

Implementing a Service Objects Web Service into a PHP Project

Introduction

This guide provides a step by step on how to implement a Service Objects DOTS web service into a simple web form using PHP. This project will be using the PHP implementation of Address Geocoder but will be similar for any SOAP implementation of a Service Objects web service. This guide will be very basic and should help serve as a foundation to integrating one of Service Object’s web services into your PHP application.

What to Expect:

We will demonstrate the implementation in two parts. The first part will demonstrate the process of calling and displaying the results from a Service Objects web service and the second part will include a more applicable example of how to integrate some useful business logic around the validated data from the Service Objects web service.

Audience

Any developer (beginner to expert), that is interested in integrating a DOTS web service into PHP web application.

Requirements

- PHP IDE– *NetBeans was used in this example, but the steps should be relatively similar for another PHP IDE*
- Basic Working Knowledge of PHP and a script based programming
- A working License Key(trial or production) from Service Objects. [Click here](#) to obtain a free trial key for the service you are most interested in.

Contents

Setting Up Your Project.....	2
Designing Your Web Form and Adding the Code.....	6
Failover Configuration.....	7
Processing the Results From A Service Objects Web Service	8
Implementing Business Logic into Your Web Application.	11
Conclusion.....	15

Implementing a Service Objects Web Service into a PHP Project

Setting Up Your Project

Launch NetBeans and select **File**, and then **New Project** to create the web application. In the New Project window select **PHP Application** and then click next as seen in the figure below.

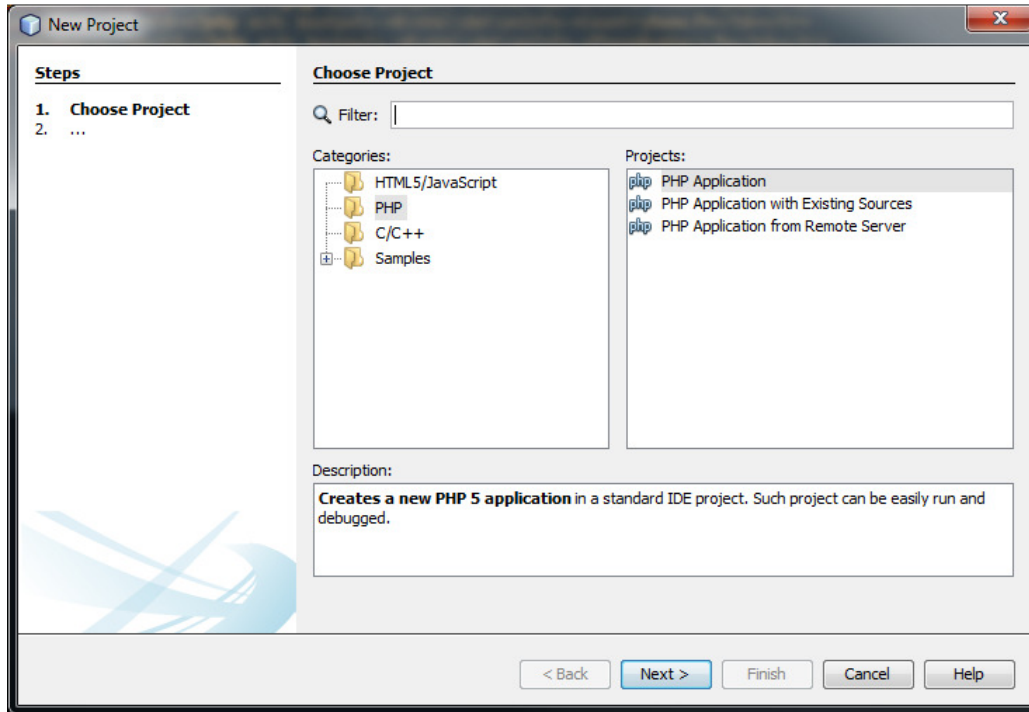


Figure 1

On the next screen, give the project an appropriate name and storage location.

