ICG LUX Rest Interface

Change Log

Version	Author	Date	Change Description
1.0	Scott Zimmer	12/27/2014	Initial Version
1.1	Bob Novas	12/30/2014	- added JSON objects - added consumed and response fields - added response codes - added GoDaddy Cert (not in jre cacerts) added optional rolename to add project acl
1.2	Bob Novas	1/7/2015	- added Usage Notes section - documented GET and DELETE methods for User and Project - changed form of User and Project apis slightly - replaced username with userId in most services - added Watchboard services
1.3	Bob Novas	1/13/2015	- POST and PUT on Projects and Watchboards can set ACLs on the object - corrected some incorrect Response Codes - tweaked some grammar.
1.4	Bob Novas	11/13/2015	- added AOI for Managed AOI
1.5	Bob Novas	12/1/2015	- updated AOI per implementation
1.6	Bob Novas	12/11/2015	- added note about accept query parameter
1.7	Bob Novas	2/26/2016	- added aoi tagQuery parameter
1.8	Bob Novas	8/3/2016	- added 3 levels to TOC, added page numbers.
1.9	Bob Novas	8/18/2016	- added POST/PUT/DELETE AOIs
1.10	Bob Novas	8/23/2016	- added section on JWTs
1.11	Bob Novas	9/21/2016	- added note about encoding + as %2B
1.12	Tom Foughner	1/22/2018	- added rule calls

TABLE OF CONTENTS

Table of Contents

Background

The LUX REST API is a json and XML programming interface to the ICG LUX system.

General Information

Base URI

https://dev4.icgsolutions.com/lux/rest/v2. The REST services are json by default. XML can be used by appending .xml to the full URI.

REST Service Public Key

GoDaddyG2RootCert.crt

Header Formats

Authorization: Bearer <JWT>
Accept: application/json
Content-Type: application/json

Usage Notes

An admin user creates a project or a watchboard and makes themselves the owner. Then, that admin user can see the projects and watchboards that they've created. The admin can adds a user to a project or a watchboard as an ADMIN, PARTICIPANT, or WATCHER.

Service Resources

Users

Get Users

Verb	GET
Template	/users
Produces	application/json
Consumes	None
Request Headers	Authorization: Bearer <jwt> Accept: application/json
Response Headers	TBD
Produced Entity	List of User objects
Consumed Entity	None
Response Codes	200 Success 401 Authentication failed 403 Not Authorized (caller must be an Admin user) 500 Server Error

Get User by UserId

Verb	GET
Template	/users/{userId} OR /users?userId={userId}
Produces	application/json
Consumes	None
Request Headers	Authorization: Bearer <jwt> Accept: application/json
Response Headers	TBD
Produced Entity	User object
Consumed Entity	None
Response Codes	200 Success 401 Authentication failed 403 Not Authorized (caller is not an Admin user or the userId specified is not the userId of the user making the call) 404 Not Found (userId doesn't exist, only returned if the user is Authorized) 500 Server Error

Get User by Username

Verb	GET
Template	/users?username={username}
Produces	application/json
Consumes	None

Request Headers		**Authorization: Bearer <jwt> Accept: application/json**
Response Headers		**TBD**
Produced Entity		**User object**
Consumed Entity		**None**
Response Codes		**200 Success 401 Authentication failed 403 Not Authorized (caller is not an Admin user or the username specified is not the username of the user making the call) 404 Not Found (username doesn't exist, only returned if the user is Authorized) 500 Server Error**

Create User

Verb		**POST**
Template		**/users**
Produces		**application/json**
Consumes		**application/json**
Request Headers		**Authorization: Bearer <jwt> Accept: application/json**
Response Headers		**TBD**
Produced Entity		**User object**
Consumed Entity		**User object username - type:String unique: yes, min length: 4, max length: 50 id – must be null**
Response Codes		**201 Success 400 Invalid request (i.e., User entity validation failed) 401 Authentication failed 403 Not Authorized (caller is not an Admin user) 409 Conflict (username already exists) 500 Server Error**

Delete User by UserId

Verb		**DELETE**
Template		**/users/{userId}**
Produces		**None**
Consumes		**None**
Request Headers		**Authorization: Bearer <jwt>**
Response Headers		**TBD**
Produced Entity		**None**
Consumed Entity		**None**

