RL-Glue C/C++ Codec 2.0 Manual

Brian Tanner :: brian@tannerpages.com

Contents

1 Introduction

This document describes how to use the C/C++ RL-Glue Codec, a software library 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 documentation provided with that project.

---

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

This codec will allow you to create agents, environments, and experiment programs in C and/or C++.

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/c-c-codec`

This document should be used in combinations with the technical manual for RL-Glue 3.0. The RL-Glue 3.0 technical manual explains the data types, function prototypes, etc. for the main C/C++ RL-Glue implementation that is used by this codec. The RL-Glue 3.0 technical manual can be found in the `docs` section of the RL-Glue project or online:
`http://rl-glue.googlecode.com/svn/trunk/docs/tech_html/index.html`

1.1 Software Requirements

This project requires that RL-Glue has been installed on your computer. It has no additional requirements beyond RL-Glue: nothing more exotic than a C compiler, Make, etc. This codec uses a configure script that was created by GNU Autotools[3], so it should compile and run without problems on most *nix platforms (Unix, Linux, Mac OS X, Windows using CYGWIN[4]).

1.2 Getting the Codec

You can get the codec a number of ways, including from source as a .tar.gz file, or as a binary distribution.

All of the official downloads of the C/C++ codec can be found here:
`http://code.google.com/p/rl-glue-ext/wiki/CandCPP`

You may also check the code out directly from the subversion:
`svn checkout http://rl-glue-ext.googlecode.com/svn/trunk/projects/codecs/C C-Codec`

1.3 Installing from a Binary Package

1.3.1 Intel Mac OX 10.3+ Package

This distribution is an installer package bundled into a Mac Disk Image (.dmg). This is a graphical installer application and should be fairly self explanatory. This distribution comes with an uninstall

---

[2]`http://www.apache.org/licenses/LICENSE-2.0.html`
[3]`http://sources.redhat.com/autobook/`
[4]`http://www.cygwin.com/`

script that can be used to remove this codec from your system.

1.4 Installing From Source

The .tar.gz distribution of the codec was made with autotools, which means that you shouldn't have to do much work to get it installed.

1.4.1 Simple Codec Install

If you are working on your own machine, it is usually easiest to install the headers and libraries into **/usr/local**, which is the default installation location but requires *sudo* or *root* access.

The steps are:

```
>$ ./configure
>$ make
>$ sudo make install
```

Provided everything goes well, the headers have now been installed to **/usr/local/include** and the libs to **/usr/local/lib**.

NOTE: On many Linux systems, **/usr/local** is not actually on the library and header search paths by default, but **/usr** surely is. In this case, you may want to follow the instructions in Section 1.4.3 system with **--prefix=/usr**.

1.4.2 Install Codec when RL-Glue is in a custom location

If **configure** can't find RL-Glue installed on your machine, it will give you an error like the following:

```
checking for rlConnect in -lrlgluenetdev... no
configure: error: RL-Glue library not found.
You must have RL-Glue installed to use this codec.

If you have not downloaded it please see http://glue.rl-community.org/
If you do have it installed in a non-standard location you may need to use the
--with-rl-glue=/path/to/rlglue
command line switch to specify where the rl-glue root is located.
```

If you installed RL-Glue to some place other than **/usr/local**, say **/Users/btanner/tmp/rlglue**, you could do:

```
>$ ./configure --with-rl-glue=/Users/btanner/tmp/rlglue
>$ make
>$ sudo make install
```

1.4.3 Install Codec To Custom Location (maybe without *root* access)

You might want to install the codec to a location other than the default of `/usr/local`.

If you don't have *sudo* or *root* access on the target machine, you can install the codec in your home directory (or other directory you have access to). If you install to a custom location, you will need set your `CFLAGS` and `LDFLAGS` variables appropriately when compiling your agents, environments, and experiments. See Section 2.2 for more information.

For example, maybe we want to install the codec to `/Users/btanner/tmp/rlglue`. This will **not** clobber RL-Glue if it is already installed to this location, it will install beside it. The commands are:

```
>$ ./configure --prefix=/Users/btanner/tmp/rlglue
>$ make
>$ make install
```

Provided everything goes well, the headers and libraries have been respectively installed to `/Users/btanner/tmp/rlglue/include` and `/Users/btanner/tmp/rlglue/lib`.

1.4.4 Uninstall

If you decide that you don't want the RL-Glue C codec on your machine anymore, you can easily uninstall it. The procedures varies a tiny bit depending on if you installed it to the default location, or somewhere custom.

1.4.5 Codec Installed To Default Location

```
>$ ./configure
>$ sudo make uninstall
```

This will remove all of the headers and libraries from `/usr/local`.

