Tutorial: Testing REST Web Services

Top Previous Next

In this section you shall learn how to test a RESTful web services API using Rapise. We shall be using a demo application called **Library Information System** that has a dummy RESTful web service API available for learning purposes. You can access this sample application at http://www.libraryinformationsystem.org, and its RESTful web service API can be found at: www.libraryinformationsystem.org/Services/RestService.aspx.

What is REST and what is a RESTful web service?
REpresentational State Transfer (REST) is a style of software architecture for distributed systems such as the World Wide Web. REST has emerged as a web API design model that offers greater simplicity over other web service protocols such as SOAP and XML-RPC.

A RESTful web API (also called a RESTful web service) is a web API implemented using HTTP and REST principles. Unlike SOAP-based web services, there is no "official" standard for RESTful web APIs. This is because REST is an architectural style, unlike SOAP, which is a protocol.

Overview

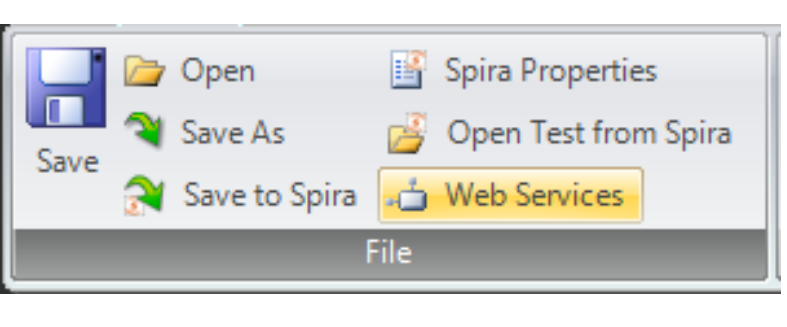
Creating a REST web service test in Rapise consists of the following steps:

1. Using the REST query builder to create the various REST web service requests and verify that they return the expected data in the expected format.
2. Parameterizing these REST web service requests into reusable templates and saving as Rapise learned objects.
3. Writing the test script in Javascript that uses the learned Rapise web service objects.

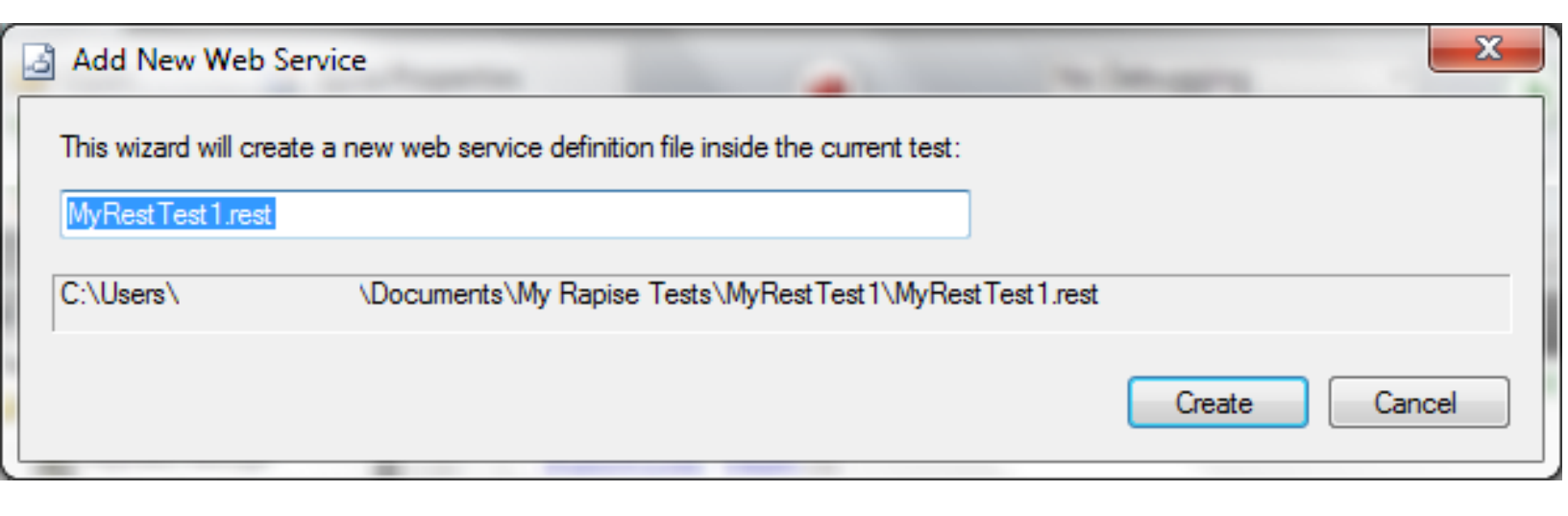
We shall discuss each of these steps in turn.

1. Using the REST Query Builder

Create a new test in Rapise called MyRestTest1.sstest. Once you have created it, click on the "Web Services" icon in the Test ribbon to add a new web service definintion to your test project:



This will display the Add New Web Service dialog box:

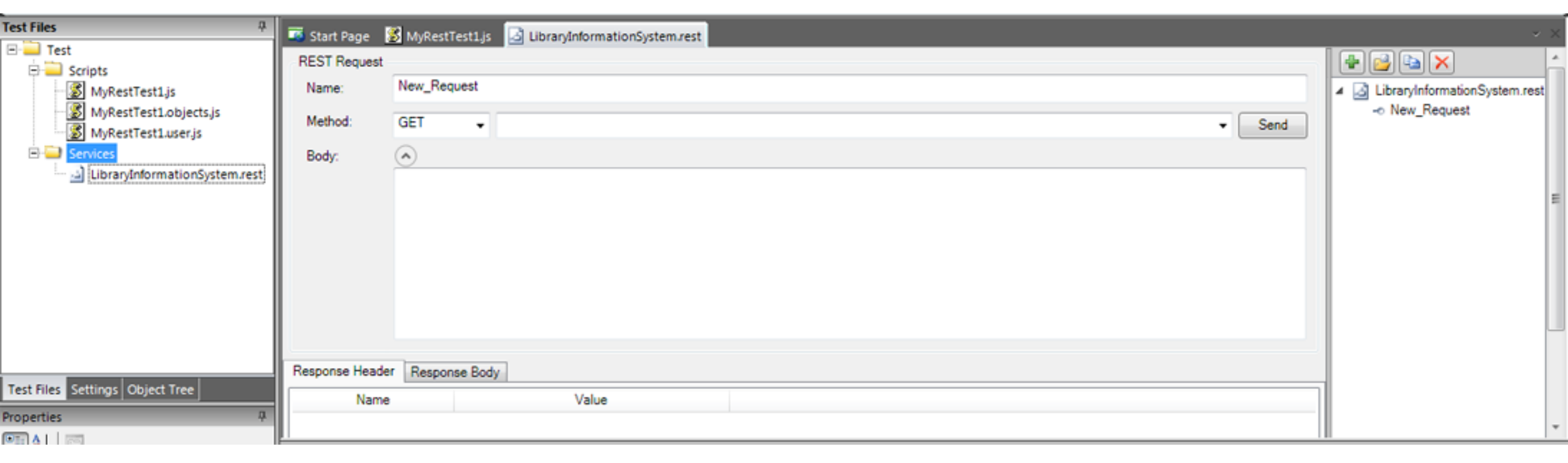


This wizard will create a new web service definition file inside the current test:

MyRestTest1.rest

C:\Users\ \Documents\My Rapise Tests\MyRestTest1\MyRestTest1.rest

Enter the name of the web service that you're going to add, in this case enter **"LibraryInformationSystem.rest"** and click "Create". This will add the REST web services definition file to your test project:



Name: New_Request
Method: GET

You will see on the right hand side, there is a new document editor for the .rest file. This is the REST web services query form. It lets you send test HTTP requests to the web service under test and inspect the output being returned.

If you open up API documentation for our sample application (www.libraryinformationsystem.org/Services/RestService.aspx) you will see that it exposes several operations for retrieving, adding, updating and deleting books and authors in the system. For this tutorial we shall perform the following operations:

1. Get the special SessionID to identify our test session
2. Get a list of books in the system
3. Add a new book to the system and verify that it was added

According to the documentation that means we will need to send the following requests:

(i) Get a Unique Session

URL:	http://www.libraryinformationsystem.org/Services/RestService.svc/session
Method:	GET
Returns:	Unique session ID that is passed to other requests to keep data separate for different demo users

(ii) Get this list of books

URL:	http://www.libraryinformationsystem.org/Services/RestService.svc/book?session_id={session_id}
Method:	GET
Returns:	Array of book objects

(iii) Add a new book to the list

URL: http://www.libraryinformationsystem.org/Services/RestService.svc/book?session_id={session_id}
Method: POST
Body: Pass a populated book object:

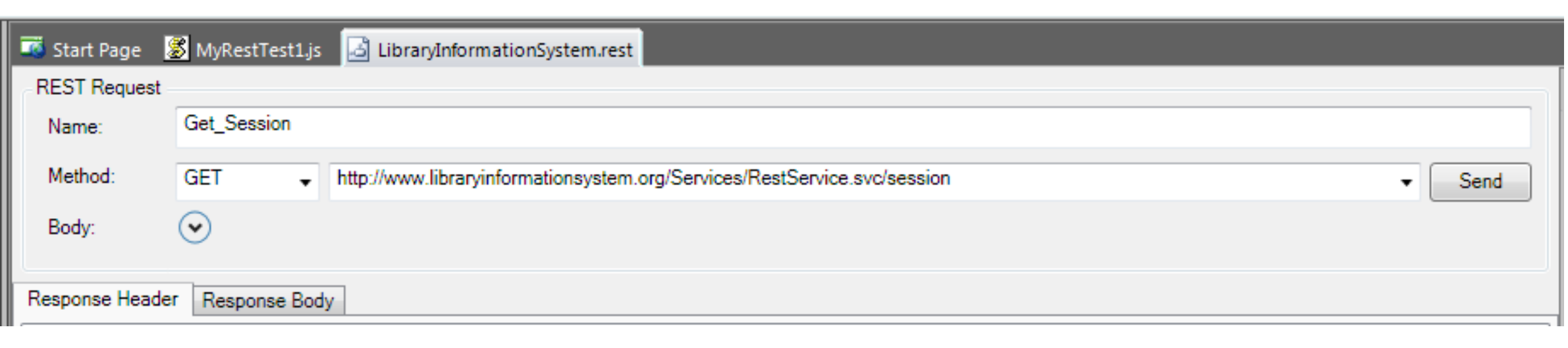
```
{
    "Name": "Book Name",
    "AuthorId": 1,
    "GenreId": 1,
}
```