Response Codes		**201 Success** **401 Authentication failed** **403 Not Authorized (caller is not an Admin user)** **404 Not Found (userId doesn't exist, only returned if the user is Authorized)** **500 Server Error**

Delete User by Username

Verb		**DELETE**
Template		**/users?username={username}**
Produces		**application/json**
Consumes		**None**
Request Headers		**Authorization: Bearer <jwt>**
Response Headers		**TBD**
Produced Entity		**None**
Consumed Entity		**None**
Response Codes		**201 Success** **401 Authentication failed** **403 Not Authorized (caller is not an Admin)** **404 Not Found (username doesn't exist, only returned if the user is Authorized)** **500 Server Error**

Projects

Get Projects

Verb		**GET**
Template		**/projects**
Produces		**application/json**
Consumes		**None**
Request Headers		**Authorization: Bearer <jwt>** **Accept: application/json**
Response Headers		**TBD**
Produced Entity		**List of Project objects visible to caller**
Consumed Entity		**None**
Response Codes		**200 Success** **401 Authentication failed** **403 Not Authorized** **500 Server Error**

Get Project

Verb	GET
Template	/projects/{projectId}
Produces	application/json
Consumes	None
Request Headers	Authorization: Bearer <jwt> Accept: application/json
Response Headers	TBD
Produced Entity	Project object
Consumed Entity	None
Response Codes	200 Success 401 Authentication failed 403 Not Authorized (caller is not an Admin user or the projectId specified is not visible to the user making the call) 404 Not Found (projectId doesn't exist, only returned if the user is Authorized) 500 Server Error

Get Project Rules

Verb	GET
Template	/projects/{projectId}/rules
Produces	application/json
Consumes	None
Request Headers	Authorization: Bearer <jwt> Accept: application/json
Response Headers	TBD
Produced Entity	Array of Rule objects
Consumed Entity	None
Response Codes	200 Success 401 Authentication failed 403 Not Authorized (the projectId specified is not visible to the user making the call) 404 Not Found (projectId doesn't exist, only returned if the user is Authorized) 500 Server Error

Create Project

Verb	POST
Template	/projects

Produces		**application/json**
Consumes		**application/json**
Request Headers		**Authorization: Bearer <jwt> Accept: application/json**
Response Headers		**TBD**
Produced Entity		**Project object**
Consumed Entity		**Project object: id – must be null name - type: string, unique: yes, required: yes, max length: 50 description - type: string, required: yes, max length: 1000 owner - type: string, required: no, default: authenticated user aclEntries – type: [AclEntry], required: no**
Response codes		**201 Success 400 Invalid request (i.e., Project entity validation failed) 401 Authentication failed 403 Authorization failed 409 Project already exists 500 Server Error**

Update Project

Verb		**PUT**
Template		**/projects**
Produces		**application/json**
Consumes		**application/json**
Request Headers		**Authorization: Bearer <jwt> Accept: application/json**
Response Headers		**TBD**
Produced Entity		**Project object**
Consumed Entity		**Project object: id – required: yes name - type: string, unique: yes, required: no, max length: 50 description - type: string, required: no, max length: 1000 owner - type: string, required: no, default: authenticated user aclEntries – type: [AclEntry], required: no**
Response codes		**201 Success 400 Invalid request (i.e., Project entity validation failed) 401 Authentication failed 403 Authorization failed 409 Project already exists 500 Server Error**

Delete Project

Verb	DELETE
Template	/projects/{projectId}
Produces	None
Consumes	None
Request Headers	Authorization: Bearer <jwt>
Response Headers	TBD
Produced Entity	None
Consumed Entity	None
Response Codes	201 Success 401 Authentication failed 403 Not Authorized (caller is not an Admin user or the projectId specified is not delete-able by the user making the call) 404 Not Found (projectId doesn't exist, only returned if the user is Authorized) 500 Server Error

Add User Project ACL

Verb	POST
Template	/projects/{projectId}/acls/{userId}
Produces	None
Consumes	application/json
Request Headers	Authorization: Bearer <jwt> Accept: application/json
Response Headers	TBD
Produced Entity	None
Consumed Entity	ACL object userId must be null roleName is one of {"WATCHER", "PARTICIPANT", "ADMIN"}
Response codes	204 Success 400 Invalid request 401 Authentication failed 403 Authorization failed 404 Project or User do not exist 409 ACL already exists (same projectId and userId) 500 Server Error

