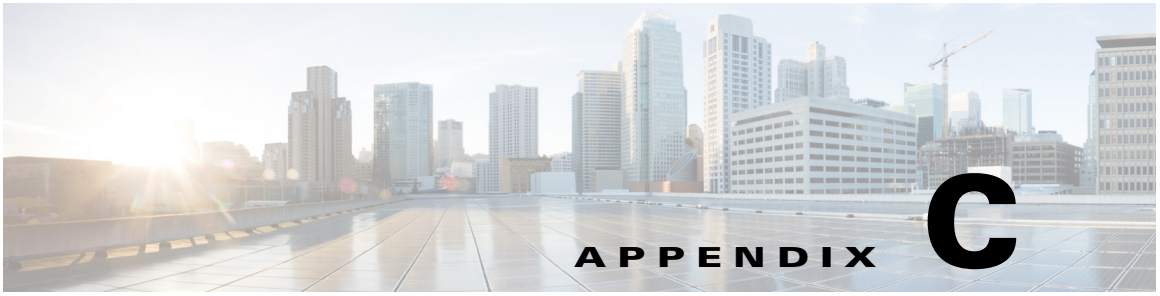


Appendix

Overview

There are many ways to invoke the RESTful services available from Enterprise Scheduler. In the examples below, the services are called by a simple Java client using the `HttpURLConnection` class. In addition to this method, you can also use a wide variety of third party frameworks such as the Spring Framework RestTemplate or the Apache CXF Framework.

REST API From Browser

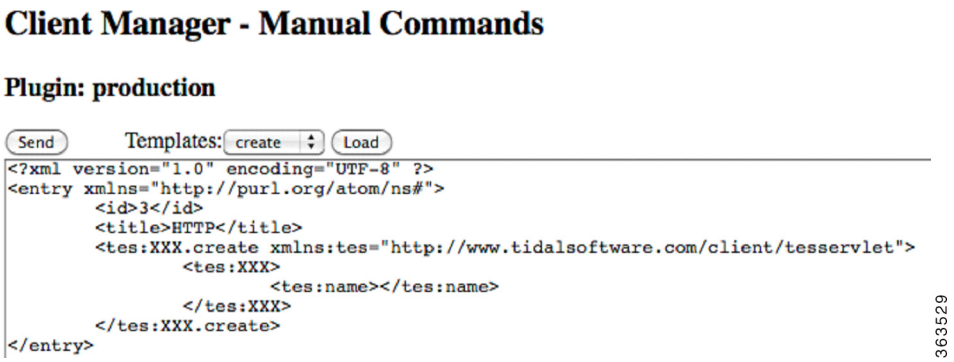
Before writing code to invoke the REST API, one could first browse the services available via a browser. In a live Enterprise Scheduler environment, the URL where the API can be reached is as follows:

For example:

Figure C-1 REST API browser view

Clicking on the links issues a "GET" request to the API. A "POST" request to the API can also be issued from the browser by using the "Manual Commands (Post)" link.

Figure C-2 REST API Post Screen



REST API Security Notes

The calls to the REST API are subjected to the same security restrictions as the same user accessing Scheduler UI. In Code Example 1 below, a call is issued to get all of the available jobs. The list of available jobs returned is determined by the username used in the API call.

Java Client REST API Examples

The following Java client issues a GET request to the REST API. This is the equivalent of clicking on the ApiObject link as described in the REST API From Browser section. This example retrieves all of the jobs currently defined in the Scheduler environment. The username and password pair is Base64 encoded and passed to the server as the "Authorization" property.

An XML document containing a list of jobs is returned from this call.

Code Example 1 - GET Request

```
public static void postRequest() throws Exception
{
URL url = new URL("http://www.companyscheduler.com:8080/api/tes-6.2/post");
HttpURLConnection conn = (HttpURLConnection) url.openConnection();
conn.setRequestMethod("GET");
conn.setDoInput(true);
conn.setDoOutput(true);

String userNamePassword = "myusername:mypassword";
userNamePassword =
            new
String(org.apache.commons.codec.binary.Base64.encodeBase64(userNamePassword
                .getBytes()));
    conn.setRequestProperty("Authorization", userNamePassword);

    conn.connect();
    BufferedReader reader = new BufferedReader(new InputStreamReader(
                conn.getInputStream()));
    String resp = "";
    String next = null;
    while ((next = reader.readLine()) != null)
```

```
        resp += next;
    System.out.println(resp);
    }
```

Code Example 2 shows a POST request issued to the TES REST API. The URL for issuing a POST request is always the same: http://<hostname>:<port>/api/<DSP Name>/ post.

In the post request, the command to be executed is sent in an URL encoded string. In this particular example, a POST request is sent to insert a job into the schedule. The <id> is the id of the job. Other parameters include <startdate> - the requested runtime for the job; <vars> - local job variable overrides; <params> - local job parameter overrides; and <deps> - the Y/N value for whether or not to override the job's dependencies.