Implementing a Service Objects Web Service into a PHP Project

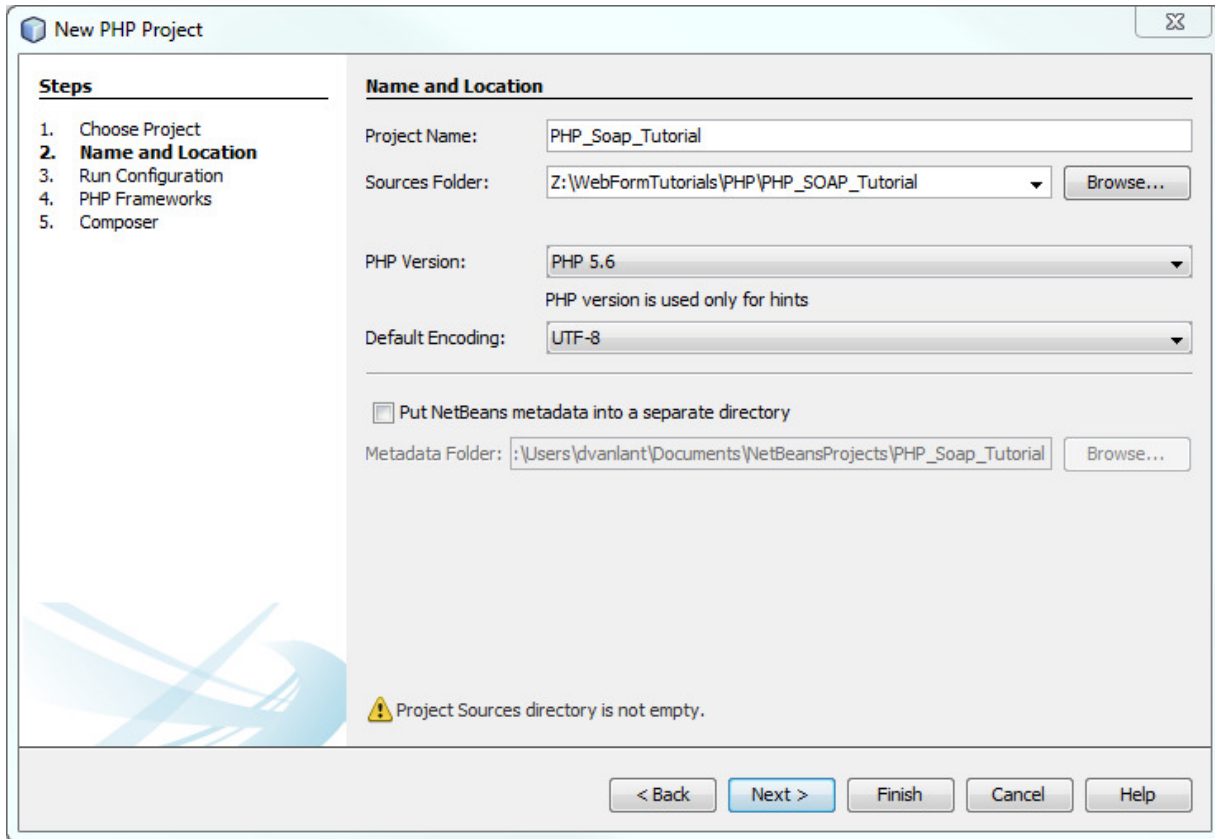


Figure 2

Now that the project has been created, we will need to add a php file to function as our input form and to handle the call to the Service Objects web service. To do this, right click the project folder, select **New** and then select **PHP File** and you will be prompted to name the file to be added. For this example, we will name the PHP file called **soapCall.php** which will serve as our input form, make the call to the web service and then process the results. This can be seen in figure 3.

