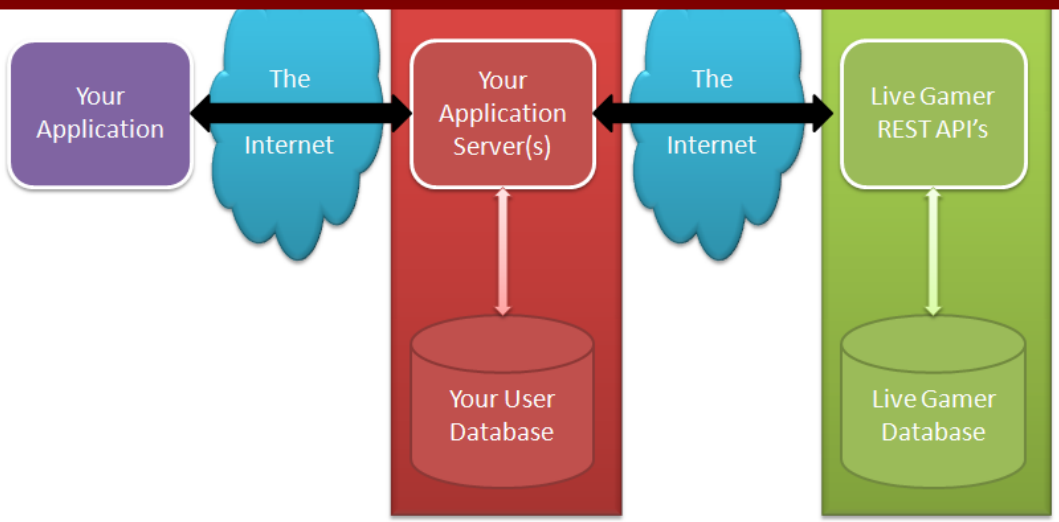
Tutorial 1: Calling the Elements REST API

ARCHITECTURE OVERVIEW

You can access most of Element's functionality through the Elements REST API. Each method accepts a set of parameters, and returns data in XML format. After setting up your server, your Live Gamer Account Manager will provide you with credentials.

The Elements REST API includes security parameters so that hackers cannot spoof your application. These must not be visible in the client application. For this reason, your client app will communicate with your servers, which will perform authentication and send requests to Elements. A typical architecture looks like:

TUTORIAL OVERVIEW



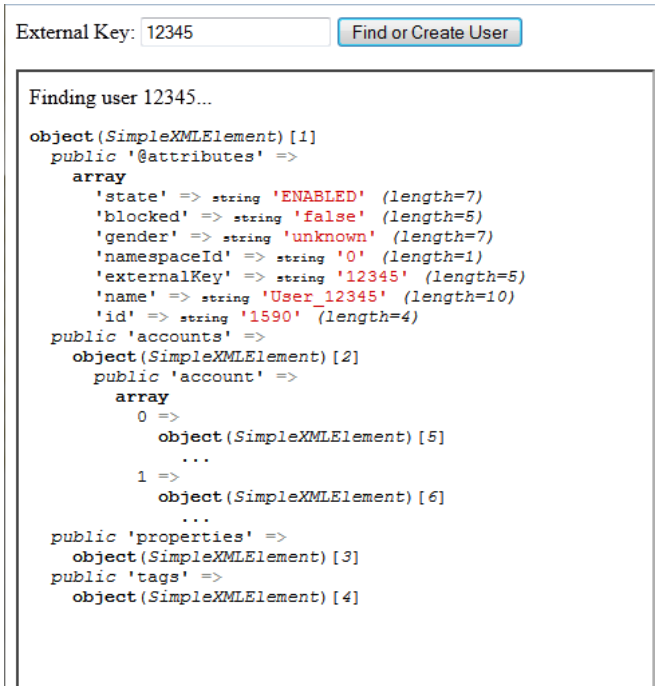
This tutorial will show how to call two common Elements methods:

- Find User
- Create User

We will implement it using HTML and PHP with cURL. To run this tutorial, you will need to set up a web server that supports PHP. The example code can be unzipped into your web directory and run from there. For your own applications, you can use any language that supports making HTTP requests using the POST, GET, DELETE and PUT methods.