Returns: Single book object that has its BookId populated

The first request will be to get the unique session ID that we will need to pass to the other requests. This is needed by our sample application to prevent testing by different users interfering with each other. To create this request, simply enter the following information on the REST Request form:

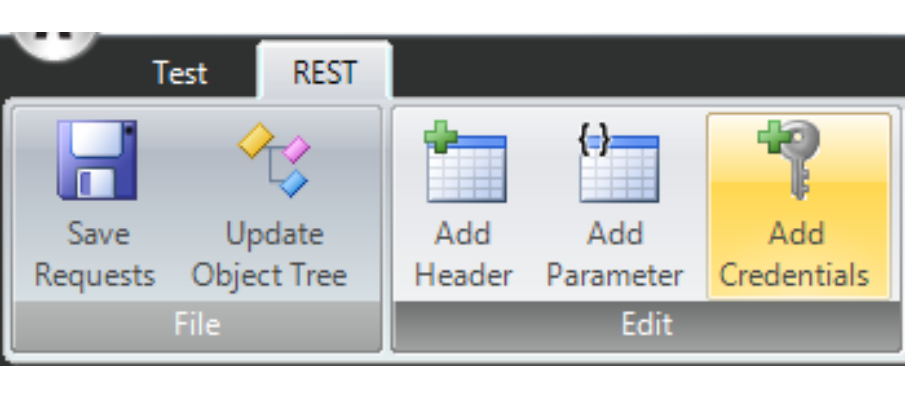
- **Name:** Get_Session
- **Method:** GET
- **URL:** http://www.libraryinformationsystem.org/Services/RestService.svc/session

You should now have it populated as illustrated below:

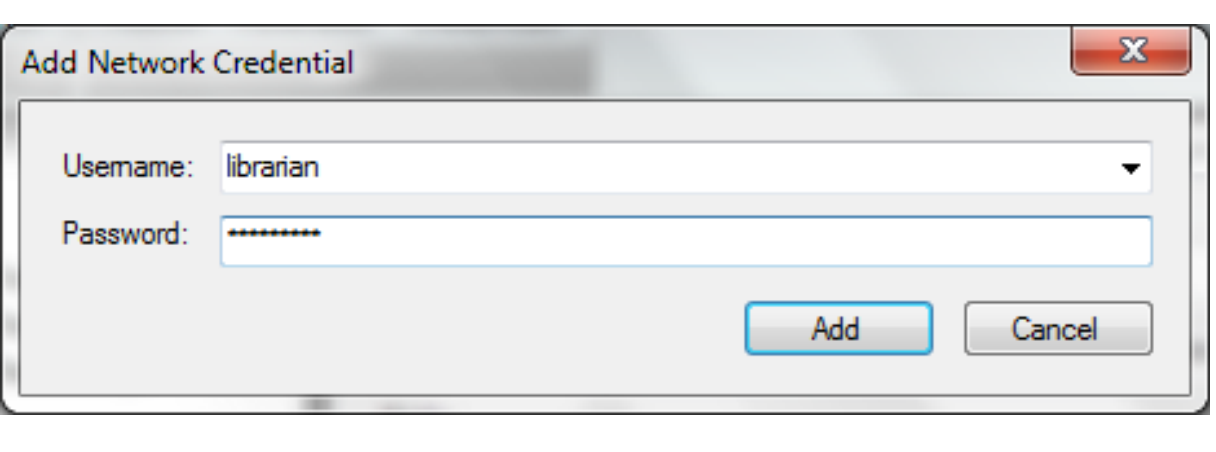


Name: Get_Session
Method: GET http://www.libraryinformationsystem.org/Services/RestService.svc/session

This web service request requires that we pass credentials by means of HTTP Basic authentication. So click on the "REST" tab in the Rapise ribbon and click on the "Add Credentials" button.



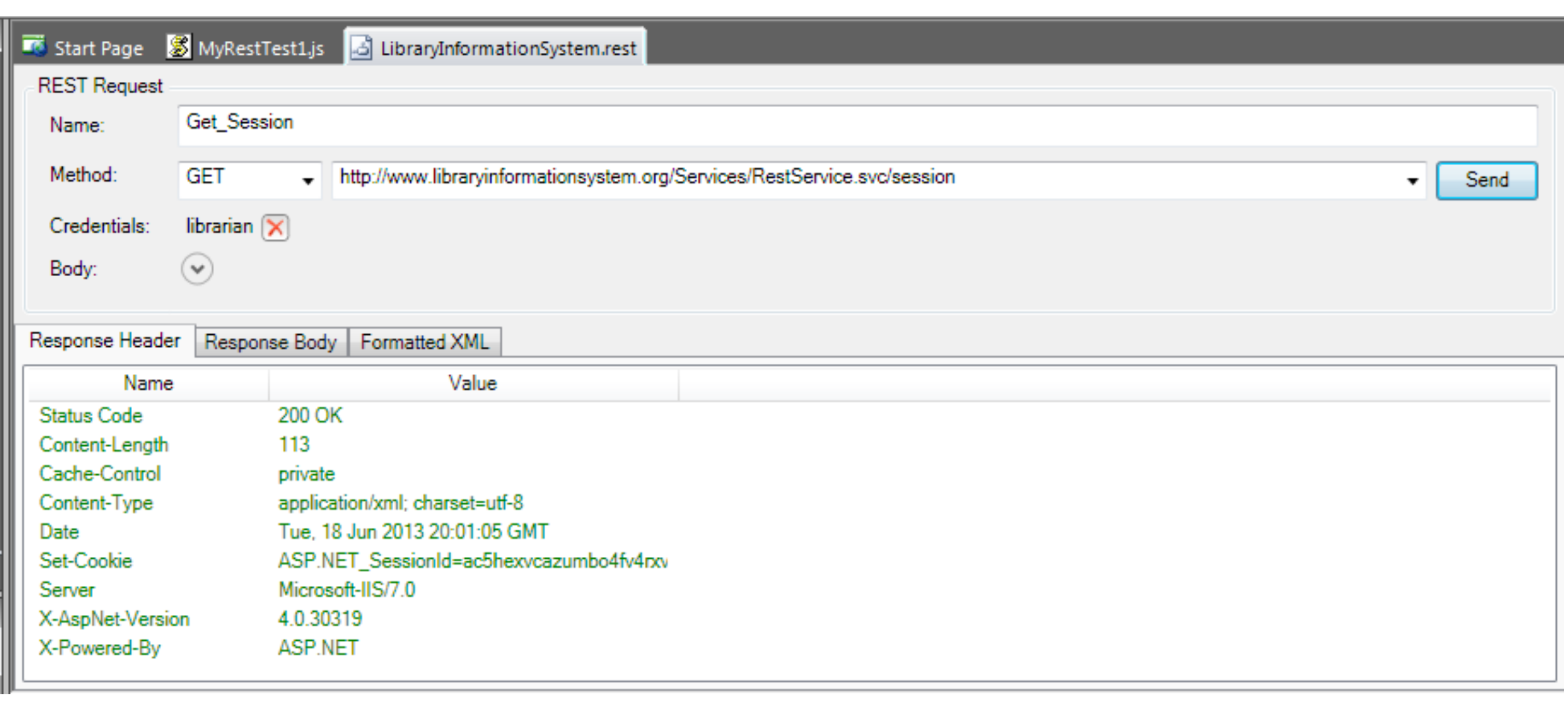
This will display the "Add Credentials" dialog box:



Username: librarian
Password: •••••••••

Enter **librarian** as both the username and password and click "Add".

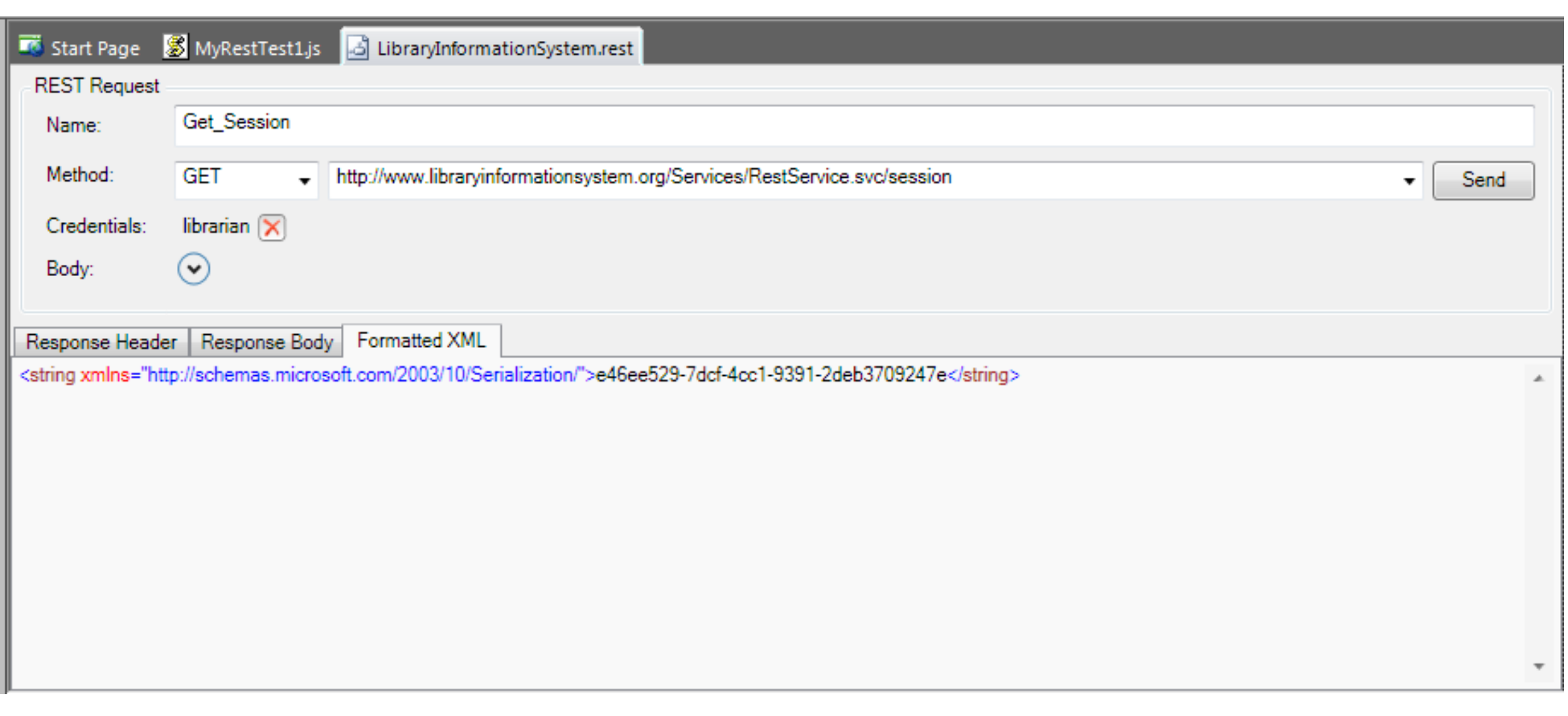
Now click the "Send" button and the request will get sent to the web service:



Name: Get_Session
Method: GET http://www.libraryinformationsystem.org/Services/RestService.svc/session
Credentials: librarian

Name	Value
Status Code	200 OK
Content-Length	113
Cache-Control	private
Content-Type	application/xml; charset=utf-8
Date	Tue, 18 Jun 2013 20:01:05 GMT
Set-Cookie	ASP.NET_SessionId=ac5hexvcazumbo4fv4rxv
Server	Microsoft-IIS/7.0
X-AspNet-Version	4.0.30319
X-Powered-By	ASP.NET

The Response Header tab will display the headers coming back from the web service. The Status Code **200 OK** means that the request succeeded and that data was returned. If you click on the "Formatted XML" tab, you will see the XML serialized data returned from the web service:



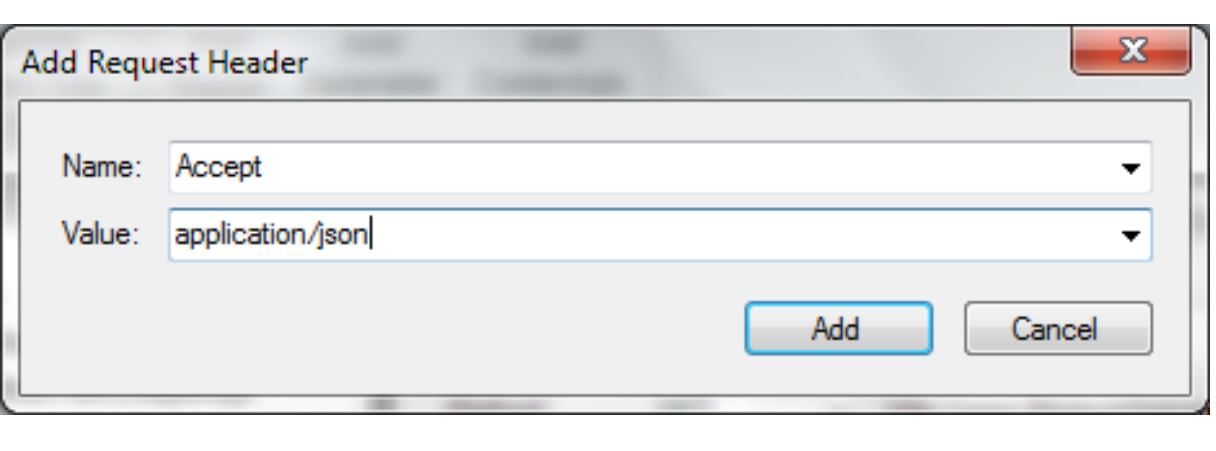
Name: Get_Session
Method: GET http://www.libraryinformationsystem.org/Services/RestService.svc/session
Credentials: librarian

```
<string xmlns="http://schemas.microsoft.com/2003/10/Serialization/">e46ee529-7dcf-4cc1-9391-2deb3709247e</string>
```

Since Rapise uses JavaScript as its scripting language, it is usually easier to work with JSON (JavaScript Object Notation) serialized data rather than XML. In the case of the sample Library Information System web service, you can change the format that it accepts and retrieves by sending two special HTTP headers:

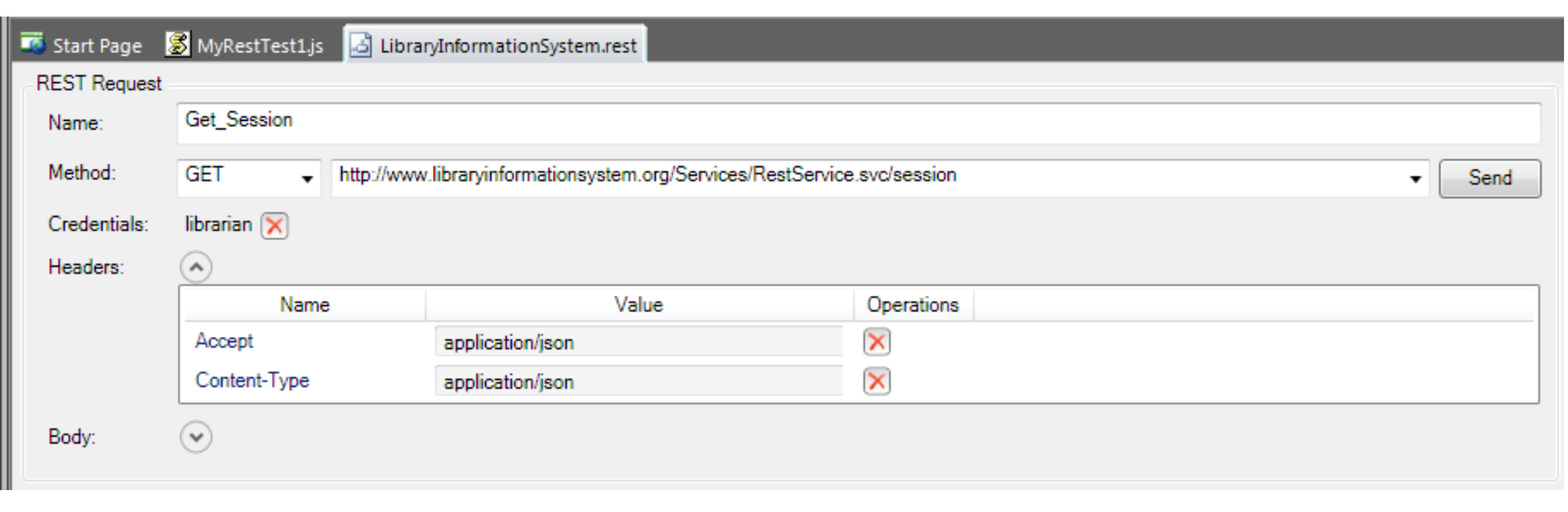
- **Content-Type:** application/json
- **Accept:** application/json

To add these headers to the request, simply click on the "Add Header" button in the REST ribbon tab. This will display the following dialog box:



Name: Accept
Value: application/json

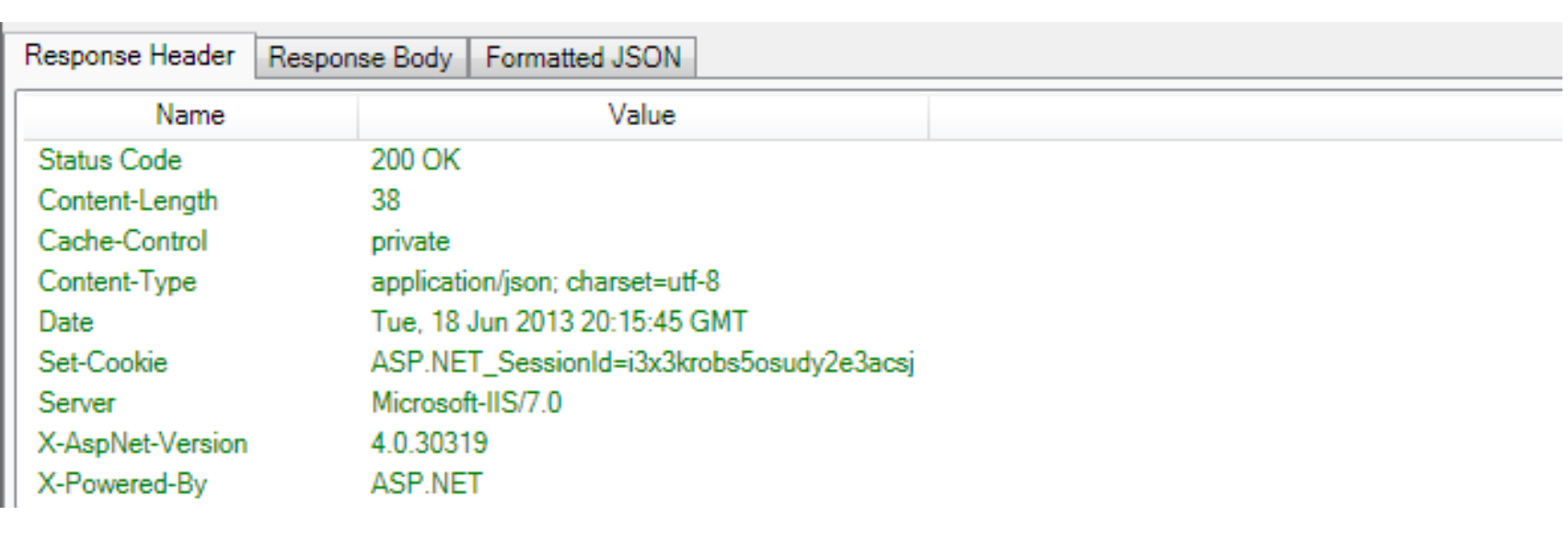
Choose the HTTP Header **"Accept"** from the list and enter **"application/json"** as the value. Repeat for the **"Content-Type"** header. You should now have the following populated request:



