RL-Glue Python Codec 2.0 Manual

Brian Tanner :: brian@tannerpages.com

Contents

1 Introduction

This document describes how to use the Python 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.

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

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

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/python-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 Python.

The code has some optimizations that detect if you have NumPy[3] installed, and will speed up some parts of the code automatically in that case. We're working on looking at more ways to transparently make this codec faster.

Possible Contribution: Someone with Python experience could help us find out what version of Python is required to use this codec, and could help us update the codec to be as robust as possible to older versions. Also, if someone wants to speed up this codec, we have some ideas but we are not really Python programmers.

1.2 Getting the Codec

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

The .tar.tz distribution can be found here:
`http://code.google.com/p/rl-glue-ext/wiki/Python`

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

1.3 Installing the Codec

There are two options on how you can install the Python codec, depending on whether you'd like to use the automated method, or if you'd like to setup your Python paths yourself.

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

1.3.1 setup.py

You can install the Python codec the "usual" way, using the `setup.py` script that is made possible by Python Distutils[4].

There are many options for exactly how to build and where to install the codec. If you are interested in a custom install or installing to a custom location, we recommend reading the Python installation docs.

To see a list of the available options:

```
>$ cd /path/to/codecs/Python
>$ python setup.py --help
```

To install the Python codec into your system (you may need sudo or root access to do this):

```
>$ cd /path/to/codecs/Python
>$ python setup.py install
```

If you are not able to install the codec because it requires root access on your system and you do not have these privileges, you can do a local install of the codec in your user directory. Please see the Python installation docs for more details.

After installing the Python codec, you will be able to access all of the RL-Glue Python classes and functions without needing to specify them in your `$PYTHONPATH` environment variable.

Removing the Codec
Distutils does not provide an automated way to remove the Python modules that it installs, because in general there may be dependencies between modules and it does not want to break your system. We recommend that you only remove the installed Python codec from your system manually if you know what you are doing. Future versions of the codec will install over the existing one.

1.3.2 The no-install install

You may prefer not to install the codec, to instead specify the codec source files in your `$PYTHONPATH` variable when necessary. In this case, there is no real "installation" for the codec per-se. The important thing is to put the source code somewhere that is easy to get at.

For the rest of this section, we'll assume you've put this project in a subdirectory of your home directory called PythonCodec, so the `src` directory is at: `~/PythonCodec/src`.

Python will want to know where the codec source files are, so we'll frequently use code like:

[4] `http://docs.python.org/distutils/index.html`

```
>$ PYTHONPATH=~/PythonCodec/src python do_something_with_codec.py
```

You can make your life easier by adding the path to the codec source to your `PYTHONPATH` environment variable, something like (in Bash):

```
>$ export PYTHONPATH=~/PythonCodec/src:$PYTHONPATH
```

Now your commands can be less cluttered, the same as if you had installed the codec:

```
>$ python do_something_with_codec.py
```

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 Python 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.

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/skeleton_agent.py
```

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
>$ python skeleton_agent.py
```

You will see something like:

```
RL-Glue Python Agent Codec Version: 2.0 (Build 250)
     Connecting to 127.0.0.1 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.

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/skeleton_environment.py
```

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
>$ python skeleton_environment.py
```

You will see something like:

```
RL-Glue Python Environment Codec Version: 2.0 (Build 250)
     Connecting to 127.0.0.1 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.

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 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/skeleton_experiment.py`

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
>$ python skeleton_experiment.py
```

You will see something like:

```
Experiment starting up!
RL-Glue Python Experiment Codec Version: 2.0 (Build 250)
     Connecting to 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.

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 Running All Three Components 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 &
>$ python skeleton_agent.py &
>$ python skeleton_environment.py &
>$ python skeleton_experiment.py &
```

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-beta-1, Build 848:856
RL-Glue is listening for connections on port=4096

>$ PYTHONPATH=~/PythonCodec/src python skeleton_agent.py &
RL-Glue Python Agent Codec Version: 2.0 (Build 250)
     Connecting to 127.0.0.1 on port 4096...
     Agent Codec Connected
     RL-Glue :: Agent connected.

>$ PYTHONPATH=~/PythonCodec/src python skeleton_environment.py &
RL-Glue Python Environment Codec Version: 2.0 (Build 250)
     Connecting to 127.0.0.1 on port 4096...
     Environment Codec Connected
     RL-Glue :: Environment connected.

>$ PYTHONPATH=~/PythonCodec/src python skeleton_experiment.py &
Experiment starting up!
RL-Glue Python Experiment Codec Version: 2.0 (Build 250)
     Connecting to 127.0.0.1 on port 4096...
     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(Python) by Brian Tanner.


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

```
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, Python edition!
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  42 steps  1.0 total reward  1 natural end
Episode 1  28 steps  1.0 total reward  1 natural end

Episode 2  96 steps  -1.0 total reward  1 natural end
Episode 3  52 steps  1.0 total reward  1 natural end
Episode 4  100 steps  0.0 total reward  0 natural end
Episode 5  1 steps  0.0 total reward  0 natural end
Episode 6  82 steps  1.0 total reward  1 natural end


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


----------Summary----------
It ran for 66 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 Python 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`

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[5] is manually because there is not yet a task spec builder for Python.

