# A Tutorial for GNUnet 0.9.x (Java version)

Florian Dold

April 17, 2013

## 1 Getting Started

### 1.1 Installing GNUnet

This tutorial assumes that you have GNUnet $\geq$ 0.9.5 installed on your system. Instructions on how to do this can be found at `https://gnunet.org/installation`, or in the C version of the GNUnet tutorial. Make sure that you run `configure` with the option `--enable-javaports`, in order to allow Java clients to connect to GNUnet services. Start your GNUnet peer with the command `gnunet-arm -s` and convince yourself that the default GNUnet services are running by typing `gnunet-arm -I`.

**Exercise:** Read the first three chapters of the GNUnet C tutorial, available at `https://gnunet.org/svn/gnunet/doc/gnunet-c-tutorial.pdf`.

### 1.2 Other Dependencies

Make sure that you have OpenJDK 7 or later installed on your system. GNUnet-Java uses Gradle for its build system, visit `http://www.gradle.org/` for installation instructions, and install version 1.5 or later.

**Exercise:** Execute the command `gradle --version` from your shell, and verify that you correctly installed the right version.

### 1.3 Installing GNUnet-Java

Check out the latest version of GNUnet-Java with subversion

```
$ svn checkout https://gnunet.org/svn/gnunet-java/
```

And compile the project with

```
$ cd gnunet-java
$ gradle assemble
```

To test whether your GNUnet-Java installation is working, run

```
$ ./bin/gnunet-nse
```

which should display the estimated current size of the network. If the program keeps running whithout producing any output, your GNUnet peer is probably not running.

Throughout the tutorial it will be useful to consult the Javadoc for GNUnet-Java, either build it yourself with

```
$ gradle javadoc # docs will be put in ./build-gradle/docs/
```

or use the online version at `https://gnunet.org/javadoc/`.

## 2 A simple GNUnet-Java extension

Check out the template directory for GNUnet-Java extensions

```
$ svn checkout https://gnunet.org/svn/gnunet-java-ext/
```

The extension template contains a build system and some example code.

Before you can compile the examples in the extension template, make sure you have the following environment variables set, so that the build system can find both the `gnunet-java.jar` and its dependencies

```
$ export GNJ_HOME=gnunet-java/gradle-build/libs # or wherever gnunet-java.jar is
$ export GNJ_DEPS=gnunet-java/lib # or wherever the dependencies of GNUnet-Java are
```

Build the example code in the template with

```
$ cd gnunet-java-ext
$ gradle assemble
```

and run

```
$ ./bin/gnunet-ext
```

which should print "`hi`" on your terminal.

### 2.1 The Basics

This is the most basic skeleton for a GNUnet-Java application:

```
import org.gnunet.util.*;
public class HelloGNUnet {
    public static void main(String[] args) {
            new Program(args) {
                public void run() {
                    System.out.println("Hello, GNUnet");
                }
            }.start();
}
```

Calling `start` initializes GNUnet-Java, parses the command line, loads configuration files and starts the task scheduler, with the code in the `run` method as initial task.

**Exercise:** Try to get the code above running. Place your code in the `src/` directory of the template and execute `gradle assemble`. Copy and modify the example shell-wrapper `bin/gnunet-ext` so that you can run your own program with it. Remember that the environment variables `GNJ_HOME` and `GNJ_DEPS` have to be set appropriately.

### 2.2 Adding and using command line arguments

Command line options are added by annotating members of your `org.gnunet.util.Program` subclass with the Option-annotation.

Here is a simple example:

```
new Program(args) {
    @Option(
        shortname = "n",
        longname = "name",
        action = OptionAction.STORE_STRING,
        description = "the name of the person you want to greet")
    String name;
[...]
}
```

You can now specify a value for the member `name` at the command line, either by the long name with two dashes (`--name=Foo` / `--name FOO`) or the short name (`-n FOO`) with one dash.

Inside of the `run` method, the field `name` will then be initialized with the passed argument, or `null` if the option has not been passed.

The Option annotation can not only be used with Strings, but also with booleans and numbers. These are a few of the available options:

- `STORE_STRING`: Store a string in a `String` variable
- `STORE_NUMBER`: Store a number in a member of primitive type
- `SET`: Set a `boolean` to `true`

By default, the following arguments are available on the command line:

- `-h` / `--help` shows the help text
- `-v` / `--version` shows version information
- `-c` / `--config` specify an additional configuration file to load
- `-L` / `--log` specify the log level
- `-l` / `--logfile` specify a file to write the logs to

You can change the about text and the version information by overriding the `getVersion` or `getAboutText` methods in your `Program` subclass.

