RL-Glue Java Extension (Codec) 2.0 Manual

Brian Tanner ::brian@tannerpages.com

February 3, 2009

Contents

1 Introduction

This document describes how to use the RL-Glue Java Extension, a software library (or codec) that provides socket-compatibility with the RL-Glue Reinforcement Learning software library.

For general information and motivation about the RL-Glue[1] project, please refer to the documen-

[1] `http://glue.rl-community.org/`

tation provided with that project.

This codec will allow you to create agents, environments, and experiment programs in Java.

This software project is licensed under the Apache-2.0[2] license. We're not lawyers, but our intention is that this code should be used however it is useful. We'd appreciate to hear what you're using it for, and to get credit if appropriate.

This project has a home here:
`http://glue.rl-community.org/Home/Extensions/java-codec`

1.1 Software Requirements

To run agents, environments, and experiments created with this codec, you will need to have RL-Glue (or a surrogate) installed on your computer.

Compiling and running components with this codec requires Java 1.5 or higher. You can find out what version you have by doing the following at the command-line:

```
>$ java -version
```

This codec is distributed as a compiled Java JAR file, so you do not need to compile it in order to use it.

If you are a developer and you want to compile the codec from source, you will need the Apache Ant[3] build system. You probably need JUnit and Subversion installed as well. We'll try to make those optional dependencies in the future.

1.2 Getting the Codec

The codec can be downloaded either as a tarball or can be checked out of the subversion repository where it is hosted.

The tarball distribution can be found here:
`http://code.google.com/p/rl-glue-ext/downloads/list`

To check the code out of subversion:
`svn checkout http://rl-glue-ext.googlecode.com/svn/trunk/projects/codecs/Java Java-Codec`

Technically all you really **need** is the JAR archive of the codec:
`svn export http://rl-glue-ext.googlecode.com/svn/trunk/projects/codecs/Java/products/JavaRLGlueCodec.jar`

[2] `http://www.apache.org/licenses/LICENSE-2.0.html`
[3] `http://ant.apache.org/`

1.3 Installing the Codec

This codec can either be installed into a Java extensions directory, or it can be used uninstalled in "free-float" mode by specifying it in the Java `classpath` when compiling and running Java classes.

The advantage of installed mode is that your system will always know about the RL-Glue classes, so the code required to compile and run Java classes is much cleaner. The downside of installing is that Java will always find the extension classes first, meaning that you can not easily move between different versions of the classes just by specifying a different classpath.

Below are two examples of how you would work with the code, and then you can choose for yourself whether to install or not. This manual's instructions will all be written as if you have installed, for clarity.

```
//Not installed (free-float) mode
>$ javac -cp path/to/codecs/Java/products/JavaRLGlueCodec.jar MyAgent.java
>$ java -cp path/to/codecs/Java/products/JavaRLGlueCodec.jar:. MyAgent

//Installed mode
>$ javac MyAgent.java
>$ java MyAgent
```

To install the RL-Glue Java Extension, do the following:

```
//This location will depend on if you have the developer distribution
//or the user distribution

//User Distribution do this
>$ cd path/to/downloaded/codec/Java/

//Developer Distribution do this
>$ cd path/to/downloaded/codec/Java/products

//Both distributions install the same way
>$ java -jar JavaRLGlueCodec.jar --install
```

This will provide you with a numbered list, prompting you to choose a Java extension folder to install to that is appropriate for your system.

1.4 Removing the Codec

If you have installed the RL-Glue Java Extension, it can be removed automatically with the command:

```
>$ java org.rlcommunity.rlglue.codec.RLGlueCore --uninstall
```

Alternatively, it can be removed any time by manually deleting the JAR file from the appropriate extensions folder.

2 Sample Project

We have included two example projects with this codec, located in the `examples` directory. Each project contains an agent, environment, and experiment written for this Java codec. The two projects are `skeleton` and `mines-sarsa-sample`.

The `skeleton` contains all of the bare-bones plumbing that is required to create an agent/environment/experiment with this codec and might be a good starting point for creating your own components.

The `mines-sarsa-sample` contains a fully functional tabular Sarsa learning algorithm, a discrete-observation grid world problem, and an experiment program that can run these together and gather results. More details below in Section 2.5.

In the following sections, we will describe the skeleton project. Running and using the `mines-sarsa-sample` is analogous.

