bytes.

CLICK\_HOME: the absolute path where Click is installed.

WRITE\_CONF: the absolute path where the deployment utility will remotely copy the Click/Blackadder configurations in each Blackadder node. The same is going to be used to remotely copy the produced topology.graphML file at the network node that will run the Topology Manager.

USER: The username of the user that will be used when ssh-ing network nodes (for retrieving mac addresses and running Click) and copying configuration files.

SUDO: True if the deployment utility will use *sudo* when remotely executing commands to network nodes.

OVERLAY\_MODE: The mode in which Blackadder will run at the testbed. Currently, Blackadder can run on top of Ethernet ("mac") or Raw IP Sockets ("ip").

### Defining a Network

A network is defined in the configuration file as follows:

```
network = {  
    nodes = (  
        { ...node1  
        },  
        { ...node2  
        }  
    );  
};
```

A configuration file can currently store a single network. The abovementioned global parameters are valid for the whole network (including all nodes and all connections).

### Defining a Network Node

A network node is defined in the configuration file as follows:

```
{  
    testbed_ip = "";  
    running_mode = "";  
    label = "";  
    role = ["", ""];  
    connections = (  
        { ...node1  
        },  
        { ...node2  
        }  
    );  
};
```

```

{
    ... connection 1
},
{
    ... connection 2
}
);
}

```

testbed\_ip: The IP address (in dotted decimal format) to which the deployment utility will copy the configuration files and remotely execute commands.

running\_mode: The mode in which Blackadder will run in this node. Use “user” for user-space (as user-space process) or “kernel” for kernel space (as a Linux module). Note that in a testbed some nodes may run in user-space while others run in kernel-space.

label: The Label of that network node. The label is used when sending requests to the Rendezvous Node. The Topology Manager also keeps track of the nodes in the network using their labels. The size of the label must be `BLACKADDER_ID_LENGTH` bytes.

role: if omitted or `role[]` then the network node has no special functionality. Use `role[“RV”,“TM”]` if the node is the Rendezvous Node and the Topology Manager or use the above keywords separately to place the (extra) functionalities to different nodes.

### Defining a Network Connection

A network connection is always unidirectional and is defined within the context of a network node as follows:

```

{
    to = "00000002";
    src_ip = "10.0.1.18";
    dst_ip = "10.0.1.19";
}

```

to: the destination node (its label)

If the network runs on top of IP:

src\_ip: the source IP address that will be used when sending to the Raw IP Socket.

dst\_ip: the destination IP address that will be used when sending to the Raw IP Socket.

If the network runs on top of Ethernet:

src\_if: the network interface from which publications will be sent. The deployment will use this and remotely acquire the respective MAC address. E.g. use “eth0” or “tap0” and Blackadder will use a physical interface or a virtual one (probably over a VPN), respectively.

dst\_if: the network interface to which publications will be sent. The deployment will use this and remotely acquire the respective MAC address.

src\_mac: the MAC address of the source network interface. This parameter is optional and if given, the deployment script will not acquire the MAC address remotely. Helpful when the network contains nodes that will not be online at the time the deployment utility will run.

dst\_mac: the MAC address of the destination network interface. This parameter is optional and if given, the deployment script will not acquire the MAC address remotely.

For the exact syntax check the sample\_topology.cfg file in the deployment folder as well as the configuration file grammar in <http://www.hyperrealm.com/libconfig/>.

## Running the Deployment Utility

The Deployment Utility accepts flags to perform a number of different operations. Usage:

```
./deploy [-t <string>] [-d <string>] -c <string> [-m] [-a] [--]  
          [--version] [-h]
```

Where:

-t <string>, --tgzfile <string>

tar gzipped file that gets transferred and extracted at user home folders on all experiment targets. The experiment targets are read from the -c <file>

-d <string>, --experimentfile <string>

Experiment description deployment file that contains information to remotely deploy applications and locally collect their STDOUT (Experimental Feature).

-c <string>, --configfile <string>

(required field, the tool will complain without it)

Configuration file that contains the node descriptions OR describes connections as well (when -a is not used)

-m, --montoolstub

Enable Java monitor tool stub in the Click configuration files. This injects counters in click configs that are inspected at runtime via port 55555. It is automatically omitted when kernel versions are described in the configuration files.

-a, --auto

Enable graph autogeneration - an autogenerated.cfg and edgevertices.cfg files are emitted at WRITE\_CONF folder. The former contains the autogenerated graph to repeat the experiment and the later the leaf nodes of that graph.

-h, --help

Displays usage information and exits.