Name: Get_Session
Method: GET http://www.libraryinformationsystem.org/Services/RestService.svc/session
Credentials: librarian
Headers:

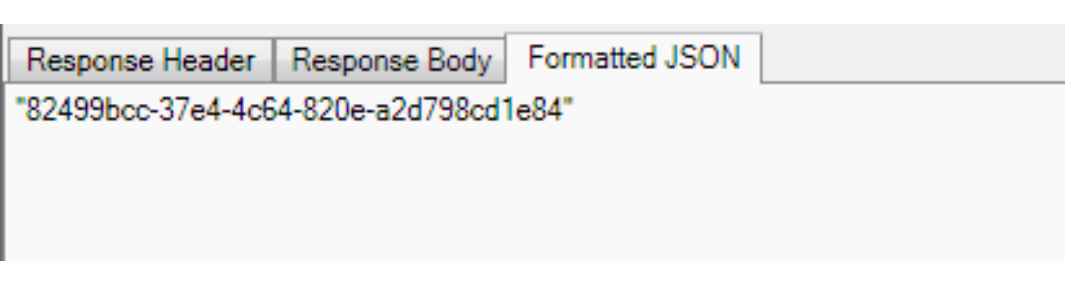
Name	Value	Operations
Accept	application/json	☒
Content-Type	application/json	☒

Now click the "Send" button and the request will get sent to the web service:



Name	Value
Status Code	200 OK
Content-Length	38
Cache-Control	private
Content-Type	application/json; charset=utf-8
Date	Tue, 18 Jun 2013 20:15:45 GMT
Set-Cookie	ASP.NET_SessionId=i3x3krobs5osudy2e3acsj
Server	Microsoft-IIS/7.0
X-AspNet-Version	4.0.30319
X-Powered-By	ASP.NET

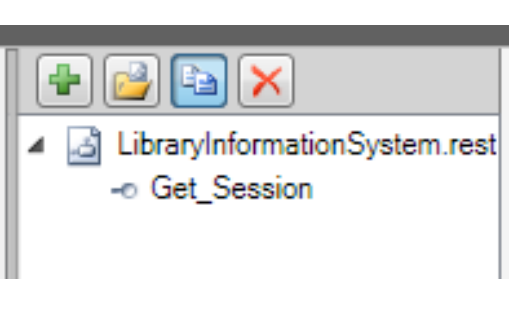
The Response Header tab will display the headers coming back from the web service. Note that the returned Content-Type is listed as "application/json" as requested. If you click on the "Formatted JSON" tab, you will see the JSON serialized data returned from the web service:



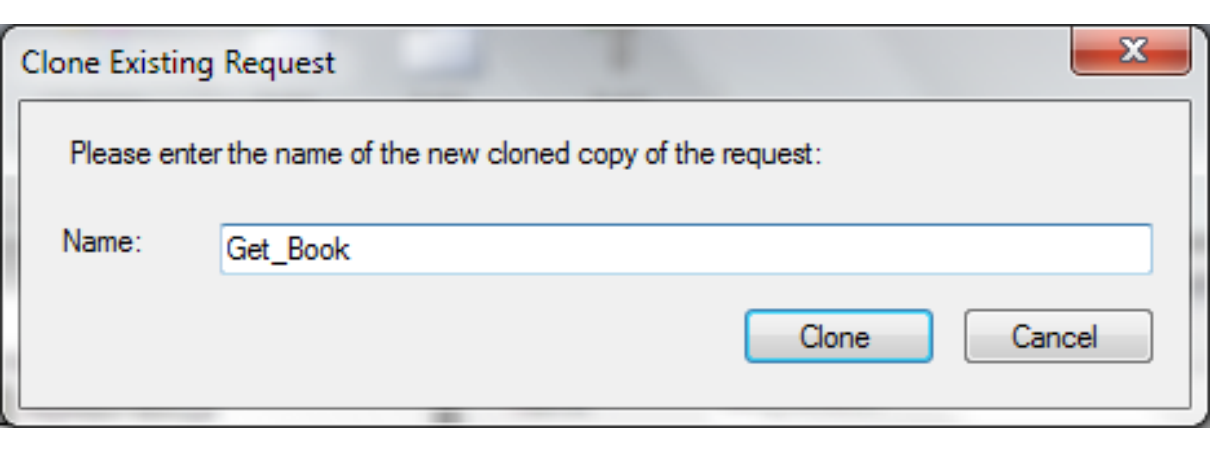
```
"82499bcc-37e4-4c64-820e-a2d798cd1e84"
```

We have now completed the creation of our first test operation. Click on the "Save Requests" button in the Rapise REST Ribbon to make sure our changes have been saved.

Now click on the "Clone request" icon in the REST request explorer in the right-hand side of the screen:



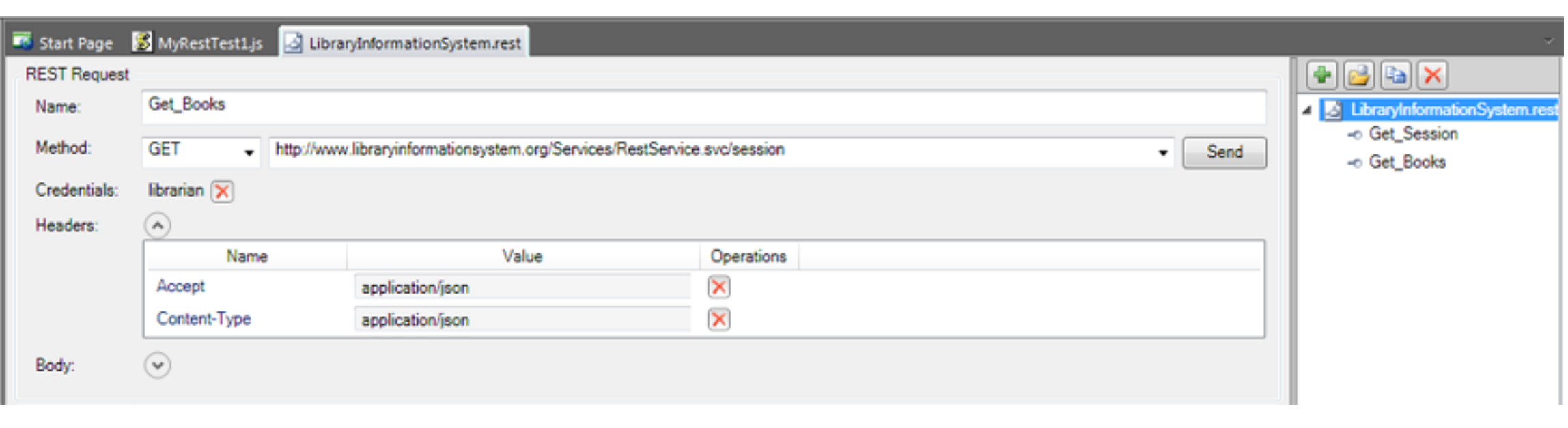
This will display the Clone Request dialog box. This lets us create a new REST request that contains the headers and authentication already defined on our existing request. This will save time over creating a new REST request from scratch:



Please enter the name of the new cloned copy of the request:

Name: Get_Book

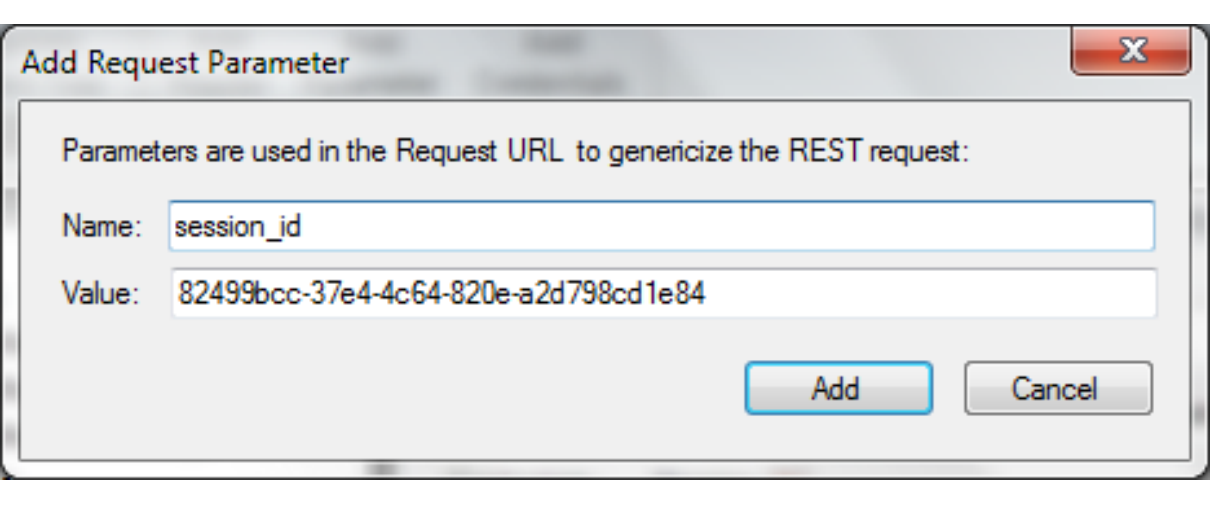
Enter the name **"Get_Books"** in the dialog box and click the "Clone" button. This will create a new REST request with this name:



Name: Get_Books
Method: GET http://www.libraryinformationsystem.org/Services/RestService.svc/session
Credentials: librarian
Headers:

Name	Value	Operations
Accept	application/json	☒
Content-Type	application/json	☒

For this request we need to pass through the SessionID in the querystring. Rather than hardcoding it in the URL, we can make use of the parameterization feature of Rapise. Click on the **"Add Parameter"** button in the Rapise REST Ribbon. This will display the "Add Request Parameter" dialog box:



