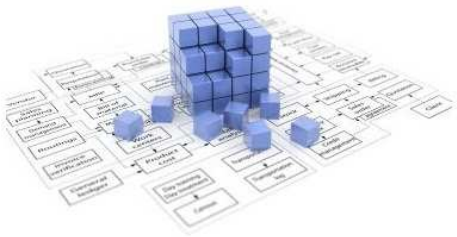
PSG REST API



Introduction:

Starting with September 2013, the PSG application server includes a new API for data integration with 3rd party applications, applications that do not use the PSG client interfaces. It is a REST API that implements basic CRUD operations based on JSON data encapsulation.

The PSG RESTAPI provides a single URI that acts as the service endpoint.

You will be able to access the REST API service endpoint using the GET/POST/PUT/DELETE REST HTTP.

The PSG REST API is intended to offer services to external applications; JSON is implemented as first option. This feature opens the server for different applications like HTML5 + JQuery or server applications data exchange.

Please check the latest documentation from:
http://www.psgsdk.com/downloads/psg_rest_api.pdf

Please contact the technical support for any questions you may have support@psgsdk.com .

Index:

1. PSG Application server

The PSG application server is a part of PSG SDK from "Programming Solutions Group". The PSG application model is client/server n-tier using HTTPS as communication protocol and REST services.

Documentation and downloads can be found at http://www.psgsdk.com.

PSG application server offers services to PSG clients over TCP/IP using HTTPS; the programming model reduces the learning curve for client server applications by encapsulating functionalities. However the PSG services can be accessed only by a PSG client application; the PSG REST API is created to offer services to 3rd party application in order to extend the PSG solutions or to create integrations with other systems that need a communication mechanism.

2. PSG REST API Implementation

PSG implements basic CRUD operations by default as open REST services; the API offers the possibility to modify and extend the available services as it is needed.

Implements:

- Query (GET) - get a query result as JSON string
- Insert (CREATE) – insert records into database tables
- Update – update a table record
- Delete – delete a table record

PSG uses HTTPS for security and basic authorization.

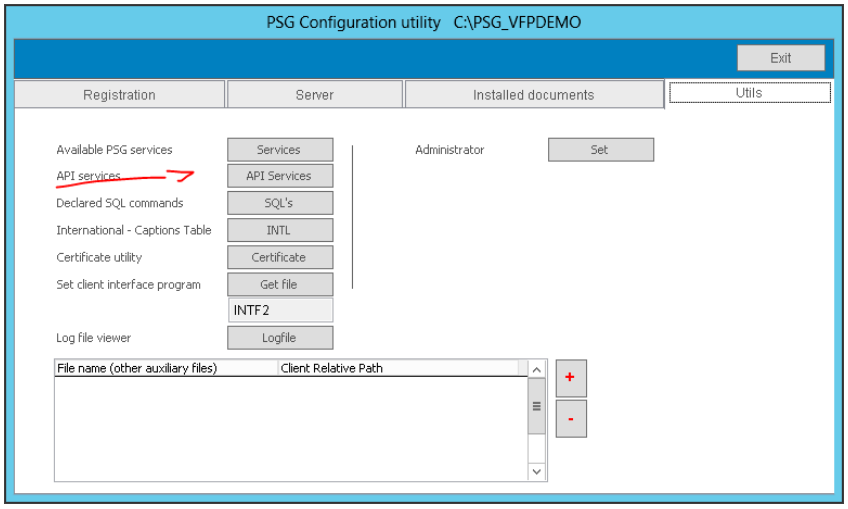
Script programming languages are Javascript, Vbscript, C# and VFP.

Default services are pointed to the database and used by the implemented PSG application; this can be changed in services code.

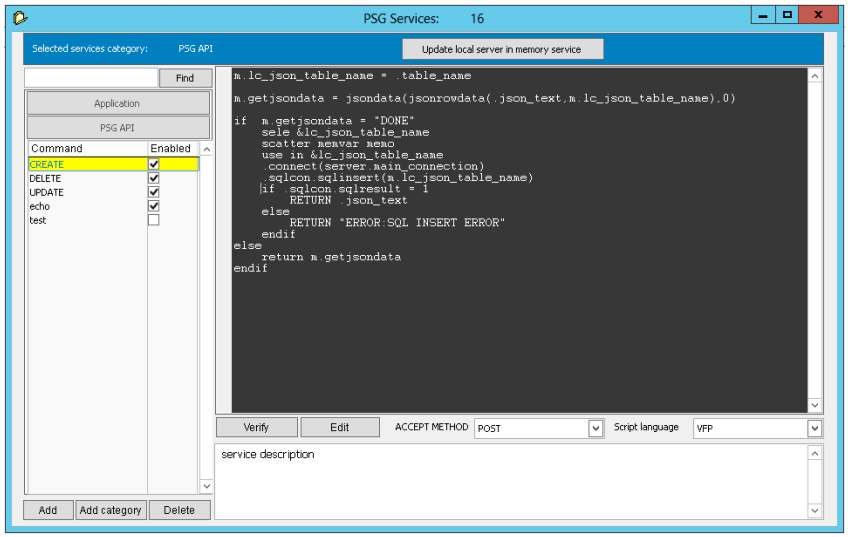
2.1. PSG API services management

Implemented services can be controlled from PSG server configuration tool using the "PSG API Commands" user interface found in configuration "Utils" page.