Update User Project ACL

Verb	PUT
Template	/projects/{projectId}/acls/{userId}
Produces	None
Consumes	application/json
Request Headers	Authorization: Bearer <jwt> Accept: application/json
Response Headers	TBD
Produced Entity	None
Consumed Entity	ACL object userId must be null roleName is one of {"WATCHER", "PARTICIPANT", "ADMIN"}
Response codes	204 Success 400 Invalid request 401 Authentication failed 403 Authorization failed 404 ACL does not exist for Project/User tuple 500 Server Error

Remove User Project ACL

Verb	DELETE
Template	/projects/{projectId}/acls/{userId}
Produces	None
Consumes	None
Request Headers	Authorization: Bearer <jwt>
Response Headers	TBD
Response codes	204 Success 400 Invalid request 401 Authentication failed 403 Authorization failed 404 ACL does not exist 500 Server Error

Rules

Create / Update a Rule

Verb	**POST**
Template	**/rules**
Produces	**application/json**
Consumes	**application/json**
Request Headers	**Authorization: Bearer <jwt> Accept: application/json Content-Type: application/json**
Response Headers	**TBD**
Produced Entity	**Rule object**
Consumed Entity	**Rule object id – must be null for new rule name – must be unique for the project assigned status – must be ACTIVE or USER_DISABLED classification – must be valid if classification is enabled for system justification – text justification if classification is enabled for system ruleLogic – must be valid for system streamName – must be a name of a stream in the system formName – must be a name of a form in the system project.id – must be valid project rest user has access to**
Response Codes	**200 Success 401 Authentication failed 403 Not Authorized 500 Server Error**

Get Rule

Verb	**GET**
Template	**/rules/{ruleId}**
Produces	**application/json**
Consumes	**None**
Request Headers	**Authorization: Bearer <jwt> Accept: application/json**
Response Headers	**TBD**
Produced Entity	**Rule object**
Consumed Entity	**None**
Response Codes	**200 Success 401 Authentication failed 403 Not Authorized (caller is not an Admin user or the ruleId specified is not visible to the user making the call)**

		404 Not Found (ruleId doesn't exist, only returned if the user is Authorized) 500 Server Error

Delete Rule

Verb		DELETE
Template		/rules/{ruleId}
Produces		application/json
Consumes		None
Request Headers		Authorization: Bearer <jwt>
Response Headers		TBD
Produced Entity		None
Consumed Entity		None
Response Fields		None
Response Codes		201 Success 401 Authentication failed 403 Not Authorized (caller is not an Admin user or the ruleId specified is not delete-able by the user making the call) 404 Not Found (ruleId doesn't exist, only returned if the user is Authorized) 500 Server Error

Watchboards

Get Watchboards

Verb	GET
Template	/watchboards
Produces	application/json
Consumes	None
Request Headers	Authorization: Bearer <jwt> Accept: application/json
Response Headers	TBD
Produced Entity	List of Watchboard objects visible to caller
Consumed Entity	None
Response Codes	200 Success 401 Authentication failed 403 Not Authorized 500 Server Error

Get Watchboard

Verb	GET
Template	/watchboards/{watchboardId}
Produces	application/json
Consumes	None
Request Headers	Authorization: Bearer <jwt> Accept: application/json
Response Headers	TBD
Produced Entity	Watchboard object
Consumed Entity	None
Response Codes	200 Success 401 Authentication failed 403 Not Authorized (caller is not an Admin user or the watchboardId specified is not visible to the user making the call) 404 Not Found (watchboardId doesn't exist, only returned if the user is Authorized) 500 Server Error

Create Watchboard

Verb	POST
Template	/watchboards
Produces	application/json
Consumes	application/json

Request Headers	**Authorization: Bearer <jwt> Accept: application/json**
Response Headers	**TBD**
Produced Entity	**Watchboard object**
Consumed Entity	**Watchboard object id – must be null name - type: string, unique: yes, required: yes, max length: 50 description - type: string, required: yes, max length: 1000 owner - type: string, required: no, default: authenticated user aclEntries – type: [AclEntry], required: no contents – type: [WatchboardContent], required: no**
Response codes	**201 Success 400 Invalid request (i.e., Project entity validation failed) 401 Authentication failed 403 Authorization failed (caller is neither an admin user nor the specified owner) 409 Project already exists 500 Server Error**

