# Project 2

## Mobility and Replication for PDA processors

In this project, you will extent tour RESTful API from the previous project to support replication of PDA processors and client mobility with monotonic-write client consistency.

**What to do:**

**Task 5a [95% of points].**
Provide support for CRUD operations for replica groups of PDA processors.

A replica group consists of PDA processors with the same specification. Each replica group has a unique id (gid) and is accessible through its RESTful address. identified with a positive integer group id. Members of a replica group are specified by their RESFTful address.

You should have a replica server that is responsible for managing replica groups. There might be multiple servers running PDA processors.

To create a replica group, a user sends a request to the replica server providing the id of the replica group, an array of URLs for its PDA members and a JSON specification for the members. The replica server, create the replica group and it also creates (or replaces existing) member PDA processors with the given code.

You should also implement additional RESTful methods for basic management of replica groups: retrieve the array of addresses of the member PDA processors, reset all its member PDA processors, close all its member PDA processors, and delete the replica group and all its member PDA processors.

Further, to emulate client mobility, the replica server support a connect method for a given replica group, which returns the address of a random member PDA processor to the caller.

Moreover, you should implement two new methods for PDA processors. A method that returns a PDA processor's JSON specification, and a method that allows an existing PDA processor to join a replica group. When, a PDA processor joins a replica group it adopts the specification of the current member PDAs (if the group is non-empty).

Your implementation should provide for a monotonic writes client-centric consistency model. To this end, you should make appropriate changes to your PDA processors to maintain state information that enables you to efficiently provide monotonic writes client-centric consistency.

Furthermore, a client will be processing a single input-stream during a single session with a replica group, and the client may interact with multiple member PDA processors of a replica group. To efficiently support monotonic-writes client-centric consistency, a client will maintain some session state information (aka session cookie). A client should include its session cookie in appropriate RESTful API calls to PDA processors; and calls by a client to a PDA processor that were given a session cookie may return an updated value of the session cookie (as a JSON msg) for the client. A reset call returns a pristine (e. initial/empty) session cookie to a caller client.

**RESTful API**

The extended RESTful API you should support is given below. Let ***base*** be the URL for your servers: eg. http://localhost:8080/
As usual, any warning or error messages generated when calling methods of a PDA should be returned in the HTTP response.
Full credit will be given only to correct implementation with exactly-once RPC semantics.

| HTTP method | URL | HTTP Request Parameters | Meaning |
|---|---|---|---|
| GET | base/pdas | | Return a JSON list of ids of PDAs available at the server |
| PUT | base/pdas/*id* | pda_code | Create/replace at the server a PDA with the given id and the specification provided in the body of the request; calls open() method of PDA processor |
| PUT | base/pdas/*id*/reset | | Call reset() method |
| PUT | base/pdas/*id*/tokens/*position* | token_value, session_cookie | Present a token at the given position |
| PUT | base/pdas/*id*/eos/*position* | session_cookie | Call eos() with no tokens after (excluding) position |
| GET | base/pdas/*id*/is_accepted | session_cookie | Call and return the value of is_accepted() |
| GET | base/pdas/*id*/stack/top/*k* | session_cookie | Call and return the value of peek(k) |
| GET | base/pdas/*id*/stack/len | session_cookie | Return the number of tokens currently in the stack |
| GET | base/pdas/*id*/state | session_cookie | Call and return the value of current_state() |
| GET | base/pdas/*id*/tokens | session_cookie | Call and return the value of queued_tokens() |
| GET | base/pdas/*id*/snapshot/*k* | session_cookie | Return a JSON message (array) with three components: the current_state(), queued_tokens(), and peek(k) |
| PUT | base/pdas/*id*/close | | Call close() |
| DELETE | base/pdas/*id*/delete | | Delete the PDA with name from the server |
| | | | |
| GET | base/replica_pdas | | Return list of ids of replica groups currently defined |
| PUT | base/replica_pdas/*gid* | pda_code, group_members | Define a new replica group with the given member PDA addresses sharing the specification given in pda_code; create/replace the group members (as needed). |
| PUT | base/replica_pdas/*gid*/reset | | Reset the member PDAs |
| GET | base/replica_pdas/*gid*/members | | Return a JSON array with the addresses of its members |
| GET | base/replica_pdas/*gid*/connect | | Return the address of a random member that a client could connect to |
| PUT | base/replica_pdas/*gid*/close | | Close the members PDAs |
| DELETE | base/replica_pdas/*gid*/delete | | Delete replica group and all its members |
| PUT | base/pdas/*id*/join | replica_group | The PDA id joins the replica group with the given address |
| GET | base/pdas/*id*/code | | Return JSON specification code of the PDA |
| GET | base/pdas/*id*/c3state | | Return JSON message of the state information maintained to support client-centric consistency. |

## Task 6 [5% of points].

Implement a driver along the lines of the Task 5 to create a replica group of PDA processors, process an input-stream by contacting a random PDA processor of the group for each input token, and upon completion delete the replica group. Your driver should include additional appropriate calls demonstrating the behavior of your implementation of Task 5.

Your driver for this task can be a simple HTML file that contains a sequence of hyperlinks for the appropriate HTTP requests to your server; or a Bash script that sends the relevant HTTP requests to your server.

**What to submit:**

Submit a .tar.gz archive with your

- complete GoLang code
- sample input files (with PDA specs or marshalled token-streams)
- a Bash script for startup/shutdown of your server
- a (Bash/HTML) script file with commands demonstrating the execution of your client/driver with different command-line arguments
- sample output (text output of screen shots) of the execution of your client
- README file (with relevant documentation and usage guidance)