# RL-Glue Lisp Codec 1.0 Manual

Gabor Balazs
gabalz@gmail.com

## Contents

# 1 Introduction

RL-Glue codecs provide TCP/IP connectivity to the RL-Glue reinforcement learning software library. This codec makes possible to create agent, environment and experiment programs in the Common Lisp programming language.

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

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

## 1.1 Software requirements

We highly recommend the usage of the *ASDF library*[3], because the codec components are packaged this way.

The codec provides two (ASDF) packages, one of them (rl-glue-codec) is the codec itself which provides the connectivity with the RL-Glue component, the other (rl-glue-utilities) contains utilities which can be helpful during using the codec.

Required libraries for the codec (rl-glue-codec package).

usocket (http://common-lisp.net/project/usocket/)

Required libraries for the utilities (rl-glue-utils package).

none

These libraries are very portable among various Lisp implementations, so we hope the codec can be used most of them. However the performance section can worth to take a glance before choosing among them. If you have problems with your favorite, do not hesitate to contact us.

## 1.2 Getting the codec

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

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 co http://rl-glue-ext.googlecode.com/svn/trunk/projects/codecs/Lisp Lisp-Codec
```

[1] http://glue.rl-community.org/
[2] http://www.apache.org/licenses/LICENSE-2.0.html
[3] http://common-lisp.net/project/asdf/

## 1.3 Installation

The codec can be installed by ASDF-Install or manually copying the files.

ASDF is designed so to use symbolic links for organizing the package directory structure. If the file system does not support it (e.g. *FAT*[4], *NTFS*[5]), a few tricks have to be applied to get it work. More information can be found on http://bc.tech.coop/blog/041113.html or in the FAQ section of http://www.cliki.net/asdf. If you have this kind of system, just ignore the symbolic link descriptions of this manual and solve the problem described on these pages.

The following parts assume that the ASDF library has been installed previously. More information about this can be found on http://common-lisp.net/project/asdf/.

Optionally, if you want to put the compiled files user locally under your home instead of the ASDF source directory, you can use the ASDF-Binary-Locations library. More information about this can be found on http://common-lisp.net/project/asdf-binary-locations/.

Change to the Lisp-Codec directory.

```
$ cd Lisp-Codec
```

### 1.3.1 Installation by ASDF-Install

You need the ASDF-Install library for this. If you do not have this, the installation instructions can be found on http://common-lisp.net/project/asdf-install/.

On *Windows*[6] ASDF-Install is only supported officially under *Cygwin*[7], but it can be possible to use it without that (you have to test it on your own). A description can be found on http://sean-ross.blogspot.com/2007/05/asdf-install-windows.html.

Create the compressed ASDF packages.

```
$ tool/make-asdf-package.sh rl-glue-codec
$ tool/make-asdf-package.sh rl-glue-utils
```

Start the Lisp interpreter.

```
$ lisp
*
```

Set up ASDF, ASDF-Install (optionally ASDF-Binary-Locations) according to their manual.

Install the packages.

```
* (asdf-install:install "./rl-glue-codec.tar.gz")
* (asdf-install:install "./rl-glue-utils.tar.gz")
```

[4] http://en.wikipedia.org/wiki/File_Allocation_Table
[5] http://en.wikipedia.org/wiki/NTFS
[6] http://en.wikipedia.org/wiki/Microsoft_Windows
[7] http://www.cygwin.com/

### 1.3.2 Manual installation

This method assumes that you have installed the library requirements previously. The steps below are described for UNIX like systems, but the adaptation for other ones is straightforward.

If you downloaded the tarball, just copy the sources into the ASDF source directory.

```
$ cp -r src/rl-glue-codec asdf-source-directory
$ cp -r src/rl-glue-utils asdf-source-directory
```

If you checked out the SVN repository, you probably do not want to have the `.svn` directories under your asdf-source-directory. You can copy by skipping them with these commands.

```
$ cd src
$ find rl-glue-codec -name .svn -prune -o \( \! -name *~ -print0 \) |
  cpio -pmd0 asdf-source-directory
$ find rl-glue-utils -name .svn -prune -o \( \! -name *~ -print0 \) |
  cpio -pmd0 asdf-source-directory
$ cd ..
```

Create the symlinks (if your filesystem supports them).