Update Watchboard

Verb	**PUT**
Template	**/watchboards**
Produces	**None**
Consumes	**application/json**
Request Headers	**Authorization: Bearer <jwt> Accept: application/json**
Response Headers	**TBD**
Produced Entity	**None**
Consumed Entity	**Watchboard object id – required: yes name - type: string, unique: yes, required: no, max length: 50 description - type: string, required: no, max length: 1000 owner - type: string, required: no, default: authenticated user aclEntries – type: [AclEntry], required: no contents – type: [WatchboardContent] objects, required: no**
Response codes	**204 Success 400 Invalid request (i.e., Project entity validation failed) 401 Authentication failed 403 Authorization failed (caller is neither an admin user nor the specified owner) 409 Project already exists 500 Server Error**

Delete Watchboard

Verb	DELETE
Template	/watchboards/{watchboardId}
Produces	application/json
Consumes	None
Request Headers	Authorization: Bearer <jwt>
Response Headers	TBD
Produced Entity	None
Consumed Entity	None
Response Fields	User object
Response Codes	201 Success 401 Authentication failed 403 Not Authorized (caller is not an Admin user or the watchboardId specified is not delete-able by the user making the call) 404 Not Found (watchboardId doesn't exist, only returned if the user is Authorized) 500 Server Error

Add User Watchboard ACL

Verb	POST
Template	/watchboards/{watchboardId}/acls/{userId}
Produces	None
Consumes	application/json
Request Headers	Authorization: Bearer <jwt> Accept: application/json
Response Headers	TBD
Produced Entity	None
Consumed Entity	ACL object userId must be null roleName is one of {"WATCHER", "PARTICIPANT", "ADMIN"}
Response codes	204 Success 400 Invalid request 401 Authentication failed 403 Authorization failed 404 Project or User do not exist 409 ACL already exists (same projectId and userId) 500 Server Error

Update User Watchboard ACL

Verb	PUT
Template	/watchboards/{watchboardId}/acls/{userId}
Produces	None
Consumes	application/json
Request Headers	Authorization: Bearer <jwt> Accept: application/json
Response Headers	TBD
Produced Entity	None
Consumed Entity	ACL object userId must be null roleName is one of {“WATCHER”, “PARTICIPANT”, “ADMIN”}
Response codes	204 Success 400 Invalid request 401 Authentication failed 403 Authorization failed 404 ACL does not exist for Project/User tuple 500 Server Error

Remove User Watchboard ACL

Verb	DELETE
Template	/watchboards/{watchboardId}/acls/ {userId}
Produces	None
Consumes	None
Request Headers	Authorization: Bearer <jwt>
Response Headers	TBD
Response codes	204 Success 400 Invalid request 401 Authentication failed 403 Authorization failed 404 ACL does not exist 500 Server Error

Area Of Interest (AOI)

Get AOIs

Verb	GET
Template	/aois [?projectId=<projectId>] [&tagQuery="<tag1>{,<tag2>, ...}"] [&accept=<media type>]
Produces	application/vnd.google-earth.kml+xml (KML), application/vnd.google-earth.kmz (KMZ), application/vnd.geo+json (GeoJSON) application/octet-stream (ESRI shapefile)
Consumes	None
Request Headers	Authorization: Bearer <jwt> or x.509 certs, configured by spring Accept: application/vnd.google-earth.kml+xml[1] application/vnd.google-earth.kmz application/vnd.geo+json application/octet-stream
Response Headers	TBD
Produced Entity	kml, kmz, GeoJSON, ESRI shapefile as requested
Consumed Entity	None
Response Codes	200 Success 401 Authentication failed 403 Not Authorized 500 Server Error

The datatype returned is controlled by the value of the Request Accept header. The caller can request KML, KMZ, GeoJSON, or an ESRI shapefile, which is delivered as a zip archive of a folder containing the shapefile and friends.

This service method can alternately take an accept query parameter that uses the same media types as the Request Accept header, as listed above. This is particularly useful if you want to test the service using a browser – you can easily form the query for any output format.

This query can be filtered by an optional projectId query parameter. Only an admin user is authorized to get all AOIs. A user must have at least WATCHER access to a project to be authorized to get AOIs for a particular project.