Note: these examples assume that the RL-Glue Java Extension is installed, so they do not specify adding `JavaRLGlueCodec.jar` to the classpath every time.

2.1 Skeleton Agent

We have provided a skeleton agent with the codec that is a good starting point for agents that you may write in the future. It implements all the required functions and provides a good example of how to compile a simple agent.

The pertinent file is:

```
examples/skeleton/SkeletonAgent.java
```

This agent does not learn anything and randomly chooses integer action 0 or 1.

You can compile and run the agent like:

```
>$ cd examples/skeleton
>$ javac SkeletonAgent.java
>$ java SkeletonAgent
```

You will see something like:

```
RL-Glue Java Agent Codec Version: 2.0 (Build:465:481M)
     Connecting to 127.0.0.1 on port 4096...
```

This means that the SkeletonAgent is running, and trying to connect to the `rl_glue` executable server on the local machine through port 4096!

You can kill the process by pressing `CTRL-C` on your keyboard.

The Skeleton agent is very simple and well documented, so we won't spend any more time talking about it in these instructions. Please open it up and take a look.

2.2 Skeleton Environment

We have provided a skeleton environment with the codec that is a good starting point for environments that you may write in the future. It implements all the required functions and provides a good example of how to compile a simple environment. This section will follow the same pattern as the agent version (Section 2.1). This section will be less detailed because many ideas are similar or identical.

The pertinent file is:

`examples/skeleton_environment/SkeletonEnvironment.java`

This environment is episodic, with 21 states, labeled $\{0, 1, \ldots, 19, 20\}$. States $\{0, 20\}$ are terminal and return rewards of $\{-1, +1\}$ respectively. The other states return reward of 0. There are two actions, $\{0, 1\}$. Action 0 decrements the state number, and action 1 increments it. The environment starts in state 10.

You can compile and run the environment like:

```
>$ cd examples/skeleton
>$ javac SkeletonEnvironment.java
>$ java SkeletonEnvironment
```

You will see something like:

```
RL-Glue Java Environment Codec Version: 2.0 (Build:192:239M)
     Connecting to 127.0.0.1 on port 4096...
```

This means that the SkeletonEnvironment is running, and trying to connect to the `rl_glue` executable server on the local machine through port 4096!

You can kill the process by pressing `CTRL-C` on your keyboard.

The Skeleton environment is very simple and well documented, so we won't spend any more time talking about it in these instructions. Please open it up and take a look.

2.3 Skeleton Experiment

We have provided a skeleton experiment with the codec that is a good starting point for experiment that you may write in the future. It implements all the required functions and provides a good example of how to compile a simple experiment. This section will follow the same pattern as the agent version (Section 2.1). This section will be less detailed because many ideas are similar or identical.

The pertinent files are:

`examples/skeleton/SkeletonExperiment.java`

This experiment runs `RL_Episode` a few times, sends some messages to the agent and environment, and then steps through one episode using `RL_step`.

```
>$ cd examples/skeleton
>$ javac SkeletonExperiment.java
>$ java SkeletonExperiment
```

You will see something like:

```
Experiment starting up!
RL-Glue Java Experiment Codec Version: 2.0 (Build:192:239M)
     Connecting to 127.0.0.1 on port 4096...
```

This means that the SkeletonExperiment is running, and trying to connect to the `rl_glue` executable server on the local machine through port 4096!

You can kill the process by pressing `CTRL-C` on your keyboard.

The Skeleton experiment is very simple and well documented, so we won't spend any more time talking about it in these instructions. Please open it up and take a look.

2.4 Putting it all together

At this point, we've compiled and run each of the three components, now it's time to run them with the `rl_glue` executable server. The following will work from the examples directory if you have them all built, and RL-Glue installed in the default location:

```
>$ cd examples/skeleton
>$ rl_glue &
>$ javac *.java
```

```
>$ java SkeletonAgent &
>$ java SkeletonEnvironment &
>$ java SkeletonExperiment
```

If RL-Glue is not installed in the default location, you'll have to start the `rl_glue` executable server using it's full path (unless it's in your `PATH` environment variable):

```
>$ /path/to/rl-glue/bin/rl_glue &
```

