

Ruby Demo

Scott Hurrey

```
{% assign sluggedName = page.name | replace: 'md' %}  
modified: {{ page.last_modified_at | date: '%b-%d-%y' }}
```

Demo using Ruby

The rest demo script demonstrates authenticating a REST application, management and use of the authorization token, and creating, updating, discovering, and deleting supported Learn objects.

Prerequisites

- You must register a developer account and application in the Developer Portal
- You must register your application in Learn
- You must also configure the script as outlined in the README for the project

This Ruby command line Application allows you to:

- Authenticate
- Create, Read, and Update a Data Source
- Create, Read, and Update a Term
- Create, Read, and Update a Course
- Create, Read, and Update a User
- Create, Read, and Update a Membership
- Delete created objects in reverse order of create - membership, user, course, term, datasource.

All generated output is sent to the terminal.

This is not meant to be a Ruby tutorial. It will not teach you to write code in Ruby. It will, however, give a Developer familiar with Ruby the knowledge necessary to build a Web Services integration.

Assumptions

This help topic assumes the Developer:

- is familiar with Ruby
- has installed Ruby and the Ruby rest-client gem.
- has obtained a copy of the source code and built it in conjunction with the project README.md file.
- has a REST-enabled Learn instance.

Code Walkthrough

To build an integration with the Learn REST Web Services, regardless of the programming language of choice, can really be summed up in two steps:

1. Use the Application Key and Secret to obtain an OAuth 2.0 access token, as described in the Basic Authentication document.
2. Call the appropriate REST endpoint with the appropriate data to perform the appropriate action.

Authorization and Authentication

The REST Services rely on OAuth 2.0 Bearer Tokens for authentication. A request is made to the token endpoint with a Basic Authorization header containing the base64-encoded key:secret string as its key. The token service returns a JSON object containing the Access Token, the Token Type, and the number of seconds until the token expires. The token is set to expire after one hour, and subsequent calls to retrieve the token will return the same token with an updated expiry time until such time that the token has expired. There is no refresh token and currently no revoke token method.

The Ruby code handles this with the following code:

```
bb_rest = RestClient::Resource.new $AUTH_PATH, $KEY, $SECRET
bb_rest.post('grant_type=client_credentials', :accept => :json){ |response, request,
↪ result, &block|
    case response.code
    when 200
        p "It worked !"
        token = JSON.parse(response)
        $access_token = token['access_token']
        $auth = "Bearer " + $access_token
        p 'Access-Token: ' + $access_token
    else
        p response.to_s
        response.return!(request, result, &block)
    end
}
```

The JSON response is serialized into the Token object, and you may then retrieve those values from that object.

Calling Services

The individual service calls are handled in succession in the restdemo.rb file. Each operation and object combination creates the JSON body by instantiating the appropriate JSON object in the form of a String (required if you need to control the content-type, which you do) when necessary, and then generating the appropriate HTTP Request, shipping it to Learn, and serializing the JSON response back into the appropriate object.

End points are generally defined as `/learn/api/public/v1/<objecttype>/<objectId>`. Object ID can be either the pk1, like `_1_1`, or as the batchuid. This value should be prepended by `externalId:`, like `externalId:test101`.

For example, to retrieve a course by the pk1 `_1_1`, you would call **GET** `/learn/api/public/v1/courses/_1_1`. To retrieve by the batchuid `test101`, you would call **GET** `/learn/api/public/v1/courses/externalId:test101`.

Create is sent to Learn as a HTTP POST message with a JSON body that defines the object. The endpoint should omit the `objectId`, as this will be generated on creation.

Read is sent to Learn as a HTTP GET message with an empty body. The endpoint should include the `objectId` being retrieved.

Update is sent to Learn as a HTTP PATCH message with a JSON body that defines the object. The endpoint should include the `objectId` being updated.

Delete is sent to Learn as a HTTP DELETE message with empty body. The endpoint should include the `objectId` being deleted.

Datasources

Create

```

payload = "{ \"externalId\": \"BBDN-DSK-RUBY\", \"description\": \"Demo Data Source used for
↳ REST Ruby Demo\" }"
RestClient.post($DSK_PATH, payload, :content_type => :json, :accept => :json,
↳ :Authorization => $auth){ |response, request, result, &block|
  case response.code
    when 201
      p "It worked !"
      datasource = JSON.parse(response)
      $dsk_id = datasource['id']
      p 'Create Datasource: dsk_id=' + $dsk_id
    else
      p response.to_s
      response.return!(request, result, &block)
    end
  end
}

```

Read

```

RestClient.get($DSK_PATH + $dsk_id, :content_type => :json, :accept => :json,
↳ :Authorization => $auth){ |response, request, result, &block|
  case response.code
    when 200
      p "Got Datasource !" + response.to_s
    else
      p response.to_s
      response.return!(request, result, &block)
    end
  end
}

