JAL Library

The Java Abstraction Layer (JAL) library is the Java wrapping of Abstraction Layer Bundle Protocol (ALBP). JAL library uses JNI methods to use the ALBP library written in C. It allows users to use Bundle Protocol sockets on Java programs without caring about DTN implementation running on the machine. According to ALBP library, the supported DTN implementations are DTN2, ION and IBR-DTN.

Requirements

To install JAL library you need:

- Java JDK 1.7 or superior.
- Ant.
- ALBP library.
- Linux based system (because of compatibility of ALBP library).
- At least one of the DTN implementations supported by ALBP library (ION, DTN2 and IBR-DTN).

Installation

The installation of JAL is pretty easy, it just needs 3 steps:

1. Download and compile ALBP library

The first step is to download and compile the ALBP library. To perform this operation you should run the make command in a terminal. The make command requires some parameters to let it know which implementations you want the ALBP library work with. You can run the following commands to get the help message.

```
$ cd <ALBP_folder>
$ make
```

Here is an example of compiling properly the library for working only with ION implementation.

```
$ cd <ALBP_folder>
$ make ION_DIR=<ION_folder>
```

2. Download and install JAL library

Once the ALBP library is compiled you need to download, compile and install the JAL library. Once downloaded and unzipped the library, you should proceed to compile and install the C module for the JNI calls; you can perform it by running the following commands in a terminal.

```
$ cd <JAL_folder>
$ make ION_DIR=<ION_folder> AL_BP_DIR=<ALBP_folder>
$ sudo make install
```

Note that `<ALBP_folder>` must be the absolute path and should be the same as the previous command because the JAL library must use the ALBP library to work!
Now the al_bp library is correctly installed and ready to be used from the JAL library. To build the JAL library is necessary to run the command

```
$ ant
```

that will compile the library and will prepare the javadoc and the .jar file that will be saved in /dist/lib/JAL_<version>.jar, where <*version*> will be the current version.

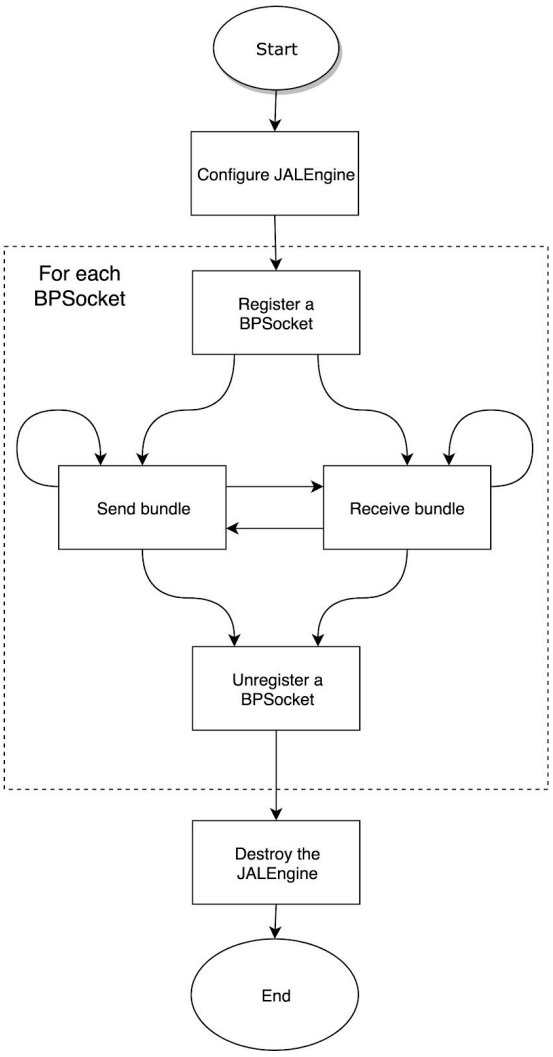
3. Import and use the library

Once created the .jar file you can import it in your project and use it. After including the .jar file you can specify to your IDE the internal folder “doc” which contains the javadoc documentation.

How to use the JAL library

To use the JAL library you have to perform some steps.

1. Configure the JALEngine to force one of the two schemes required by the standard (IPN and DTN); in case you are forcing IPN scheme you must set also the IPNSchemeForDTN2 because DTN2 working with IPN scheme doesn’t know which is its local number. It’s important to remember that this operation is optional and the configuration is for an advanced use of the JAL library, for normal users the default is the preferred choice.
2. Initialize the JALEngine by calling the init method. Also this operation is optional because when registering a BPSocket the JALEngine will be automatically initialized, allowing users not to worry about that.
3. Register a BPSocket. Registering a BPSocket is the equivalent as binding a port to the local driver TCP/UDP. You can register as many sockets as you want. You can register using only DTN scheme, only IPN scheme or both.
4. Send a bundle. This let you to send a bundle through a BPSocket. The bundle will be sent using the endpoint ID used on registration phase (step 3). You can repeat this phase as many times as you want.
5. Receive a bundle. This let you to receive a bundle addressed to the endpoint ID used on registration phase (step 3). You can repeat this phase as many times as you want.
6. Unregister a BPSocket. This is the dual of the step 3. It unbinds a BPSockets. You have to unregister every BPSocket before closing the program.
7. Destroy the JALEngine. This operation must be called when you don’t want to use BPSockets anymore and serves to deallocate properly the data from the C structures. It is automatically called when the library for JNI calls is unloaded from the JVM, that time usually coincides with the ending of the program therefore is required that your application destroys on his own the JALEngine when no more BPSockets will be created.