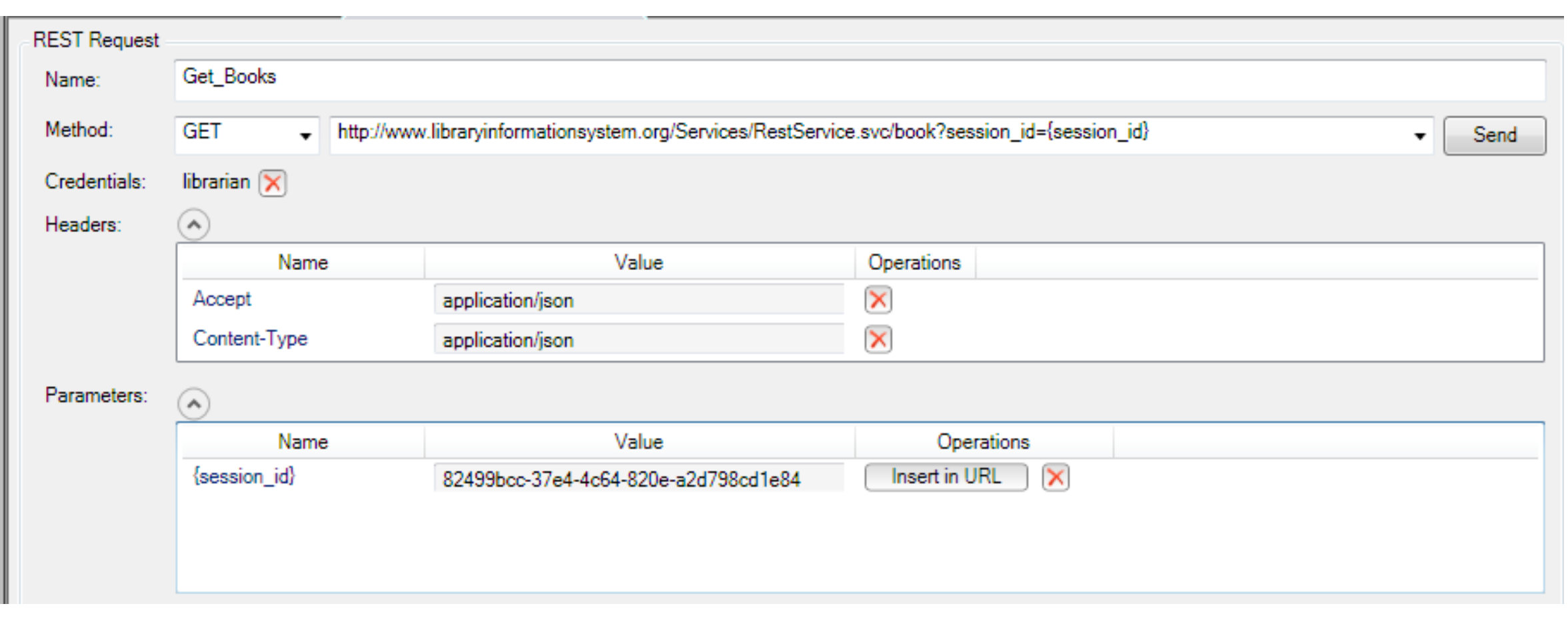
Parameters are used in the Request URL to genericize the REST request:

Name: session_id
Value: 82499bcc-37e4-4c64-820e-a2d798cd1e84

Click the "Add" button and the parameter will be added to the request. Now change the URL to:

```
URL: http://www.libraryinformationsystem.org/Services/RestService.svc/book?session_id=
```

Then position the caret at the end of this URL and click the "Insert in URL" button. This will insert the parameter token in the URL at the specified point:



Name: Get_Books
Method: GET http://www.libraryinformationsystem.org/Services/RestService.svc/book?session_id={session_id}
Credentials: librarian
Headers:

Name	Value	Operations
Accept	application/json	☒
Content-Type	application/json	☒

Parameters:

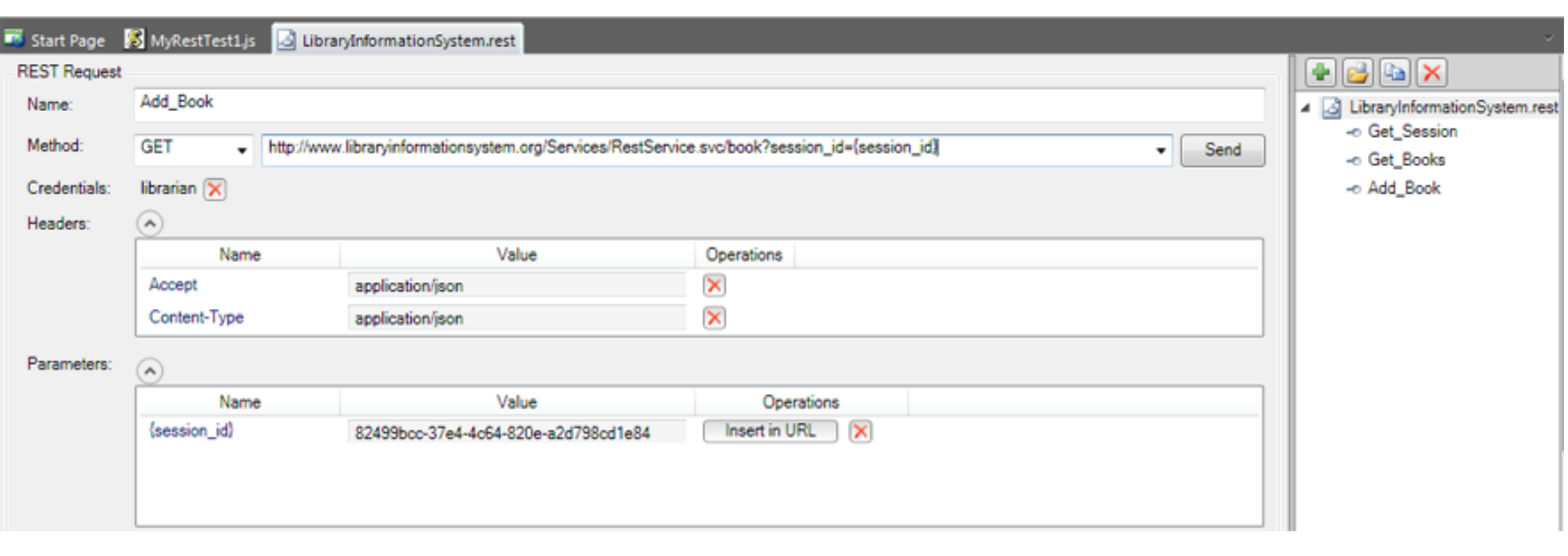
Name	Value	Operations
{session_id}	82499bcc-37e4-4c64-820e-a2d798cd1e84	Insert in URL ☒

Now click the "Send" button and the request will get sent to the web service. This will return the list of books serialized as a JSON array of objects:

```
[
 {
  "Author": {
   "Age": 125,
   "Id": 3,
   "Name": "Arthur Conan Doyle"
  },
  "AuthorId": 3,
  "Genre": {
   "Id": 2,
   "Name": "Murder & Mystery"
  },
  "GenreId": 2,
  "Id": 1,
  "Name": "Hound of the Baskervilles"
 },
 {
  "Author": {
   "Age": 125,
   "Id": 3,
   "Name": "Arthur Conan Doyle"
  }
```

We have now completed the creation of our second test operation. Click on the "Save Requests" button in the Rapise REST Ribbon to make sure our changes have been saved.

Now click on the "Clone request" icon in the REST request explorer in the right-hand side of the screen. Enter the name **"Add_Book"** in the dialog box and click the "Clone" button. This will create a new REST request with this name:



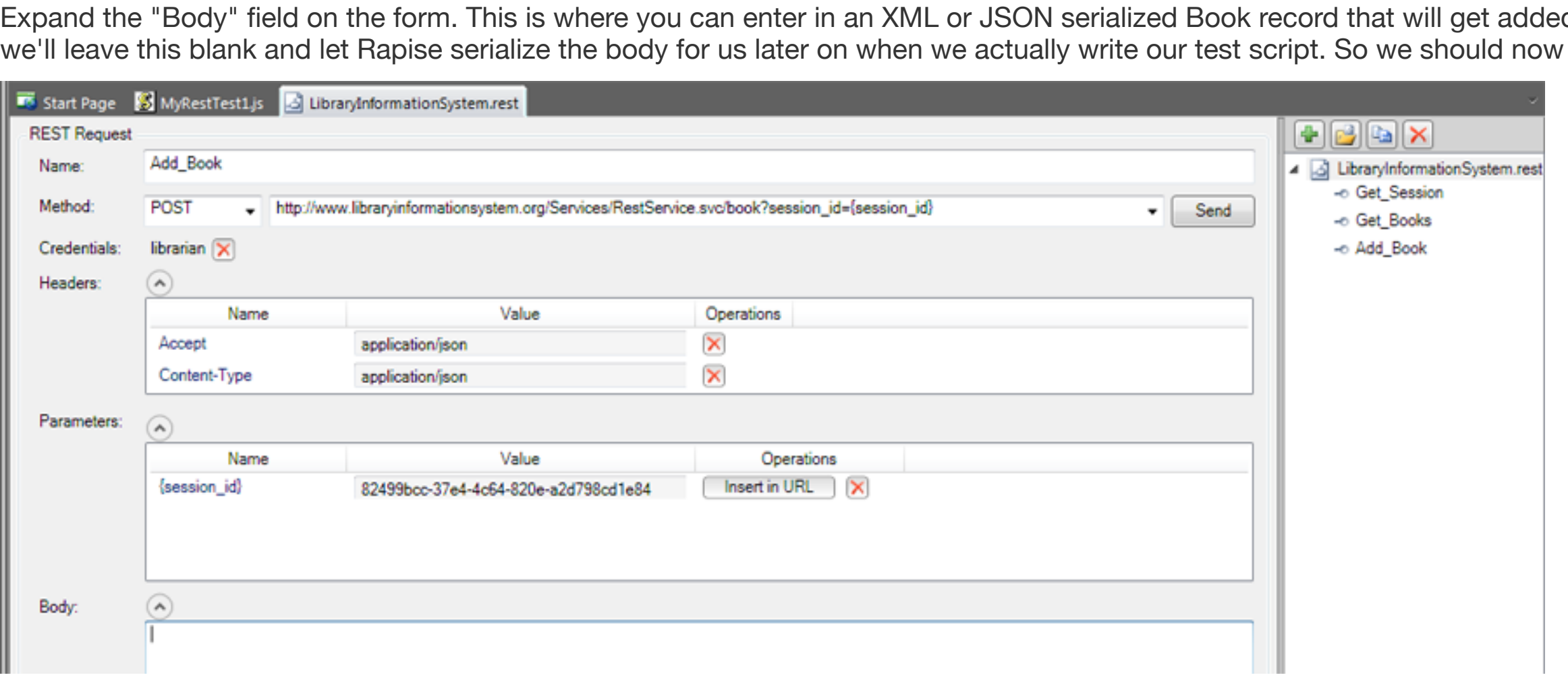
Name: Add_Book
Method: GET http://www.libraryinformationsystem.org/Services/RestService.svc/book?session_id={session_id}
Credentials: librarian

Name	Value	Operations
Accept	application/json	☒
Content-Type	application/json	☒

Name	Value	Operations
{session_id}	82499bcc-37e4-4c64-820e-a2d798cd1e84	Insert in URL ☒

This operation will add a new book to the system, so it's a POST request. Change the Method type in the dropdown list from "GET" to **"POST"**.