Implementing a Service Objects Web Service into a PHP Project

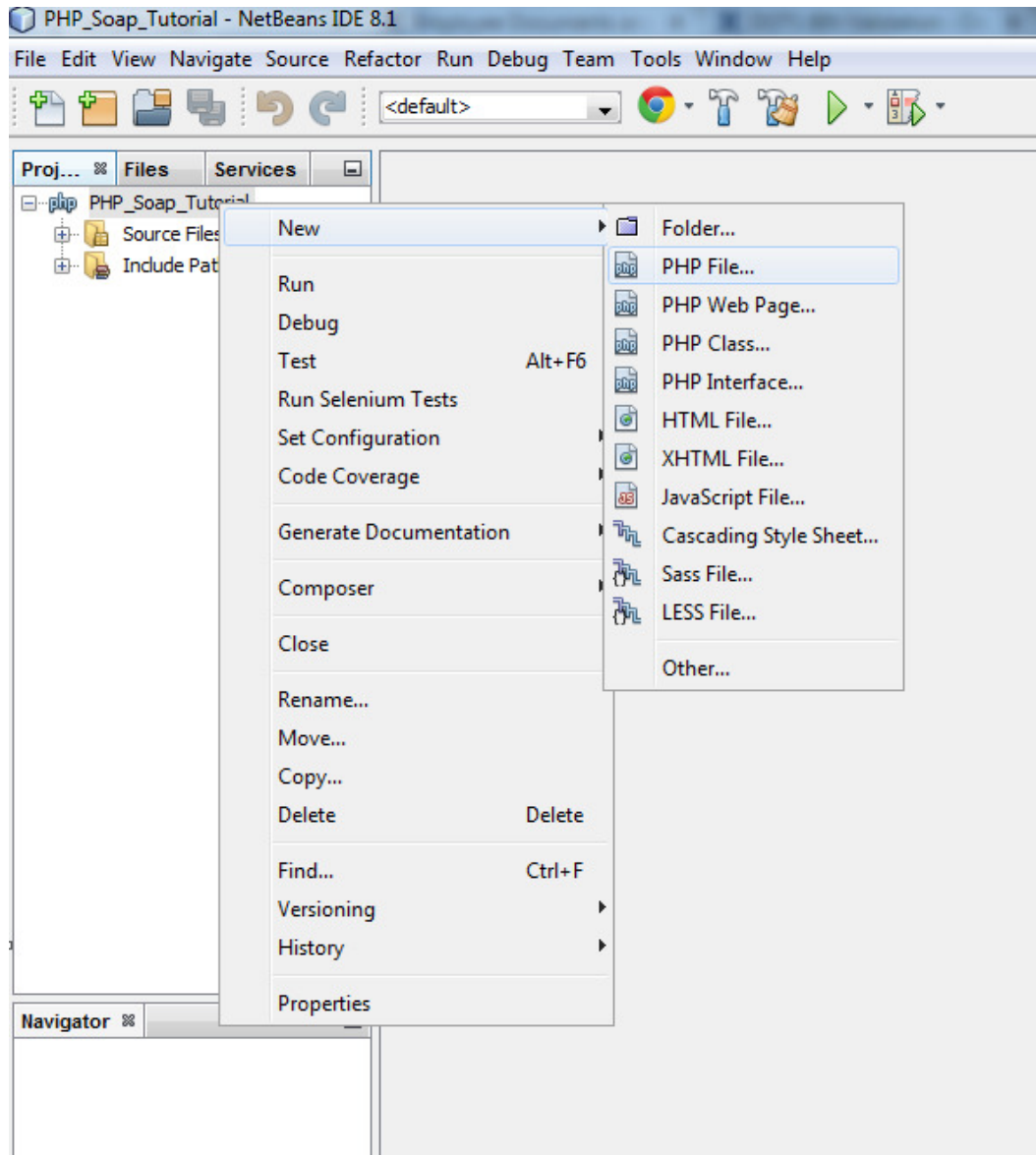


Figure 3

Now that our PHP file has been properly added, we'll need to set up the Run Configuration so we can properly test it. To do this select **Run** at the top of the NetBeans program and then select **Set Project Configuration** and then **Customize...** In the **Run As** selection, highlight **PHP Built-in Web Server(running on built-in web server)**. Ensure that the name of the php file that was just added to the project is placed in the **Router Script** section. This can be seen in the screen shot below.