```

Update

```

payload = "{ \"externalId\": \"BBDN-DSK-RUBY\", \"description\": \"Demo Data Source used for
↳ REST Ruby Demo - Updated\" }"

RestClient.patch($DSK_PATH + $dsk_id, payload, :content_type => :json, :accept =>
↳ :json, :Authorization => $auth){ |response, request, result, &block|
  case response.code
    when 200
      p 'Updated Datasource: ' + response.to_s
    else
      p response.to_s
      response.return!(request, result, &block)
    end
  end
}

```

Delete

```

RestClient.delete($DSK_PATH + $dsk_id, :content_type => :json, :accept => :json,
↳ :Authorization => $auth){ |response, request, result, &block|
  case response.code
    when 204
      p "Datasource Deleted !"
    else
      p response.to_s
      response.return!(request, result, &block)
    end
  end
}

```

Terms

Create

```
payload = "{ \"externalId\": \"BBDN-TERM-RUBY\", \"dataSourceId\": \"\" + $dsk_id + "\",
↪ \"name\" : \"REST Demo Term - Ruby\", \"description\": \"Term Used For REST Demo - Ruby\",
↪ \"availability\" : { \"available\" : \"Yes\" } }"
RestClient.post($TERM_PATH, payload, :content_type => :json, :accept => :json,
↪ :Authorization => $auth){ |response, request, result, &block|
  case response.code
  when 201
    p "It worked !"
    term = JSON.parse(response)
    $term_id = term['id']
    p 'Create Term: term_id=' + $term_id
  else
    p response.to_s
    response.return!(request, result, &block)
  end
}
```

Read

```
RestClient.get($TERM_PATH + $term_id, :content_type => :json, :accept => :json,
↪ :Authorization => $auth){ |response, request, result, &block|
  case response.code
  when 200
    p "Got Term !" + response.to_s
  else
    p response.to_s
    response.return!(request, result, &block)
  end
}
```

Update

```
payload = "{ \"externalId\": \"BBDN-TERM-RUBY\", \"dataSourceId\": \"\" + $dsk_id + "\",
↪ \"name\" : \"REST Demo Term - Ruby\", \"description\": \"Updated Term Used For REST Demo -
↪ Ruby\", \"availability\" : { \"available\" : \"Yes\" } }"
RestClient.patch($TERM_PATH + $term_id, payload, :content_type => :json, :accept =>
↪ :json, :Authorization => $auth){ |response, request, result, &block|
  case response.code
  when 200
    p 'Updated Term: ' + response.to_s
  else
    p response.to_s
    response.return!(request, result, &block)
  end
}
```

Delete

```
RestClient.delete($TERM_PATH + $term_id, :content_type => :json, :accept => :json,
↪ :Authorization => $auth){ |response, request, result, &block|
  case response.code
  when 204
    p "Term Deleted !"
  else
    p response.to_s
  end
}
```

```

        response.return!(request, result, &block)
    end
}

```

Course

Create

```

payload = "{ \"externalId\" : \"BBDN-Java-Ruby-Demo\", \"courseId\" :
↪ \"BBDN-Java-Ruby-Demo\", \"name\" : \"Course Used For REST Demo - Ruby\", \"description\" :
↪ \"Course Used For REST Demo - Ruby\", \"allowGuests\" : \"false\", \"readOnly\" :
↪ \"false\", \"termId\" : \"\" + $term_id + "\", \"dataSourceId\" : \"\" + $dsk_id + "\",
↪ \"availability\" : { \"available\" : \"Yes\" } }"
RestClient.post($COURSE_PATH, payload, :content_type => :json, :accept => :json,
↪ :Authorization => $auth){ |response, request, result, &block|
  case response.code
    when 201
      p "It worked !"
      course = JSON.parse(response)
      $course_id = course['id']
      p 'Create Course: course_id=' + $course_id
    else
      p response.to_s
      response.return!(request, result, &block)
    end
  end
}

```

Read

```

RestClient.get($COURSE_PATH + $course_id, :content_type => :json, :accept => :json,
↪ :Authorization => $auth){ |response, request, result, &block|
  case response.code
    when 200
      p "Got Course !" + response.to_s
    else
      p response.to_s
      response.return!(request, result, &block)
    end
  end
}

```

Update

```

payload = "{ \"externalId\" : \"BBDN-Java-Ruby-Demo\", \"courseId\" :
↪ \"BBDN-Java-Ruby-Demo\", \"name\" : \"Course Used For REST Demo - Ruby\", \"description\" :
↪ \"Updated Course Used For REST Demo - Ruby\", \"allowGuests\" : \"false\", \"readOnly\" :
↪ \"false\", \"termId\" : \"\" + $term_id + "\", \"dataSourceId\" : \"\" + $dsk_id + "\",
↪ \"availability\" : { \"available\" : \"Yes\" } }"
RestClient.patch($COURSE_PATH + $course_id, payload, :content_type => :json, :accept
↪ => :json, :Authorization => $auth){ |response, request, result, &block|
  case response.code
    when 200
      p 'Updated Course: ' + response.to_s
    else
      p response.to_s
      response.return!(request, result, &block)
    end
  end
}