This query can also be filtered by an optional tagQuery query parameter. The tagQuery searches the AOI tags field. A tagQuery with a list of tags searches for any of the tags (the "OR").

[1] see https://developers.google.com/kml/documentation/kml_tut#kml_server

Get AOI

Verb	GET
Template	/aois/{aoiId} [?accept=<media type>]
Produces	application/vnd.google-earth.kml+xml (KML) application/vnd.google-earth.kmz (KMZ) application/vnd.geo+json (GeoJSON) application/octet-stream (ESRI shapefile)
Consumes	None
Request Headers	Authorization: Bearer <jwt> or x.509 certs, configured by spring Accept: application/vnd.google-earth.kml+xml application/vnd.google-earth.kmz application/vnd.geo+json application/octet-stream
Response Headers	TBD
Produced Entity	kml, kmz, GeoJSON, ESRI shapefile as requested
Consumed Entity	None
Response Codes	200 Success 401 Authentication failed 403 Not Authorized (caller is not an Admin user or the aoiId specified is not accessible to the user via some project to which the user has WATCHER access) 404 Not Found (aoiId doesn't exist, only returned if the user is Authorized) 500 Server Error

The datatype returned is controlled by the value of the Request Accept header. The caller can request KML, KMZ, GeoJSON, or an ESRI shapefile, which is delivered as a zip archive of a folder containing the shapefile and friends.

This service method can alternately take an accept query parameter that uses the same media types as the Request Accept header, as listed above. This is particularly useful if you want to test the service using a browser – you can easily form the query for any output format.

Note that to use this feature in a browser you must encode the "+" (in application/vnd.geo+json) as %2B like this: application/vnd.geo%2Bjson. Otherwise the + will turn into a blank space.

Create AOI

Verb	POST
Template	/aois
Produces	application/json
Consumes	multipart/form-data
Request Headers	Authorization: Bearer <jwt> Accept: application/json
Response Headers	TBD
Produced Entity	AoiStatusResults object
Consumed Entity	json – string in geojson format (id – is ignored) managed – optional string containing “true” or “false” default is true – AOI will be managed
Response Codes	201 “envelope” Success. Check AoiStatusResults for the results of the individual AOI create(s). 400 Invalid request (i.e., entity validation failed) 401 Authentication failed 403 Not Authorized (jwt does not reference a recognized user) 500 Server Error

POST creates a new AOI. The name of the AOI as expressed in the geojson properties name field must be unique in the LUX system. The feature id field is ignored. If the managed form field is not present, the AOI will be created “managed”. Otherwise, the managed field should contain the text “true” or “false” to explicitly indicate whether or not to create the AOI managed.

POST returns an AoiStatusResults object. A feature collection can create more than one AOI. AoiStatusResults reports the name, (lux) id, and status of the posted AOIs.

Update AOI

Verb	PUT
Template	/aois
Produces	application/json
Consumes	multipart/form-data
Request Headers	Authorization: Bearer <jwt> Accept: application/json
Response Headers	TBD
Produced Entity	AoiStatusResults object
Consumed Entity	json – string in geojson format (id – the LUX id of the object) managed – optional string containing "true" or "false" if omitted, PUT does not change the current value.
Response Codes	201 "envelope" Success. Check AoiStatusResults for the results of the individual AOI update(s). 400 Invalid request (i.e., entity validation failed) 401 Authentication failed 403 Not Authorized (jwt does not reference a recognized user) 500 Server Error

PUT updates existing AOI(s). The name of the AOI as expressed in the geojson properties name field must be unique in the LUX system. The feature id field must be present and an AOI must exist with that Id in LUX.

If the managed form field is not present, the AOI will be created "managed". Otherwise, the managed field should contain the text "true" or "false" to explicitly indicate whether or not to create the AOI managed.

PUT returns an AoiStatusResults object. A feature collection can update more than one AOI. AoiStatusResults reports the name, (lux) id, and status of any updated AOIs.

