SPK and Associates

www.spkaa.com
Ph: 888-310-4540

SPK and Associates
900 E Hamilton Ave, Ste.100
Campbell, CA 95008

How to Obtain Electric Commander Metrics for Use in a GWT Plugin

Developed by Electric Cloud, Electric Commander is an extremely versatile tool for enhancing and automating your organization's software development build-test-deploy cycle. Commander makes it possible to eliminate unnecessary lag time between phases of the development process, such as hand-offs between your development team and QA, but almost more importantly, it introduces standardization, repeatability, and reliability to your process. Those three things are key ingredients for producing meaningful metrics.

Out of the box Commander generates a variety of useful statistics as it executes jobs: When a job was launched, who launched it, how long did the whole thing take, how long did specific steps take, what was the outcome, why did a step fail, where artifacts were stored, etc. Commander also allows users to create their own properties, specific to their needs. All of these properties are exposed to the user through a variety of tools, one of which is the GWT (Google Web Toolkit) Commander SDK.

A primary use for the GWT Commander SDK is the creation of custom dashboards to display pertinent metrics. Keep reading as we describe the general process of using the GWT Commander SDK to extract data from Commander for the purpose of developing meaningful metrics.

David Hubbell
SPK Software Engineer

www.spkaa.com
Ph: 888-310-4540

SPK and Associates
900 E Hamilton Ave, Ste.100
Campbell, CA 95008

The first step towards obtaining a set of useful data is defining our search criteria. In order to do this we create a FindObjectsRequest object and add the appropriate search filters.

```
FindObjectsRequest req = getRequestFactory().createFindObjectsRequest( ObjectType.job );

req.addFilter( new FindObjectsFilter.EqualsFilter( "projectName", proj ) );
req.addFilter( new FindObjectsFilter.EqualsFilter( "scheduleName", sched ) );
req.addFilter( new FindObjectsFilter.GreaterThanFilter( "start", startDate ) );
```

We can also modify the sort order to suit your needs

```
req.addSort( "start", Order.ascending );
```

The second step involves defining the callback – the code that processes the response for the specific search request we have defined.

```
req.setCallback( new FindObjectsResponseCallback() {
        @Override
        public void handleResponse( FindObjectsResponse response ){
                // code to handle the respose
        }

        @Override
        public void handleError( CommanderError error ){
                // code to handle error responses
        }
});
```

The third step is to submit the request to the server using the doRequest.

```
doRequest( req );
```

This is the basic process for performing a search with a single set of criteria. When doRequest is called the browser communicates its request to the server and waits for a response. When the server sends its response, the code inside the handleResponse or handleError methods of our anonymous FindObjectsResponseCallback object is executed. Inside those methods is where the actual work of parsing the response takes place.

Sometimes, especially when dealing with large chunks of data, the server can take several seconds to return with its response. We want to minimize the number of requests

www.spkaa.com
Ph: 888-310-4540

SPK and Associates
900 E Hamilton Ave, Ste.100
Campbell, CA 95008

we send to the server, so when we have multiple pieces of information, even for unrelated things, it is best to group them together in an array and send the entire array in a single request. Because each FindObjectsRequest element in the array will have already defined its own individual handleResponse method when setting its callback, we know they will be processed appropriately.

Bundling multiple searches into a single request requires that we use a ChainedCallback object when submitting the request to the server. In addition to letting us bundle our searches, ChainedCallback objects provide ability to define actions that should take place only after all the results of been returned. We don't know how quickly or in what order we will get responses from the server for the various requests we send. Waiting until all the data is in before performing calculations is vitally important if we hope to compare data across multiple search results.

