EIEvent: A quick start guide to processing events with the EI process

Russell G Almond
ralmond@fsu.edu

1 Installation

1.1 Prerequisites

The following piece of software need to be installed on your system. Follow local instructions.

Programming Environment You will need some basic command line programming tools. MacOS X users, download Xcode from the app store and install the command line options. Windows users there is a low memory path and a high memory path. Low memory: install the RTools package from (`http://cloud.r-project.org/bin/windows/Rtools/`). High memory, with more tools install Cygwin (`https://www.cygwin.com/`). Cygwin has most of the GNU command line tools (e.g., `grep` and `sed`) which are very useful managing text files.

Subversion (svn) You will need a subversion command line client to be able to download the development version EIEvent from the server. MacOS X, this is included in the Xcode command line tools. Windows, I like Tortoise SVN (`https://tortoisesvn.net`), but you need to do a custom install to request the command line tools.

ownCloud Get a username (usually first letter of your first name and your last name) and password from Russell. Next point your browser at `https://pluto.coe.fsu.edu/owncloruld`. You should now be able to use the web browser interface. The Physics Playground project should be in the folder `Shared/NSFCyberlearning`. You can download a client from `https://owncloud.org/download/`. This will make ownCloud operate like Dropbox. When it asks you for a repository name, put in `https://pluto.coe.fsu.edu/owncloruld`.

Mongo Database Follow the instructions at `https://www.mongodb.com/download-center/` to download and install MongoDB. Note that you want to install both the mongo database server and command line client (`mongo`), but also the `mongoimport` and `mongoexport` tools.

jq This is a json manipulation program which is used for filtering. You can get it from `https://stedolan.github.io/jq/downlaod`.

R Download and install R from `https://cloud.r-project.org/`.

R Studio Download and install R Studio from `https://www.rstudio.com/`.

R packages Download and install the following R packages: `jsonlite`, `mongolite`, `futile.logger` and `R.utils`. To do this from R Studio, select `Tools > Install Packages ...` then enter the packages names.

1.2 EIEvent Code

There are two local R packages necessary for running `EIEvent`. The first is called `Proc4`, and it provides some basic support services (support for writing JSON messages to the database and reading them back). The second is called `EIEvent`.

You can load these packages two ways: You can grab the latest tarballs[1] from Pluto, or you can download the working copy using Subversion.

[Pluto, by the way, is `https://pluto.coe.fsu.edu` ("coe" is College of Education). If you memorize this address, a lot of the URLs in this document simplify. For example, the ownCloud repository is at `https://pluto.coe.fsu.edu/` + `owncloud`. Bugzilla is at `https://pluto.coe.fsu.edu/` + `bugzilla`, &c.]

The simpler solution is to go to `https://pluto.coe.fsu.edu/Proc4/` and look for the latest versions of the `Proc4` and `EIEvent` packages. Download these to your local machine. You can do this through R Studio, by going to `Tools > Install Packages ...` and then select install from local file. You will be prompted to select one of the files you saved in the previous steps. You can also install these files from the command line using the command `R CMD INSTALL EIEvent.tar.gz`. (Replace the last with the full pathname if you are not in the correct directory. Windows users may need `Rcmd` instead of `R CMD`).

A more robust, and only slightly more difficult to implement, solution is to download the code from the Subversion (svn) repository `https://pluto.coe.fsu.edu/svn/commonn/`. (BTW, you can browse that location with a web browser to see the latest files, but its better to use to use svn to check the files out to make sure everything works together.) Subversion is a source code control system (similar to `git`). It works as follows: a programmer, or a user, "checks out" a copy of the latest version from the server, making a local copy. Unlike Dropbox, the local version does not automatically sync with the server, that needs to be done manually. When a programmer finishes making changes (and testing them), they "commit" the changes to the server. Then anybody who needs the fixed version can "update" their local copy to get the changes from the server. There are other features, related to merging changes from multiple programmers, and going back to previous versions, but for basic use, "check out", "commit" and "update" are all you need to know.

[1]The `.tar` extension is used by convention with "tape archive" files, which are used by Linux systems to bundle files together. The `.gz` extension means the archive has been compressed with `gnuzip` (sometimes other compression algorithms are used). These are called "tarballs".

An R package actually lives in a directory with a given structure. The name is the name of the package. There are some required files, `DESCRIPTION` and `NAMESPACE` which describe the package; a collection of source files in the directory `R`, manual pages in the directory `man`, and possibly other files and directories. In particular, things in the directory `inst` (for "install") get installed along with the library. I use that directory for test files, and for configuration scripts (in the `conf`) folder.

