RL-Glue Matlab Codec 1.0 Manual

Brian Tanner :: brian@tannerpages.com

Contents

1 Introduction

This document describes how to use the Matlab RL-Glue Codec, a software library that provides socket-compatibility with the RL-Glue Reinforcement Learning software library. Matlab is a brand new codec, created specifically for the RL-Glue 3.0 release. Special thanks (or pokes) should go to Dale, Doina, and Yaki who gave voices to the countless others who probably have scorned us for not supporting Matlab earlier. It turned out to not even be that difficult :)

For general information and motivation about the RL-Glue[1] project, please refer to the documentation provided with that project.

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

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

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/matlab-codec`

1.1 Software Requirements

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

Compiling and running components with this codec requires Matlab. The codec was developed on Matlab 7.6.0.x, it has not been tested extensively on other versions. This Matlab codec uses the RL-Glue Java Extension, which means that Matlab needs to be running with the Java Virtual Machine enabled (it is by default). The Java extension does not need to be installed independently.

Possible Contribution: Someone with Matlab experience could help us find out what version of Matlab is required to use this codec, and could help us update the codec to be as robust as possible to older versions.

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/Matlab Matlab-Codec`

1.3 Installing the Codec

This codec is package with a Matlab "installer" which copies the codec source files and the RL-Glue Java Extension JAR file to a user-configurable location and adds them to your Matlab path and Matlab Java classpaths. This is the recommended way of using the codec. Alternatively, you can skip the installer and setup these paths on your own. This manual will assume that you are using the installer.

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

To run the installer, use the `installRLGlue()` function from within Matlab in the main directory of the package you downloaded. This will suggest installing Matlab to your home directory at `~/rl-glue/codecs/matlab`. If you would prefer an alternate location, you can call the function with a path, like:

```
>>installRLGlue('~/desired/path/to/codec')
```

2 Gotchas!

2.1 Disconnect your Agent/Environment/Experiment

Matlab is not really multithreaded. This means that when Matlab is listening for a command from the network, it cannot be interrupted easily. Issuing the `CTRL-c` command will not bring Matlab back when it is waiting to connect to `rl_glue`. It also means in some crash cases, Matlab will hang, or Matlab will come back to you, but the socket will not close properly so it will hang next time you try to make a connection.

If your agent/environment/experiment crashes or terminates prematurely, you should manually disconnect your Matlab component so that the `Java` socket gets cleaned up. Depending on what you were running, the code would be:

```
>> disconnectAgent()                %For agents
>> disconnectEnvironment()          %For environments
>> disconnectGlue()                 %For experiments
```

If you don't disconnect manually in these cases, Matlab can get hung up and you'll have to restart it.

If Matlab is otherwise hanging because it is blocked waiting on the network, you can sometimes bring it back by using `CTRL-c` and then making a network operation happen. For example, if you started up an agent, and then decided you want to stop it, you could first do `CTRL-c`. There will be no feedback that anything happened, Matlab will stay hung. Then, go to your terminal/console and start `rl_glue`. The `CTRL-c` you did in Matlab will be processed after the agent connects to `rl_glue` and you will be able to `disconnectAgent()` and continue.

Possible contribution: help make Matlab networking more reliable. If anyone has experience, ideas, or enthusiasm, we would very much like to make Matlab more robust to communication and network problems like those described above. They are quite frustrating. If you are interested in helping, please contact us.

2.2 One Component At A Time

Because Matlab is inherently single threaded, we had limit each Matlab instance to running a single component at a time. This means that you can only run ONE of an agent OR environment OR experiment in Matlab at a time. If you are able to open multiple Matlab instances, you can run one component per instance. We think that the usual case will probably be to run your agent in Matlab, but run the experiment and environment in another language, like Java, C, or Python.

We may lift find a clever way to lift this restriction in the future, or create a local glue engine so that all three can run together inside Matlab without the network.

3 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 Matlab 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 3.5.

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

3.1 Agents

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 create and run a simple agent.

There are several functions that need to be written. They are all contained in a single Matlab source file:

```
examples/skeleton/skeleton_agent.m
```

The `skeleton_agent.m` file has a public function that returns a structure with function pointers to all of the required RL-Glue functions. The structure looks like this:

```
>> theAgent=skeleton_agent()
```

```
theAgent =

       agent_init: @skeleton_agent_init
      agent_start: @skeleton_agent_start
       agent_step: @skeleton_agent_step
        agent_end: @skeleton_agent_end
    agent_cleanup: @skeleton_agent_cleanup
    agent_message: @skeleton_agent_message
```

Alternatively, these different functions could each be in their own `skeleton_agent_`{init, start, step, end, cleanup, message}`.m` files. This is a personal choice.