The application we build in this tutorial tries to find a specified user in Elements by searching for an external key (your ID), and if one does not exist, creates a new user. The results are displayed.

The tutorial app looks like this when complete:



So, without further ado, let's get started.

STEP 1: CREATE A WEB PAGE TO DISPLAY RESULTS

This is the HTML page the user will interact with. It consists of a form where the user can type an external Key, a button to initiate the call to Elements, and an iFrame to receive the results.

```
<html>
<body>
<form id="getuser-form" name="getuser-form" action="display_user.php" method="POST" target="userDataFrame" >
            <label>External Key:</label>
            <input type="text" name="externalKey">
            <input type="submit" value="Find or Create User">
            </select>
</form>
<iFrame name='userDataFrame' id='userDataFrame' width='100%' height='100%' src='about:blank' > </iFrame>
</body>
</html>
```

STEP 2: CREATE SERVER-SIDE SCRIPT TO RECEIVE HTML POST

First, some background. Your server side code serves enable you to:

- Authenticate the user
- Insert private parameters to prevent spoofing attacks

It is also a good place to:

- Cache the results of API calls.
- Insert a retry mechanism for when API calls fail due to timeouts.
- Parse the XML you receive from Elements and repackage it for your application.
- Add functionality like calling the Live Gamer Health Check periodically.
- Record the results of successful transactions (more about this in ***Tutorial 3: Handling Live Gamer Callbacks***).

Of those, this tutorial only inserts the private parameters to ensure security (in ***Step 3***).

Here is the PHP that receives the POST from the preceding HTML:

```
<?php
include_once("lg_send_request.php");

// Try to find the user
echo ("Finding user ".$_POST['extrnalKey']."...<BR>\n" );
$userXML = sendRequest( 'user', "externalKey=".$_POST['externalKey'], 'GET' );

// If they dont exist, create a new user.
if( !$userXML ){
        echo ("User Not Found...<BR>\n");
        echo ("Creating user ".$_POST['externalKey'].".<BR>\n");
        $userXML = sendRequest(
                'user',
                '&name=User_'.$_POST['externalKey'].'&namespaceId=0'.'&externalKey='.$_POST['externalKey'],
                'POST' );
}

// Print out the user, or an error
if( $userXML ){
        $userObj = simplexml_load_string($userXML);
        var_dump( $userObj );
}
else{
        echo( "ERROR: Unable to find or create user ID " . $_POST['externalKey'] );
}
?>
```

Let's look deeper at this code. First, it includes 'lg_send_request.php', which handles the communication with Elements (see ***Step 3***).

Next, it calls 'sendRequest' (in lg_send_request.php) and tries to find the user by the external key that was passed in from the HTML form.

```
$userXML = sendRequest( 'user',  "&externalKey=".$_POST['userId'], 'GET' );
```

Key point: An external key in Elements is an ID that you provide. The typical use case is to set the external key to your user ID, so that you can reference users with a single ID (yours) throughout your application.

If a user with the specified external key exists in Elements, it will be returned as a block of XML in $userXML. If it does not exist, the return value will be FALSE.

Next, the code checks whether $userXML is false, and if it is sends a request to create the user:

```
if( !$userXML ){
        $userXML = sendRequest(
                'user',
                'name=User_'.$_POST['externalKey'].
                '&namespaceId=0'.
                '&externalKey='.$_POST['externalKey'],
                'POST' );
}
```

This second request also calls the 'user' service in Elements, but this time it is a POST and includes additional query parameters necessary to create a user:

- The user **name** is an arbitrary string, but it must be unique and cannot contain spaces.
- The **namespace ID** should be 0. *This parameter is required, but is being deprecated*.
- The **external Key** is an arbitrary string, and should be set to your user ID. This allows you to find the user using your ID, by calling Find User with the external key as we are doing here.

