

RL-Glue Python Codec 2.0 Manual

Brian Tanner :: brian@tannerpages.com

Contents

1	Introduction	2
1.1	Software Requirements	2
1.2	Getting the Codec	3
1.3	Installing the Codec	3
2	Agents	3
3	Environments	4
4	Experiments	5
5	Putting it all together	6
6	Who creates and frees memory?	8
7	Advanced Features	8
8	Codec Specification Reference	8
8.1	Types	8
8.1.1	Simple Types	8
8.1.2	Structure Types	9
8.2	Functions	10
8.2.1	Agent Functions	10

8.2.2	Environment Functions	10
8.2.3	Experiments Functions	10
9	Changes and 2.x Backward Compatibility	11
9.1	Agent/Environment Loading	11
10	Frequently Asked Questions	12
11	Credits and Acknowledgements	12
11.1	Contributing	12

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-Glue¹ project, please refer to the documentation provided with that project.

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

This software project is licensed under the Apache-2.0² 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://rl-glue-ext.googlecode.com>

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.

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.

¹<http://glue.rl-community.org/>

²<http://www.apache.org/licenses/LICENSE-2.0.html>

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/Python Python-Codec
```

1.3 Installing the Codec

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 document, 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:

```
>$ 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 just:

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

To be as general as possible, our examples will use the long version.

POSSIBLE CONTRIBUTION: IF someone wants to investigate the options for having a script “install” the python modules into a system classpath, or distribute “compiled” Python modules, we would like to have your help!

2 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 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
>$ PYTHONPATH=~/.PythonCodec/src 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.

POSSIBLE CONTRIBUTION: If you take a look at the agent and you think it's not easy to understand, think it could be better documented, or just that it should do some fancier things, let us know and we'll be happy to do it!

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

The pertinent file is:

```
examples/skeleton_environment/skeleton_environment.py
```

This environment is episodic, with 21 states, labeled $\{0, 1, \dots, 19, 20\}$. 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
>$ PYTHONPATH=~/.PythonCodec/src 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.

POSSIBLE CONTRIBUTION: If you take a look at the environment and you think it's not easy to understand, think it could be better documented, or just that it should do some fancier things, let us know and we'll be happy to do it!

4 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 2). 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
>$ PYTHONPATH=~/.PythonCodec/src 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.

POSSIBLE CONTRIBUTION: If you take a look at the experiment and you think it's not easy to understand, think it could be better documented, or just that it should do some fancier things, let us know and we'll be happy to do it!

5 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 &
>$ PYTHONPATH=~/.PythonCodec/src python skeleton_agent.py &
>$ PYTHONPATH=~/.PythonCodec/src python skeleton_environment.py &
>$ PYTHONPATH=~/.PythonCodec/src 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: 2:e:1_[i]_[0,20]:1_[i]_[0,1]:[-1,1]

-----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

6 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.

7 Advanced Features

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).

Someone should write this later (Opportunity to contribute!).

8 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.

8.1 Types

8.1.1 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`

8.1.2 Structure Types

All of the major structure types (observations, actions, random seed keys, and state keys) extend the `RL_Abstract_Type` class, which has three 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)
class Random_seed_key(RL_Abstract_Type)
class State_key(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:
    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`.

8.2 Functions

8.2.1 Agent Functions

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

8.2.2 Environment Functions

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

8.2.3 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):

# (State_key) -> void
def RL_set_state(sk):

# (Random_seed_key) -> void
def RL_set_random_seed(rsk):

# () -> State_key
def RL_get_state():

# () -> Random_seed_key
def RL_get_random_seed():

```

9 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.

9.1 Agent/Environment Loading

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

The old strategy was:

```

PYTHONPATH=~/.PythonCodec/src:/agent/src/path python -c \
    \"import rlgue.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 rlg glue.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:

```
PYTHONPATH=~/.PythonCodec/src 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.

However, if you love the old way, you can still do it like:

```
PYTHONPATH=~/.PythonCodec/src:/agent/src/path python -c \
    \"import rlg glue.agent.AgentLoaderScript\" agentName
```

10 Frequently Asked Questions

We're waiting to hear your questions!

10.1 Where can I get more help?

10.1.1 Online FAQ

We suggest checking out the online RL-Glue C/C++ 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.

10.1.2 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.

10.2 What's up with the Task Spec Parser?

We've distributed the existing task spec parser from the previous version of RL-Glue with this codec. However, it is not officially supported as we are trying to ratify a new and improved task specification language.

We will release an updated codec with certified task spec parser once the new language has been ratified.

For more information, please go here:

<http://glue.rl-community.org/Home/rl-glue/task-spec-language>

11 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.

11.1 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: 294 \$

Last Updated By: \$Author: brian@tannerpages.com \$

Last Updated : \$Date: 2008-10-01 21:24:00 -0600 (Wed, 01 Oct 2008) \$

\$URL: <https://rl-glue-ext.googlecode.com/svn/trunk/projects/codecs/Python/docs/PythonCodec.tex>