When you check out a project from svn, it makes a local copy on your computer. The easiest way to do this is to create an R project using R Studio. In R Studio, go to "New Project", then select from "Source Code Control" and the "Subversion" options. (If the subversion option is not active on your machine, this is because R Studio cannot find the command line subversion tool. Make sure that you installed the command line tools for subversion.) This will open a dialog box. Type in:

Repository Path `https://pluto.coe.fsu.edu/svn/common/Proc4/trunk`. (The first part is the URL of the repository, `Proc4` indicates that you want the Proc4 project, and `trunk` means that you want the most recent version.)

Username You can leave this blank. (You only need the username for committing changes in this repository.) If you want to make changes to the code, talk to Russell about getting an SVN username and password.

Project Name `Proc4`

Local Directory Whatever you like. I use `/home/ralmond/Projects`, a directory called "Projects" under my home directory.

If all goes well, you should see a dialog box talking about downloading files from the server (don't worry, there aren't a lot here) and when that finishes, you should see `DESCRIPTION` and other files in the `File` window tab in R Studio. Look for a window tab marked `Build` (this is by default in the upper left.) There should be a button marked "Compile and Reload" (or maybe "Build and Load", it seems to be marked different things in different versions). Press on that button. You should then see a bunch of messages and finally, the package should successfully load. (You may see a message about `timestamp` begin hidden by a new function; this is normal.)

Congratulations! You have successfully installed Proc4, it should now load every time you type `library(Proc4)` just like a normal R package.

Now, do the same thing for the `EIEvent` package. You will create a new project for this in R Studio. The only things you will change are the repository pathname, `https://pluto.coe.fsu.edu/svn/common/EIEvent/trunk` and the name of the project `EIEvent` (make sure you have the funky capitalization correct). Compile and load this package as well.

We are almost done. We need to note what to do when somebody finds a bug. First, the person who found the bug needs to go do Bugzilla and enter the bug. Bugzilla is at `https://pluto.coe.fsu.edu/bugzilla`. It requires a username and password, but anybody with a `fsu.edu` or `my.fsu.edu` can automatically enroll. Select the "Physics Playground" project and the "EI" component. Make sure there is enough information there so that the programmer can recreate the bug. Include (a) what you were trying to do, (b) what you expected to happen, (c) what actually happened, (d) steps to recreate, and (e) attach any special test files (e.g., new rules, or event sets).

When the bug is fixed and the fix is tested, the programmer will check in the changes to the svn repository, and send word around to update the code. To do this open the project which was affected (Proc4 or EIEvent; if both, you need to do this one at a time) in R Studio. Then look for a window tab called "SVN" (by default this is in the upper right). Look for the gear icon (and the word "More") and select the "Update" option. The changes should download. Then do the "Compile and Restart" option again. If both packages are affected, you need to do them in either order.

1.3 Physics Playground Specific Data

The R packages have the generic `EIEvent` system. We also need *PhysicsPlayground* specific rules and other configuration data. This is located on the ownCloud server (`https://pluto.coe.fsu.edu/c`
You can either use the desktop client, or download through the web. (The latter might be a better option if you are tight on disk space on your computer, as there is a lot of stuff in there, including everything that is in the Physics Playground Dropbox.)

For most people, the Physics Playground folder is at `Shared/NSFCyberlearning`. There are 3 subfolders in their of interest.

EvidenceID This contains both the configuration files and test data for the Evidence Identification (`EIEvent`) process.

EvidenceAc This contains both the configuration files and test data for the Evidence Accumulation (`EABN`) process.

FSUSSp2019Data This directory contains the downloaded data from the May 2019 FSUS field trial, as well as the configuration files used for that.

2 Configuring EIEvent

When `EIEvent` is running as a server process, the configuration files should be placed in a directory `/usr/local/share/Proc4`. (On *nix, including MacOS, this will need to be created by a super-user using `sudo mkdir /usr/local/share/Proc4`, but then can be assigned write permissions to the regular account using `sudo chown <username> /usr/local/share/Proc4`.)