There are many additional parameters that can be passed in when creating users (country, gender, etc). The more you fill out the user object, the more data will be available for analytics. If Live Gamer is acting as Seller of Record, you may be required to include some of these (like country).

The final block uses PHP functions to display the user, or an error message if both the find and create user calls failed:

```
if( $userXML ){
        $userObj = simplexml_load_string($userXML);
        var_dump( $userObj );
}
else{
        echo( "ERROR: Unable to find or create user ID " . $_POST['externalKey'] );
}
```

STEP 3: CALL ELEMENTS USING CURL

The last step of this process is to call Elements:

```
<?php
$partner_id         = '$apiactor';
$access_key         = 'CnJwZfSDor';
$appFamilyId        = '1360614558681';
$v2_api_url         = 'https://eval.tfelements.com/tfel2rs/v2/';

function sendRequest($path = '', $queryParams = NULL, $method = 'GET' )
{
        $m_response = null;
        $url = $GLOBALS['v2_api_url'] . $path;

        $queryParams =
                'auth.partnerId='.$GLOBALS['partner_id'].
                '&auth.accessKey=' .$GLOBALS['access_key'].
                '&auth.appFamilyId=' .$GLOBALS['appFamilyId'].
                '&auth.namespaceId=0'.
                '&'.$queryParams;

        if( ($curl_handle = curl_init()) )
        {
                switch($method)
                {
                        case "GET":
                                $url .= '?' . $queryParams;
                                break;

                        case "POST":
                                curl_setopt($curl_handle, CURLOPT_POST, true);
                                curl_setopt($curl_handle, CURLOPT_POSTFIELDS, $queryParams);
                                break;

                        case "DELETE":
                                $url .= '?' . $queryParams;
                                curl_setopt($ch, CURLOPT_CUSTOMREQUEST, 'DELETE');
                                return FALSE;
                }
                curl_setopt($curl_handle, CURLOPT_URL, $url);
                curl_setopt($curl_handle, CURLOPT_SSL_VERIFYPEER, FALSE);
                curl_setopt($curl_handle, CURLOPT_RETURNTRANSFER, TRUE);
                curl_setopt($curl_handle, CURLOPT_TIMEOUT, 5);

                $m_response = curl_exec($curl_handle);
                $m_response_code = curl_getinfo($curl_handle, CURLINFO_HTTP_CODE);
                curl_close($curl_handle);

                if($m_response_code < 200 || $m_response_code >= 300 )
                {
                        //echo "<BR>ERROR:<BR>\n";
                        //echo $url."<BR>\n";
                        //echo $m_response."<BR>\n";
                        return FALSE;
                }
        }
        return $m_response;
}
?>
```

This function has several blocks. Let's discuss each one.

First, we have the credentials to use when calling the Elements API. Your credentials are unique to you, so you should get them from your Live Gamer Account Manager:

```
<?php
$partner_id         = '$apiactor';
$access_key         = 'CnJwZfSDor';
$appFamilyId        = '1360614558681';
$v2_api_url         = 'https://eval.tfelements.com/tfel2rs/v2/';
```

Next, we define one method – submitRequest, that takes three parameters:

```
function sendRequest($path = '', $queryParams = NULL, $method = 'GET' )
{
```

- **$path** is the path to the user service, including any path parameters for that API. You can check the API documentation pages (https://eval.tfelements.com/tfel2rs/doc/v2) to see what parameters each service accepts, and whether they are path parameters or query parameters. A couple of examples of how this might look are:
- user – This calls 'find user by external id' (external ID is a query parameter)
- user/12345 – This calls 'find user by user ID' (user ID is a path parameter)
- **$queryParams** are additional parameters that go at the end of the URL. These parameters should be separated by a '&'. For example, the create user call is passing in three parameters, so the resulting value looks like this:
- name=User_12345&namespaceId=0&externalKey=12345
- **$method** is the REST method to use – Live Gamer uses GET, PUT, POST and DELETE.

