C# AND PHP INTEGRATION

Kishan Jainandunsing, PhD
Senzo Labs

Introduction

A desired essential functionality of applications is client-server integration with the cloud. This can be for database and/or compute services. Client platforms run either of the following 5 mainstream operating systems: MS Windows, Linux, MacOS, iOS or Android. Each of these OSes support their own native programming languages and frameworks with client-server programmatic constructs.

Similarly, there are many server-side languages and frameworks in deployment. Some of the mainstream ones include Python, Node.JS and PHP.

Here we focus on the MS Windows .NET framework and C# language for the client side, and on PHP for the server side. We show how to develop data exchange interfaces between a C# client and PHP server for access to database services in the cloud. We specifically illustrate developing these interfaces using the C# WebClient and HttpWebRequest/HttpWebResponse constructs.

WebClient

C# function – client-side:

```
using System.Net;
public Boolean UploadFileToDatabase(string content, string filename) {
   byte[] s = Encoding.ASCII.GetBytes(content); // convert the string formatted file content to an array of bytes
   // convert byte array into a string with each byte separated by ^ to avoid special chars in the PHP
   string byte_content = string.Join("^", s);
   WebClient client = new WebClient(); // instantiate a web socket
   NameValueCollection inputs = new NameValueCollection(); // instantiate an API parameters + input container
   inputs.Add("filename", filename); // add the file name parameter and value to the container
   inputs.Add("content", byte_content); // add the content parameter and content to the container
  string APIurl = “http://myApiUrl/myAPI.php"; // the API call URL for the web socket
  byte[] isSuccess = client.UploadValues(APIurl, inputs); // execute the "POST" API call
  string isDone = Encoding.UTF8.GetString(isSuccess); // convert the returned byte array to a string
  if (isDone != “false") return true;
  else  return false;
}
```

The above C# function converts the string passed to it via *content* into a byte array. This byte array is then converted into a string of bytes with the bytes separated by a ‘^’ character as the string *byte_content*. A new WebClient object is instantiated as *client* together with its *NameValueCollection* as *inputs* to hold the file name

and content (the latter as the byte string). The *client* object is then called with the API URL and the inputs as the parameters.

The server responds with the output of the PHP script in the API URL, which is read as a byte array by the client. This byte array is converted to a string and tested if it is *true* or *false*.

PHP script – server-side:

```
<?php
$filename = $_POST['filename'];
$content = $_POST['content']; //a byte string with each byte separated by a ^
$content_arr = explode('^',$content); //convert into array of bytes
$filecontent = implode(array_map("chr", $content_arr)); //convert into ASCII string
$retval = "false";
$sql = "insert into MY_TABLE (FILE_NAME,CONTENT,TIME_STAMP)
      values ('$filename','$filecontent,CURRENT_TIMESTAMP)";
if(mysqli_query($db, $sql)) $retval = "true";
echo $retval;
?>
```

The above PHP script simply reads in the filename and the file content passed to it via POST and uploads the content together with the file name and a date-time stamp to the database as a record in MY_TABLE.

The file content is passed to it as a string of bytes where the bytes are separated by a '^'. This string is the converted into an array of bytes via the *explode()* function, which is then converted into an array of ASCII characters via the *array_map()* function, and this array is then converted into an ASCII string via the *implode()* function.

HttpWebRequest/HttpWebResponse

C# function – client-side:

```
using System.Net;
public string getUserInfo(string uid, string state)
{
   string data = null;
    // create the URL with the API call and parameters
   string APIcall = "http://myApiUrl/myAPI.php? uid=" + uid + "&state=" + state;
    // make the data request via the API call
   HttpWebRequest request = (HttpWebRequest)WebRequest.Create(APIcall);
    myHttpWebRequest.Method = "GET"; // GET or POST, default is GET
   // get the response
   HttpWebResponse response = (HttpWebResponse)request.GetResponse();
   if (response.StatusCode == HttpStatusCode.OK) // check if the request-response succeeded
   {
       Stream receiveStream = response.GetResponseStream(); // create a stream object
       // create a stream buffer for the stream object to capture the response into
       StreamReader readStream = (response.CharacterSet == null)?
             new StreamReader(receiveStream)  : new StreamReader(receiveStream,
             Encoding.GetEncoding(response.CharacterSet));
             data = readStream.ReadToEnd(); // read the data out of the stream buffer
           readStream.Close(); //release stream resource for re-use
    }
    response.Close(); // release response resource for re-use
    return data;
}
```

The above C# function forms the API call to the PHP script on the server side in customary URL format as would be entered into a browser.

Next it instantiates a *WebRequest* object and binds it to the URL for the API call via the *WebRequest.Create()* method. *HttpWebRequest* by default issue a GET API call. To issue a POST API call requires the explicit setting POST via *HttpWebRequest.Method = "POST"* (in the example above: *request.Method = "POST"*).

Next, the *GetResponse()* method checks if the API call succeeded. If so, then a stream object, *response*, is created for the *WebRequest* instance. A buffer, *readStream*, is created and the server-side data stream from the PHP script is read into the string object *data*.

Note: It is important to properly close the *StreamReader* and the *HttpWebResponse* resources for re-use. Otherwise the server keeps opening new sockets and new resource requests will be refused. In the above example connections to the database by other applications will be refused.

PHP script – server-side:

```
<?php
$uid = $_GET['uid'];
$state = $_GET['state'];
```

```
$sql = "select FIRST,LAST from USERS where USER_ID = '$uid' and STATE = '$state'";
$result = mysqli_query($db, $sql);
$row = mysqli_fetch_array($result, MYSQLI_ASSOC);
echo json_encode($row);
?>
```

The above PHP script simply reads the data provided to it via the parameter ‘uid’ and performs a SQL query on the table USERS to retrieve the record which match ‘uid’ for table field USER_ID. The retrieved record is then encoded as a JSON in string format.

The JSON string is read by the WebClient instance on the client side via the method readStream.ReadToEnd(). From here onward the data can be further parsed using string parsing/manipulating methods available for C# string objects.

Large Data Uploads

There are key differences between the GET and POST API call methods, specifically concerning the amount of data to upload. The GET method limits the amount of data and the limit is browser dependent. The POST method has no such limit and is therefore the preferred choice when uploading entire file contents, for instance.