Listing 1: Copy the script files from the library to the configuration directory

```
## Edit the next line reflect how you want to set things up.
P4local <- "/Users/ralmond/Projects/PhysicsPlayground/EvidenceID"
EIHome <- library(help="EIEvent")$path
file.copy(file.path(EIHome,"conf","EIini.R"),
          file.path(P4local,"EIini.R")
file.copy(file.path(EIHome,"conf","EILoader.R"),
          file.path(P4local,"EILoader.R"
file.copy(file.path(EIHome,"conf","EIEvent.R"),
          file.path(P4local,"EIEvent.R")
## Optional:  Unix users only
file.copy(file.path(EIHome,"conf","EIEvent"),
          file.path(P4local,"EIEvent")
```

Alternatively, if you are not running as a server (which is most of you), you can put that configuration directory anywhere. Choose a location that makes sense to you, and write down the pathname. You will need to modify the configuration files to point to that directory.

2.1 Copy the R script files to your working directory.

There are three R files needed to setup and run the EI process: `EIini.R`—common initialization code,—`EILoader.R`—loads the rules and other configuration information,—and `EIEvent.R`—script for running the engine. (There is also a file called `EIEvent` in the same place which is a bash shell script for running the EIEvent as a server, but most people won't need that.) These are supplied in the `EIEvent`, but copy them into the local configuration directory for ready access.

The R commands in Listing 1 will do the copying for you. Set the variable `P4local` to the directory you picked. (Windows users, the backslash character ('\') has a special meaning in R: it indicates that the next character in a string has a special interpretation. You need to either double the backslashes ('\\') or use a Unix-style forward slash ('/') in the pathnames.)

2.2 Download default configuration files from ownCloud

There are three kinds of data files in which the rules and other necessary configuration for `EIEvent` reside. The easiest way to get started is to copy the files from `Physics Playground` to use as templates. You can copy these out of the `EvidenceID` or the the `FSUSSp2019Data/Sp2019Rules` folder (if going from the latter folder, be sure not to change those files, as they are archival copies). The three types of files are as follows, with more information about the formats below:

Context Tables These are spreadsheets (csv files) which map between the Unity and internal names for the levels, as well as describe which context sets the levels belong in. The current files are `ContextSketching.csv` and `ContextManipulation.csv`.

Rule Files These are JSON files which list the rules in JSON format. For no particularly important reason, these live in the `Rules` subfolder of `EvidenceID`. The current rule files are `CombinedRules.json` (all of the scoring rules), `CorrectDuration.json` (the reduced set of scoring rules used during run time) and `TrophyHallRules.json` (special rules for passing the bank balance and solved level information back to the game).

Default Student Records The `EIEvent` process starts a new student by copying a default student record, so one is needed for each application supported. The file `defaultSR.json` has default student records for the four basic app.

2.3 Application IDs

Part of the `EIEvent` system (and indeed all of the processes in the four process framework) is the application ID (the `app` field of most objects) which defines what application it goes with. The syntax is a url-like structure that defines exactly who owns the design for the assessment. Consider the example, `ecd://epls.coe.fsu.edu/PhysicsPlayground/Sp2019/linear`. The `ecd:` indicates that this assessment was designed with evidence-centered assessment design. The string `epls.coe.fsu.edu` defines the organization (the Educational Psychology and Learning Systems department of the College of Education at Florida State University) that holds the designs. The string `PhysicsPlayground/Sp2019/linear` defines the assessment and particular version.

The `Proc4` package defines the special application ID, `ecd://epls.coe.fsu.edu/P4test` to use for testing. The *Physics Playground* project defined the following application IDs.

- `ecd://epls.coe.fsu.edu/PhysicsPlayground/Sp2019/linear`—The linear sequencing condition.
- `ecd://epls.coe.fsu.edu/PhysicsPlayground/Sp2019/adaptive`—The adaptive sequencing condition.
- `ecd://epls.coe.fsu.edu/PhysicsPlayground/Sp2019/linear`—The user controlled sequencing condition.

These should be used for analysis of operational data, and the testing one for testing.

The `EIEvent` process maintains separate collections of rules, contexts, student records (including the context) and event queues for each application ID. (Actually, these are the same collections in the database, but the process filters the collection so that it only looks at the ones with the appropriate application ID.) In particular, that means that the configuration information needs to be loaded once for each application ID being used.

There is nothing sacred about this list, and in fact the application ID is mostly a simple string in the code, so any application ID could be used for testing. However, the loader scripts would need to be modified to use the new application ID.

2.4 The Loader Script

The script `EILoader.R` loads the configuration information into the Mongo database. The `EIEvent` process keeps both its configuration information (the rules, contexts and default student record) and its data (the events, student records and messages) in the same database. The loader script reads this configuration information from a file and stores it in the database. It (or part of it) needs to be revisited when the rules, contexts or default student record is updated.

