A Tutorial for GNUnet 0.10.x (Java version)

Florian Dold

June 4, 2014

Although GNUnet is primarily developed in the C programming language, it is also possible to write peer-to-peer applications with GNUnet in Java. The GNUnet-Java project provides bindings to a subset of existing GNUnet components, as well as the infrastructure to develop new components. GNUnet-Java is less stable and less complete than the C version. Please report any bugs or feature requests at `https://gnunet.org/bugs/`.

1 Getting Started

1.1 Installing GNUnet

This tutorial assumes that you have GNUnet $\geq$ 0.10.x installed on your machine. 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 to 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 6 or later installed on your system.

2 A simple GNUnet-Java extension

The simplest way to create a new GNUnet component in Java is to copy the template extension project, which already contains a build system and sample code.

Obtain the extension template from the subversion repository:

```
$ svn export https://gnunet.org/svn/gnunet-java-ext/ my-extension
$ cd my-extension
```

The template project uses Gradle[1] as a build system. Simply use the `./gradlew` script to run build tasks. The wrapper script will automatically download the correct version of Gradle on the first run. Alternatively, download Gradle$\geq$1.11 yourself, and use the `gradle` command directly instead of the wrapper.

Build the template project by running the `assemble` task:

[1] `http://gradle.org`

```
$ ./gradlew assemble
```

This will download all direct and transitive dependencies. Gradle stores all dependencies in an internal cache. Run

```
$ ./gradlew copyDeps
```

in order to copy all dependencies into the `lib/` folder[2].

Check if you're on the right track by running the Java client for GNUnet's network size estimation service:

```
$ java -cp 'lib/*' org.gnunet.nse.NetworkSizeEstimation
```

This should print something like `est: 42.3 dev: 3.14 t: Sun Apr 06 23:40:37 CEST 2014` to your terminal. If the program hangs, check if your peer is running and correctly configured.

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() {
                public void run() {
                    System.out.println("Hello,␣GNUnet");
                }
            }.start(args);
}
```

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

Exercise: Get the code above to execute by placing it in a `*.java` file in `src/main/java/`, running the `assemble` task from Gradle, and invoking `java` with the right parameters. The source code in the extension template should follow the the Maven Standard Directory Layout[a] for application code, tests and resources.

[a] https://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html

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 Argument-annotation.

Here is a simple example:

```
import org.gnunet.util.getopt.*;
[...]
new Program(args) {
    @Argument(
        shortname = "n",
        longname = "name",
```

[2] This is not strictly necessary—you can also ask Gradle for the classpath pointing to the internal cache. However, having all dependencies in one folder is more convenient, especially when using shell wrappers for Java entry points later on.

```
        action = Argument.STORE_STRING,
        description = "the␣name␣of␣the␣person␣you␣want␣to␣greet")
    String name = "Jane␣Doe"; // default value if option is missing
[...]
}
```

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.

Before the `run` method is called, the field `name` will be set to the given argument. If the `name` option is missing, the field will keep the value specified in the constructor or the field's initializer.

The `Argument` 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 `makeHelpText` or `makeVersionDescription` methods in your `Program` subclass.

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

2.3 Shell Wrappers

It is usually more convenient to have a shell wrapper for each entry point than it is to pass the main class and classpath to the JVM manually every time. The shell wrappers in the `bin/` directory of the template set the JVM classpath classpath relative to the location of the script file. The shell wrappers should therefore always be kept in the `bin/` directory.

Exercise: Copy the wrapper `bin/my-ext` and modify it to call your `HelloGnuNet` class.

2.4 More Documentation

The documentation for `gnunet-java` generated by Javadoc is available at https://gnunet.org/javadoc/.

3 The statistics API

The statistics service allows to store numbers under a subsystem and a name. These values are available to other components, even after your program quits.

3.1 Connecting to the statistics service

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

The Statistics constructor is called with the configuration, provided by the method `getConfiguration` of the `Program` class. The configuration contains the necessary information (port numbers, socket paths, ...) to establish a connection to the statistics service. As with most API calls in GNUnet-Java, this operation is asynchronous, meaning that the above statement does not wait for the connection to be established, but returns immediately.

Always remember to explicitly destroy your `Statistics` instance by calling its `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 to, for instance, set the value named "# bytes sent" to the value 42.

```
statistics.set("gnunet-java-hello", "#␣bytes␣sent", 42, true);
```

The last parameter (`true`) indicates that the value should be stored persistently (persistent values are stored even if the statistics service restarts).

3.3 Retrieving statistics

Retrieving a value is slightly more complex. Because of the asynchronous nature of the GNUnet-Java APIs, the `get` 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 `get` method.

This example retrieves the statistics value "# Requests Served" for the subsystem "gnunet-java-hello"

```
Cancelable getCancel = statistics.get(RelativeTime.SECOND, "gnunet-java-hello",
        "#␣Requests␣Served", 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() {
        // called if the service does not respond after the
        // specified timeout (one second)
        System.out.println("timeout␣occured");
    }
});
```

Exercise: Read the Javadoc of the statistics service. What other operations can be done on statistics values, other than reading and writing them?

Exercise: Write a program that increments a statistics value each second. Check the result with the `gnunet-statistics` command line tool. Hint: Use `Scheduler.addDelayed` to run a function after a timeout.

4 Sending encrypted messages

The CORE service is one of the most important components of GNUnet, and allows sending encrypted messages to directly connected peers. Be aware that, depending on the used transport protocol, messages sent by CORE arrive with varying reliability.

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. The same message format is used both for communication between GNUnet services and clients, as well as between peers in the network.

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

Additionally, you have to register your new message type with GNUnet-Java, giving it a unique message type number. 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 type 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/main/java/org/gnunet/construct/MsgMap.txt`, which allows the system to instantiate the right Java class when de-serializing a message from its binary representation.

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 embedding 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.messages.MonitorPutMessage` in the GNUnet-Java source code[a]. This message uses a variety of different annotations, try to understand the purpose of each member's annotation.

[a] https://gnunet.org/svn/gnunet-java/

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 void visit(MyFooMessage m) {
      // do something
    }
    public void visit(MyBarMessage m) {
      // do something else
    }
}
```

After specifying your message handler, the `init` method has to be called with a callback object. This starts the handshake with the CORE service, and 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 CORE is ready to transmit your message. Call the `transmit` method in the `MessageSink` with a `GnunetMessage.Body` in order transmit it to CORE. to finally transmit it. The header of the message is automatically added to your message body.