PSG server configuration interface starts from the "PSG server manager" application shortcut.



API service command button:



Services are stored under CATEGORIES; each service can be written in one of the available programming languages.
For each service corresponding "ACCEPT METHOD" should be set (GET, POST, PUT, and DELETE).

If the server is started (for development servers) press the "Update local server in memory service" button to apply changes to the services to working server. Available services are loaded when server starts and kept in memory for faster access.

2.2. Services scripts

REST API services are implemented as scripts; for the developer's convenience several programming languages can be used: Javascript, Vbscript, C#, VFP. Please check C# SDK documentation for more info related to C# service implementation.

Programming language is selected from the right lower corner of scripts editor.

For different purposes, REST services uses a different "accept method"; this can be set using "Accept Method" ComboBox.

Navigation through services list uses a menu by category (left side of the user interface). Services can be disabled using the attached checkbox.

Objects, methods and properties of the server user connection can be used into scripts.

Sample Javascript (GET JSON data); implements service for data request as JSON, accepted method is GET.

***

```
connection.connect(server.main_connection);
sqlcon.cname =  'fcustomers';
sqlcon.sqlcommand = "select * from fcustomers";
sqlcon.sqlexec();
connection.response.body = connection.data2json('fcustomers',1);
```

****

2.3. Script objects, methods and properties

(Upper objects heritage)

Objects passed as references:

SEERVER – object to access server methods and properties
CONNECTION – object to access user connection methods and properties
SQLCON – a SQL connection used for data access to SQL servers

Each of these objects can be used in the script code.

You can see below methods and properties useful for REST API purposes.

2.3.1 SERVER object:

Methods list:
MSG – put a message on the server window (developer mode/debugging purposes)

Javascript sample:

```
Server.msg("something here");
```

Properties list:

main_connection - connection string to PSG application database in use

Javascript sample:

```
connection.connect(server.main_connection);
```

2.3.2. CONNECTION object:

CONNECTION is the main object to be used within scripts; it is the upper object of script interpreter. Next are presented the useful methods and properties for basic API services.

Methods list:

connect connect(<conn string>)	Set the default connection string and connect the database Sample (Javascript): connection.connect(server.main_connection);
data2json data2json(<query result name>)	Transforms a query data result to Json string Sample (Javascript): connection.data2json("mydata"); See also READ / CRUD operations.
data2jsonpart	Transform a data result to a partial JSON response, used to create a JSON data set (collection of data arrays), useful when multiple responses for one request are needed to increase the page load speed. Sample (Javascript) : …. var d1 = cursor2jsonpart(connection.sqlcon.cname,1); …. connection.response.body = '{"data":{'+d1+','+d2+'}}';

jsonrowdata jsonrowdata(<json row string>,)	By default a CREATE/UPDATE/DELETE request will come as a single JSON "row". jsonrowdata transform a JSON row to a JSON table with one row only to be used next by jsondata. Sample (Javascript): Connection.jsonrowdata(connection.json_text,table_name)
jsondata jsondata(<json source>,<source type>)	Used to transform a JSON data/table string to table, "source type" is 0 for string and 1 for JSON text file Sample (Javascript): jsondata(jsonrowdata(connection.json_text, lc_json_table_name),0)

get_parameter(<parameter number>,<1=value,2=name>)
 	
or
 	
get_parameter_byname(<parameter name>)
	A list of parameter can be sent with the request, the parameters are sent in an array, and this method will return a parameter in list.
	
	Sample (Javascript):
	orderid = connection.get_parameter(1,1);
	orderid = connection.get_parameter_byname('orderid');
	Sample URI (need authorization basic):
	https://psgamazon.servehttp.com:30200/psgapi/orderdetails?orderid=O3BZ0PTWLAAA

sqlcon object	SQLCON object is an object in "connection" It is passed as an object but can be used also from "connection". See SQLCON methods and properties next. Sample (Javascript): connection.sqlcon.cname = "test";
Gettime	GET a date-time value

Properties list:

table_name	table name to be used with other methods, it is the name of the "database table"/"local data set" to apply service command to. Sample (Javascript): connection.table_name
json_text	JSON text that comes with the REQUEST, passed to the USER connection Sample (Javascript):

	connection.json_text
username	User name used by the client to access the API
Remoteip	User computer IP
Usrcode	PSG user ID (string 10)

2.3.3. SQLCON object:

A database connection object used to send SQL commands and query the database.