Remove AOI

Verb	DELETE
Template	/aois/ {aoiId}
Produces	None
Consumes	None
Request Headers	Authorization: Bearer <jwt>
Response Headers	TBD
Response codes	204 Success 400 Invalid request 401 Authentication failed 403 Authorization failed 404 AOI does not exist 500 Server Error

LUX JSON Objects

User Object

JSON:

```
{
      “id”: “<id of User>”
            “username”: “<username of User>”,
}
```

ACL Object

JSON:

```
{
            “userId”: “<id of acl’d User>”,
            “role”: “WATCHER” | “PARTICIPANT” | “ADMIN”
}
```

Project Object

JSON:

```
{
“id”: "<id of Project>"
“name”: "test project 2"
“description”: "test, test, test"
“privateAccess”: false
“active”: true
“aclEntries”: [
            {
“userId”: "<id of acl’d User>",
“role”: "ADMIN"
},
{
“userId”: "<id of acl’d User>",
“role”: "PARTICIPANT"
}]
“owner”: "<id of owner>"
“createdBy”: "<id of creator>"
```

```
“createdDate”: "2014-11-08T17:04:10.0+0000"
}
```

Rule Object

JSON:

```
{
“id”: "<id of rule, null for new ,rule>",
“version”: null for new rules,
“status” : “<status value”>,
“name”: "rule name",
“description” : ”<optional>”,
"formName":"<name of valid form in system, ie AIS FORM, etc >",
"streamName":"<name of valid stream in system, ie AIS FORM, etc >",
"project": {"id":"<id of target project, user must have acl access to project"},
"ruleLogic":"<rule logic, make sure to justify the “ in the rule>",
"classification":"<valid class value, null if unclass system>”,
“justification” : “<classification justification, null for unclass system>”,
"iconPath":"< optional url for rule icon, defaults to
             googleearth/images/ltblu-blank.png>"
}
```

WatchboardContent Object

JSON:

```
{
       “id”: “<id of child watchboard or project>”,
       “type”: “WATCHBOARD” or “PROJECT”,
              “name”: “entity name”,
              “position”: 1..32
       }
```

Watchboard Object

JSON:

```
{
       "id": "<id of Watchboard object>",
```

```
        "name": "does this show up",
        "description": "test",
        "owner": "<id of owner>",
        "aclEntries": [{
                "userId ": "<id of acl'd User>",
                "role": "PARTICIPANT"
        },
        {
                "userId ": "<id of acl'd User>",
                "role": "PARTICIPANT"
        }],
        "contents": [{
                "id": "<id of child Watchboard or Project",
                "type": "WATCHBOARD" or "PROJECT",
                "name": "entity name",
                "position": 1..32
        }]
}
```

AoiStatusResults Object

JSON:

```
{
	"statusCode": <integer, 0 indicates success, 1 indicates failure>,
	"statusMessage": "OK" or “Failed”,
	“aois”: [{
		“id”: “<uid – LUX AOI Id”,
		“name”: “<name of AOI>”,
		“statusCode”: <integer, 0 indicates success>,
		“statusMessage”: “OK” or “<error message>”
	}, …]
}
```

JSON Web Token (JWT) Primer

A JWT is a JSON object that contains a number of claims. The claims include the issuer (this is used to find the key when verifying the signature), the username the JWT represents (this is the user that LUX UI will use for access, e.g., ACLs, for this request), and an expiration date/time for the JWT, among others.

We show two ways of creating, signing and verifying JWTs – with a shared (common) key, and with a PKI Public/Private key pair.

Using a Shared (Common) Key

The LUX Engine uses a (static, non-expiring) JWT with the admin username that was signed by a key shared with the LUX UI to make REST requests on the LUX UI. The engine's JWT is in the lux.properties file. The key that signed this JWT is in the icg.sk file in the LUX UI jwt-keys folder (e.g., /usr/local/tomcat/webapps/lux/WEB-INF/classes/jwt-keys). The UI uses this key to verify requests with JWTs from the "icg" issuer, as determined by the jwt-keys.properties file also in the jwt-keys folder. The icg.sk key is 512 random bits, e.g., 64 bytes, base64 encoded. This makes for a relatively short signed JWT.

The steps necessary to create and use JWTs using your own Shared Key credential are:

1. Create a shared key with 64 random bytes and base64 encode it.
2. Add the shared key to the LUX UI jwt-keys folder
3. Create JWT(s) using the shared key
4. Use these JWT(s) in your application

Create shared key

Using a cryptographically secure random number generator (e.g., java.security.SecureRandom nextBytes(byte[])), create a 64 byte array of random bits. Base64 encode this array and write the result to a file named shared-key.sk.