You should see output like the following if it worked:

```
>$ rl_glue &
RL-Glue Version 3.0-beta-1, Build 848:856
      RL-Glue is listening for connections on port=4096

>$ java SkeletonAgent &
RL-Glue Java Agent Codec Version: 2.0 (Build:192:239M)
     Connecting to 127.0.0.1 on port 4096...
     Agent Codec Connected
     RL-Glue :: Agent connected.

>$ java SkeletonEnvironment &
RL-Glue Java Environment Codec Version: 2.0 (Build:192:239M)
     Connecting to 127.0.0.1 on port 4096...
     Environment Codec Connected
     RL-Glue :: Environment connected.

>$ java SkeletonExperiment
Experiment starting up!

RL-Glue Java Experiment Codec Version: 2.0 (Build:192:239M)
     Connecting to 127.0.0.1 on port 4096...
     Experiment Codec Connected
     RL-Glue :: Experiment connected.

     Skeleton agent parsed the task spec.
     Observation have 1 integer dimensions
     Actions have 1 integer dimensions
     Observation (state) range is: 0 to 20
     Action range is: 0 to 1
     Reward range is: -1.0 to 1.0
     RL_init called, the environment sent task spec:
VERSION RL-Glue-3.0 PROBLEMTYPE episodic DISCOUNTFACTOR 1.0
OBSERVATIONS INTS (1 0 20)  ACTIONS INTS (1 0 1)  REWARDS (1 -1.0 1.0)  EXTRA
```

```
----------Sending some sample messages----------
Agent responded to "what is your name?" with: my name is skeleton_agent, Java edition!
Agent responded to "If at first you don't succeed; call it version 1.0" with: I don't know how

Environment responded to "what is your name?" with: my name is skeleton_environment, Java editi
Environment responded to "If at first you don't succeed; call it version 1.0" with: I don't kno


----------Running a few episodes----------
Episode 0  10 steps  -1.0 total reward  1 natural end
Episode 1  10 steps  -1.0 total reward  1 natural end
Episode 2  10 steps  -1.0 total reward  1 natural end
Episode 3  10 steps  -1.0 total reward  1 natural end
Episode 4  10 steps  -1.0 total reward  1 natural end
Episode 5  1 steps  0.0 total reward  0 natural end
Episode 6  10 steps  -1.0 total reward  1 natural end


----------Stepping through an episode----------
First observation and action were: 10 and: 0


----------Summary----------
It ran for 10 steps, total reward was: -1.0
```

2.5 Going Further – Mines Sarsa Example Project

The `skeleton` sample project is extremely limited and only shows the mechanics of how RL-Glue components are structured using the Java codec. The `mines-sarsa` sample project is much richer.

2.5.1 Sample-Mines-Environment

The mines environment is internally a two-dimensional, discrete grid world where the agent receives a penalty per step until reaching a goal state, hopefully without stepping on any exploding landmines along the way. The (x,y) state is flattened into a discrete, scalar observation for the agent. This environment can receive special messages from the experiment program to print the current state to the screen, and also to toggle between random starting states and a fixed starting-state specified by the experiment.

The task specification string[4] is created in a semi-automated way using the Java RL-Glue Extension task spec parser/builder.

2.5.2 Samples-Sarsa-Agent

The SARSA agent is a tabular learning agent that uses $\epsilon - greedy$ exploration as described in Reinforcement Learning: An Introduction by Sutton and Barto.

The SARSA agent parses the task specification string using the Java RL-Glue Extension task spec parser. This agent can receive special messages from the experiment program to pause/unpause learning, pause/unpause exploring, save the current value function to a file, and load the the value function from a file.

2.5.3 Sample-Experiment

The sample experiment program runs the show. First, it alternates running the agent in the environment for a number of episodes, and telling the agent to pause learning so that the current performance can be evaluated. These results are saved to a comma-separated-value file.

The sample experiment then tells the agent to save the value function to a file, and then resets the experiment (and agent) to initial conditions. After verifying that the agent's initial policy is bad, the experiment tells the agent to load the value function from the file. The agent is evaluated again using this previously-learned value function, and performance is dramatically better.

Finally, the experiment sends a message to specify that the environment should use a fixed (instead of random) starting state, and runs the agent from that fixed start state for a while.