```

Delete

```

RestClient.delete($COURSE_PATH + $course_id, :content_type => :json, :accept => :json,
  ↳ :Authorization => $auth){ |response, request, result, &block|
    case response.code
      when 204
        p "Course Deleted !"
      else
        p response.to_s
        response.return!(request, result, &block)
      end
    end
  }

```

Users

Create

```

payload = "{ \"externalId\" : \"bbdnrestdemorubyuser\", \"userName\" : \"restrubyuser\",
  ↳ \"password\" : \"Bl@ckb0ard!\", \"studentId\" : \"restrubyuser\", \"dataSourceId\" : \"\" +
  ↳ $dsk_id + "\", \"name\" : { \"given\" : \"Ruby\", \"family\" : \"Rest Demo\" }, \"contact\"
  ↳ : { \"email\" : \"developers@blackboard.com\" }, \"availability\" : { \"available\" :
  ↳ \"Yes\" } }"

RestClient.post($USER_PATH, payload, :content_type => :json, :accept => :json,
  ↳ :Authorization => $auth){ |response, request, result, &block|
    case response.code
      when 201
        p "It worked !"
        user = JSON.parse(response)
        $user_id = user['id']
        p 'Create User: user_id=' + $user_id
      else
        p response.to_s
        response.return!(request, result, &block)
      end
    end
  }

```

Read

```

RestClient.get($USER_PATH + $user_id, :content_type => :json, :accept => :json,
  ↳ :Authorization => $auth){ |response, request, result, &block|
    case response.code
      when 200
        p "Got User !" + response.to_s
      else
        p response.to_s
        response.return!(request, result, &block)
      end
    end
  }

```

Update

```

payload = "{ \"externalId\" : \"bbdnrestdemorubyuser\", \"userName\" : \"restrubyuser\",
  ↳ \"password\" : \"Bl@ckb0ard!\", \"studentId\" : \"restrubyuser\", \"dataSourceId\" : \"\" +
  ↳ $dsk_id + "\", \"name\" : { \"given\" : \"Ruby\", \"family\" : \"Rest Demo\", \"middle\" :
  ↳ \"updated\" }, \"contact\" : { \"email\" : \"developers@blackboard.com\" },
  ↳ \"availability\" : { \"available\" : \"Yes\" } }"

RestClient.patch($USER_PATH + $user_id, payload, :content_type => :json, :accept =>
  ↳ :json, :Authorization => $auth){ |response, request, result, &block|

```

```

case response.code
when 200
p 'Updated User: ' + response.to_s
else
p response.to_s
response.return!(request, result, &block)
end
}

```

Delete

```

RestClient.delete($USER_PATH + $user_id, :content_type => :json, :accept => :json,
↳ :Authorization => $auth){ |response, request, result, &block|
  case response.code
  when 204
    p "User Deleted !"
  else
    p response.to_s
    response.return!(request, result, &block)
  end
}

```

Memberships

Create

```

payload = "{ \"courseRoleId\" : \"Student\", \"dataSourceId\" : \"\" + $dsk_id + "\",
↳ \"availability\" : { \"available\" : \"Yes\" } }"

RestClient.put($COURSE_PATH + $course_id + '/users/' + $user_id, payload,
↳ :content_type => :json, :accept => :json, :Authorization => $auth){ |response,
↳ request, result, &block|
  case response.code
  when 201
    p "It worked !"
    membership = JSON.parse(response)
    $created = membership['created']
    p 'Create Membership: ' + $created
  else
    p response.to_s
    response.return!(request, result, &block)
  end
}

```

Read

```

RestClient.get($COURSE_PATH + $course_id + '/users/' + $user_id, :content_type => :json,
↳ :accept => :json, :Authorization => $auth){ |response, request, result, &block|
  case response.code
  when 200
    p "Got Membership !" + response.to_s
  else
    p response.to_s
    response.return!(request, result, &block)
  end
}

```

Update

```

payload = "{ \"userId\" : \"\" + $user_id + "\", \"courseId\" : \"\" + $course_id + "\",
↪ \"courseRoleId\" : \"Instructor\", \"dataSourceId\" : \"\" + $dsk_id + "\", \"availability\"
↪ : { \"available\" : \"Yes\" } }"

RestClient.patch($COURSE_PATH + $course_id + '/users/' + $user_id, payload,
↪ :content_type => :json, :accept => :json, :Authorization => $auth){ |response,
↪ request, result, &block|
  case response.code
  when 200
    p 'Updated Membership: ' + response.to_s
  else
    p response.to_s
    response.return!(request, result, &block)
  end
}

```

Delete

```

RestClient.delete($COURSE_PATH + $course_id + '/users/' + $user_id, :content_type =>
↪ :json, :accept => :json, :Authorization => $auth){ |response, request, result, &block|
  case response.code
  when 204
    p "Membership Deleted !"
  else
    p response.to_s
    response.return!(request, result, &block)
  end
}

```

Conclusion

All of the code snippets included in this document are included in a sample REST Demo Ruby application available on GitHub. There is a README.html included that talks more specifically about building and running the code. Feel free to review the code and run it against a test or development Learn instance to see how it works.