Methods list:

SQLEXEC	Execute declared SQL command Sample (Javascript): sqlcon.sqlcommand = "select * from table_name"; sqlcon.sqlexec();

JSQLINSERT	Insert a row from memory variables.
	
	Sample (Javascript):
	sqlcon.set_new_row();
	sqlcon.set_row_field("test_id",ctest_id,1) ;
	sqlcon.set_row_field("name",cname,0);
	sqlcon.set_row_field("description",cdescription,0);
	sqlcon.jsqlinsert("test");

JSQLUPDATE	Send a SQL record update
	Sample (Javascript):
	sqlcon.set_new_row();
	sqlcon.set_row_field("test_id",ctest_id,1) ;
	sqlcon.set_row_field("name",cname,0);
	sqlcon.set_row_field("description",cdescription,0);
	sqlcon.jsqlupdate("test");

SQLSTRINGCONNECT	Connects to database, connection string must be a valid ODBC connection string, returns -1 for a failed connection.
SQLDISCONNECT	Disconnect a established connection
SET_NEW_ROW	Declare a new row array to be used next by Insert and

	Update commands. See previous JSQLINSERT sample.
SET_ROW_FIELD	Declare a new row field to row array to be used next by Insert and Update commands. See previous JSQLINSERT sample.

Using transactions with SQLCON:

By default transactions are not used, in order to declare a transaction you should check the used database SQL commands and use SQLCOMMAND and SQLEXEC to send the proper command to start/rollback/commit/close a transaction. SQLCON it is not bound to a specific database server.

Properties list:

CNAME	Name of the QUERY result data set/cursor Sample (Javascript): sqlcon.cname = "test";

SQLRESULT	A status of executed command/method. 1 = OK, -1 SQL server side errors occurs.
	
	Sample (Javascript):
	if (sqlcon.sqlresult = 1)
	{
	connection.response.body = "DONE OK";
	}

SQLCOMMAND	SQL statement to be sent to the server by SQLEXEC method. Become empty after used by SQLEXEC. Sample (Javascript): sqlcon.sqlcommand = "select * from test"

3. Implemented CRUD Operations

PSG REST API services are open to be modified or extended according to the application's needs. Basic implemented services are as close to standards as possible; there are different standards for REST SERVICES; next we treat the basic implementation of CRUD operations as understood by default by PSG REST API.

Basic needed services to extend access to an application are related to data exchange.
PSG API requires HTTPS and basic authorization at least.

PSG REST API implements by default:

- Query (GET) – Query the database and return a JSON string (it uses the GET accept method)
- CREATE – Inserts a new record to a database table
- UPDATE – Updates an existing table record
- DELETE – Deletes a table record

3.1 READ

CREATE service insert a new record to the database.
PSG REST API implements a default CREATE SERVICE.

Service request is sent using "POST" accept method, the URI has table reference like:

URI: https://<server address> : < server port >/<API access point>/<server table name>

"Server port" needed when different by the default server port (443).
"server table name" is the actual database table name to be used.
"API access point" is the single entry point for the PSG REST API, default is set to "psgapi".

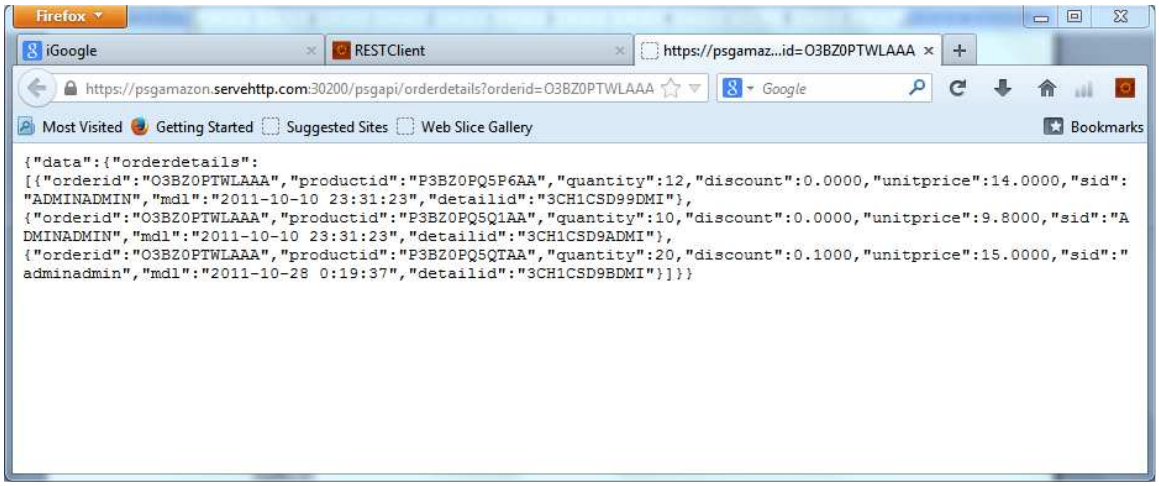
And the insert row request is sent using the REQUEST.BODY:

Request body Sample:
{"id_test":"M36419JSJX","name":"test name","description":"Hello world!"}

The body should be one JSON row with all needed fields to do the INSERT otherwise the database server will send back an SQL error.