Implementing a Service Objects Web Service into a PHP Project

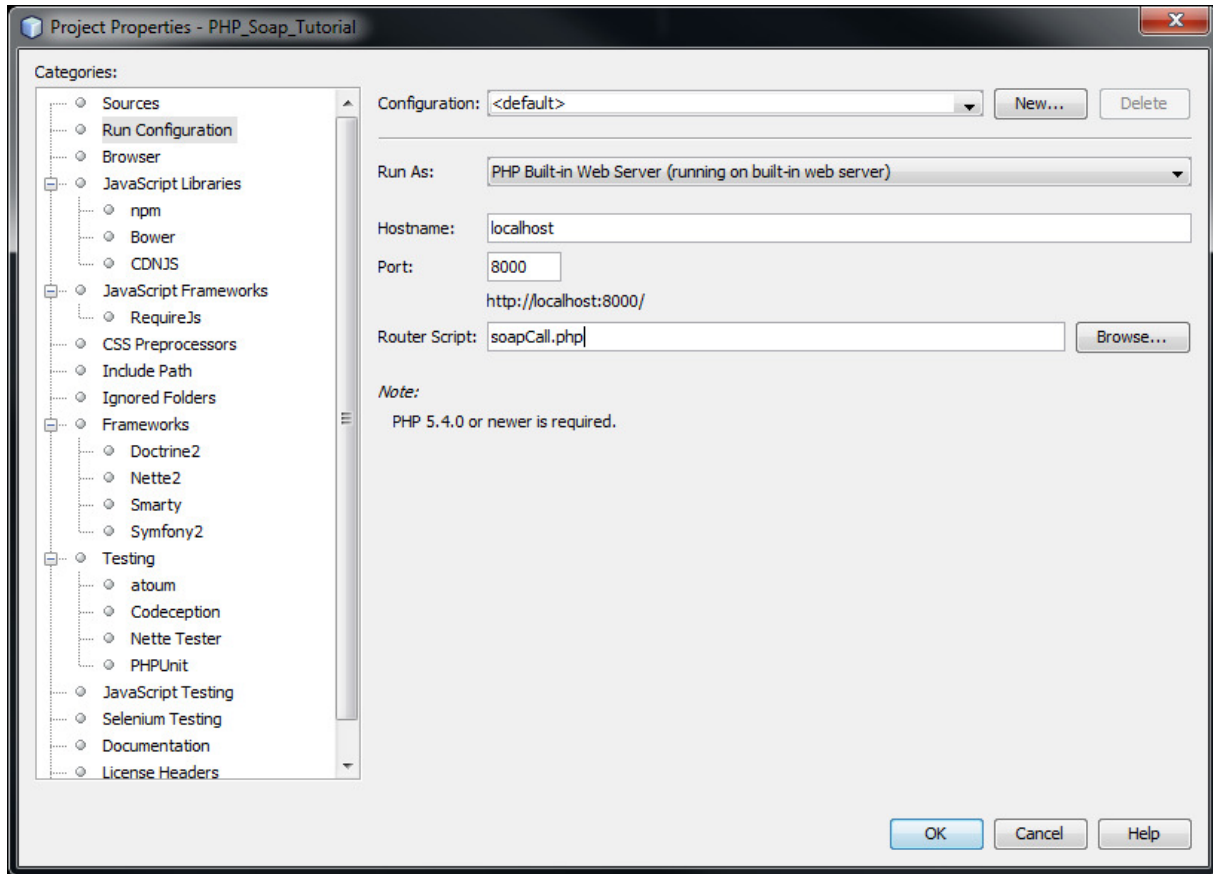


Figure 4

Implementing a Service Objects Web Service into a PHP Project

Designing Your Web Form and Adding the Code

Now that we have successfully created the web application and added the necessary php page, we need to fill in script with the code that will receive the inputs, make the call to the Service Objects web service and process the results for the user. This code can be found in the **SoapCall_Contents.txt** file that has been included with this tutorial.

```
// variable cleanup before generating url
$Address = trim($Address);
$City = trim($City);
$State = trim($State);
$PostalCode = trim($PostalCode);
$LicenseKey = trim($LicenseKey);

$params['Address'] = $Address;
$params['City'] = $City;
$params['State'] = $State;
$params['PostalCode'] = $PostalCode;
$params['LicenseKey'] = $LicenseKey;

$url = "http://trial.serviceobjects.com/gcr/GeoCoder.asmx?WSDL";

//use backup url once given purchased license key
$backupURL = "http://trial.serviceobjects.com/gcr/GeoCoder.asmx?WSDL";

try{

    //Instantiates the SoapClient
    $client = new SoapClient($url);
    //Calls the GetBestMatch_V3 operation
    $outputs = $client->GetBestMatch_V3($params)->GetBestMatch_V3Result;
```

Figure 5: Basic Soap Call Configuration

The first thing the code does is instantiate the variables that input form will be passing to the rest call. It also sets the input-form parameters to our PHP variables.

The code will also display a simple input form with a table that has inputs for **Address**, **City**, **State**, **PostalCode**, and **LicenseKey**. There is also submit button that will send the user entered information to the script behind the form and make the actual SOAP call to the web service.

After the user has entered valid address information into the input form, the script will then trim any whitespace on the input variables and then pass then create the backup and primary URLs to use for the SOAP client. The code will then initialize the SOAP client and then pass the parameters to the **GetBestMatch_V3** operation.

Implementing a Service Objects Web Service into a PHP Project

Failover Configuration

Within the code that makes the call to the web service, we've included failover configuration. In the event that you design your own SOAP call, we highly recommend implementing a failover call into your code. This is done by checking if the response from the webservice is null or if the error object has a **Number** of 4; which would indicate a fatal error. In the event that either of these instances occurs, the code will call the backup endpoint. We strongly recommend implementing this fail over configuration into your project. With failover enabled you can ensure that your application will function as normal in the event that the production endpoint is unavailable.