3 Who creates and frees memory?

The RL-Glue technical manual has a section called *Who creates and frees memory?*. The general approach recommended there is to make a copy of data you want to keep beyond the method it was given to you. The same rules of thumb from that manual should be followed when using the Java codec. Both sample projects make copies of observations and actions in the appropriate way, and they can be used as a guide.

[4] `http://glue.rl-community.org/Home/rl-glue/task-spec-language`

4 Advanced Features

4.1 Task Specification Parser

As of fall 2008, we've updated the task specification language:
`http://glue.rl-community.org/Home/rl-glue/task-spec-language`

The new task specification string parser should be used in environments to create task specification strings in `env_init`. Both sample environments provide an example of creating a task spec in this way. There are also several advanced examples of this in the RL-Library[5]. The task spec parser/builder can also be used by agents to decode the task spec string for `agent_init`. The sample sarsa agent in Section 2.5.2 demonstrates how to do this.

4.2 Connecting on custom ports to custom hosts

This section will explain how to set custom target IP addresses (to connect over the network) and custom ports (to run multiple experiments on one machine or to avoid firewall issues). Sometimes you will want run the `rl_glue` server on a port other than the default (4096) either because of firewall issues, or because you want to run multiple instances on the same machine.

In these cases, you can tell your Java agent, environment, or experiment program to connect on a custom port and/or to a custom host using the environment variables `RLGLUE_PORT` and `RLGLUE_HOST`.

For example, try the following code:

```
> $ RLGLUE_PORT=1025 RLGLUE_HOST=yahoo.ca java SkeletonAgent
```

That command could give output like:

```
RL-Glue Java Agent Codec Version: 2.0 (Build:390M)
Connecting to yahoo.ca on port 1025...
```

This works for agents, environments, and experiments. In practice, yahoo.ca probably isn't running an RL-Glue server.

You can specify the port, the host, neither, or both. Ports must be numbers, hosts can be hostnames or ip addresses. Default port value is 4096 and host is 127.0.0.1.

If you don't like typing these variables every time, you can export them so that the value will be set for future calls in the same session:

[5] `http://library.rl-community.org`

```
> $ export RLGLUE_PORT=1025
> $ export RLGLUE_HOST=mydomain.com
```

Remember, on most *nix systems, you need `superuser` privileges to listen on ports lower than 1024, so you probably want to pick one higher than that.

5 Codec Specification Reference

This section will explain how the RL-Glue types and functions are defined for this codec. This isn't meant to be the most exciting section of this document, but it will be handy.

Instead of re-creating information that is readily available in the JavaDocs, we will give pointers were appropriate.

5.1 Types

5.1.1 Simple Types

Unlike the C/C++ codec, we will not be using `typedef` statements to create special labels for the types. In Java:

- *reward* is `double`
- *terminal* is `int` (1 for terminal, 0 for non-terminal) We hope to replace these with boolean eventually.
- *messages* are `String`
- *task specifications* are `String`

5.1.2 Structure Types

All of the major structure types (observations, actions, random seed keys, and state keys) extend the `RL_Abstract_Type` class.

All of the types are listed in the `org.rlcommunity.rlglue.codec.types` package.

5.2 Functions

5.2.1 Agent Functions

All agents **should implement** `org.rlcommunity.rlglue.codec.AgentInterface`.

5.2.2 Environment Functions

All environments **should implement** `org.rlcommunity.rlglue.codec.EnvironmentInterface`.

5.2.3 Experiments Functions

All experiments **can call** the static methods in `org.rlcommunity.rlglue.codec.RLGlue`.

6 Changes and 2.x Backward Compatibility

There were many API/Interface changes from RL-Glue 2.x to RL-Glue 3.x. For those that are at the level of the API and project organization, please refer to the the RL-Glue overview documentation.

6.1 RL-Glue Split and Package/Jar Naming Changes

With the RL-Glue 3.x project, we've decided to keep the codecs separate from the main RL-Glue project. This is so the codecs can be more agile while the main RL-Glue project stays very stable.

Since things were changing, we took an opportunity to fix some of our long-term gripes with the Java codec. The name of the JAR file has changed from `RL-Glue.jar` to `JavaRLGlueCodec.jar`.