In the example that follows we see how we can create multiple FindObjectRequests, group them together in an array and then make a single connection to the server to send those requests all at once with a ChainedCallback object.

```
private void getTableInfo() {
        // PROJECTS_TO_MONITOR is an array of Strings representing the names of projects
        // similarly, JOBS_TO_MONITOR contains the names of scheduled jobs we're interested in
        ArrayList<FindObjectsRequest> allRequests = new ArrayList<FindObjectsRequest>();
        for( String proj : PROJECTS_TO_MONITOR ){
                for( String sched : JOBS_TO_MONITOR ){
                        allRequests.add( infoRequest( proj, sched ) );
                }
        }
        sendRequests( allRequests );
}

        private FindObjectsRequest infoRequest( String proj, final String sched ){
        FindObjectsRequest req = getRequestFactory().createFindObjectsRequest( ObjectType.job );

        req.addFilter( new FindObjectsFilter.EqualsFilter( "projectName", proj ) );
        req.addFilter( new FindObjectsFilter.EqualsFilter( "scheduleName", sched ) );
        req.addFilter( new FindObjectsFilter.GreaterThanFilter( "start",startDate ) );

        req.addSort( "start", Order.ascending );

        req.setCallback( new FindObjectsResponseCallback() {
                @Override
                public void handleResponse( FindObjectsResponse response ){
```

www.spkaa.com
Ph: 888-310-4540

SPK and Associates
900 E Hamilton Ave, Ste.100
Campbell, CA 95008

```
                int numPassed = 0;
                int numFailed = 0;
                for( Job job : response.getJobs()){
                        if( JobOutcome.success == job.getOutcome() ){
                                numPassed++;
                        }
                        else
                                numFailed++;
                        }
                        PassFailGraph graph = new PassFailGraph (numPassed, numFailed );
                        updateTable( sched, 1, graph );
                }

                @Override
                public void handleError( CommanderError error ){}
        });
        return req;
}

private void sendRequests( ArrayList<FindObjectsRequest> requests ){
        FindObjectsRequest[] allRequests = new FindObjectsRequest[ requests.size() ];
        requests.toArray(allRequests);
        doRequest( new ChainedCallback() {
                @Override
                public void onComplete() {
                        // something we want to do after all results are in...
                        compareSuccessRatesAcrossJobs();
                }
        }, allRequests );
}
```

Sometimes obtaining certain types of information requires sending multiple requests to the server. For instance, obtaining the elapsedTime property of a step requires that we first search for the group of jobs we want to examine, obtain their jobId properties, and then create a GetPropertyRequest utilizing the jobId value as part of its property request string. Creating the GetPropertyRequest is very similar to the process for creating a FindObjectsRequest object.

www.spkaa.com
Ph: 888-310-4540

SPK and Associates
900 E Hamilton Ave, Ste.100
Campbell, CA 95008

```
private void findElapsedTimeForSpecifiedJobs() {
        ...
        GetPropertyRequest[] propertyRequests =
                        new GetPropertyRequest[ JOBS_TO_MONITOR.length ];
        propertyRequests = createElapsedTimeRequests();

        doRequest( new ChainedCallback() {
                @Override
                public void onComplete(){
                        // do stuff....
                }
        }, propertyRequests );
}

private GetPropertyRequest[] createElapsedTimeRequests() {
        ArrayList<GetPropertyRequest> propertyRequests =
                        new ArrayList<GetPropertyRequest>();
        for( String jobId : myArrayOfJobIds ) {
                for( final String stepName : STEPS_TO_MONITOR ) {
                        GetPropertyRequest propReq =
                                getRequestFactory().createGetPropertyRequest();
                        propReq.setPropertyName(
                                "/jobs/"+jobId+"/jobSteps/"+ stepName +"/elapsedTime" );
                        propReq.setCallback( new DefaultPropertyCallback( this ) {
                                @Override
                                public void handleResponse( Property property ) {
                                        // process response ...
                                }
                                @Override
                                public void handleError( CommanderError error ) {}
                        });
                        propertyRequests.add( propReq );
                }

        }
        GetPropertyRequest[] requestArray =
                         new GetPropertyRequest[ propertyRequests.size() ];
        return propertyRequests.toArray( requestArray );
}
```