```
if (empty($outputs) || (isset($outputs->Error) && $outputs->Error->Number = "4"))
{
    $backupClient = new SoapClient($backupURL);
    $outputs = $client->GetBestMatch_V3($params)->GetBestMatch_V3Result;
}
```

Figure 6: Failover Configuration

As noted in the screen shot above, if the code detects a null response or **Number** of 4 it will call the backupURL and attempt to get a valid response. After this, the code will then proceed as normal and display a response if one was received from the backup URL.

In this code both the primary and backup URLs are pointed to the same server, but once a production key is purchased there should be 2 service references added to the project: one for the primary Service Objects endpoint and one for the backup:

<http://ws.serviceobjects.com/qcr/GeoCoder.asmx?WSDL>

<http://wsbackup.serviceobjects.com/qcr/GeoCoder.asmx?WSDL>

Implementing a Service Objects Web Service into a PHP Project

Processing the Results From A Service Objects Web Service

After the SOAP call has completed, the code will then display the whatever response was received from the service. First the code will check to see if an error response was returned from the service. If one was returned, the application will display the **Desc** and **Number** results from the Address Geocode service. This can be seen below.

DOTS Geocoder US

Address:

City:

State:

PostalCode:

LicenseKey:

Web Service Results

Outputs	Values
Desc	Please provide a valid license key for this web service.
Number	2

Figure 7

If valid inputs are given to the input form, then service will return a valid response and the resulting values will be displayed for the user. Below is an example of a valid response from the Geocoder service.

Implementing a Service Objects Web Service into a PHP Project

DOTS Geocoder US

Address:
City:
State:
PostalCode:
LicenseKey:

Web Service Results

Outputs	Values
Latitude	34.418014
Longitude	-119.696477
Zip	93101-7602
Tract	0009.00
Block	2039
Level	S
LevelDescription	The address matched exactly at the property location.
StateFIPS	06
CountyFIPS	083

Figure 8

One thing to note about the Geocoder service is that if the service cannot find a street level match for the inputs it receives, then it will attempt to find a match at the most accurate level that it can. Below is a listing of types of matches the Geocoder service can make from most accurate to least accurate.