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
>$ theAgent=skeleton_agent();
>$ runAgent(theAgent);
```

`skeleton_agent()` creates a struct with function pointers to the other `skeleton_agent` methods. `runAgent(theAgent)` then connects to RL-Glue and runs one step at a time until RL-Glue disconnects.

Alternatively, for a more interactive experience, you can run the agent manually one step at a time:

```
>$ cd examples/skeleton/
>$ theAgent=skeleton_agent()
>$ connectAgent(theAgent);
>$ runAgentLoop(theAgent);    %run one step
>$ runAgentLoop(theAgent);    %run one step
>$ runAgentLoop(theAgent);    %run one step
   ...
```

Using this method, you can stop and examine what your agent is learning, and potentially modify, visualize, or analyze it however you like.

You will see something like:

```
>> runAgent(theAgent)
RL-Glue Matlab Agent Codec Version: 1.0 ($Revision: 444 $)
    Connecting to rl_glue at host: localhost on port 4096
```

This means that the skeleton_agent 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, and then starting `rl_glue` on your local machine (this will cause Matlab to come back from Java, and process your kill command). Don't forget to `disconnectAgent()`.

We're sorry that killing an agent (or environment) is so kludgy. This is the first version of the Matlab codec, so there are some kinks to get out. We welcome your expertise in helping make this codec better!

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.

3.2 Environments

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 3.1). This section will be less detailed because many ideas are similar or identical.

The pertinent file is:

```
examples/skeleton/skeleton_environment.m
```

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
>$ theEnv=skeleton_environment();
>$ runEnvironment(theEnv);
```

You will see something like:

```
RL-Glue Matlab Environment Codec Version: 1.0 ($Revision: 444 $)
    Connecting to rl_glue at host: localhost on port 4096
```

This means that the skeleton_environment 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, and then starting `rl_glue` on your local machine (this will cause Matlab to come back from Java, and process your kill command). Don't forget to `disconnectEnvironment()`.

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.

3.3 Experiments

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 3.1). This section will be less detailed because many ideas are similar or identical.

The pertinent files are:

```
examples/skeleton/skeleton_experiment.m
```

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
>$ skeleton_experiment();
```

You will see something like:

```
Experiment starting up!
RL-Glue Matlab Experiment Codec Version: 1.0 ($Revision: 444 $)
    Connecting to rl_glue at host: 127.0.0.1 on port 4096
```

This means that the skeleton_experiment 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, and then starting `rl_glue` on your local machine (this will cause Matlab to come back from Java, and process your kill command). Don't forget to `disconnectGlue()`.

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.

3.4 Putting it all together

At this point, we've run each of the three components, now it's time to run them with the `rl_glue` executable server. As mentioned in Section 2.2, in Matlab we can only run ONE of the agent, experiment, or environment at a time. So, for this example we'll run the `Java SkeletonAgent` and `Java SkeletonEnvironment` with the `Matlab skeleton_experiment`. You could really run any agent and environment of your choice.

The following will work if you have the `rl_glue` socket server installed on your path(the default location), and the Java RL-Glue Extension[3] installed.

For this, we'll use a separate terminal window in addition to the Matlab interpreter. In a separate terminal window, do:

```
>$ cd /path/to/Java/codec/examples/skeleton
>$ rl_glue &
>$ javac *.java
>$ java SkeletonAgent &
>$ java SkeletonEnvironment &
```

In the Matlab interpreter:

```
>$ cd examples/skeleton
>$ skeleton_experiment();
```

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 &
```

In the Matlab window, you should see the following if it worked:

```
>> skeleton_experiment()
Experiment starting up!
RL-Glue Matlab Experiment Codec Version: 1.0 ($Revision: 444 $)
    Connecting to rl_glue at host: 127.0.0.1 on port 4096
    Experiment Codec Connected
RL_init called, the environment sent task spec: VERSION RL-Glue-3.0 PROBLEMTYPE episodic
DISCOUNTFACTOR 1.0 OBSERVATIONS INTS (0 20)  ACTIONS INTS (0 1)
REWARDS (-1.0 1.0)  EXTRA skeleton_environment(Matlab) by Brian Tanner.

----------Sending some sample messages----------
```