Expand the "Body" field on the form. This is where you can enter in an XML or JSON serialized Book record that will get added to the system. For now we'll leave this blank and let Rapise serialize the body for us later on when we actually write our test script. So we should now have:



Name: Add_Book
Method: POST http://www.libraryinformationsystem.org/Services/RestService.svc/book?session_id={session_id}
Credentials: librarian

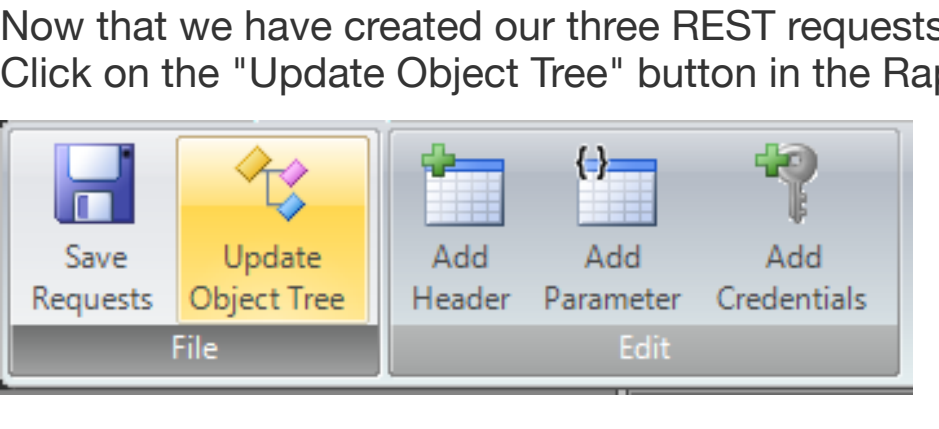
Name	Value	Operations
Accept	application/json	☒
Content-Type	application/json	☒

Name	Value	Operations
{session_id}	82499bcc-37e4-4c64-820e-a2d798cd1e84	Insert in URL ☒

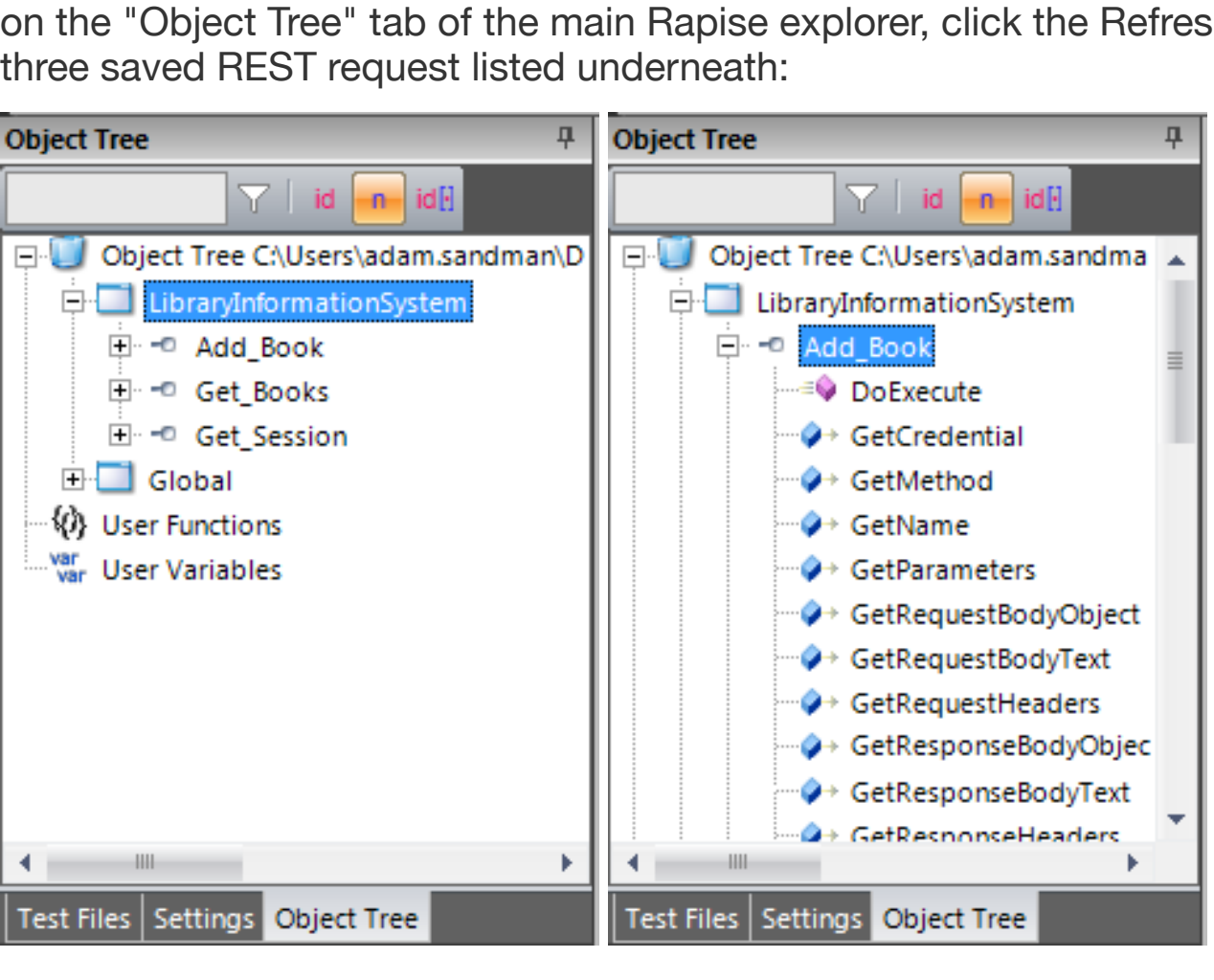
We have now completed the creation of our third test operation. Click on the "Save Requests" button in the Rapise REST Ribbon to make sure our changes have been saved.

2. Saving the REST Requests as Objects

Now that we have created our three REST requests, the next step is to actually create the Rapise objects that we can use in our JavaScript test scripts. Click on the "Update Object Tree" button in the Rapise REST Ribbon to tell Rapise to update the Object Tree with our new requests:



Rapise will open a command prompt window in the background and then display a confirmation message once the Object Tree has been updated. Click on the "Object Tree" tab of the main Rapise explorer, click the Refresh icon and you will see the "LibraryInformationSystem" heading displayed, with the three saved REST request listed underneath:

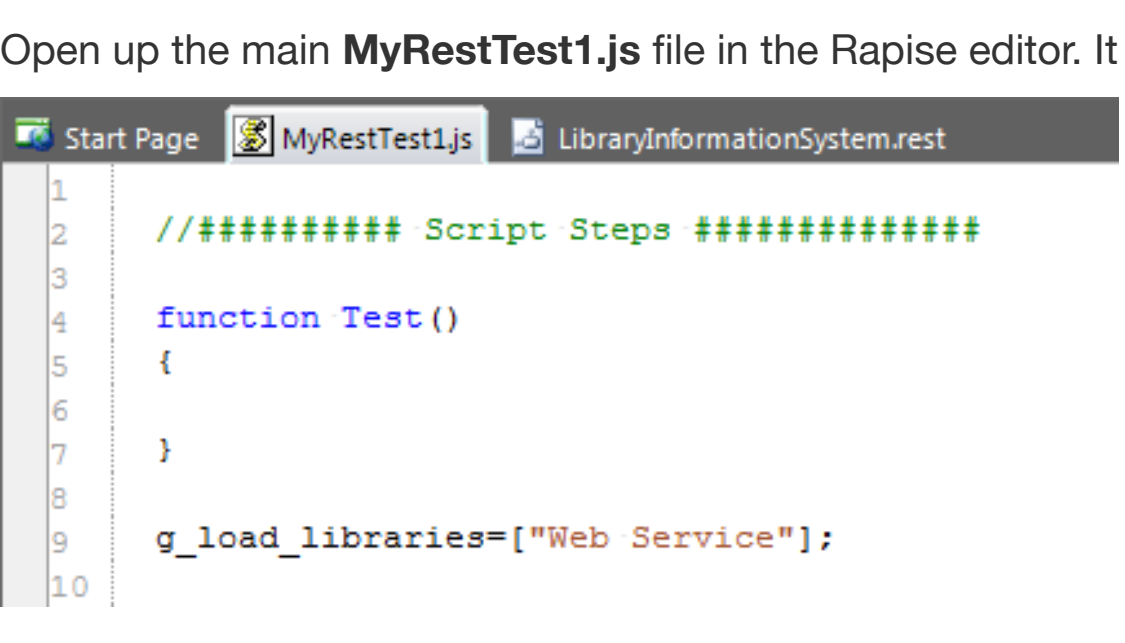


If you expand one of the REST requests (e.g. Add_Book), you'll see that it has a single operation **"DoExecute"** that executes the web services and a series of properties available for inspecting or updating any part of the REST request prior to it being sent to the server.

In the next section we shall illustrate how you can write a test script using these learned objects.