```
$ cd asdf-system-directory
$ ln -s asdf-source-directory/rl-glue-codec/rl-glue-codec.asd rl-glue-codec.asd
$ ln -s asdf-source-directory/rl-glue-utils/rl-glue-utils.asd rl-glue-utils.asd
$ cd -
```

## 1.4 Uninstallation

The codec can be uninstalled by ASDF-Install or manually depending on how it was installed. If you do not need the dependent libraries installed for the codec, you have to uninstall them one by one.

### 1.4.1 Uninstallation by ASDF-Install

You had to install the codec by ASDF-Install to use this method.

Start the Lisp interpreter.

```
$ lisp
*
```

Set up ASDF, ASDF-Install according to their manual.

Uninstall the packages.

```
* (asdf-install:uninstall :rl-glue-codec)
* (asdf-install:uninstall :rl-glue-utils)
```

### 1.4.2 Manual uninstallation

Delete the source files.
```
$ rm -r asdf-source-directory/rl-glue-codec
$ rm -r asdf-source-directory/rl-glue-utils
```

Delete the symlinks (if you have them).
```
$ rm -r asdf-system-directory/rl-glue-codec.asd
$ rm -r asdf-system-directory/rl-glue-utils.asd
```

If you use ASDF-Binary-Locations, do not forget to delete the compiled Lisp files from its directory.

## 1.5 Credits and acknowledgment

Gabor Balazs wrote the Lisp codec. Great!

### 1.5.1 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.

# 2 Using the codec

This section describes how the codec can be used to create agents, environments and experiments in Lisp, and how they can be glued into a running system.

## 2.1 Packages

The codec has an own (ASDF) package named `rl-glue-codec`.
```
* (asdf:oos 'asdf:load-op :rl-glue-codec)
```

One can use the proper package qualified symbol names, e.g.
```
* (defclass my-agent (rl-glue-codec:agent) ...)
* (defmethod rl-glue-codec:env-init ((env my-env)) ...)
```

Or one can import all the symbols into one's package by the `:use` directive, and use the codec symbols without any package qualification (the further examples will assume this case).
```
* (defpackage :my-package (:use :rl-glue ...) ...)
```

There are a few utilities for the codec which can be useful. These can be accessed in the `rl-glue-utils` (ASDF) package. More information about them can be found in their section.

## 2.2 Types

There is an abstract data type, `rl-abstract-type`, which can contain integers, floating point numbers and a character string. Its slots can be accessed by the `int-array`, `float-array` and `char-string` functions. There are two macros for number array creation, `make-int-array` and `make-float-array`, which automatically set the type of the contained elements according to the codec requirements. The usage of them is strongly suggested.

| | |
|---|---|
| observation | rl-abstract-type |
| action | rl-abstract-type |
| reward | double-float |
| terminal | boolean |
| task specification | string |
| state key | rl-abstract-type |
| random seed key | rl-abstract-type |

## 2.3 Agents

On writing an agent first, you have to create an own agent class, e.g.

```
* (defclass my-agent (agent) ...)
```

Second, implement the following methods for it.

```
* (defmethod agent-init ((agent my-agent) task-spec) ...)
* (defmethod agent-start ((agent my-agent) first-observation) ...)
* (defmethod agent-step ((agent my-agent) reward observation) ...)
* (defmethod agent-end ((agent my-agent) reward) ...)
* (defmethod agent-cleanup ((agent my-agent)) ...)
* (defmethod agent-message ((agent my-agent) input-message) ...)
```

A detailed description of the methods can be obtained this way.

```
* (documentation #'<method-name> 'function)
```

When your agent is ready, you can run it.

```
* (run-agent (make-instance 'my-agent)
             :host "192.168.1.1"
             :port 4096
             :retry-timeout 10)
```

It will try to connect your agent to an RL-Glue component listening on `192.168.1.1` and port `4096`, waiting 10 seconds between the trials. Its detailed description can be checked this way.

```
* (documentation #'run-agent 'function)
```

It will prompt something like this, where each dot denotes a connection trial.