## Deploying BlackAdder on PlanetLab

### Software Versions

Until recently (September 2011), the Click modular router did not support properly the PlanetLab Sliver port registration mechanism, but Click 2.0.1 and later versions support this. If you are not sure about the version you have you can check the developers' comment that is located in the top of the *elements/userlevel/rawsocket.cc* file in your Click distribution. If it has a note from 2011 about the port registration mechanism update then your Click distribution is up-to-date.

## Building Instructions for PlanetLab

### Building Blackadder

PlanetLab allocates slivers on PlanetLab nodes for your account. You can log in to these nodes using passwordless SSH with your user account name. Your Click binary on PlanetLab can only be executed on your account home folder of each allocated sliver, e.g: */home/certhple\_purs1*.

So when you build Click and Blackadder for the PlanetLab target, create the same folder pattern on your development host and use the appropriate `--prefix` command when configuring Click and Blackadder. In our example we would create */home/certhple\_purs1* on our development host and add `--prefix=/home/certhple_purs1` when executing the *configure* script.

### Building Blackadder TM

PlanetLab nodes have the Fedora Core 8 Linux distribution with the *yum* package manager.

Blackadder's C++-based TM will need *igraph* that is available as a package and can be installed with *yum* on the node that you intend to use as the TM node.

Note: PlanetLab slivers are resource constrained. A TM that executes on a sliver can easily become the performance bottleneck, especially for large-scale experiments. It is advised that the TM runs on a dedicated server machine in our own network. All you need is a publicly accessible IP address and firewall configuration for an open UDP port.

## Deployment Utility Usage for PlanetLab

Blackadder's universal deployment tool has extended support for PlanetLab. It can do the following:

### ***i) Retrieve the allocated PlanetLab nodes directly from planet-lab.eu and output a node input file***

In the *deployment* folder you will find a Python script that does this: *planetlabgetnodes.py*

Edit this file, locate the Python dictionary that is depicted below and add your planet-lab.eu username and password:

```
auth = { 'AuthMethod' : 'password',
         'Username'   : '<planetlab user name>',
         'AuthString' : '<planetlab user passwd>',
       }
```

This file will access planet-lab.eu, retrieve the sliver hostnames that you have allocated, resolve their IP addresses and produce the so-called node input file with name *planetlab.cfg* which has the following format:

```
network = {
  nodes = (
    {
      testbed_ip = "193.63.75.18";
    },
    {
      testbed_ip = "128.232.103.201";
    },
    {
      testbed_ip = "195.148.124.73";
    }
  )
}
```



```
    );  
};
```

Before you invoke the deployment tool, the generated file should be enriched with the following information (which should be placed at the top of the file, also replace appropriate fields with what applies in your case):

```
BLACKADDER_ID_LENGTH = 8;  
LIPSIN_ID_LENGTH = 8;  
CLICK_HOME = "/home/certhple_purs1/";  
WRITE_CONF = "/tmp/";  
USER = "certhple_purs1";  
SUDO = true;  
OVERLAY_MODE = "ip";
```

### ***ii) Transfer a .tar.gz bundle and automatically decompress it at home folders on all slivers***

In case you need to transfer Click, Blackadder and other application binaries, you can appropriately create a .tar.gz bundle of them. Keep in mind that it has to be a .tar.gz because tar zxvf is the command that will be issued remotely for decompression and everything gets transferred and decompressed at the home folders on the PlanetLab slivers, so the home folder will be the root reference. The deployment tool accepts a flag, *filename* as follows: *-t <tgzfile>* and commences file transfer and remote decompression before it deploys the blackadder instances. This mode of operation is, of course, optional during deployment.

### ***iii) Autogenerate graphs from a node input file using the Barabasi-Albert model for scale-free networks. The autogenerated graph is saved in standard deployment tool configuration file format and can be re-used in experiments.***

Doing this is optional and depends on the experiment needs. The user may directly define a standard configuration file format and use that on PlanetLab, rather than following this approach. If you have allocated many PlanetLab nodes and you need an autogenerated graph then you may use this approach. For this to work, the deployment tool expects a node file format as described in (i). This file contains all the PlanetLab nodes along with global configurations. The deployment tool should be invoked with the *-c <node configfile> -a* flag combination. *-c* designates the configuration file name and *-a* lets the tool know that the configuration file format has plain nodes without connections and that the tool should use Albert-Barabasi model to define connections.

The tool finds the node that has the least amount of total hops from the rest of the nodes on the autogenerated graph and sets this as the Rendezvous and TM node. You can always change that in the autogenerated file.