The JAR file is now distributed in:
`rl-glue-ext/projects/codecs/java/products/JavaRLGlueCodec.jar`

In the previous incarnation of the Java codec, the classes were in a very shallow heirarchy: `rlglue.RLGlue`, `RLGlue.types`, etc. For the updated release, we've moved to a richer package description that is more in line with other Java projects in the Reinforcement Learning community. The new package hierarchy is:
`org.rlcommunity.rlglue`

We've also moved a few of the class/interfaces around. The most notable change is that instead of `Agent` and `Environment` interfaces, we now have `AgentInterface` and `EnvironmentInterface`. Also, instead of these interfaces being in their own package, they are now in: `org.rlcommunity.rlglue.codec`.

Updating existing code might seem like a lot of work, but it's easier than it seems. The main changes from the user end are: 1) Using the right jar (JavaRLGlueCodec.jar instead of RL-Glue.jar) (Not necessary if you are using RL-Viz)

2) Change classes that implement `Agent` and `Environment` to `AgentInterface` and `EnvironmentInterface`

3) Change package imports. Find and replace:

```
rlglue.RLGlue                    ==>    org.rlcommunity.rlglue.codec.RLGlue
```

```
rlglue.types.                    ==>    org.rlcommunity.rlglue.codec.types.
rlglue.agent.Agent               ==>    org.rlcommunity.rlglue.codec.AgentInterface
rlglue.environment.Environment   ==>    org.rlcommunity.rlglue.codec.EnvironmentInterface
```

4) `org.rlcommunity.rlglue.codec.types.Reward_observation` has been renamed to: `org.rlcommunity.rlglue.codec.types.Reward_observation_terminal`

5) `Random_seed_key` and `State_key` have been removed

Those few things should cover most of it. If someone makes a very strong case, we can probably create a codec that is completely compatible with the old naming conventions. However, I assure you that I uploaded several tens of thousands of lines of code for the RL-Viz, RL-Library, and BT-AgentLib projects in only about 30 minutes total.

POSSIBLE CONTRIBUTION: If we've missed anything, or there is an easier way, please let us know!

6.2 Agent/Environment Loading

Historically, there have been a few different approaches for loading agents and environments.

The first (original) method was to call the main() method of the AgentLoader/EnvironmentLoader class, passing it the name of the Agent/Env class you wanted to load. Hopefully you remembered to put it in your classpath. This approach inevitably lead to lots of typing to load an agent, as frustrating run-time failures when paths weren't set right. Here is an example of how that looks:

```
java org.rlcommunity.rlglue.codec.util.AgentLoader myAgent
```

The **new and improved** method is to put a `main` method inside your agent/environment that calls the applicable loader so that the agent/env can load itself. So, in your agent (for example) you'd have code like:

```
import org.rlcommunity.rlglue.codec.util.AgentLoader;
/* rest of agent here */
//This would be inside MyAgent.java
public static void main(String[] args){
        AgentLoader theLoader=new AgentLoader(new MyAgent());
        theLoader.run();
}
```

So, for the price of that tiny bit of code inside your agent, you can now run the agent class directly:

```
java myAgent
```

We feel that this is a useful step forward, and will be encouraging this approach.

7 Frequently Asked Questions

We're waiting to hear your questions!

7.1 Where can I get more help?

7.1.1 Online FAQ

We suggest checking out the online RL-Glue Java Codec FAQ:
`http://glue.rl-community.org/Home/Extensions/java-codec#TOC-Frequently-Asked-Questions`

The online FAQ may be more current than this document, which may have been distributed some time ago.

7.1.2 Google Group / Mailing List

First, you should join the RL-Glue Google Group Mailing List:
`http://groups.google.com/group/rl-glue`

We're happy to answer any questions about RL-Glue. Of course, try to search through previous messages first in case your question has been answered before.

8 Credits and Acknowledgements

Andrew Butcher originally wrote the RL-Glue network library and first version of this codec. Thanks Andrew.

Brian Tanner has since grabbed the torch and has continued to develop the codec.

8.1 Contributing

If you would like to become a member of this project and contribute updates/changes to the code, please send a message to rl-glue@googlegroups.com.

Document Information

`Revision Number: $Rev: 497 $`
`Last Updated By: $Author: brian@tannerpages.com $`

Last Updated : $Date: 2009-02-02 00:23:20 -0700 (Mon, 02 Feb 2009) $
$URL: https://rl-glue-ext.googlecode.com/svn/trunk/projects/codecs/Java/docs/JavaCodec.tex $