Listing 2 shows a portion of that script, with some extra annotation.

The input files are explained in detail below. A few notes about this script:

- The “CaptureListener” is explained in the next section.
- The `config.dir` needs to be customized depending on your local configuration.
- Any application ID is acceptable.
- All of the steps start with a command to clear the initial values from the database, that can be omitted, and it will just add stuff.
- There can be any number of context tables, but the `*INITIAL*` one is required.
- The rules code assume that the rules are in a subdirectory called `Rules`. That may need adjustment.
- The way the code is written, the `CorrectDuration` rules replace the `CombinedRules` before they are loaded. Adjust as needed.
- The `defaultSR.json` file has records for all four applications; this likely requires editing.
- The last line assumes that `mongoimport` is available on the search path (command line).

When the `EILoader.R` script is adjusted to your satisfaction, you can run it from the command line: `R --slave <EILoader.R`. You can also run it from R Studio by running the entire script.

Listing 2: EILoader script (with just one application)

```
## Initial Setup
library(EIEvent)
flog.threshold(INFO)
cl <- new("CaptureListener")

## Point this at your configuration directory
config.dir <- "/home/ralmond/ownCloud/Projects/NSFCyberlearning/EvidenceII

## Create an EIEngine object to communicate with database
engTest <- EIEngine(app="ecd://epls.coe.fsu.edu/P4test",
                    listeners=list(capture=cl))

## Read in context tables
sketchCon <- read.csv(file.path(config.dir,"ContextSketching.csv"))
manipCon <- read.csv(file.path(config.dir,"ContextManipulation.csv"))
initCon <- data.frame(CID="*INITIAL*",Name="*INITIAL*",Number=0)

## Clear old context tables and load new ones.
engTest$clearContexts()
engTest$addContexts(initCon)
engTest$addContexts(sketchCon)
engTest$addContexts(manipCon)

## Read in rules
sRules <- lapply(fromJSON(file.path(config.dir, "Rules",
                                    "CombinedRules.json"),FALSE),
                 parseRule)
## Note: as written, this overrides the previous line
sRules <- lapply(fromJSON(file.path(config.dir, "Rules",
                                    "CorrectDuration.json"),FALSE),
                 parseRule)
tRules <- lapply(fromJSON(file.path(config.dir, "Rules",
                                    "TrophyHallRules.json"),FALSE),
                 parseRule)
## Clear old rules and load
engTest$rules$clearAll()
engTest$loadRules(sRules)
engTest$loadRules(tRules)

## Load default student records.
engTest$userRecords$clearAll(TRUE)   #Clear default records
system2("mongoimport",args=sprintf("-d␣EIRecords␣-c␣States␣--jsonArray␣%s
                                   file.path(config.dir,"defaultSR.json"))
```

2.5 Context Descriptions

The `context` field of a rule object determines which game levels a are applicable for a given rule. This can be one of three things. A name of a specific game level (rule is applicable to only one level), a name of a set of levels (e.g., `Sketching`, rule is applicable to all levels in that set), or the special keyword `ALL` (rule is applicable to all levels). The middle category requires information about which rules belong in which level set. This is the role of the context tables.

Figure 1 shows an excerpt from the file `ContextSketching`. There are three required columns: `CID`, `Number` and `Name`. For game levels, the `Name` and `Number` columns should be exactly as they are in the game engine. Any discrepancy in the name reported by the game and in the context table will result in an "unknown level" warning. Only `ALL` rules will be processed for that level. There are also some levels with negative numbers; these are context sets. The `CID` (context id) is a compressed version of the name which corresponds to variable naming conventions (e.g., no spaces or other special characters).

Table 1: Selected rows and columns from the `ContextSketching` table.

CID	Number	Name	Sketching	RampLevels
Sketching	-100	Sketching Levels	0	0
RampLevels	-105	Ramp Levels	1	0
LeverLevels	-110	Lever Levels	1	0
CloudyDay	89	Cloudy Day	1	0
ClownChallenge	18	Clown Challenge	1	1
Cornfield	119	Cornfield	1	1
CosmicCave	112	Cosmic Cave	1	0
CrazySeesaw	95	Crazy Seesaw	1	0

There should be a column in the spreadsheet corresponding to each of the level sets. The name of the column should be the `CID` of the corresponding level set. The contents of the column should be 0 or 1; 1 if the level in the row belongs to the set in the column and 0 otherwise. Additional columns can be placed in the spreadsheet for documentation purposes.