3.2 READ

It is no service with this name, instead one service should be implemented for each query request; we use this approach because each query can be different from a standard request and needs proper code. Standard QUERY request is sent to a REST API using GET accept method and parameters into URI like in the next sample:



URI: https://<server address>:<server port>/<API access point>/<table/service name>?<parameter name>=<parameter value>& …

Server port is needed when different by the default server port (443). The answer comes into RESPONSE.BODY as standard JSON table like the one in the picture.

Resulted JSON string looks like:


```
{"data":{"orderdetails":[{"orderid":"O3BZ0PTWLAAA","productid":"P3BZ0PQ5P6AA","quantity":
12,"discount":0.0000,"unitprice":14.0000,"sid":"ADMINADMIN","mdl":"2011-10-10
23:31:23","detailid":"3CH1CSD99DMI"},{"orderid":"O3BZ0PTWLAAA","productid":"P3BZ0PQ5Q1AA"
,"quantity":10,"discount":0.0000,"unitprice":9.8000,"sid":"ADMINADMIN","mdl":"2011-10-10
23:31:23","detailid":"3CH1CSD9ADMI"},{"orderid":"O3BZ0PTWLAAA","productid":"P3BZ0PQ5QTAA"
,"quantity":20,"discount":0.1000,"unitprice":15.0000,"sid":"adminadmin","mdl":"2011-10-28
0:19:37","detailid":"3CH1CSD9BDMI"}]}}
```


One JSON string can be tested with http://www.jsoneditoronline.org/ online editor.
Check the samples paragraph below for more details.

The service implemented for QUERY requests can be used to implement other needed services depending on your application's demands.

The QUERY services are implemented using the services editor using:

- Desired programming language
- Accept method should be set to "GET"
- A service name (not necessary to be the same as the database table name)

Sample script (Javascript):

```
connection.connect(server.main_connection);
sqlcon.cname =  'fcustomers';
sqlcon.sqlcommand = "select * from fcustomers";
sqlcon.sqlexec();
connection.response.body = connection.data2json('fcustomers',1);
```

3.3 UPDATE

UPDATE service updates a record into the database table.
PSG REST API implements a default UPDATE SERVICE.

Service request is sent using "PUT" accept method, the URI has table reference like:

URI: https://<server address> : < server port >/<API access point>/<server table name>

"Server port" needed when different by default server port (443).
"server table name" is the actual database table name to be used.
"API access point" is the single entry point for the PSG REST API, default is set to "psgapi".

And the update row request is sent using the REQUEST.BODY:

Request body Sample:
{"id_test":"M36419JSJX","name":"test name","description":"Hello world!"}

The body should be one JSON row with all needed fields to do the INSERT otherwise the database server will send back an SQL error.

The request command looks pretty much the same as CREATE excepting the accept method that is "PUT" in this case.

We do understand that the primary key is the first field in JSON row, which is used by default by PSG API UPDATE service. This is required.

3.4 DELETE

DELETE service deletes a new record into the database table.
PSG REST API implements a default UPDATE SERVICE.

Service request is sent using "PUT" accept method, the URI has table reference like:

URI: https://<server address> : < server port >/<API access point>/<server table name>

"Server port" needed when different by default server port (443).
"server table name" is the actual database table name to be used.
"API access point" is the single entry point for the PSG REST API, default is set to "psgapi".

And the update row request is sent using the REQUEST.BODY:

Request body Sample:
{"id_test":"M36419JSJX"}

The body should be one JSON row with all needed fields to do the INSERT otherwise the database server will send back an SQL error.

The request command looks pretty much the same as CREATE excepting the accept method that is “DELETE” in this case.

We do understand that the primary key is the only in JSON row, which is used by default by PSG API UPDATE service. DELETE should be used carefully with not primary key fields as it will delete all records that match the expression.

DELETE can be implemented using parameters instead a JSON body request. In which case the method should be like:

4. Cross-Origin Resource Sharing (CORS) and JSONP

For WEB applications that use a standard Internet browser like Firefox, IE or Chrome:

Internet browsers comply with the “Same-origin policy” http://en.wikipedia.org/wiki/Same_origin_policy.
“The policy permits scripts running on pages originating from the same site – a combination of scheme, hostname, and port number – to access each other's methods and properties with no specific restrictions, but prevents access to most methods and properties across pages on different sites. Same-origin policy also applies to XMLHttpRequest and to robots.txt.”