3. Writing REST Test Scripts

Open up the main **MyRestTest1.js** file in the Rapise editor. It will initially consist of a single empty function Test():



```
//########## Script Steps ##############

function Test()
{

}

g_load_libraries=["Web Service"];
```

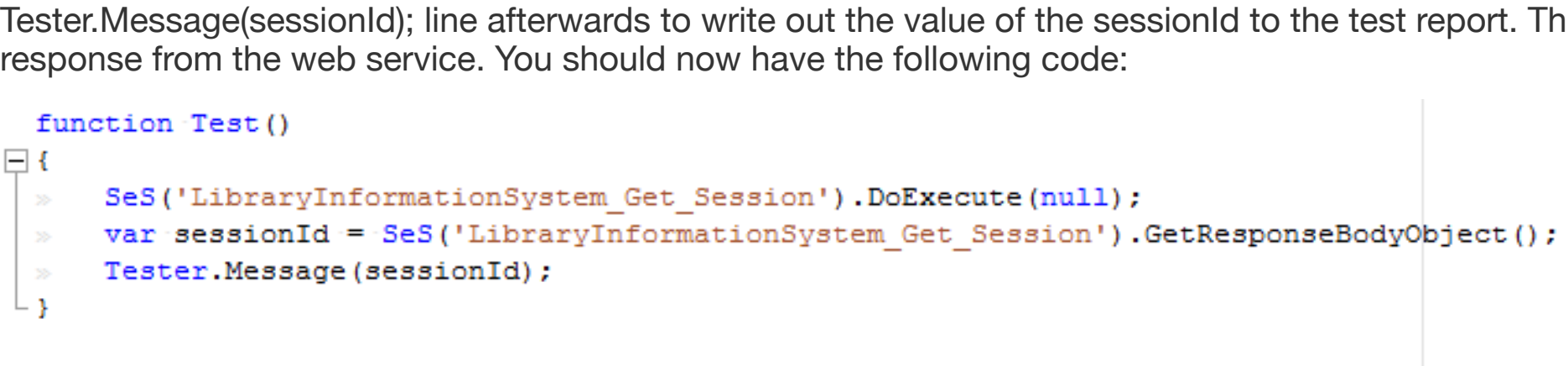
The first task is to get a new SessionId from the server using the **Get_Session** operation. To do this, drag the **"DoExecute"** operation from under the **"Get_Session"** object into the script editor, in between the opening and closing braces of the Test() function:

```
//########## Script Steps ##############

function Test()
{
    SeS('LibraryInformationSystem_Get_Session').DoExecute(null);
}

g_load_libraries=["Web Service"];
```

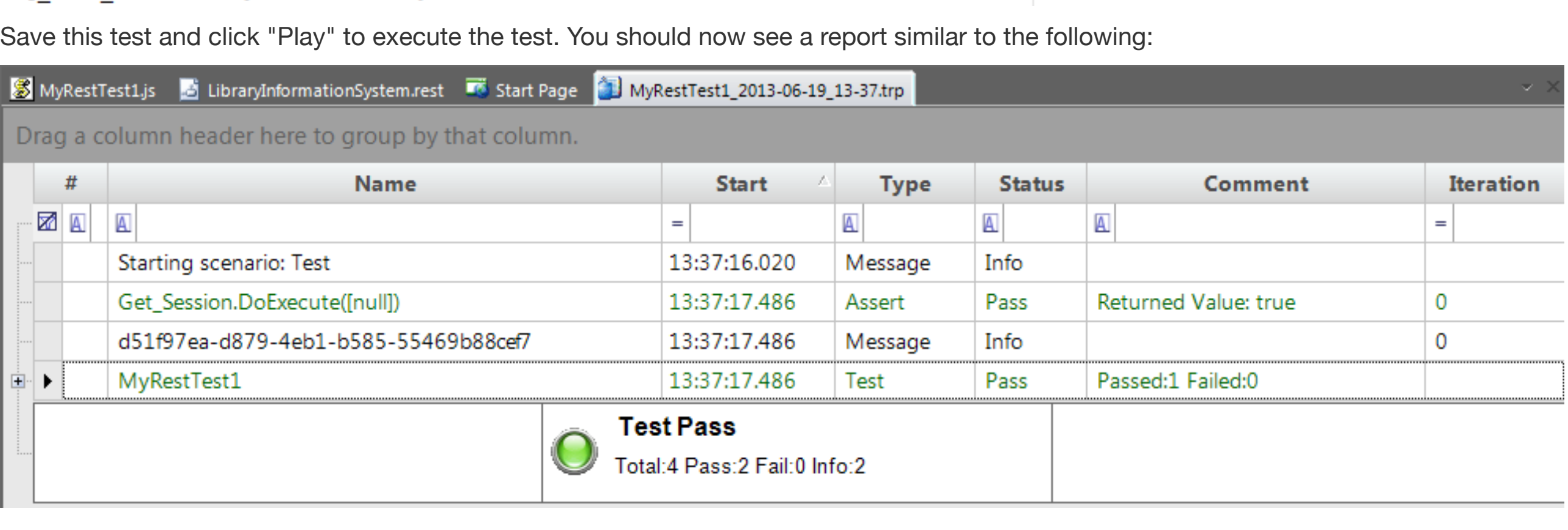
This will execute the web serviced and return the SessionId. To actually access the retrieved value, you need to drag the **"GetResponseBodyObject"** property to the script editor, under the previous line. Then add the JavaScript code var sessionId = to actually store the value. We will also add a Tester.Message(sessionId); line afterwards to write out the value of the sessionId to the test report. This will help us make sure we are getting back a valid response from the web service. You should now have the following code:



```
function Test()
{
    SeS('LibraryInformationSystem_Get_Session').DoExecute(null);
    var sessionId = SeS('LibraryInformationSystem_Get_Session').GetResponseBodyObject();
    Tester.Message(sessionId);
}

g_load_libraries=["Web Service"];
```

Save this test and click "Play" to execute the test. You should now see a report similar to the following:



#	Name	Start	Type	Status	Comment	Iteration
	Starting scenario: Test	13:37:16.020	Message	Info		
	Get_Session.DoExecute([null])	13:37:17.486	Assert	Pass	Returned Value: true	0
	d51f97ea-d879-4eb1-b585-55469b88cef7	13:37:17.486	Message	Info		0
	MyRestTest1	13:37:17.486	Test	Pass	Passed:1 Failed:0	

Test Pass
Total:4 Pass:2 Fail:0 Info:2

Now we need to add the code to get the list of books. To do that, simply drag the **"DoExecute"** operation from under the **"Get_Books"** object into the script editor. Then change the (null) argument to instead provide the session id as a Javascript dictionary:

```
SeS('LibraryInformationSystem_Get_Books').DoExecute({"session_id":sessionId});
```

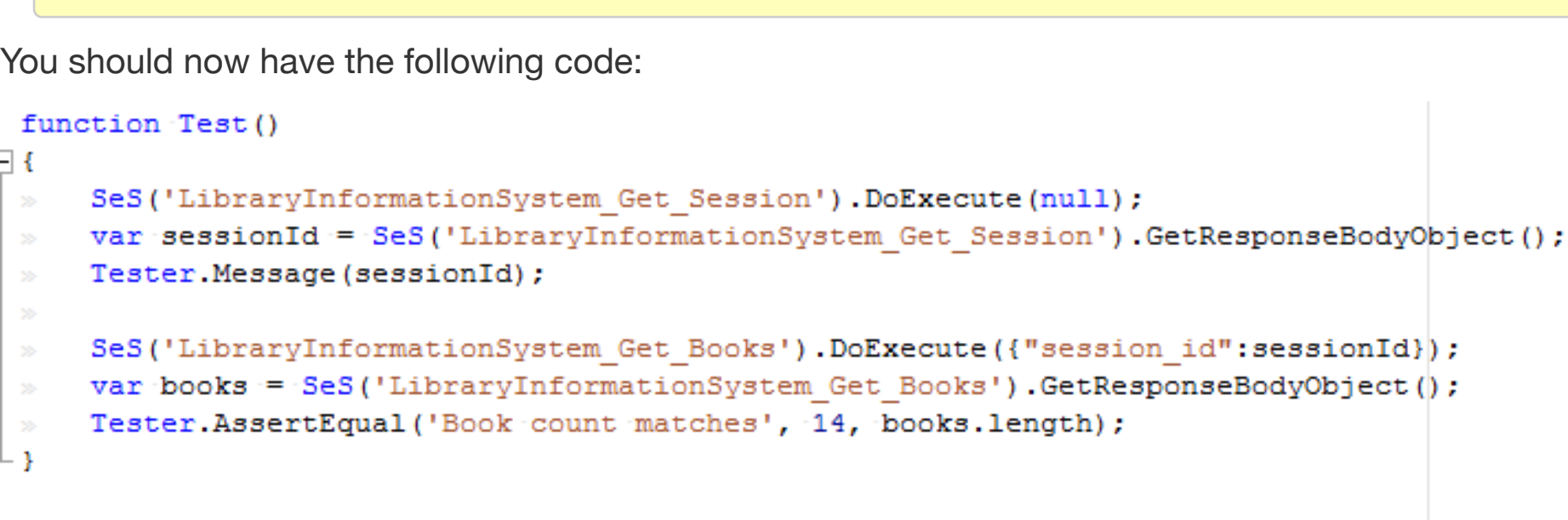
To get the list of books as a JavaScript array, drag the **"GetResponseBodyObject"** property to the script editor, under the previous line. Then assign the value of this property to a variable such as "books":

```
var books = SeS('LibraryInformationSystem_Get_Books').GetResponseBodyObject();
```

Now we can add code to test that the number of books returned matches the expected value. Type in the following code:

```
Tester.AssertEqual('Book count matches', 14, books.length);
```

You should now have the following code:



```
function Test()
{
    SeS('LibraryInformationSystem_Get_Session').DoExecute(null);
    var sessionId = SeS('LibraryInformationSystem_Get_Session').GetResponseBodyObject();
    Tester.Message(sessionId);

    SeS('LibraryInformationSystem_Get_Books').DoExecute({"session_id":sessionId});
    var books = SeS('LibraryInformationSystem_Get_Books').GetResponseBodyObject();
    Tester.AssertEqual('Book count matches', 14, books.length);
}

g_load_libraries=["Web Service"];
```

Finally we need to add the code to add a new book to the system. To do that, simply drag the **"DoExecute"** operation from under the **"Add_Book"** object into the script editor. Then change the (null) argument to instead provide the session id as a Javascript dictionary:

```
SeS('LibraryInformationSystem_Add_Book').DoExecute({"session_id":sessionId});
```