Add the shared key to the LUX UI jwt-keys folder

Copy the shared key to the LUX UI tomcat webserver jwt-keys folder.

```
$ cd /usr/local/tomcat/webapps/lux/WEB-INF/classes/jwt-keys
$ cp /<path-to-key-gen-folder>/shared-key.sk .
```

Edit the jwt-keys.properties file in the webserver jwt-keys folder and add a line referencing the new key.

```
shared-key=shared-key.sk
```

Create JWTs using the shared key

Build the JWT-Tool project and use the runit.bat or runit.sh command to create a JWT. The form of the command is

```
./runit.sh <issuer> <config file path> <username> [expiration – MM/dd/yyyy | millis]
```

The following notes apply.

- the <config file path> must point to a jwt-keys.properties file. There is one in the JWT-Tool project resources folder, in the jwt-keys folder.
- The <issuer> must appear to the left of the "=" in a line in the in the jwt-keys.properties file. The right side of the "=" must name a file containing a shared key.
- The <username> should be that of an existing LUX user.
- For a JWT with a long expiration time, express the expiration time as MM/dd/yyyy. For a JWT with a short expiration time (e.g., one that expires in 60 seconds) express the expiration time in milliseconds.

Use JWTs in your application

For long duration JWTs, you can copy and paste the JWT into your application in some fashion. For short duration JWTs, you will probably want to use the same code as is used in JWT-Tool to create JWTs on the fly. Alternately, you can create JWTs using your own code and libraries as you chose.

Using a PKI Public/Private key pair

PKI is an asymmetric key algorithm. You use a private key to sign a JWT and the corresponding public key to verify the signature on a received JWT.

The steps necessary to create and use JWTs using your own PKI credentials are:

1. Create a public/private key pair in DER encoding.
2. Add the public key to the LUX UI jwt-keys folder
3. Create JWT(s) using the private key
4. Use these JWT(s) in your application

Create public/private key pair

Execute the following commands in a shell.

```
# create a public/private key pair
$ ssh-keygen -b 2048
   file       ./id_rsa
   passphrase <leave blank>
   passphrase <leave blank>

# convert the private key to DER format and unencrypt it for signing JWTs
$ openssl rsa -in ./id_rsa -outform DER -out pub.key.der          # private key
$ openssl pkcs8 -topk8 -nocrypt -outform der -in pub.key.der -inform der -out pub.nocrypt-key.der

# convert the public key to a Certificate (you won't be signing this certificate)
$ openssl req -new -x509 -key id_rsa -days 1024 -out pub.cer.crt

# dump the keys to see what's in them (optional)
$ openssl rsa -in pub.key.der -inform DER -text
$ openssl rsa -in pub.nocrypt-key.der -inform DER -text
$ openssl x509 -in pub.cer.crt -text
```

Add the key pair to the LUX UI jwt-keys folder

Copy the public key certificate to the LUX UI tomcat webserver jwt-keys folder.

```
$ cd /usr/local/tomcat/webapps/lux/WEB-INF/classes/jwt-keys
$ cp /<path-to-key-gen-folder>/pub.cer.crt .
```

Edit the jwt-keys.properties file in the webserver jwt-keys folder and add a line referencing the new public key certificate. (You haven't created any JWT's yet.)

```
pub-key=pub.cer-key.der
```

Create JWTs using the new key pair

Build the JWT-Tool project and use the runit.bat or runit.sh command to create a JWT. The form of the command is

```
./runit.sh <issuer> <config file path> <username> [expiration – MM/dd/yyyy | millis]
```

The following notes apply.

- The <config file path> must point to a jwt-keys.properties file. There is one in the JWT-Tool project resources folder, in the jwt-keys folder.
- The <issuer> must appear to the left of the "=" in a line in the in the jwt-keys.properties file. The right side of the "=" must name a file containing an unencrypted private key in DER format, as was generated above in `pub.nocrypt-key.der`. The LUX UI server currently only handles RSA keys.
- The <username> should be that of an existing LUX user.
- For a JWT with a long expiration time, express the expiration time as MM/dd/yyyy. For a JWT with a short expiration time (e.g., one that expires in 60 seconds) express the expiration time in milliseconds.

Use JWTs in your application

For long duration JWTs, you can copy and paste the JWT into your application in some fashion. For short duration JWTs, you will probably want to use the same code as is used in JWT-Tool to create JWTs on the fly. Alternately, you can create JWTs using your own code and libraries as you chose.