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 handleError() {
        // do something
    }
```

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

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

5 Establishing channels to remote peers with CADET

In contrast to CORE, the CADET (Confidential Ad-hoc Decentralized End-to-End Transport) service [3] can send messages reliably (if requested) over channels to distant peers, who must not necessarily be directly connected. The following code connects to the CADET service, and waits for connections on port 42:

[3]Formerly known as MESH service

```
Cadet m = new Cadet(cfg, inboundChannelHandler, messageHandler, 42);
```

The `inboundChannelHandler`'s `onInboundChannel` is called whenever another peer wants to establish a connection to our peer on port 42. The `messageHandle` must be a Runabout instance, and implement visit methods analogously to the CORE message handler in the previous section.

The following snippet establishes a channel to the given peer on port 42, where the channel should not buffer data (first boolean argument) and be reliable (second boolean argument).

```
Channel c = m.createChannel(targetPeer, 42, true, true);
```

A channel can be used to send messages, which are first queued and then sent to the CADET service:

```
c.send(myMessage);
```

Using this way of sending messages may cause the message queue of the channel to fill up quickly. To prevent this, wrap the message in an `Envelope`, which can invoke a notification callback once the message has been sent to the service:

```
Envelope ev = new Envelope(myMessage);
ev.notifySent(myNotifySentHandler);
c.send(myMessage);
// use ev.cancel() to abort sending the message
```

Note that the notification is called when the local CADET service accepts the message for further transmission, not when the target peer receives the message.

Exercise: Write a netcat-style tool that allows to interactively send and receive a stream of text on the command line over CADET.

6 Managing a peer's egos

An ego in GNUnet is a name tied to a key pair. Egos can represent the identity of actual users, organisations, or more abstract entities. Managed by the `IDENTITY` service, egos are entirely local to your peer.

For looking up the key of egos, there is a convenient helper function:

```
Identity.lookup(getConfiguration(), "my-ego-name", new IdentityCallback() {
    @Override
    public void onEgo(Identity.Ego ego) {
        System.out.println("public␣key:␣" + ego.getPublicKey());
        });
    }
    @Override
    public void onError(String errorMessage) {
        System.err.println("lookup␣failed:␣" + errorMessage);
    }
});
```

Creating, renaming and deleting egos requires a handle to the identity service:

```
Identity identity = new Identity();
identity.connect(getConfiguration(), null);
```

The second parameter of `connect`, which is null in the above code, can be a listener object of type `IdentityListCallback`, and is called whenever an identity is added, deleted or changed.

After connecting, identities can be created like this:

```
identity.create(myEgoName, new IdentityContinuation() {
    @Override
    public void onError(String errorMessage) {
      System.out.println("create␣failed:␣" + error message);
    }
    @Override
    public void onDone() {
      System.out.println("create␣successful");
    }
});
```

Renaming and deleting egos is done by similar means.

7 Using the GNU Name System

Resolving and publishing name records in the network can be done with GNS, a secure and decentralized alternative to the widely used Domain Name System. Currently, GNUnet-Java only supports resolving names.

Names must be looked up in a zone, which is simply an ego. Run `gnunet-gns-import.sh` (distributed with the main GNUnet package) in order to set up GNS.

Verify that this created the master-zone ego:

```
$ gnunet-identity -d # display all egos
```

Create an A-record in your your master-zone with:

```
$ gnunet-namestore -z master-zone -a -n myrecord -t A -V 1.2.3.4 -e never
```

The following snippet assumes that `master` is the ego with the name "master-zone", as retrieved with the `Identity.lookup` function.

```
final Gns gns = new Gns(getConfiguration());
// look up an A record
long typeId = GnsRecord.getIdFromString("A");
gns.lookup("myrecord.gnu", master.getPublicKey(), typeId, GNS_LOOKUP_OPTION_DEFAULT,
        null, new LookupResultProcessor() {
    @Override
    public void process(GnsRecord[] records) {
        for (GnsRecord record : records) {
            System.out.println("Record␣" +
                record.getRecordData().asRecordString());
        }
    }
});
```

A GnsRecord contains the record stored in binary form. Calling getRecordData on the record instantiates an object whose class is specific to the record type (e.g. ARecordData), or an object of type UnknownRecordData if GNUnet-Java does not support the given record type.

Exercise: Read more about GNS at https://gnunet.org/gns-namestore-editing.

Exercise: Look at the Javadoc for org.gnunet.gns.record. What types of records are currently supported, which are missing?

8 Other useful APIs

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.

The API to the TESTBED service, which allows to manage multiple peers for testing and evaluating GNUnet components, is partially implemented.

Among the most important APIs still missing are FS (filesharing) and NAMESTORE (manages GNS zone entries).

9 Writing your own client and service

GNUnet is split up into many components, with every component running 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.

9.1 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 configuration 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.

9.2 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.

9.3 Writing a service

To implement your own service, 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 choosing 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(args);
    }
}
```

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 sending messages with 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.