```
RL-Glue Lisp Agent Codec Version 1.0-RC2, Build 345
        Connecting to 192.168.1.1:4096 .... ok
```

## 2.4 Environments

On writing an environment, first you have to create an own environment class, e.g.
```
* (defclass my-env (environment) ...)
```

Second, implement the following methods for it.
```
* (defmethod env-init ((env my-env)) ...)
* (defmethod env-start ((env my-env)) ...)
* (defmethod env-step ((env my-env) action) ...)
* (defmethod env-cleanup ((env my-env)) ...)
* (defmethod env-message ((env my-env) input-message) ...)
```

A detailed description of the methods can be obtained this way.
```
* (documentation #'<method-name> 'function)
```

When your environment is ready, you can run it.
```
* (run-env (make-instance 'my-env)
           :host "192.168.1.1"
           :port 4096
           :retry-timeout 10)
```

It will try to connect your environment to an RL-Glue component listening on `192.168.1.1` and port `4096`, waiting 10 seconds between the trials. Its detailed description is here.
```
* (documentation #'run-env 'function)
```

It will prompt something like this, where each dot denotes a connection trial.
```
RL-Glue Lisp Environment Codec Version 1.0-RC2, Build 345
        Connecting to 192.168.1.1:4096 .... ok
```

## 2.5 Experiments

On writing an experiment, first you have to create an own experiment class, e.g.
```
* (defclass my-exp (experiment) ...)
```

Second, implement your experiment. For this the codec provides client functions which hides the necessary buffer handling and network operation. These are the following.

`rl-init`, `rl-start`, `rl-step`, `rl-cleanup`, `rl-close`, `rl-return`, `rl-num-steps`, `rl-num-episodes`, `rl-episode`, `rl-agent-message`, `rl-env-message`.

A detailed description of these functions can be obtained this way.
```
* (documentation #'<function name> 'function)
```

Do not forget to call the `rl-close` function at the end of your experiment, because it closes the network connection and so terminates the RL-Glue session.

## 2.6 Running

After you have an agent, an environment and an experiment, which could be written on any of the supported languages, you can connect them by RL-Glue.

First start the server.
```
$ rl_glue
```

You should see this kind of output on the server side.
```
RL-Glue Version 3.0-RC2, Build 909
RL-Glue is listening for connections on port=4096
        RL-Glue :: Agent connected.
        RL-Glue :: Environment connected.
        RL-Glue :: Experiment connected.
```

Then start the agent, the environment and the experiment.

The output of the Lisp agent.
```
RL-Glue Lisp Agent Codec Version 1.0-RC2, Build 345
        Connecting to 127.0.0.1:4096 . ok
```

The output of the Lisp environment.
```
RL-Glue Lisp Environment Codec Version 1.0-RC2, Build 345
        Connecting to 127.0.0.1:4096 . ok
```

The output of the Lisp experiment.
```
RL-Glue Lisp Experiment Codec Version 1.0-RC2, Build 345
        Connecting to 127.0.0.1:4096 . ok
```

## 2.7 Utilities

The utilities has an own (ASDF) package named `rl-glue-utils`.
```
* (asdf:oos 'asdf:load-op :rl-glue-utils)
```

### 2.7.1 Task specification parser

This is a parser for the *task specification language*[8]. Only the RL-Glue version 3.0 specification type is supported.

The parser can return a `task-spec` object from a specification string, e.g.
```
* (parse-task-spec
    "VERSION RL-Glue-3.0 PROBLEMTYPE episodic DISCOUNTFACTOR 1
     OBSERVATIONS INTS (3 0 1) ACTIONS DOUBLES (3.2 6.5) CHARCOUNT 50
     REWARDS (-1.0 1.0) EXTRA extra specification")
#<TASK-SPEC>
```

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

The `task-spec` object has the following slots.

`version`, `problem-type`, `discount-factor`,
`int-observations`, `float-observations`, `char-observations`,
`int-actions`, `float-actions`, `char-actions`,
`rewards`, `extra-spec`

Their names are appropriately show their functionalities according to the task specification documentation. The `int-` and `float-` observation and action slot values contain `int-range` and `float-range` objects appropriately. These have the `repeat-count`, `min-value` and `max-value` slots. For the latter two there are three special symbols which are `'-inf`, `'+inf` and `'unspec`. They are used to represent the NEGINF, POSINF and UNSPEC specification keywords.

The `task-spec` type supports the `to-string` generic function with which it can be converted to a string. There are also two other helper functions, namely `across-ranges` and `ranges-dimension`, which can be useful for range vector handling. Their documentation can be checked by the `documentation` function.