What’s the idea? REST API’s generally are created to offer services to other applications/sites and do not comply with this basic rule. In order to avoid situations when a WEB application can not use a REST API there are generally two known options:

- JSONP (http://en.wikipedia.org/wiki/JSONP), basically the JSON string is encapsulated into a function.
    - To specify that JSON should be used you must add a new parameter to the request
        - callback=yourfunctionname OR jsonp=yourfunctionname
        - the PSG server will use <yourfunctionname> to encapsulate the response
        - client side a JavaScript function <yourfunctionname> must exist to use the JSNOP string sent back
- CORS / Cross-Origin Resource Sharing are W3C specifications that’s better to be used instead JSONP
        - Implements response headers and request headers that should be used
        - PSG application server implements the required headers to comply with W3C rules http://www.w3.org/TR/2013/CR-cors-20130129/

PSG application server accepts both methods to be used. However recommended method is CORS. JSONP is recognized as a not standardized hack or workaround.

5. Samples

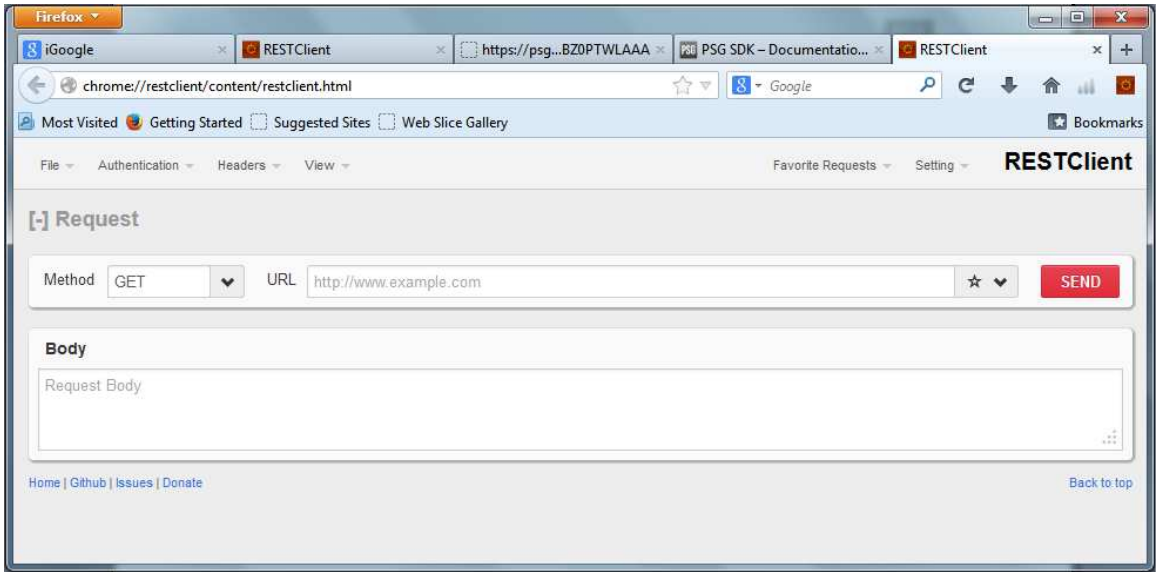
Please check the latest release of this document for updated information.

By default PSG application server use a self signed certificate, to be sure that you can access next samples you must navigate to the server address and add the certificate into accepted list; otherwise the samples described below may not work. PSG can use certificates signed by a certified authority to avoid this situation.

The sample server address is: https://psgamazon.servehttp.com:30200/ ; basic AMAZON EC2 Windows 2012 server. An Amazon AMI virtual machine can be used for implementing your own server for detailed testing and check into AMAZON cloud.

API access point is "psgapi".

In order to test PSG REST API services you need a client to sent proper requests that verify the standard. One client is provided by a Firefox add-on from: https://addons.mozilla.org/ro/firefox/addon/restclient/ .



You will need to set the "authentication" to basic:

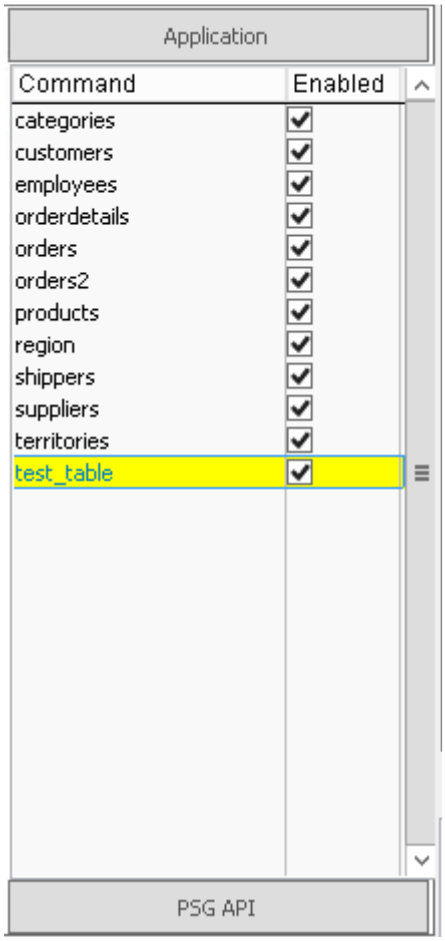
- Username = "Administrator"
- Password = "admin"