### ***iv) Build and distribute Click configuration files as well as remotely start Blackadder instances in testbed nodes***

This is typical behavior in a testbed that the tool will always do that as a result of any proper configuration. The user does not have to define extra parameters for this.

## Running Example Applications

### Samples

Several sample applications are included in Blackadder's release.

In `~/examples/samples`, run `make` to compile these applications.

There are also simple examples for the SWIG-based Python, Ruby and Java bindings in language-specific directories under `examples/swig-samples`. Similar examples for the C binding can be found in the `examples/c-samples` directory.

### Video Streaming

#### C++ version

Run `make` in `~/examples/video_streaming`

VLC is required to run this example application.

Video Publisher: run `./video_publisher`

Using the console-based interface, advertise videos by typing the exact name of each video.

Advertise the catalogue of videos by typing `.` and then press enter.

Video Subscriber: run `./video_subscriber`

Upon initiation the subscriber subscribes to the video catalogue and receives it by the publisher.

Then, type the exact name of one of the advertise videos to join the video stream.

#### Java version

The Java version of the video streamer provides a graphical user interface for both the publishing and subscribing to video and media streams. Using the graphical version of the publisher, we can currently publish video files and audio files. We can also publish Internet media streams such as Internet radio and YouTube media (although some YouTube media will not currently work).

There are no particular requirements for running the demo although there are some things you'll need to do - I'll assume a blank install here on something like Debian:

1) Install VLC (luggage or later should be ok).

2) You will need to build the Java bindings (instructions are included in the Java bindings directory - `/BAv0.2/lib/bindings/java-binding`). Make sure you get the correct location of your `jni.h` file in the Makefile and then do a 'make'.

3) Edit the `project.properties` file and correct the path to your specific

"`java-binding/jni/eu_pursuit_client_BlackadderWrapper.o`" file.

4) You must add the necessary jar files to your lib folder if they are not already there.

- The Apache Commons Codec
  - <http://archive.apache.org/dist/commons/codec/binaries/commons-codec-1.5-bin.zip>
- The JNA libraries
  - <http://java.net/projects/jna/sources/svn/content/tags/3.3.0/jnalib/dist/jna.jar>
  - <http://java.net/projects/jna/sources/svn/content/tags/3.3.0/jnalib/dist/platform.jar>

- The VLCj jar
  - <http://vlcj.googlecode.com/files/vlcj-1.1.5.1.jar>

5) In the video demo directory, type 'ant' to build the project. You may need to edit the build.xml file and change line 9 to point to your v0.2 location.

6) You need to start blackadder. For example, by running "sudo click sample.conf" in the blackadder src directory.

7) You're now ready to run the app! The two commands are:

```
ant publisher
ant subscriber
```

Run the publisher first, add some videos and then run the subscriber. You will need to refresh the catalog whenever you change it in the publisher. The video should start when you click on subscribe - it should stop when you unsubscribe. If you then re-subscribe, it should start again from the beginning.

You should **not** subscribe to more than one video at a time (as a single user). To subscribe to another video, you should unsubscribe from the previous one first. Also, please note that this is a demo and is likely to crack under pressure....

## Voice over Blackadder

The Java-based voice application is located in *{blackadder\_dir}/examples/java-voice*. The application uses the non-SWIG Java binding, therefore, before running the application, you need to compile and build the respective binding following the instructions above.

Once you have the Java binding built, track the location of the binding object file (*eu\_pursuit\_client\_BlackadderWrapper.so*). Edit the *config.txt* file located in the java-voice directory and set the *JNI\_PATH* variable to point to the location on the Java binding's object file.

You also need to download and add jar files for *Visual Swing for Eclipse* into the *java-voice/lib* folder if they are not already there. The file to download and to extract is:

[http://visualswing4eclipse.googlecode.com/files/vs4e\\_0.9.12.l20090527-2200.zip](http://visualswing4eclipse.googlecode.com/files/vs4e_0.9.12.l20090527-2200.zip)

To build the Java sources just run "*ant*" (*ant* is also required to build the Java binding's Java sources). In case you modify the Java binding itself, make sure to copy the respective jar in the application's *lib* folder.

To start the application, first start the Blackadder network and then run

```
./run-application.sh -f conf_file
```

with "*conf\_file*" pointing to the location of your *config.txt*.

One the pop-up window, choose whether you want to make the call or wait for incoming calls. One of the two participants has to "Make call" (caller), providing his/her name and the callee's name as well. Then presses the "Call" button. The other participant awaits for the call (callee). Press "Receive call", enter you name (it has to match the callee name set by the caller) and accept the call.