An XML document acknowledging the job insert is returned.

Code Example 2 - POST Request

```
public static void postRequest() throws Exception
{
  URL url = new URL("http://www.companyscheduler.com:8080/api/tes-6.2/post");
  HttpURLConnection conn = (HttpURLConnection) url.openConnection();
  conn.setRequestMethod("POST");
  conn.setDoInput(true);
  conn.setDoOutput(true);

  conn.setRequestProperty("Content-Type","application/x-www-form-urlencoded");

  String userNamePassword = "myusername:mypassword";
  userNamePassword =
            new
String(org.apache.commons.codec.binary.Base64.encodeBase64(userNamePassword
                .getBytes()));
  conn.setRequestProperty("Authorization", userNamePassword);

  String postCommand = "<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=\"yes\" ?>"
                + "<atom:entry xmlns:atom=\"http://purl.org/atom/ns#\">"
+ "<atom:id>1</atom:id><atom:title>api</atom:title>"
+ "<Job.insert>"
+ "<id>2</id>"
+ "<startdate>20110812</startdate>"
+ "<vars></vars>"
+ "<params></params>"
+ "<deps>N</deps>"
+ "</Job.insert>"
+ "</atom:entry>";

  String payload = "data="+ java.net.URLEncoder.encode(postCommand, "ISO-8859-1");

  conn.setRequestProperty("Content-Length", Integer.toString(payload.getBytes().length));
  conn.setFixedLengthStreamingMode(payload.getBytes().length);

  DataOutputStream out = new DataOutputStream(conn.getOutputStream());
  out.writeBytes(payload);
  out.flush();

  BufferedReader reader = new BufferedReader(new InputStreamReader(
                conn.getInputStream()));
  String resp = "";
  String next = null;
  while ((next = reader.readLine()) != null)
```

```
    resp += next;
  System.out.println(resp);
}
```

Session Management

In both previous code examples, the calls establish new sessions on the server. For typical applications that make repeated calls to the REST API, the best practice is to reuse the established sessions so that the server does not create excessive number of active sessions, which eventually could cause it to run out of memory.

Code Example 3 is an extension of Code Example 1 showing the usages of a cookie for session management.

Code Example 3 - Use Session Cookie

```
public static void tesGetRequestWithSession() throws Exception
{
String sessionID = null;

for (int i=0; i<10; i++)
{
    URL url = new URL(
            "http://www.mycompanyscheduler.com:8080/api/tes-6.2/Job.getList");

    HttpURLConnection conn = (HttpURLConnection) url.openConnection();
    conn.setRequestMethod("GET");
    conn.setDoInput(true);
    conn.setDoOutput(true);

    if (sessionID == null)
    {
      String userNamePassword = " myusername:mypassword ";
      userNamePassword =
            new
String(org.apache.commons.codec.binary.Base64.encodeBase64(userNamePassword
                .getBytes()));
      conn.setRequestProperty("Authorization", userNamePassword);
    }
    else
    {
      conn.setRequestProperty("Cookie", sessionID);
    }

    conn.connect();
    BufferedReader reader = new BufferedReader(new InputStreamReader(
              conn.getInputStream()));
    String resp = "";
    String next = null;
    while ((next = reader.readLine()) != null)
       resp += next;
    System.out.println(resp);

    //extract cookie
    String setCookies = conn.getHeaderField("Set-Cookie");
    if (setCookies != null && sessionID == null)
    {
      String cookies[] = setCookies.split(";");
      if (cookies.length > 0)
        sessionID = cookies[0];
```

```
    }
  }
}
```

Execute a Query with Conditions

In Code Example 1, a `GET` request was issued to get a list of all jobs. The `GET` request can accept additional parameters such that the list of jobs returned can be filtered further.

If one needs to get a list of jobs that match a specific name pattern, the `GET` request URL can be constructed as follows:

```
URL url = new
URL("http://www.mycompanyscheduler.com:8080/api/tes6.2/Job.getList?query=(Job.name LIKE
'%name%')")
```

In this case a where clause (`Job.name LIKE '%name%'`) is sent. The `where` clause must be URL encoded. Similarly, other queries using other field names in the Job object can be constructed.

The same also be achieved using a `POST` request. The `POST` payload is below. In addition to the **queryCondition**, using the `POST` one could also specify columns needed.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<atom:entry xmlns:atom="http://purl.org/atom/ns#">
    <atom:id>1</atom:id>
    <atom:title>api</atom:title>
    <Job.getList>
    <selectColumns>
        id,ownerid,parentid,parentname,runtimeusername
    </selectColumns>
    <queryCondition>
        (Job.name LIKE '%name%')
</queryCondition>
    </Job.getList>
</atom:entry>
```