Set the METHOD depending what it is tested, the URL and the BODY if needed, next press SEND button.

Check next samples for basic CRUD using PSG REST API and Chrome://restclient

Query sample: get data from the server as standard JSON table.

The PSG sample server implements next data services:



The sample URL request is:

https://psgamazon.servehttp.com:30200/psgapi/<table name>

"orderdetails" require one parameter like

https://psgamazon.servehttp.com:30200/psgapi/orderdetails?orderid=O3BZ0PTWLAAA

The used method is GET.

Server response:

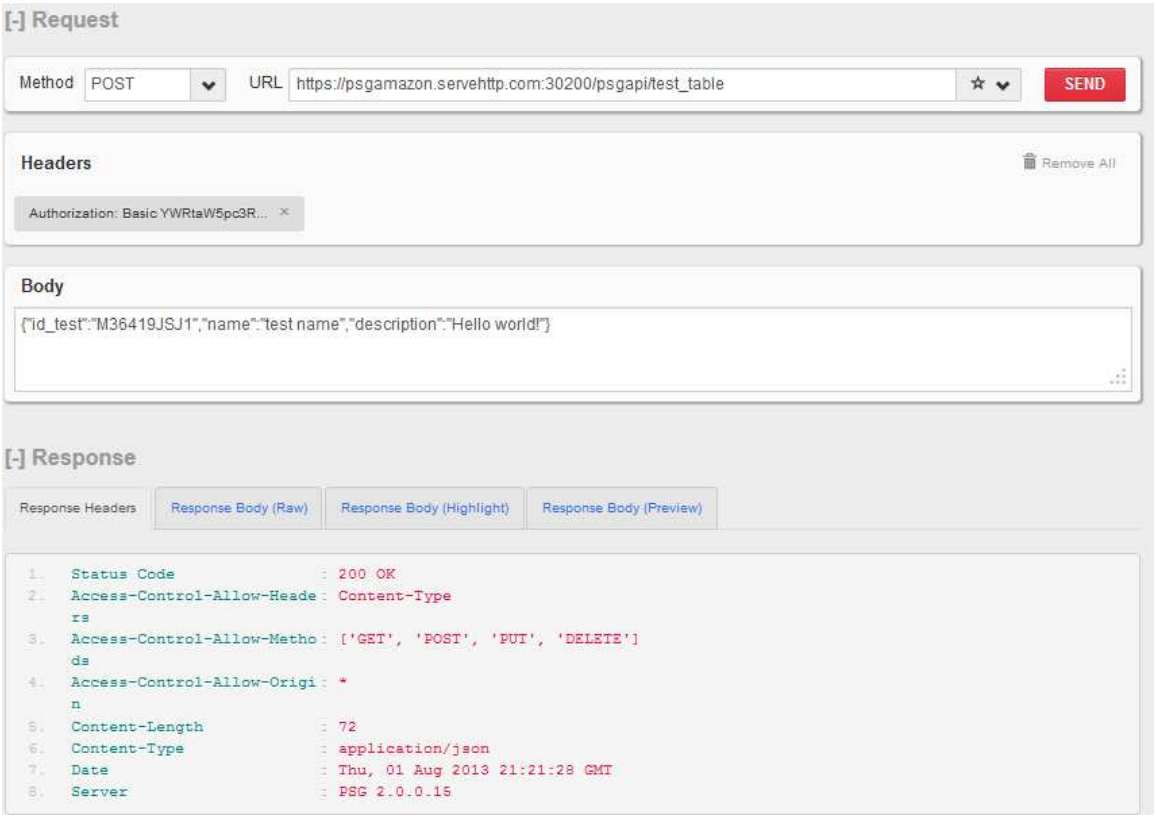
- `Status Code: 200 OK`

Or error messages like 400,401 or 407 if errors occurs with the request (400 Bad request/ 401 – authorization failed/ 407 resource not found)