[3] `http://glue.rl-community.org/Home/Extensions/java-codec`

```
Agent responded to 'what is your name?' with: my name is skeleton_agent, Java edition!
Agent responded to 'If at first you do't succeed; call it version 1.0  '
with: I don't know how to respond to your message

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 know how to respond to your message


----------Running a few episodes----------
Episode 0  10 steps   -1.000000 total reward  natural end 1
Episode 1  10 steps   -1.000000 total reward  natural end 1
Episode 2  10 steps   -1.000000 total reward  natural end 1
Episode 3  10 steps   -1.000000 total reward  natural end 1
Episode 4  10 steps   -1.000000 total reward  natural end 1
Episode 5  1 steps   0.000000 total reward  natural end 0
Episode 6  10 steps   -1.000000 total reward  natural end 1


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


----------Summary----------
It ran for 10, total reward was: -1.000000
```

Congratulations, you have run an RL-Glue experiment using the Matlab codec! You can mix and match the Matlab agent/environment/experiment with the appropriate sample skeleton agent/environment/experi from any other codec.

3.5 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 Matlab codec. The `mines-sarsa` sample project is much richer.

3.5.1 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 land-mines 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.

3.5.2 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.

3.5.3 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 plotted with error-bars in a Matlab figure.

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.

4 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 Matlab codec. The observations and actions can be copied using the same techniques as the RL-Glue Java Extension.

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

5 Advanced Features

5.1 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 Matlab codec uses the task spec parser implementation from the RL-Glue Java Extension. This task spec parser/builder can be used in environments to create task specification strings for `env_init`. The sample mines environment in Section 3.5.1 provides 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 3.5.2 demonstrates how to do this.

5.2 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 Matlab agent, environment, or experiment program to connect on a custom port and/or to a custom host using the `RL_set_port()` and `RL_set_host()` Matlab functions.

For example, the following code:

```
>> RL_set_port(4097);
>> RL_set_host('yahoo.ca')
>> cd examples/skeleton
>> skeleton_experiment();
```

That command could give output like:

```
RL-Glue Matlab Experiment Codec Version: 1.0 ($Revision: 444 $)
     Connecting to rl_glue at host: yahoo.ca on port 4097
```

This works for agents, environments, and experiments. In practice though, remember that 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.

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

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.

6 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 MatlabDocs, we will give pointers were appropriate.

6.1 Types

6.1.1 Simple Types

Unlike the C/C++ codec, we will not be using `typedef` statements to create special labels for the types. Since Matlab is loosely typed, these things aren't so hard and fast:

- *reward* is `double`
- *terminal* is `int` (1 for terminal, 0 for non-terminal) We hope to replace these with boolean eventually.
- *messages* come as Java `strings` and can be returned as Matlab strings
- *task specifications* come as Java `strings` and can be returned as Matlab strings

6.1.2 Structure Types

All of the major structure types (observations, actions) come off the network as the appropriate object from the Java codec. The Java codec manual should have all the information required to understand those objects.

So in a given Matlab method, like
`function theAction=skeleton_agent_step(theReward, theObservation)`, `theObservation` is actually of type: `org.rlcommunity.rlglue.codec.types.Observation`.

Java and Matlab play very well together, so you can do things like:

```
>> testObs=org.rlcommunity.rlglue.codec.types.Observation();
>> testObs.intArray=[1 2 3 4];
>> testObs.doubleArray=[0.1 0.5];
```

```
>> testObs.charArray='fun things!';
>> testObs.toString()

ans =

numInts: 4
numDoubles: 2
numChars: 11
 1 2 3 4 0.1 0.5 f u n   t h i n g s !
```

6.2 Functions

6.2.1 Agent Functions

All agent constructor functions **should set** the same functions as our Skeleton agent.

Useful utility methods for connecting, disconnecting, and running with the `rl_glue` executable server are in the `agent` directory of the Matlab codec source.

6.2.2 Environment Functions

All environment constructor functions **should set** the same functions as our Skeleton environment.

Useful utility methods for connecting, disconnecting, and running with the `rl_glue` executable server are in the `environment` directory of the Matlab codec source.

6.2.3 Experiments Functions

All experiments **can call** the methods in the `glue` directory. In this case we'll include their prototypes, because the source file is full of implementation details.

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 Matlab Codec FAQ:
`http://glue.rl-community.org/Home/Extensions/matlab-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

Brian Tanner wrote the Matlab codec. He is also responsible for creating the installer, which is pretty nifty. Yay Brian.

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: 402 $
Last Updated By: $Author: brian@tannerpages.com $
Last Updated   : $Date: 2008-12-10 19:05:12 -0800 (Wed, 10 Dec 2008) $
$URL: https://rl-glue-ext.googlecode.com/svn/trunk/projects/codecs/Matlab/docs/MatlabCodec.tex
```