There is a special context `*INITIAL*` which is used for events before the player starts a level in the game. This has number 0, and is required. The context table can be broken into a number of pieces for easier use. In particular, the current `Physics Playground` configuration has different context tables for the sketching and manipulation levels as there are different level sets that are applicable.

The command `eng$clearContexts()` clears out the previous context tables before loading the new ones. It is not necessary, but suppresses a lot of warning messages about overwriting existing contexts. Additional tables loaded later are added to the set. (Note that the function `engTest$addContexts()` takes a data frame as an argument, so these can

Listing 3: Count Air Resistance Manipulations Rule

```
[
  {
    "app": "ecd://epls.coe.fsu.edu/PPTest"
    "name": "Count Air Resistance Manipulations",
    "doc": "Increment counter if slider changed.",
    "verb": "Manipulate",
    "object": "Slider",
    "context": "Manipulation",
    "ruleType": "Observable",
    "priority": 5,
    "conditions": {
      "event.data.gameObjectType":"
         AirResistanceValueManipulator",
      "event.data.oldValue":{"?ne":"event.data.newValue"}
    },
    "predicate": {
      "!incr":{"state.observables.airManip":1}
    }
  }
]
```

be constructed in R as well as being read from a file.

[For future consideration: Matching the names was one of the points of failure for the system. We need a system for keeping the context tables in sync between EI, EA, and the game engine.]

2.6 Rules of Evidence

The rules of evidence file is a collection of `EIEvent::Rule` objects in JSON format. The format is documented elsewhere, so only some notes related to loading the files are given here. Listing 3 shows an example. Note that the JSON file should contain a list of rules; the lists should be surrounded by square brackets ('[',']'), and be separated by commas.

A few things to note when preparing the rules files.

- R seems to insists that all strings (including keywords) are enclosed in quotes, while Mongo and jq are more relaxed about this. It is best practice to make sure that the names are in quotes.

- The `name` is used as a key in the database, so the names should be unique.
- The `doc` field is optional (for humans only) but highly recommended.
- The `app` field is ignored when the rules are read in through the `engTest$loadRules()` command. The command will adjust this field so it matches `app(eng)`. On the other hand, it is necessary if reading the rules into the database directly using `mongoimport`.
- The `context` field must be an exact string match for one of the context IDs (values of the `CID`) column, or the special keyword `"ALL"`.
- It is recommended to use priority 5 for normal priority rules, and lower numbers for rules that must be run early and higher numbers for rules that must be run late.

The function `eng$rules$clearAll()` removes the existing rules (recommended to avoid name duplication errors) and `engTest$loadRules()` loads a list of `Rule` objects into the database, setting the `app` field to match `app(eng)` in the process. The function `parseRule` takes the output of `fromJSON` and coverts it into a rule object. Thus, the recommended way to parse a rule file is `lapply(fromJSON(`*`filename`*`, FALSE), parseRule)`.

2.7 Default Student Records

Observable variables which represent a count, an elapsed time, or a set, may need initialization. If these observables are specific to a level, this should happen with a Reset Rule for that variable. On the other hand, if the rules are to operate with the entire playing session, their initial values may need to be included in the default student record. For example, the default student record sets the initial bank balance of each player to $0. Listing 4 gives the default student record for the test application.

The `uid` should always be `"*DEFAULT*"`. The observables can be set as needed. The default record is loaded into the `States` collection in the `EIRecords` database, using a simple `mongoimport`. The function `eng$userRecords$clearAll(TRUE)` clears all the records including the default record (if `TRUE` is omitted, the default record is not cleared). The shell command `mongoimport -d EIRecords -c States --jsonArray defaultSR.json` is used to load the records (the `system2` command invokes the `mongoimport` function).

The complete `defaultSR.json` file has default records for all four applications. It may need to be edited depending on the goals.

3 Running EIEvent

The `EIEvent` process is run as a server or as a standalone application in basically the same way, using the script `EIEvent.R`. The script file has a number of `if (interactive())` calls

Listing 4: Default Student Record (test application only)

```
[
  {
    "app": "ecd://epls.coe.fsu.edu/P4test",
    "uid": "*DEFAULT*",
    "context": "*INITIAL*",
    "oldContext": "*INITIAL*",
    "timers": null,
    "flags": null,
    "timestamp": "2019-05-01 00:00:00 EDT",
    "observables": {
      "trophyHall": [],
      "bankBalance": 0
    }
  }
]
```

that make it behave slightly differently in the two modes. In particular, the configuration is set up to facilitate debugging in interactive mode, and to run quietly in server mode. Also, if an event file is supplied, then the `EIEvent` will score all of the events and then stop; if not, it will continue running accepting new events as they come into the database.