1.4.6 Codec Installed To Custom Location

You'll need to make sure that either you haven't reconfigured the directory you downloaded from, or, if you removed/changed that already, you have to run configure again the exact same way as

when you installed it. For example:

```
>$ ./configure --prefix=/Users/btanner/tmp/rlglue
>$ make uninstall
```

That's it! This will remove all of the headers and libraries from **/Users/btanner/tmp/rlglue**.

2   Sample Project

We have included three example projects with this codec, located in the `examples` directory. The `skeleton-example` and `mines-sarsa-sample` projects each contain an agent, environment, and experiment written for this C/C++ codec.

The `skeleton-example` 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 3.1.

the `custom_integrated_env` in Section 3.1.4 shows how you can adapt this codec to help programs not originally meant for RL-Glue (Atari emulators, real-time strategy games, etc.) work with RL-Glue.

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

The `README` file in the specific examples directories may also be helpful in getting started.

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 files are:

```
examples/skeleton-example/SkeletonAgent.c
examples/skeleton-example/Makefile
```

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

If RL-Glue and this codec have been installed in the default location, `/usr/local`, then you can compile and run the agent like:

```
>$ cd examples/skeleton-example
>$ make
>$ ./SkeletonAgent
```

You will see something like:

```
RL-Glue C Agent Codec Version 1.0-alpha-3, Build 192:208M
Connecting to host=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.

See Section 2.2 if RL-Glue or this Codec are not installed in default locations.

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.

We will spend a little bit talking about how to compile the agent, because not everyone is comfortable with using a `Makefile`. To compile the agent from the command line, you could do:

```
>$ cc SkeletonAgent.c -lrlutils -lrlagent -o SkeletonAgent
```

On some platforms, you may need to add `-lrlgluenetdev`

It might be useful to break this down a little bit:

cc The C compiler. You could also use `gcc` or `g++`, etc.

SkeletonAgent.c Compile the SkeletonAgent.c source file.

-lrlutils Link to the RLUtils library, which comes with the RL-Glue project. That library contains convenience functions for allocating and cleaning up the structure types. If you don't use these convenience functions, you don't need this library. See the RL-Glue 3.0 technical manual for more information about RLUtils.

-lrlagent Link to the RLAgent library of the codec. This is where the main agent loop is defined. The main agent loop connects to the `rl_glue` executable server and dispatches commands sent by the glue.

-lrlgluenetdev Link to the RLGlueNetDev library from the RL-Glue project. This library is automatically linked through `rlagent` on most platform (except notably Cygwin). RLGlueNetDev provides implementations of the low level network code that is used by all three parts of the codec, as well as the `rl_glue` executable server.

7

2.2 Custom Flags for Custom Installs

If RL-Glue **or** this codec have been installed in a custom location (for example: `/Users/joe/glue`), or if `/usr/local` isn't in your computers default library and header search paths, then you will need to set the header search path in `CFLAGS` and the library search path in `LDFLAGS`. You can either do this each time you call make, or you can export the values as environment variables. These instructions apply to agents, environments, and experiment programs.

To do it on the command line:

```
>$ CFLAGS=-I/Users/joe/glue/include LDFLAGS=-L/Users/joe/glue/lib make
```

That might turn out to be quite a hassle while you are developing. In that case, you can either update the `Makefile` to include these flags, or set an environment variable. If you are using the bash shell you can `export` the environment variables:

```
>$ export CFLAGS=-I/Users/joe/glue/include
>$ export LDFLAGS=-L/Users/joe/glue/lib
>$ make
```

In some cases, you may be able to compile and link your programs without incident, but you receive shared library loading errors when you try to execute them, as mentioned in Gotchas! (Section 2.5.2).

The reason for these errors is that the search path of the loader does not include `/usr/local/lib`. This problem has both temporary and permanent fixes.

To temporarily fix the problem, you can set `LD_LIBRARY_PATH` (Linux) or `DYLD_LIBRARY_PATH` (OS X) environment variables, like:

```
>$ export LD_LIBRARY_PATH=/Users/joe/glue/lib
```

In some cases (64-bit linux) you may have to use this approach even when RL-Glue and this codec are installed in the default locations:

```
>$ export LD_LIBRARY_PATH=/usr/local/lib
```

When you open a new terminal window, all of these environment variables will be lost unless you put the appropriate `export` lines in your shell startup script.

The permanent solution requires `root` or `sudo` access to the machine. You can edit the file `/etc/ld.so.conf` and add a line like: `/usr/local/lib` to the file. Then, if you call `sudo ldconfig`, the loader will know to look there for libraries in the future.

2.3   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 files are:

```
examples/skeleton-example/SkeletonEnvironment.c
examples/skeleton-example/Makefile
```

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.

If RL-Glue and this codec have been installed in the default location, /usr/local, then you can compile and run the environment like:

```
>$ cd examples/skeleton-example
>$ make
>$ ./SkeletonEnvironment
```

You will see something like:

```
RL-Glue C Environment Codec Version 1.0-alpha-3, Build 192:208M
Connecting to host=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.