The JSON string look like: (comes to RESPONSE.BODY)


```
{"data":{"orderdetails":[{"orderid":"O3BZ0PTWLAAA","productid":"P3BZ0PQ5P6AA","quantity":12,"discount":0.0000,"unitprice":14.0000,"sid":"ADMINADMIN","mdl":"2011-10-10 23:31:23","detailid":"3CH1CSD99DMI"},{"orderid":"O3BZ0PTWLAAA","productid":"P3BZ0PQ5Q1AA","quantity":10,"discount":0.0000,"unitprice":9.8000,"sid":"ADMINADMIN","mdl":"2011-10-10 23:31:23","detailid":"3CH1CSD9ADMI"},{"orderid":"O3BZ0PTWLAAA","productid":"P3BZ0PQ5QTAA","quantity":20,"discount":0.1000,"unitprice":15.0000,"sid":"adminadmin","mdl":"2011-10-28 0:19:37","detailid":"3CH1CSD9BDMI"}]}}
```

CREATE sample

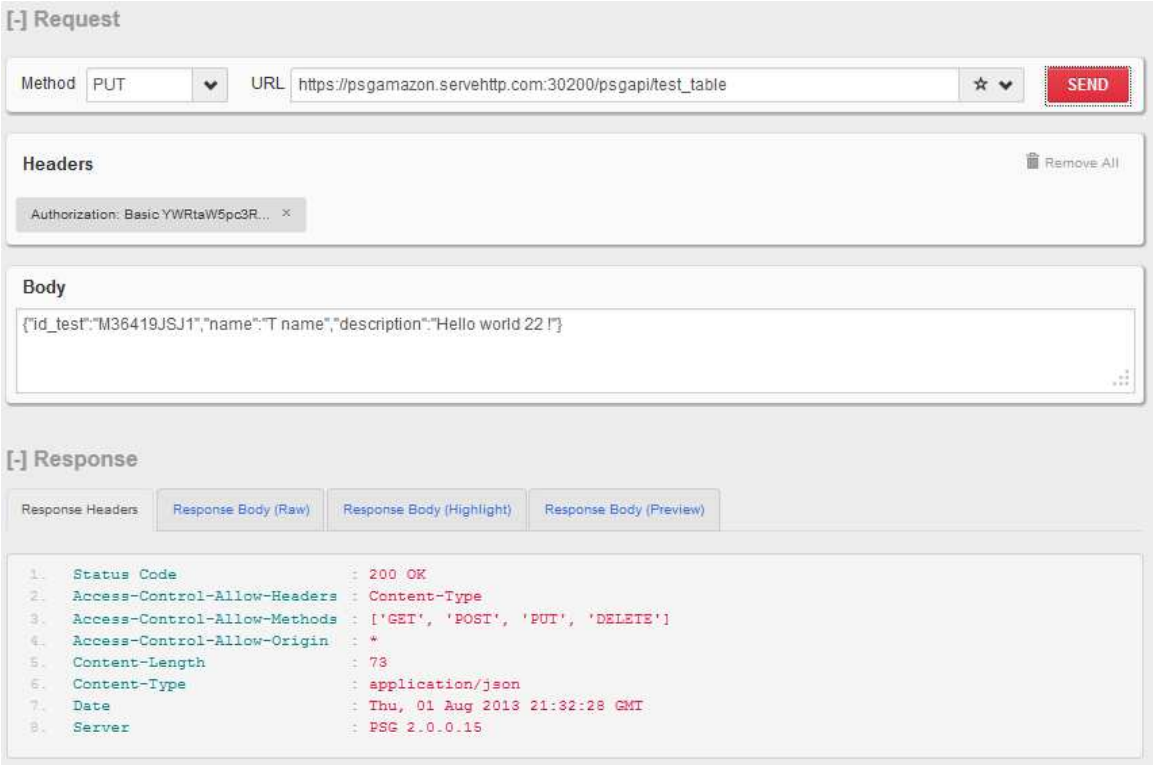


Use:

- Basic authorization as described before
- Method = "POST"
- URL = https://psgamazon.servehttp.com:30200/psgapi/test_table
- Body = {"id_test":"M3641YJSJ1","name":"test name","description":"Hello world!"}

Returns the same body as the request and that means OK, status code should be 200 ok, a second attempt with the same data row (same primary key) will generate an error (SQL error).