```
* (documentation 'rl-glue-utils:across-ranges 'function)
* (documentation 'rl-glue-utils:ranges-dimension 'function)
```

## 2.8 Examples

There is an (ASDF) package named `rl-glue-examples` located in the `example` directory.

```
* (push #P"/path/to/Lisp-Codec/example/" asdf:*central-registry*)
* (asdf:oos 'asdf:load-op :rl-glue-examples)
```

The `random-agent` chooses its actions randomly from the action space obtained from the task specification. An instance of this agent can be started by the `start-random-agent` macro of which arguments are the same as of the `run-agent` function. For connecting to a RL-Glue server on localhost and port 4096 the following is enough.

```
* (start-random-agent)
```

The `mines` environment is an episodic one. At the beginning of each episode the agents is put randomly onto the mine field. Its goal is to find the exit point without stepping to a mine. The rewards are $-1$ for each intermediate step, $-10$ for stepping to a mine and $+10$ for reaching the exit. An instance of this environment can be started by the `start-mines` macro of which arguments are the same as of the `run-env` function. For connecting to a RL-Glue server on localhost and port 4096 the following is enough.

```
* (start-mines)
```

The `episode-avg` experiment is simply runs the given number of episodes and prints a shy statistics at the end. An instance of this experiment can be started by the `start-episode-avg` macro. This requires a `num-episodes` argument which specifies the number of episodes to be run. The other arguments are the same as of the `rl-init` function. For connecting to a RL-Glue server on localhost and port 4096 the following is enough (which will run 50 episodes).

```
* (start-episode-avg 50)
```

The experiment will prompt something like this (each dot denotes an episode).

```
RL-Glue Lisp Experiment Codec Version 1.0-RC2, Build 345
        Connecting to 127.0.0.1:4096 . ok
Task spec was: VERSION RL-Glue-3.0 PROBLEMTYPE episodic DISCOUNTFACTOR 1
OBSERVATIONS INTS (0 107) ACTIONS INTS (0 3) REWARDS (-10 10) EXTRA
..................................................
--------------------------------------------------
Number of episodes: 50
Average number of steps per episode: 46.8
Average return per episode: -55.4d0
--------------------------------------------------
```

# 3 Further issues

## 3.1 Numerical encoding/decoding

Because Lisp has a platform independent internal structure for variable representation, we have to encode and decode from the platform dependent one supported by C. This generates some overhead which can be seen on the performance measurements. These functions can be found in the *rl-buffer.lisp* file.

## 3.2 64 bit architectures

By default the Lisp codec (even with 32 or 64 bit Lisp implementation) is configured for a 32 bit compiled RL-Glue component, but it is possible to reconfigure it for a 64 bit one. To do so, one has to modify the constants at the top of the *rl-buffer.lisp* file, and rewrite the floating point encoder/decoder functions. These are optimalized for 32 bit architectures because of performance issues. For a quick try, it can worth to probe the ieee-floats library at http://common-lisp.net/project/ieee-floats/. It is not used by default, because it turned to be slow on the 32 bit architecture performance tests.

## 3.3 Performance measurements

A few ad-hoc tests have been of which results should be taken with caution. These were done on a 1.8 Ghz IBM $T42$ laptop with 1 GB memory on Linux.

Test 1 contains big observations and shorter episodes. 50000 integers and 50000 double floats are allocated on each step and 200 steps are made by an episode.

Test 2 contains more typical observations and longer episodes. 5 integers and 5 doubles are allocated on each step and 5000 steps are made by an episode.

A few other codecs are also listed to make the results comparable.

| Language | Version | Steps per second | |
|---|---|---|---|
| | | Test 1 | Test 2 |
| GCC | 4.1.2 | 27.0 | 10729.6 |
| Java | 1.6.0_07 | 23.4 | 9920.6 |
| Python | 2.5.2 | 1.8 | 2423.7 |
| SBCL | 1.0.19 | 4.8 | 9107.5 |
| CMUCL | 19*d* | 4.0 | 8865.2 |
| CLISP | 2.46 | 0.6 | 2742.7 |

These measurements are just hints, they should not be used to compare the languages generally. The values may vary on different platforms and circumstances with language versions.