3.1 Configuring the initialization file

The file `EIini.R` contains a number of initialization details for the processes. The default location for this (running as a server process) is `/usr/local/share/Proc4/`, but this can be changed by editing the `EIEvent.R` file. Listing 5 contains the first part of the file, and Listing 6 contains the second part.

There are two lines in this file which need modification. If database security is turned on, then the `username` and `password` need to be set. However, by default, Mongo security is turned off, so delete the `username` and `password` options from the `EIeng.common` details. Second, adjust the logfile location to be something that makes sense on your system.

The rest of the file really doesn't need customization. It has to do with setting up listeners. When the `EIEvent` process issues a message because of a trigger rule, it is sent to the Listener. Currently, there are three kinds of listeners (these are set up in the `Proc4` package):

InjectionListener This simply saves the message in a field in the database. This is used to save the observables for the EA process.

Listing 5: EIini.R, first part

```
## These are application generic parameters
EIeng.common <- list(host="localhost",username="EI",password="secret",
                     dbname="EIRecords",P4dbname="Proc4",waittime=.25)

appstem <- basename(app)
## These are for application specific parameters
EIeng.params <- list(app=app)

logfile <- file.path("/usr/local/share/Proc4/logs",
                     paste("EI_",appstem,"0.log",sep=""))
```

UpdateListener This updates a record for the `uid` and `app` in a collection, overwriting the contents of the data (or other designated) field. This is used to update the `Player` field in the `Proc4` database, to update the bank balance and levels solved. Note that there is a `jsonEncoder` function that controls how the data are saved.

CaptureListener This is a dummy listener that is used for testing. It just builds a list of all messages created. These can be accessed using `cl$lastMessage()` and `cl$messages`.

The `Physics Playground` configuration requires the listeners in the `EIini.R` file, test configurations can get away with fewer. Note that running in interactive mode by default adds a CaptureListener to the listener set. Also note that the `EI.listenerSpecs` must be a named list. The example shown in Listing 6 should work for most purposes.

3.2 Running in manual model

The easiest way to run the `EIEvent` process is to open the file `EIEvent.R` in R Studio and run it line by line (or in chunks). The following goes over the major parts of the script noting opportunities for customization.

Listing **??** is the start; the first block gives the command arguments: these are set from the command line in server mode mode and are set in the file in interactive mode. Note that the `R.utils` package is only used for processing the command line arguments. Most of the customization is needed in this part of the file.

The arguments are as follows (the value in the parens is the name from the command line):

app (app) The application ID to run. This argument is required.

Listing 6: EIini.R, second part

```
## JSON converter for the trophyHall variables.
trophy2json <- function(dat) {
  paste('{', '"trophyHall"', ':','[',
        paste(
            paste('{"',names(dat$trophyHall),'":"',dat$trophyHall,'"}',
                  sep=""), collapse=", "), '],',
        '"bankBalance"', ':', dat$bankBalance, '}')
}

EI.listenerSpecs <-
  list("InjectionListener"=list(sender=paste("EI",appstem,sep="_"),
            dbname="EARecords",dburi="mongodb://localhost",
            colname="EvidenceSets",messSet="New Observables"),
       "UpdateListener"=list(dbname="Proc4",dburi="mongodb://localhost",
            colname="Players",targetField="data",
            messSet=c("Money Earned","Money Spent"),
            jsonEncoder="trophy2json"))
```

Listing 7: EIEvent.R, first part

```
library(R.utils)
library(EIEvent)

if (interactive()) {
  ## Edit these for the local application
  app <- "ecd://epls.coe.fsu.edu/P4test"
  loglevel <- "DEBUG"
  cleanFirst <- TRUE
  eventFile <- "/home/ralmond/Projects/EvidenceID/c081c3.core.json"
} else {
  app <- cmdArg("app",NULL)
  if (is.null(app) || !grepl("^ecd://",app))
    stop("No app specified, use '--args app=ecd://...'")
  loglevel <- cmdArg("level","INFO")
  cleanFirst <- as.logical(cmdArg("clean",FALSE))
  eventFile <- cmdArg("events",NULL)
}
```