- S – Property-level match (property-level match)
- P – Zip plus four match
- R – Zip plus three match
- T – Zip plus two match
- N – Zip plus one match
- Z – Zip level match (zip plus zero)
- C – City/state match

For example, if a bad street address is given to the service, it will then attempt to find the closest match using the zipcode(if one was provided) or the city and state. Below is screenshot that illustrates this point. This example has a bad street address and a valid zip code and will return a zip level match.

Implementing a Service Objects Web Service into a PHP Project

DOTS Geocoder US

Address:
 City:
 State:
 PostalCode:
 LicenseKey:

Web Service Results

Outputs	Values
Latitude	34.419120
Longitude	-119.703421
Zip	93101
Tract	
Block	
Level	Z
LevelDescription	The address matched at the general zip code level.
StateFIPS	06
CountyFIPS	083

Figure 9

This code can be used for testing purposes to experiment with the different types of results that service will output depending on what the user enters.

Implementing a Service Objects Web Service into a PHP Project

Implementing Business Logic into Your Web Application.

Simply displaying the results to the end user isn't a necessarily a good use of the Service Objects web service, so in this section we will go over an example of what it may look like to integrate some business logic around the results that web service returns. For example, let's assume that a user is entering their address information to determine the cost of a taxi or ride share service. If the service is able to make a street level match, then we can give the user a definite amount of money that they can expect to pay, if we find a less granular level of geocode results, the application will return less specific pricing information for a given address. For this tutorial we'll be using a very basic pricing example with a constant travel distance which may not be practical, but these can be substituted for more applicable pricing scenarios or situations.

In this section we will change the code that is implemented after our call to the service objects web service. The updated code is available in the **Updated_SoapCall_Contents.txt** file that is included with this tutorial. Copy and paste this code into the **soapCall.php** and we'll highlight some key aspects about this new logic we have implemented.

The updated code functions very similarly to the previous code in how it takes in the values from the user and then passes them to the Service Objects web service. The only notable difference is that the **LicenseKey** value has been hard coded into the program as opposed to being entered by the user.

The primary changes that have been made to our application have to do with how the response from the web service is processed. In our check for an error from the service, we have added 3 separate if statements that will check for the 3 different errors that can be returned from the web service.

```
if ( isset($outputs->Error))
{
    ?> <b>Web Service Results</b> <?php

        if($outputs->Error->Number == "1")
        {
            ?>
            <br><table border=1>
            <tr><td>An Error Occurred</td><td>No Input was received by the service. Please Contact Customer Support</td></tr>
            </table>
            <?php
        }
        //if an error Number 2 is returned, this code will display the error to the user.
        else if ($outputs->Error->Number == "2")
        {
            ?>
            <br><table border=1>
            <tr><td>An Error Occurred</td><td><?php echo isset($outputs->Error->Desc) ? $outputs->Error->Desc : "No Error Was Obtained" ?></td></tr>
            </table>
            <?php
        }

        //Error Number of 4 indicates that something seriously wrong has occurred.
        //This along with the specific inputs, time and Desc value from the service should be logged and sent to service objects.
        else if ($outputs->Error->Number == "4")
        {
            ?>
            <br><table border=1>
            <tr><td>A Fatal Error Occurred</td><td>Please Contact Customer support</td></tr>
            </table>
            <?php
        }
    }
}
```

Figure 10: Updated Error Processing Code

Implementing a Service Objects Web Service into a PHP Project

In the event that a **Number** value of “1” is returned by the service, this would indicate that the service did not receive any inputs. This would also mean that there was something wrong in how the call is set up to interact with the web service. If this error should occur then the developer or technical support team for the application should be notified of the error. Below is a screen shot of this response.

The screenshot shows a web form with the title "Enter Address to Calculate Fare Cost". It contains four input fields labeled "Address:", "City:", "State:", and "PostalCode:". Below these fields is a "Submit" button. Underneath the form, there is a section titled "Web Service Results" which contains a message box with the text: "An Error Occurred No Input was received by the service. Please Contact Customer Support".