The function starts by initializing the return variable to null, and adding your credentials to the base URL and query parameters:

```
    $m_response = null;

    // build the final path
    $url = $GLOBALS['v2_api_url'] . $path;

    // prepend the security parameters to the query parameters
    $queryParams =
            'auth.partnerId='.$GLOBALS['partner_id'].
            '&auth.accessKey=' .$GLOBALS['access_key'].
            '&auth.appFamilyId=' .$GLOBALS['appFamilyId'].
            '&auth.namespaceId=0'.
            '&'.$queryParams;
```

The resulting $url should look like this:

```
https://eval.tfelements.com/tfel2rs/v2/user
```

The resulting query parameters should look like this (without line breaks):

```
auth.partnerId=$apiactor
&auth.accessKey= CnJwZfSDor
&auth.appFamilyId=1360614558681
&auth.namespaceId=0
&name=User_12345
&namespaceId=0
&externalKey=12345
```

Once you have the URL path and parameters, use cURL to send the data to Elements. If you have used cURL before, this will all be familiar to you. If not, the steps are:

1. Initialize curl
2. Handle the query parameters, depending on whether this is a GET, POST, PUT or DELETE
3. Set other cURL options
4. Make the REST call and store the results
5. Check the result code and handle errors

Here is the code for each step:

1. Initialize cURL:

```
if( ($curl_handle = curl_init()) )
{
```

2. Package the query parameters:

```
switch($method)
{
        case "GET":
                $url .= '?' . $queryParams;
                break;

        case "POST":
                curl_setopt($curl_handle, CURLOPT_POST, true);
                curl_setopt($curl_handle, CURLOPT_POSTFIELDS, $queryParams);
                break;

        case "DELETE":
                $url .= '?' . $queryParams;
                curl_setopt($ch, CURLOPT_CUSTOMREQUEST, 'DELETE');
                return FALSE;
}
```

The query parameters are added to the end of the URL for GET and DELETE operations. They are sent url encoded in the body of the HTTP message for POST operations. Also, not all PHP implementations support CURLOPT_CUSTOMREQUEST (for DELETE), so make sure you have one that does.

3. Set cURL options:

```
curl_setopt($curl_handle, CURLOPT_URL, $url);
curl_setopt($curl_handle, CURLOPT_SSL_VERIFYPEER, FALSE);        // FOR DEBUGGING ONLY
curl_setopt($curl_handle, CURLOPT_RETURNTRANSFER, TRUE);
curl_setopt($curl_handle, CURLOPT_TIMEOUT, 5);
```

CURLOPT_SSL_VERIFYPEER should be set to TRUE when calling Elements. Elements servers should always have valid security certificates. For testing, however, you may be connecting to a server that does not have a certificate, and this parameter is necessary in that case.

4. Make the REST call, and collect the results:

```
$m_response = curl_exec($curl_handle);
$m_response_code = curl_getinfo($curl_handle, CURLINFO_HTTP_CODE);
curl_close($curl_handle);
```

Upon completion, $m_response will hold the XML that Live Gamer returns, including error text in the case of an error. $m_response_code will hold a standard HTML status code. You must check m_response_code to know whether m_response holds valid data, or an error response.

5. Check the results and handle errors:

```
                    if($m_response_code < 200 || $m_response_code >= 300 )
                    {
                            //echo "<BR>ERROR:<BR>\n";
                            //echo $url."<BR>\n";
                            //echo $m_response."<BR>\n";
                            return FALSE;
                    }
            }
            return $m_response;
}
?>
```

Elements will return standard HTML status codes. Anything in the 200-299 range indicates a successful operation. Anything outside of that range indicates an error.

NEXT STEPS

Now that you know how to create and retrieve users, ***Tutorial 2: Displaying the Paywall*** will show you how to launch the Elements payment wall to accept payments from your customers.