Creating an iHart Application (Java)

The full exported Javadocs for the iHart Java client library are available here.

Setup

First, make sure you have downloaded the latest version of the iHart project from the website (or cloned the repository from GitHub). You will need to import the `ihart` package and its contents, in order to use classes like `ihart.CVEvent`.

Eclipse

In order for Eclipse to compile and run your project, it needs to know about the iHart client library and its dependencies.

To link the jar files included in the project:

1. Open your project in Eclipse. (If you have not already created a Java project, go ahead and do so now.)
2. Under the “Project” menu, click on “Properties”. A popup window should appear.
3. Click on “Java Build Path” from the menu on the left.
4. Click on the “Libraries” tab on the top.
5. Click on “Add External JARs” on the right.
6. Navigate to where you downloaded the iHart project, and then to the “client/library/java” folder.
7. Add **both** the “ihart-library.jar” and “javax.json.jar” files. They are both required for the project to compile and run.
8. Hit “OK” at the bottom right.

At this point, you should be able to import the `ihart` package and its contents without any errors.

Command Line

If you’re planning on compiling your project from the Terminal/Command Prompt, the iHart client library and its dependencies need to be available. Whenever you compile or run your program, you will need to include the necessary jar files in the classpath.

Note: if you are unfamiliar with compiling and running Java programs from the command line, Eclipse may be a better option.

Compile your project:

To compile your project, run the following where your source ".java" files are located, and replace "<path to ihart project>" with the relative path to where you placed the iHart project.

```
javac -cp "<path to ihart project>/client/library/java/*" *.java
```

For example, to compile the PrintlnProgram example application in "ihart/client/sample/PrintlnProgram", you would first change directories to that folder, and then run:

```
javac -cp "../../library/java/*" PrintlnProgram.java
```

Run your project:

To run your project, run the following where your compiled ".class" files are located, and again replace "<path to ihart project>" with the relative path to where you placed the iHart project. You should also replace "<YourMainClass>" with the class containing your `main` method.

```
java -cp ".;<path to ihart project>/client/library/java/*" <YourMainClass>
```

If you are on OS X rather than windows, you may have to substitute a colon (`:`) for the semicolon (`;`), as below:

```
java -cp ".:<path to ihart project>/client/library/java/*" <YourMainClass>
```

Note that the "`.`" specifies that Java should refer to your application as well as the iHart libraries when running.

For example, to run the PrintlnProgram example application in "ihart/client/sample/PrintlnProgram" on Windows, you would first change directories to that folder, and then run:

```
java -cp ".;../../library/java/*" PrintlnProgram
```

For more details on the classpath option, see the official Java documentation and this Stack Overflow answer.

Development Using the iHart Java Client Library

The iHart client library is set up with a class that manages connecting to the server and receiving messages, and a set of classes that allow you to receive and process events. Similarly to Java's `ActionListener` interface, the library provides a `CVEventListener` interface with three methods:

- `public void shellsArrived(CVEvent shellEvent)`
- `public void facesArrived(CVEvent faceEvent)`
- `public void blobsArrived(CVEvent blobEvent)`

Any class that implements the `CVEventListener` interface must implement all of these methods (though the method body may be empty for one or more of them).

Parsing `CVEvents`

`CVEvents` are fired whenever information is received from the server about motion (aka "shells") or faces were detected. A generic "all blobs" event will also be dispatched whenever faces, shells, or both were detected. Every event has a type and a list of blobs associated with it, and you can get the number of blobs from the event object as well.

`CVEvents` also refer to "regions of interest" or "areas of interest," which you can read more about on the iHart website. All blobs occur within the bounds of a single region of interest. From the `CVEvent` object, you can choose to `getAllBlobs()` regardless of what region of interest they occurred in, or `getBlobsInRegion(int regionOfInterest)` to get only the blobs that occurred within the specified region.

Both methods return `List<Blob>`; you can iterate over this list to get access to each of the `Blob` objects. Blobs represent either a face or a shell that was detected by the server. Each blob has an `x` coordinate and a y coordinate (representing the top left corner of the blob), a width, a height, a type (face or shell), and the index of the region of interest it occurred in.

Note: the `x`, `y`, width, and height are stored and returned as `doubles`, in order to be as precise as possible. Many `java.swing` classes and methods require `ints` as parameters, so you may need to cast these values before using them for graphics.

Connecting to the Server

Once you have a class that implements `CVEventListener`, you can start receiving events from the server. There are a few steps involved:

1. Run the server application included in the project in "server/dist/…" (choose the Mac or Windows distribution depending on your operating system). On OS X, you'll want the `cvServer.app` file; on Windows, you should run the `cvServer.exe` file.

2. In your code, start the connection to the server by creating a `CVManager` object. This object needs to know where the server is running; usually, that will be on the same computer ("localhost" is the hostname) with the default port (5204).

```
CVManager cvManager = new CVManager("localhost", 5204);
```

3. Next, let the `CVManager` know that your object wants to receive `CVEvents`:

   ```
   cvManager.addCVEventListener(myClass);
   ```

   Where `myClass` is an instance of the class that implements `CVEventListener`.

If your application exits with a message about being unable to connect to the server, make sure you are running the server where the `CVManager` expects it to be.