See Section 2.2 if RL-Glue or this Codec are not installed in default locations.

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.

Compiling the environment is almost identical to compiling the skeleton agent, except you need to link to the RLEnvironment library instead of RLAgent.

```
>$ cc SkeletonEnvironment.c -lrlutils -lrlenvironment -o SkeletonEnvironment
```

On some platforms, you may need to add -lrlgluenetdev

2.4   Skeleton Experiment

We have provided a skeleton experiment with the codec that is a good starting point for experiments 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-example/SkeletonExperiment.c
examples/skeleton-example/Makefile
```

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`.

If RL-Glue and this codec have been installed in the default location, `/usr/local`, then you can compile and run the experiment like:

```
>$ cd examples/skeleton-example
>$ make
>$ ./SkeletonExperiment
```

You will see something like:

```
RL-Glue C Experiment Codec Version 1.0-alpha-3, Build 192:208M
Connecting to host=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.

See Section 2.2 if RL-Glue or this Codec are not installed in default locations.

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.

Compiling the experiment is almost identical to compiling the skeleton agent, except you need to link to the `RLExperiment` library instead of `RLAgent`.

```
>$ cc SkeletonExperiment.c -lrlutils -lrlexperiment -o SkeletonExperiment
```

On some platforms, you may need to add `-lrlgluenetdev`

2.5 Gotchas!

2.5.1 Crashes and Bus Errors in Experiment Program

If you are running an experiment using `RL_step`, beware that the last step (when `terminal==1`), the action will be empty. If you try to access the values of the actions in this case, you may crash your program.

2.5.2 Shared Library Loading Errors

On some machines we've used, the codec installs without incident, but when the agent/environment/experiment is run, the system gives an error message similar to:

```
>$ ./SkeletonAgent: error while loading shared libraries: librlagent-1:0:0.so.1:
cannot open shared object file: No such file or directory
```

So far this has only happened on 64-bit Linux, both a machine at the University and a personal install of 64-bit Ubuntu, Hardy Heron.

The reason for the error is that `/usr/local/lib` is not in the the loader's search path. If this happens, the operating system might have an alternate search path, and might not be looking in `/usr/local/lib` for libraries. You can troubleshoot this problem by doing:

```
>$ LD_DEBUG=libs ./RL_agent
```

If you see that `/usr/local/lib` is not in the search path, you may want to add it to your library search path using `LDFLAGS` or `LD_LIBRARY_PATH`. See Section 2.2 for more information.

3 Running the Agent, Environment, and Experiment 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.

You can either execute the following four commands in the same terminal window if you put each on in the background (& on the end), or run them in separate terminal windows:

```
>$ rl_glue
>$ skeleton-example/SkeletonAgent
>$ skeleton-example/SkeletonEnvironment
>$ skeleton-example/SkeletonExperiment
```

If RL-Glue is not installed in the default location, you'll have to start the rl_glue executable server using its 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-alpha-3, Build 848:852M
RL-Glue is listening for connections on port=4096

>$ skeleton-example/SkeletonAgent &
RL-Glue C Agent Codec Version 1.0-alpha-3, Build 192:208M
Connecting to host=127.0.0.1 on port=4096...
RL-Glue C Agent Codec :: Connected
RL-Glue :: Agent connected.

>$ skeleton-example/SkeletonEnvironment &
RL-Glue C Environment Codec Version 1.0-alpha-3, Build 192:208M
Connecting to host=127.0.0.1 on port=4096...
RL-Glue C Environment Codec :: Connected
RL-Glue :: Environment connected.

$> skeleton-example/SkeletonExperiment


Experiment starting up!
RL-Glue C Experiment Codec Version 1.0-alpha-3, Build 192:208M
Connecting to host=127.0.0.1 on port=4096...
RL-Glue C Experiment Codec :: Connected
RL-Glue :: Experiment 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(C/C++) by Brian Tanner.