**Exercise:** Add a few different command line options to your program and print them with `System.out`!

## 3 The statistics API

In this section we will use the statistics API of GNUnet-Java. The statistics service allows us to store numbers under a subsystem and a name, which are still available to you and other components of your peer after your program exits.

### 3.1 Establishing a connection with the statistics service

```
Statistics statistics = new Statistics(getConfiguration());
```

The Statistics constructor is called with the default configuration, provided by the method `getConfiguration` of the `Program` class. Calling the constructor establishes a connection to the statistics service. As with most API calls in GNUnet-Java, this operation is asynchronous. This is one of the main reasons why one has to wrap your program in the overridden `run` method of `Program`: Once all method calls are done, the run method returns, and GNUnet-Java keeps the system running until all work has been done.

Always remember that you always explicitly have to destroy your `Statistics` instance with the `destroy()` method. Otherwise there might be pending operations that prevent the termination of your program.

## 3.2 Setting statistics

You can use the newly created `statistics` handle like this to set a value:

```
statistics.set("gnunet-java-hello", "the␣answer", 42);
```

## 3.3 Retrieving statistics

Retrieving a value is a little bit more complex. Because of the asynchronous nature of the GNUnet-Java APIs, the `startGet` method does not directly return values, but a handle (implementing the interface `Cancelable`) to cancel the get request. The actual values are accessed by passing a callback object to the `startGet` method.

Example:

```
// the name parameter is the empty string, this gets all options of the specified subsystem
Cancelable getCancel = statistics.get(RelativeTime.SECOND, "gnunet-java-hello", "",
new Statistics.StatisticsReceiver {
    public void onDone() {
        System.out.println("everything␣done");
    }
    public void onReceive(String subsystem, String name, long val) {
        System.out.println(subsyste + "␣" + name + "␣" + val);
    }
    public void onTimeout() {
        System.out.println("timeout␣occured");
    }
});
```

**Exercise:** Write a program that sets statistics values, and check the result with the `gnunet-statistics` command line tool.

**Exercise:** Write a program to read and print statistics values.

# 4 The core API

The core API allows sending encrypted messages to connected peers.

## 4.1 Defining new Messages

All GNUnet messages follow a common format. Every message consists of a header (with the message size and the message type) and a body.

You can define a new type of nessage in GNUnet-Java by annotating a class with information on how to represent its members in binary format.

Additionaly, you have to register your new message type with GNUnet-Java, giving it a unique id. Here is an example:

```
@UnionCase(4242)
public class ExampleMessage implements GnunetMessage.Body {
    @UInt8
    public int age;
    @ZeroTerminatedString;
    public String name;
}
```

The `@UnionCase` annotation specifies the message id of the message body below (4242 in the example). GnunetMessage.Body is a union of messages, and ExampleMessage is one (new) member of the union.

Every time you add a new type of GNUnet message, you have to run the command

```
$ gradle msgtypes
```

This generates the file `src/org/gnunet/construct/MsgMap.txt`, which allows the system to load the right java class when reading a message from the network.

The above message then contains a value annotated with `@UInt8`: An **8**-bit **U**nsigned **int**eger. There are similar annotations for integers of other sizes, and `@Int`*N* annotations for signed integers. The second member is a String, whose binary representation appends a zero-byte to the string to mark its end.

Other useful annotations can be found in the package `org.gnunet.construct`. Among them are annotations for arrays of fixed or variable size (`@VariableSizeArray, @FixedSizeArray`), for embeding other messages in your message (`@NestedMessage` and for implementing your own message unions.

**Exercise:** Define a message that contains a 32-bit signed integer.

**Exercise:** Look at the class `org.gnunet.dht.MonitorPutMessage` in the GNUnet-Java source code. This message uses a variety of different annotations, try to understand the purpose of each member's annotation.

## 4.2 Connecting to Core

After creating a handle to core by calling the `Core` constructor, you have to specify what types of messages you are interested in. The core service will only send messages of these types to you, and only notify you of connecting peers if they share a subset of the messages you are interested in.

The `handleMessages` method allows you to specify an object of a class inheriting `Runabout`. The Runabout is a mechanism for single-argument multiple dispatch in Java. You have to define one `visit` method for every type of message you are interested in. Once `Core` receives a message, it is dispatched dynamically to the `visit` method with the appropriate signature. Note that every `visit` method, as well as the receiver's class, has to be public in order for the dynamic dispatch to work.