Listing 8: EIEvent.R, second part

```
## Adjust the path here as necessary
source("/usr/local/share/Proc4/EIini.R")

if (interactive()) {
  flog.appender(appender.tee(logfile))
} else {
  flog.appender(appender.file(logfile))
}
flog.threshold(loglevel)
```

logLevel (level) How much logging should be done by the system. The legal values are (in order of increasing verbosity): `ERROR`, `WARN`, `INFO`, `DEBUG`, `TRACE`. The default is `DEBUG` for interactive mode and `INFO` for server mode.

cleanFirst (clean) If this is true, then various collections are cleaned before starting. If false, then processing continues from where it left off before. Be careful with this one as `cleanFirst=TRUE` will destroy data.

eventFile (events) This can be either a file name or `NULL`. If it is a file name, those events will be loaded in the database and the `EIEvent` process will be configured to stop when all events are processed. Otherwise, `EIEvent` will run until it both stops and and the `AuthorizedApps` flag is set to false.

Once the `app` variable has been set, the `EIini.R` script can be loaded to setup the local configuration. Listing **??** shows the relevant part of the `EIEvent.R` script. Note that the pathname of the `EIini.R` file needs to be adjusted. It also shows setting up the log file. By default, messages are "tee"d (sent to both console and file) in interactive mode and sent only to the log file in server mode.

The next step is to setup the `EIEvent` engine and its listeners. This is shown in Listing **??**. The only real bit of customization here is related to the use of a CaptureListener, `cl`. By default, this is set up in interactive mode, but not in server mode.

The next step is to clean old stuff out of the database (if the `cleanFirst` option was set). *Be careful! This destroys data, so make sure everything is backed up.* Listing **??** shows this part of the code. The collections cleaned are the events, the user records (except for the default), the message collection (all messages are logged here) and the specific message collections associated with UpdateListner and InjectionListener objects.

A common mode to run in testing is to input a set of event records associated with a particular player and level. In this case, the default behavior is to process all of the records

Listing 9: EIEvent.R, third part

```
## Setup Listeners
listeners <- lapply(names(EI.listenerSpecs),
                    function (ll) do.call(ll,EI.listenerSpecs[[ll]]))
names(listeners) <- names(EI.listenerSpecs)
if (interactive()) {
  cl <- new("CaptureListener")
  listeners <- c(listeners,cl=cl)
}
## Make the EIEngine
eng <- do.call(EIEngine,c(EIeng.params,list(listeners=listeners),
                          EIeng.common))
```

Listing 10: EIEvent.R, fourth part

```
if (cleanFirst) {
  eng$eventdb()$remove(buildJQuery(app=app(eng)))
  eng$userRecords$clearAll(FALSE)   #Don't clear default
  eng$listenerSet$messdb()$remove(buildJQuery(app=app(eng)))
  for (lis in eng$listenerSet$listeners) {
    if (is(lis,"UpdateListener") || is(lis,"InjectionListener"))
      lis$messdb()$remove(buildJQuery(app=app(eng)))
  }
}
```

Listing 11: EIEvent.R, fifth part

```
## Process Event file if supplied
if (!is.null(eventFile)) {
  system2("mongoimport",
          sprintf('-d␣%s␣-c␣Events␣--jsonArray', eng$dbname),
          stdin=eventFile)
  ## Count the number of unprocessed events
  NN <- eng$eventdb()$count(buildJQuery(app=app(eng),processed=FALSE))
}
if (!is.null(eventFile)) {
  ## This can be set to a different number to process only
  ## a subset of events.
  eng$processN <- NN
}
```

and stop. Listing **??** shows the code for setting this up. Note that this is mostly again a call to `mongoimport -d EIRecords -c Events --jsonArray` *`file.json`*, so it could be done from the command line, too.

The line `eng$processN <- NN` deserves special mention. The `processN` field of the engine tells the engine to process N events and then stop. It decrements the counter each time through, so to run for another N events, reset the counter and run `mainLoop(eng)` as well. If `processN` is set to `Inf`, the engine will run in infinite loop mode, not stopping until it detects the stop signal from the `AuthorizedApps` collection.

The next step is to run the main loop. Listing **??** shows two ways of doing this, all at once (the first two lines of code), and one event at a time (inside the `if`) statement. The `mainLoop` function will run until `processN` events are processed (or forever if `processN` is infinite). The other three statements must be run once for each event.