Figure 11

If the **Number** value in the error object is returned as a “2” then this indicates that there was some error with the address and the service could not obtain a response. In this event, a message will be displayed to the user indicating that an error occurred and the **Desc** value that was returned in the error response. Below is an example of this response when the user does not enter a City and State or a Zip Code.

The screenshot shows the same web form as Figure 11, but with the "Address:" field filled with "123 Main Street". The "City:", "State:", and "PostalCode:" fields are empty. The "Submit" button is visible. The "Web Service Results" section shows a message box with the text: "An Error Occurred Please input either zip code or both city and state.".

Figure 12

Implementing a Service Objects Web Service into a PHP Project

If an error with a **Number** value of 4 is returned this would indicate that a Fatal error was returned from the web service. There should never be a fatal error returned in a live environment but in the event that one is returned, it would be best practice to log the error along with the specific inputs that were used that obtained this error. This information should be sent to Service Objects for further review. Below is an example of this error message

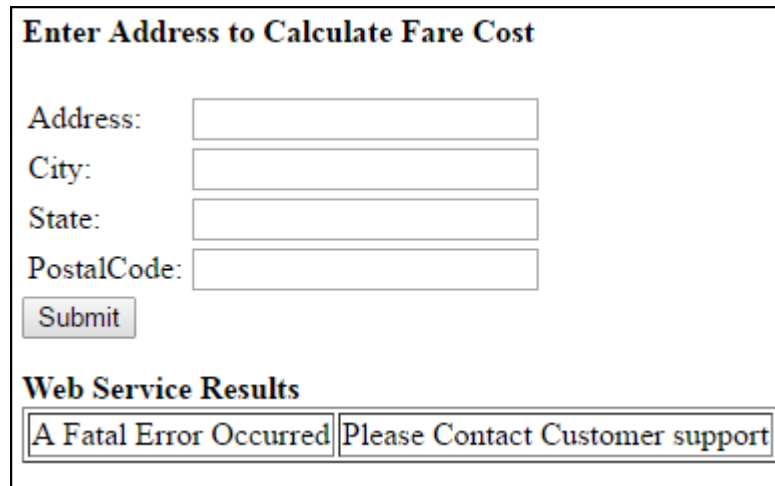


Figure 13

The Error response from the service can return different **Desc** values for the same **Number** response. If you would like to integrate different logic for each of the **Desc** values you can find them on the Developer's page for the Geocode service which is listed below.

<https://docs.serviceobjects.com/display/devguide/DOTS+Address+Geocode++--+US>

In the event that a valid response is returned from the service, the fare for the ride will be calculated depending on the **Level** value that is returned in the response object. For this example, we have made our **\$fare** a stand rate of \$1.75 per mile and our **\$milesToBeTravelled** a constant of 10 miles. We have also implemented variables named **\$MaxMilesToTravel** and **\$MinMilestoTravel**, which we will use to determine the range for the fare cost if a less granular level of match was found for the users input.

If a Street level match is found for a particular address, then the application will simply print out the calculated fare. Below is a screen shot of that example.

Implementing a Service Objects Web Service into a PHP Project

Enter Address to Calculate Fare Cost

Address:
 City:
 State:
 PostalCode:

A Specific Address Match Was Found

Estimated Total Fare:

Figure 14: Street Level Match

If a less specific match level is found, then the application will output a range that the fare could potentially cost. This is to account for the minimum and maximum distances that a user could potentially travel to . Below is an example of the Zip Level match and range of fare to be provided.

Enter Address to Calculate Fare Cost

Address:
 City:
 State:
 PostalCode:

A Zip Code Match Was Found

Estimated Total Fare:

Figure 15: Zip Level Match

Implementing a Service Objects Web Service into a PHP Project

Conclusion

That concludes our tutorial on how to create a web form that uses a SOAP call in PHP. If you have any questions please email support@serviceobjects.com and we would gladly answer any questions you may have.