UPDATE sample



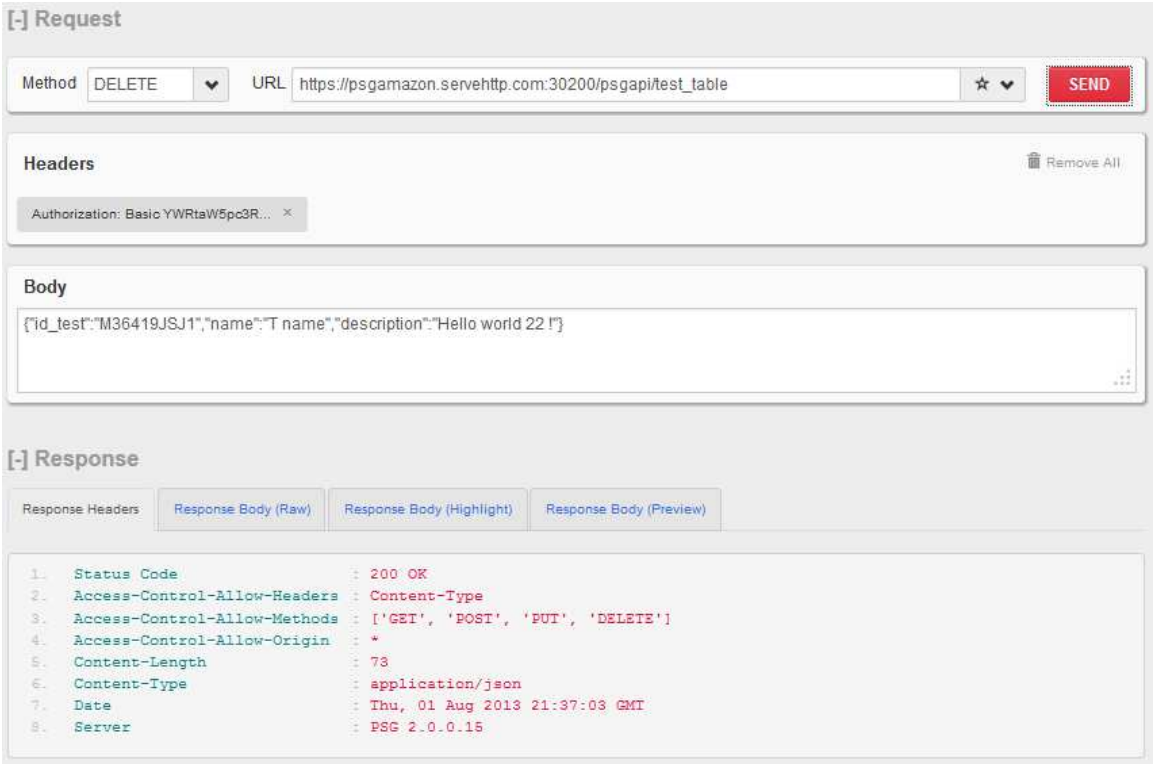
Use:

- Basic authorization as described before
- Method = "PUT"
- URL = https://psgamazon.servehttp.com:30200/psgapi/test_table
- Body = {"id_test":"M3641YJSJ1","name":"T name","description":"Hello world 22!"}

Returns the same body as the request and that means OK, status code should be 200 ok.

Depending on the database server can return OK even if the record does not exist, the database server interprets as a good statement!

DELETE sample



Use:

- Basic authorization as described before
- Method = "DELETE"
- URL = https://psgamazon.servehttp.com:30200/psgapi/test_table
- Body = {"id_test":"M3641YJSJ1"}

Returns the same body as the request and that means OK, status code should be 200 ok.

Depending on the database server can return OK even if the record does not exist, the database server interprets as a good statement!

6. Security

PSG Application server uses HTTPS / SSL3 and basic authorization.
Standard signed certificates can be used with PSG server, for flexibility the server tools includes a "certificate" utility that can be used to create self signed certificates.

Self signed certificates offer the same security but will trigger a security alert to Internet browsers that require user intervention to accept the certificate; to avoid this you should use a certificate from an authority like VeriSign or any alike can be installed in place. For other kind of applications than Internet web browsers this security alert can be avoided by your application.

Other authorization types can be implemented, please send a request to support@psgsdk.com with your specifications and requirements.

PSG application server security model implemented with PSG client's interfaces offers more control over the applications accessing the server. The PSG REST API is disabled by default and should be used with much care because the services will be available for all applications that send authorization credits (please read CREATE/UPDATE/DELETE in previous pages). Each service type can be disabled separately.

7. Licensing

The PSG REST API is a PSG application server implemented feature, it is not sold and can not be used separately from PSG application server. It is distributed only with a PSG application server. However a PSG application server can be used just to implement a REST API access point without a PSG application to run.

The PSG application server is licensed for a server users' number. Please check the PSG SDK licensing agreement for more details. PSG REST API will accept maximum connections that are limited to licensed users number multiplied by five. (Ex: for 5 declared users there are 25 connections slots available or 25 requests can be made at the same moment no matter which user provides the authorization).

There is a free license option for 5 users but does not implement HTTPS as communication protocol, HTTP only.

http://www.psgsdk.com/partners.php#model
http://www.psgsdk.com/docs/pricing/

8. Contact

PSG SDK is developed and maintained by "Programming Solutions Group LLC" and its partners.

Support website is https://www.psgsdk.com.

Please check if you have the latest releases and documentation available.
Technical articles and news can be found at http://www.psgsdk.com/docs/

Technical support is offered free of charge by email at support@psgsdk.com /
http://www.psgsdk.com/contact.php