Stripped of its logging and checks for shutdown, the main event loop basically executes the last three commands over and over again. The first command fetches the oldest unprocessed event out of the database. (This event is bound to the variable `eve`). Next the event is "handled" (the student record is extracted from the database and the applicable rules are run and finally the updated state is saved to the database). If the event is handled without error, then `out` will be the updated state (which can then be inspected). If an error occurs, the `out` will be an error object. Finally, the event is marked as processed in the database.

By cleverly using `processN` and the manual event loop statements, it is possible to rapidly scan through the first few events, and then step more slowly through the events closer to the place where there is an issue. The debugging level may be changed at any time by calling `flog.threshold` again, so it can be kept at the INFO level early on and turned up

Listing 12: EIEvent.R, sixth part

```
## Activate engine (if not already activated.)
eng$activate()
mainLoop(eng)

## This is for running the loop by hand.
if (interactive() && FALSE) {
  eve <- eng$fetchNextEvent()
  out <- handleEvent(eng,eve)
  eng$setProcessed(eve)
}
```

Listing 13: EIEvent.R, last part

```
## This shows the details of the last message.
## If the test script is
## set up properly, this should be the observables.
if (interactive() && TRUE) {
  details(cl$lastMessage())
}
```

to DEBUG or TRACE only close to the problem.

The last line of the file checks the last message posted (by the last trigger rule) and prints it out. Listing ?? shows this line. Note that `cl$lastMessage()` shows the last message and `cl$messages` shows all of the messages. The function `details` shows only the message data, for "New Observables" messages, the observables.

3.3 Running in server mode

As mentioned in the previous section, exactly the same script is used to run the process in server mode. The major difference is that the arguments, particularly the app name which is required, are taken from the command line. (The names are in parentheses.) The file `EIEvent` is a bash shell script for running the process. [I'm assuming that server mode will mostly will be run on *nix systems, so most of this section will only give instructions for Linux.] An example command is:

The `nohup` command coupled with the `&` directive (run the package in the background) causes `EIEvent` to run as a server. If an events file is supplied, it will process all of the events and then stop; if not it will continue until shut down. Once again, be careful with the `clean`

Listing 14: Launching EIEvent

```
#!/bin/bash
P4=/usr/local/share/Proc4
nohup $P4/bin/EIEvent "app=ecd://epls.coe.fsu.edu/P4test
level=INFO␣clean=TRUE" >& $P4/logs/EI_userControl3.Rout &
```

Listing 15: mongo

```
  use Proc4;
  db.AuthorizedApps.update({app:{$regex:"P4test"}},
    {"$set":{active:false}});
```

flag, as this will wipe out data.

There is a way to shut down the server gracefully after it has finished processing all events. There is a collection called `AuthorizedApps` in the `Proc4` database. In that there is a record whose `app` field matches `app(eng)` and which has a logical active field. When the event loop runs out of events to process, it checks the value of that field, and if it is false, it shuts down. If not, it sleeps for a little while (the value of the `waittime` set in `EIinit.R`) and then checks again for events or the active flag to be cleared. The code `eng$active()` called in the script sets the active flag for the application.

The active flag can be cleared from the mongo shell using commands show in Listing 15.

[Note for future versions. There is probably a need for an emergency shutdown without continuing to process the queue events. Need to add this to a future version.]

4 Notes on Workflow

Yes, there is a fair amount to set up. Partially, this is because the code is still in development, but it hopefully should settle into a fairly straightforward workflow.

1. If there are bug fixes on `Proc4` or `EIEvent`, update and recompile those packages.

2. Modify rule files (or possibly context tables) to fix problems or add new features. Note that JSON files can be edited in any text editor, and R Studio seems to provide some support (e.g., checking for matching parenthesis and missing, stray commas).

3. If necessary build a test set consisting of events related to the problem or new feature. The test sets on ownCloud in the directory `EvidenceID/c081c3Tests` are a good place to start.

4. Run `EILoader.R` to load the updated rules and other configuration information.

5. Run `EIEvent.R` to process the events in the test set, and look at the output.

6. Lather, Rinse, Repeat.

It is quite possible that this testing will uncover bugs in `EIEvent`. These should be reported on `https://pluto.coe.fsu.edu/bugzilla`. Remember that each bug report should include:

- What you expected to happen and what actually happened.
- The version of the Rules you were using, if not the current *Physics Playground* rules.
- The test script you were running (or other details such as user ID and context ID if these are from the master event files.

Be patient, please. There is only one of me.