Example:

```
public class MyMessageReceiver extends Runabout {
    public visit(MyFooMessage m) { /* do something */ }
    public visit(MyBarMessage m  { /* do something else */ }
}
```

After specifing your message handler, the `init` method has to be called with a callback object. This starts the handshake with the core service, once done the callback object's `onInit` method will be called with your peer's identity.

### 4.3 Sending a message to another peer

Before you can actually send a message, you have to wait until the core service is ready to send your message. This is done by calling the `notifyTransmitReady` method. You have to provide a callback object to this method, whose `transmit` method is invoked with a `MessageSink` object once the core is ready to transmit your message. Call the `send` method in the `MessageSink` in order to finally transmit it.

Example:

```
// arguments: messagePriority, timeout, targetPeer, messageSize, transmitter
core.notifyTransmitReady(0, RelativeTime.FOREVER, myIdentity, 42, new MessageTransmitter() {
    public transmit(Connection.MessageSink sink) {
        sink.transmit(myMessage);
    }
    public onError() {
        // do something
    }
```

You can use `Construct.getSize` to calculate the size of a message, or just do it manually.

**Exercise:** Write an echo program for core: Send a message to the local peer and receive it!

## 5 Other useful APIs

Many of GNUnet's services are not yet available as a GNUnet-Java API.

Some other useful service APIs currently implemented are NSE (in `org.gnunet.nse.NetworkSizeEstimation`), a service that gives an estimation of the current size of the network, DHT (in `org.gnunet.dht.DistributedHashTable`), a service that allows key/value pairs to be stored distributed across the network, and PEERINFO (in `org.gnunet.peerinfo.PeerInfo`), a service for retrieving information about other known peers.

## 6 Writing your own client and service

GNUnet is split up into many components, every component runs in its own process. In the previous sections you have used existing APIs to interface with other services written in C. GNUnet-Java also provides the tools necessary to both directly interface with services yourself and write completely new services.

### 6.1 The service configuration

Each service has its own configuration, specifying basic information like the executable file of the service (used by ARM), the port or socket used to reach it, as well as configuation options specific to the service.

**Exercise:** Look at the configuration file for the example service `config/greeting.conf` and try to understand the meaning of each option, by looking at the comments and the source code of the example service.

### 6.2 Writing a client

The `org.gnunet.util.Client` class allows you to connect to a GNUnet service and exchange messages with it:

```
Client myClient = new Client("myservice", configuration);
```

In the above example, the configuration values for the clients are taken from the configuration section `myservice`.

Keep in mind that all configuration files either have to be in one of the default locations, or specified on the command line with the `-c CFGFILE` option.

## 6.3 Writing a service

To implement your own service, just inherit `org.gnunet.util.Service` instead of `org.gnunet.util.Program`. The main difference between `Program` and `Service` is that the `Service` also creates a `Server`, which waits for messages from clients. You can register a Runabout to receive messages from clients with `getServer().setHandler(myRunabout)`, in similar fashion to handling messages from `core`.

Example:

```
public class MyService {
    public static void main(String... argv) {
        new Service(
            "greeting", // name of the service, for chosing the right configuration
            RelativeTime.MINUTE, // timeout for disconnecting idle clients
            true, // disallow messages of unknown type
            argv) { // command line arguments parsed by Service

            @Override
            public void run() {
                getServer().setHandler(new Server.MessageRunabout() {
                    public void visit(MyMessage m) {
                        [...]
                        getSender().receiveDone();
                    }
                });
            }
        }.start();
    }
}
```

Always remember to call `getSender().receiveDone()`, as the server does not receive further messages until `receiveDone is called`, in order to support flow control. The object returned by `getSender()` has a `notifyTransmitReady` method, which can be used to send messages to clients in a similar fashion to writing to CORE.

**Exercise:** Look at the example service implemented in `org.gnunet.ext.GreetingService`. Run the service (`gnunet-service-greeting`), and connect to it with the client program (`gnunet-greeting`). Try to understand the code, and modify both the client and the service so that can send and accept another message type.

**Exercise:** Write an API for a GNUnet service that has not been implemented yet in gnunet-java and contribute it back to the project.

# 7 The state of GNUunet-Java

The GNUnet-Java project is still somewhat unstable and under development, expect changes that break your stuff! Please report any bugs or feature requests at https://gnunet.org/bugs/.