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 Python 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 Python codec.

[5] `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 Python task spec parser 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. The Python parser (unlike Java) is not a task spec builder, so task specs must be constructed manually (for now) when using Python to implement environments.

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 Python 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 python skeleton_agent.py
```

That command could give output like:

```
RL-Glue Python Agent Codec Version: 2.0-RC1 (Build 446)
     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:

```
> $ 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 PythonDocs, 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. Since Python 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* are `strings`
- *task specifications* are `strings`

5.1.2 Structure Types

All of the major structure types (observations, actions) extend the `RL_Abstract_Type` class, which has there lists: for integers, doubles, and chars.

The class is defined as:

```
class RL_Abstract_Type:
    def __init__(self,numInts=None,numDoubles=None,numChars=None):
        self.intArray = []
        self.doubleArray = []
        self.charArray = []
        if numInts != None:
             self.intArray = [0]*numInts
        if numDoubles != None:
```

```
            self.doubleArray = [0.0]*numDoubles
        if numChars != None:
            self.charArray = ['']*numChars

    def sameAs(self,otherAbstractType):
        return self.intArray==otherAbstractType.intArray and
        self.doubleArray==otherAbstractType.doubleArray and
        self.charArray==otherAbstractType.charArray
```

The other types that inherit from `RL_Abstract_Type` but add no specialization are:

```
class Action(RL_Abstract_Type)
class Observation(RL_Abstract_Type)
```

The structure of the composite types are listed below. Note that this code is not accurate in terms of the available constructors, it is just meant to illustrate the member names.

```
class Observation_action:
    def __init__(self,theObservation,theAction):
        self.o = theObservation
        self.a = theAction

class Reward_observation_terminal:
    def __init__(self,reward, theObservation, terminal):
        self.r = reward
        self.o = theObservation
        self.terminal = terminal

class Reward_observation_action_terminal:
    def __init__(self,reward, theObservation, theAction, terminal):
        self.r = reward
        self.o = theObservation
        self.a = theAction
        self.terminal = terminal
```

The full definition are available in `types.py`.

5.2 Functions

5.2.1 Agent Functions

All agents **should implement** `rlglue.agent.Agent`.

5.2.2 Environment Functions

All environments **should implement** `rlglue.environment.Environment`.

5.2.3 Experiments Functions

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

```
# () -> string
def RL_init():

# () -> Observation_action
def RL_start():

# () -> Reward_observation_action_terminal
def RL_step():

# () -> void
def RL_cleanup():

# (string) -> string
def RL_agent_message(message):

# (string) -> string
def RL_env_message(message):

# () -> double
def RL_return():

# () -> int
def RL_num_steps():

# () -> int
def RL_num_episodes():

# (int) -> int
def RL_episode(num_steps):
```

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 Agent/Environment Loading

Historically, there was a different approach for loading agents and environments.

The old strategy was:

```
>$ python -c \"import rlglue.agent.AgentLoader\" agentName
```

That didn't seem as easy as it should be, so we changed things for this release, much like we did in the Java codec. Now, Python agents and environments can become self loading by adding a bit of code at the bottom of their source files, like:

```
#skeleton_agent.py
#top of file
from rlglue.agent import AgentLoader as AgentLoader

...

#bottom of file
if __name__=="__main__":
     AgentLoader.loadAgent(skeleton_agent())
```

Now, (as you recall) we can load the agent like:

```
>$  python agentfile.py
```

See `skeleton_environment` for instructions about how to do a similar thing for environments.

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 Python Codec FAQ:
`http://glue.rl-community.org/Home/Extensions/python-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

Mark Lee originally wrote the Python codec. Thanks Mark.

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

Jose Antonio Martin H. was kind enough to create the current task spec parser. Thanks!

James Bergstra added a bit of NumPy magic to get the Python codec speed comparable to the other languages. Thanks James.

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: 613 $
Last Updated By: $Author: brian@tannerpages.com $
Last Updated   : $Date: 2009-02-05 00:28:23 -0700 (Thu, 05 Feb 2009) $
$URL: https://rl-glue-ext.googlecode.com/svn/trunk/projects/codecs/Python/docs/PythonCodec.tex
```