Here a flow diagram of the steps that a user have to follow to use the JAL library.

Usage examples

Some simple examples follow to send and receive bundles and then a simple complete program that sends and receives bundles.

Send bundle

The following code shows how to send a bundle to *ipn:1.4* with replyTo set to *dtn:machine2/replyTo* and custody request. It registers to the DTN socket with demux token as 8 in case of IPN scheme or “sending” in case of DTN scheme.

```
BPSocket socket = null;
try {
      socket = BPSocket.register("sending", 8);
} catch (JALRegisterException e) {
      System.err.println("Error on registering.");
      System.exit(1);
```

```
}
byte[] data = "the data you want to send".getBytes();
Bundle bundle = new Bundle(BundleEID.of("ipn:1.4")); // Creates a new
bundle with destination ipn:1.4
bundle.setPayload(BundlePayload.of(data)); // Setting the payload
bundle.setReplyTo(BundleEID.of("dtn:machine2/replyTo")); // Sets the
replyTo to the uri dtn:machine2/replyTo
bundle.setExpiration(120); // Changes the default expiration setting to
120 seconds
bundle.addDeliveryOption(BundleDeliveryOption.Custody); // Add request
for custody

try {
      socket.send(bundle);
} catch (NullPointerException | IllegalArgumentException |
IllegalStateException | JALNullPointerException |
JALNotRegisteredException | JALSendException e) {
      System.err.println("Error on sending bundle.");
      System.exit(1);
}

try {
      socket.unregister(); // Closes the socket
} catch (JALNotRegisteredException | JALUnregisterException e) {
      System.err.println("Error on unregistering socket.");
      System.exit(1);
}

try {
      JALEngine.getInstance().destroy(); // Destroys the engine (after
this you can't use anymore BP sockets)
} catch (JALLibraryNotFoundException | JALInitException e) {
      System.err.println("Error on destroying JALEngine.");
      System.exit(1);
}
```

Receive bundle

To receive a bundle from a DTN socket you need to call the receive function as in the following example. It registers to the DTN socket with demux token as 9 in case of IPN scheme or “receiving” in case of DTN scheme.

```
BPSocket socket = null;
try {
      socket = BPSocket.register("receiving", 9);
} catch (JALRegisterException e) {
```

```
	System.err.println("Error on registering.");
	System.exit(1);
}

try {
	Bundle bundle = socket.receive();
} catch (JALReceptionInterruptedException e) {
	System.err.println("Reception interrupted.");
	System.exit(1);
}
catch (JALTimeoutException e) {
	System.err.println("Timed out.");
	System.exit(1);
}
catch (JALNotRegisteredException | JALReceiveException e) {
	System.err.println("Error on receiving.");
	System.exit(1);
}

try {
	socket.unregister(); // Closes the socket
} catch (JALNotRegisteredException | JALUnregisterException e) {
	System.err.println("Error on unregistering socket.");
	System.exit(1);
}

try {
	JALEngine.getInstance().destroy(); // Destroys the engine (after
this you can't use anymore BP sockets)
} catch (JALLibraryNotFoundException | JALInitException e) {
	System.err.println("Error on destroying JALEngine.");
	System.exit(1);
}
```

A simple complete program

Now that we know how to send and receive a bundle we can write a simple complete program that will receive a bundle sending the same data received from the source. It registers to the DTN socket with demux token as 10 in case of IPN scheme or “echo” in case of DTN scheme.

```
BPSocket socket = null;
try {
	socket = BPSocket.register("echo", 10);
} catch (JALRegisterException e) {
	System.err.println("Error on registering.");
```

```
	System.exit(1);
}

Bundle bundle = null;
try {
	bundle = socket.receive();
} catch (JALReceptionInterruptedException e) {
	System.err.println("Reception interrupted.");
	System.exit(1);
}
catch (JALTimeoutException e) {
	System.err.println("Timed out.");
	System.exit(1);
}
catch (JALNotRegisteredException | JALReceiveException e) {
	System.err.println("Error on receiving.");
	System.exit(1);
}

bundle.setDestination(bundle.getSource()); // We want to forward the
bundle to the source
bundle.setExpiration(120); // Sets the expiration to 120 seconds

try {
	socket.send(bundle);
} catch (NullPointerException | IllegalArgumentException |
IllegalStateException | JALNullPointerException
		| JALNotRegisteredException | JALSendException e) {
	System.err.println("Error on sending bundle.");
	System.exit(1);
}

try {
	socket.unregister(); // Closes the socket
} catch (JALNotRegisteredException | JALUnregisterException e) {
	System.err.println("Error on unregistering socket.");
	System.exit(1);
}

try {
	JALEngine.getInstance().destroy(); // Destroys the engine (after
this you can't use anymore BP sockets)
} catch (JALLibraryNotFoundException | JALInitException e) {
	System.err.println("Error on destroying JALEngine.");
	System.exit(1);
```

```
}
```

Uninstall

To uninstall the JAL library you need to run the commands below.

```
$ cd <JAL_folder>
$ make clean
$ sudo make uninstall
$ ant clean
```

After doing that, you can stop using the .jar file from your projects.