ascrPJs Engine API & Tutorials

Overview:

The Parallel JavaScript simulation engine, or PJs engine, is a tool that can be used to create simulations that are distributed across multiple webpages. Each local process in the simulation is defined by a state, which is changed based on a series of events. Events can be sent to and received from other local processes. The engine uses an optimistic algorithm that uses rollbacks, to ensure synchronization between the processes. Follow the steps below to implement a simulation using the PJs engine.

Getting Started:

1. Create a webpage for each local process that you want in the simulation.

2. Add a global script reference to jQuery on each page (<https://ajax.googleapis.com/ajax/libs/jquery/2.1.1/jquery.min.js>)

3. Add a script reference to the simulation library on each page (<http://adittechvm1.cloudapp.net/simulation/lib/simulation.min.js> or put a copy of this library on your own server and reference that).

4. Add an onload handler to the page which implements the Simulation object (following the tutorial below).

5. Install NodeJS (<https://nodejs.org/en/>) 4.2.2 or above on your supporting backend server (or computer for local testing).

6. Open a command line and navigate to the /node folder of the project.

7. Execute this command ‘node server.js [simulation domain name] [port number]’ where the simulation domain name is the domain of the pages that contain the code for the local processes, and port number is the port on which to communicate using the server.

8. Open each page on a browser and click ‘Start Simulation’ on the admin process (the page that connected to the backend first) and your simulation will start.

9. Use both the console in the browser, and the output of the node process to help debug any errors.

Creating a Simulation:

The only piece of the library that is necessary to be implemented for a page to be a local process in the simulation is the Simulation object. This object has the following parameters in the constructor:

* Simulation Name – This parameter specifies what the simulation will be called. This variable is especially important, because when other processes send a message to this process, they will need to pass this name.
* Server Root – This parameter specifies the root url of the supporting backend server. It should follow this pattern ‘http://[server domain (localhost if testing)]:[port number specified in step 6 of Getting Started]’.
* State – This parameter is a key/value pair object that contains the initial starting state of the local process. Keys of this object can be of any name except for ‘time’. The variable time is used by the engine, and must not be specified or altered at any time.
* Input Queue – This parameter is an array of Event objects that the local process will start processing once the simulation has begun. The events should be in chronological order by time stamp when added to the queue. Read more about creating event objects below.
* Draw Function – This parameter is a function which will be called after each execution of an event. This function will be passed one parameter: the current state. This function will use the current state to convey the information to the user (in the sample code a canvas is used, but this method is flexible enough to allow a user to define any methodology of conveying data).

Creating an Event:

An event defines the progress of a simulation. Events take the following parameters to be created:

* Time Stamp – This parameter is an integer that indicates at which time the execute function should be called.
* Execute Function – This parameter is a function that defines how a state should be changed and what messages should be sent when an Event is executed. The parameters are:
  + Event – This is the event that is currently being executed. This is passed as a variable so that the same execution function can be used for multiple events, and all still have access to the scheduleEvent function.
  + State – This is the current state of the process, which should be changed.

To change a state property, simply change the value of one of the keys in the state parameter. The execute function must then return the state variable, whether it was augmented or not. To send an event to the destination process defined in the constructor, call the Event.schduleEvent method which takes the following parameters:

* Simulation – The main Simulation object that was created for this page
* Event – The Event object to send

The execute function needs to contain any follow up functions in its scope in order to execute them. It is recommended that global execute functions be defined in a separate JavaScript file that is referenced by all processes pages, and calls to these global functions are used as execute functions, since long functions will need to be passed over query string. However this is not how the sample code is implemented.

* Destination – This parameter is a string that represents the name of the process to send all follow up events to. The destination can be changed dynamically, however it is best practice to have one Event have one destination, and create different Event objects for sending follow-up events to different destinations.

Sample:

To see an example implementation of this library, see the /sample folder of the project. It contains pages which implement an air traffic control simulation. You can see the simulation execute by going to the following URL’s:

<http://adittechvm1.cloudapp.net/simulation/sample/ATL.html>

<http://adittechvm1.cloudapp.net/simulation/sample/LAX.html>

<http://adittechvm1.cloudapp.net/simulation/sample/CHI.html>