----------Sending some sample messages----------
Agent responded to "what is your name?" with:
my name is skeleton_agent!
Agent 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

Environment responded to "what is your name?" with:
my name is skeleton_environment!
```

```
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  100 steps   0.000000 total reward   0 natural end
Episode 1  44 steps  -1.000000 total reward   1 natural end
Episode 2  18 steps  -1.000000 total reward   1 natural end
Episode 3  100 steps   0.000000 total reward   0 natural end
Episode 4  50 steps   1.000000 total reward   1 natural end
Episode 5  1 steps   0.000000 total reward   0 natural end
Episode 6  28 steps   1.000000 total reward   1 natural end


----------Stepping through an episode----------
First observation and action were: 10 1

----------Summary----------
It ran for 144 steps, total reward was: -1.000000
```

3.1 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 C/C++ codec. The `mines-sarsa` sample project is much richer.

More details about the `mines-sarsa` sample project can be found at their RL-Library home:
`http://library.rl-community.org/packages/mines-sarsa-sample`

3.1.1 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[5] is manually created because there is not yet a task spec builder for C/C++.

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

3.1.2 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 C/C++ 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.

3.1.3 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.

3.1.4 Customized Codec for Flexible Integration

From time to time, someone wants to connect an agent or environment to RL-Glue which comes from an third-party code base that is not easy to fit into the mod of a standard RL-Glue module. Often the integration problems are related to the third-party code base defining a main method and controlling the execution flow of the program directly (instead of waiting for commands from the `rl_glue` server).

Two such examples that have happened already are Marc Lanctot's real time strategy engine that was used in the 2008 RL-Competition, and also the project to connect the Stella Atari emulator to RL-Glue.

We support a path for integrating these projects with RL-Glue that involves customizing this C/C++ codec, rather than trying too hard to link to it. We have provided an example in: `examples/custom_integrated_env`.

The relevant files are:
`TheGame.h`
`TheGame.c`
`Custom_Integrated_Env_Codec.c`

The idea in the example is that you have a game (called `TheGame`) which was never meant to be used with RL-Glue. To integrate it, a copy of `src/RL_client_environment.c` was modified to work specifically with `TheGame`. For example, all of the calls to `env_init`, `env_start`, `env_step`, etc. were replaced with calls to functions defined in `TheGame`. The input and output to those `TheGame` functions need to be wrapped up a little bit, but the end result is a fairly easy way to connect `TheGame` to RL-Glue!

Check out the source code in the example, and feel free to ask questions in the RL-Glue Google Group!

4 Codec Specification, Memory Allocation, Types, and Function Prototypes

There is some important information about memory management practices in RL-Glue, as well as detailed information about the C/C++ RL-Glue type definitions available in the RL-Glue 3.0 technical manual. We've opted to not duplicate those 6 pages of useful information here : but you really should go read them.

The main thing to realize is that agents, environments, and experiment programs are completely interchangeable between *direct-compile* (RL-Glue project) and *socket-communication* (this codec). An agent written for either one will be compiled the same way, the only difference is what library it is linked to. So, 95% of the technical manual from the RL-Glue project applies perfectly to people using this codec!

5 Advanced Features

5.1 Connecting to Custom Hosts and Ports

Sometimes you will want to connect to the `rl_glue` server over the network instead of on the local host. Othertimes you may want to run the `rl_glue` server on a port other than 4096 either because of firewall issues, or because you want to run multiple instances on the same machine.

In these cases, you can tell this codec to connect to custom port/host combinations using the environment variables `RLGLUE_HOST` and `RLGLUE_PORT`. These are both optional parameters.

For example, try the following code:

```
> $ RLGLUE_HOST=69.64.159.1 RLGLUE_PORT=1025 skeleton-example/SkeletonAgent
```

That command could give outcome:

```
RL-Glue C Agent Codec Version 2.0-RC1, Build 277
```

```
    Connecting to host=69.64.159.1 on port=1025...
```

Alternatively, you can actually use a host name:

```
> $ RLGLUE_HOST=rlai.net skeleton-example/SkeletonAgent
```

That command could give outcome:

```
RL-Glue C Agent Codec Version 2.0-RC1, Build 277
      Connecting to host=69.64.159.1 on port=4096...
```

If you don't like typing them every time, you can export them:

```
> $ export RLGLUE_HOST=rlai.net
> $ export RLGLUE_PORT=1025
> $ skeleton-example/SkeletonAgent
```

6   Changes and 2.x Backward Compatibility

There were many changes from RL-Glue 2.x to RL-Glue 3.x. Most of them are at the level of the API and project organization, and are addressed in the RL-Glue overview documentation. The technical changes are available in the RL-Glue Technical Manual.

7   Frequently Asked Questions

We're waiting to hear more of your questions!

7.1   Where can I get more help?

7.1.1   Online FAQ

We suggest checking out the online RL-Glue C/C++ Codec FAQ:
http://glue.rl-community.org/Home/Extensions/c-c-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.

Special thanks to Scott Livingston for creating the new C/C++ task spec parser.

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: 681 $
Last Updated By: $Author: brian@tannerpages.com $
Last Updated   : $Date: 2009-02-08 20:03:55 -0700 (Sun, 08 Feb 2009) $
$URL: https://rl-glue-ext.googlecode.com/svn/trunk/projects/codecs/C/docs/C-Codec.tex $
```