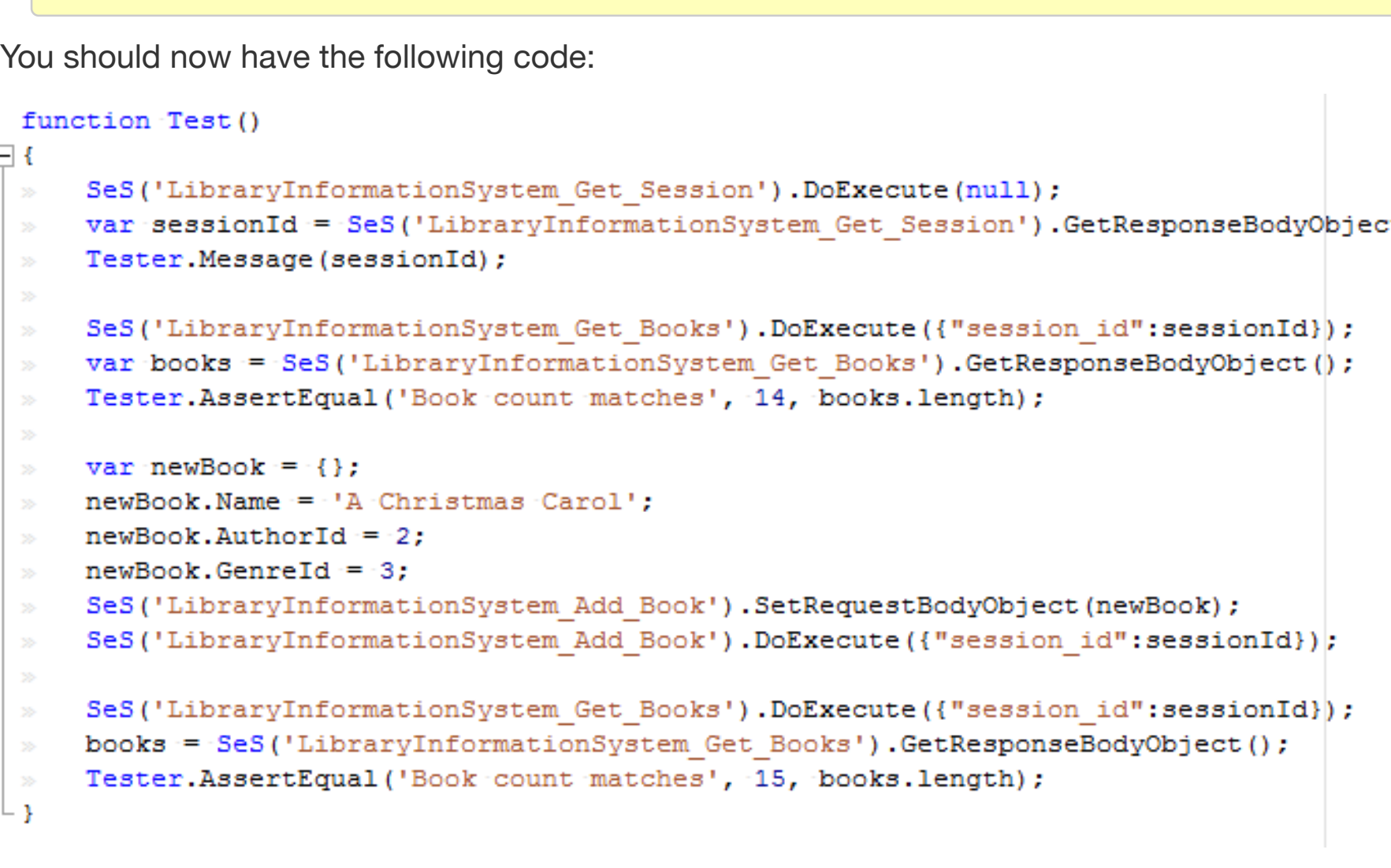
To provide the data for a new book, we will need to drag the **"SetRequestBodyObject"** property of the **"Add_Book"** object to the line **above** the DoExecute and pass in a populated JavaScript object:

```
                    var newBook = {};
                    newBook.Name = 'A Christmas Carol';
                    newBook.AuthorId = 2;
                    newBook.GenreId = 3;
                SeS('LibraryInformationSystem_Add_Book').SetRequestBodyObject(newBook);
```

Finally Add code to test that our new book was added correctly and the count has increased by one:

```
                    SeS('LibraryInformationSystem_Get_Books').DoExecute({"session_id":sessionId});
                    books = SeS('LibraryInformationSystem_Get_Books').GetResponseBodyObject();
                    Tester.AssertEqual('Book count matches', 15, books.length);
```

You should now have the following code:



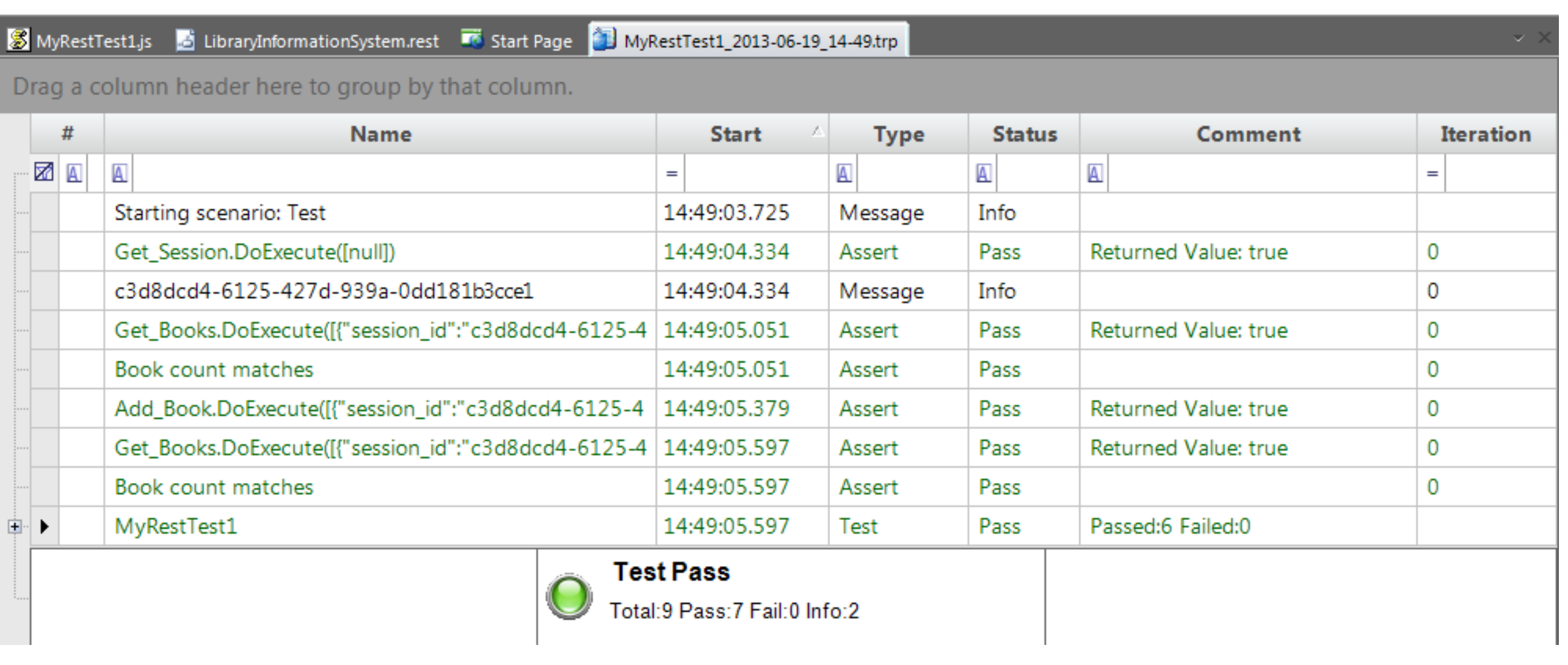
```
function Test()
{
    SeS('LibraryInformationSystem_Get_Session').DoExecute(null);
    var sessionId = SeS('LibraryInformationSystem_Get_Session').GetResponseBodyObject();
    Tester.Message(sessionId);

    SeS('LibraryInformationSystem_Get_Books').DoExecute({"session_id":sessionId});
    var books = SeS('LibraryInformationSystem_Get_Books').GetResponseBodyObject();
    Tester.AssertEqual('Book count matches', 14, books.length);

    var newBook = {};
    newBook.Name = 'A Christmas Carol';
    newBook.AuthorId = 2;
    newBook.GenreId = 3;
    SeS('LibraryInformationSystem_Add_Book').SetRequestBodyObject(newBook);
    SeS('LibraryInformationSystem_Add_Book').DoExecute({"session_id":sessionId});

    SeS('LibraryInformationSystem_Get_Books').DoExecute({"session_id":sessionId});
    books = SeS('LibraryInformationSystem_Get_Books').GetResponseBodyObject();
    Tester.AssertEqual('Book count matches', 15, books.length);
}
```

Save this test and click "Play" to execute the test. You should now see a report similar to the following:



#	Name	Start	Type	Status	Comment	Iteration
	Starting scenario: Test	14:49:03.725	Message	Info		
	Get_Session.DoExecute([null])	14:49:04.334	Assert	Pass	Returned Value: true	0
	c3d8dcd4-6125-427d-939a-0dd181b3cce1	14:49:04.334	Message	Info		0
	Get_Books.DoExecute([{"session_id":"c3d8dcd4-6125-4	14:49:05.051	Assert	Pass	Returned Value: true	0
	Book count matches	14:49:05.051	Assert	Pass		0
	Add_Book.DoExecute([{"session_id":"c3d8dcd4-6125-4	14:49:05.379	Assert	Pass	Returned Value: true	0
	Get_Books.DoExecute([{"session_id":"c3d8dcd4-6125-4	14:49:05.597	Assert	Pass	Returned Value: true	0
	Book count matches	14:49:05.597	Assert	Pass		0
	MyRestTest1	14:49:05.597	Test	Pass	Passed:6 Failed:0	

Test Pass
Total:9 Pass:7 Fail:0 Info:2

Congratulations! You have just created your first test script that tests a RESTful web service.