How to Add Custom Marker Icons to Google Map Dynamically from the Database with PHP and MySQL

[❮ PreviousNext ❯](https://www.studentstutorial.com/php/google-map-dynamically.php)

Create Database Table

To store the dynamic locations information and marker images a table is required in the database. The following SQL creates a **locations** table with some basic fields in the MySQL database.

CREATE TABLE `locations` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`latitude` varchar(20) COLLATE utf8\_unicode\_ci NOT NULL,

`longitude` varchar(20) COLLATE utf8\_unicode\_ci NOT NULL,

`name` varchar(100) COLLATE utf8\_unicode\_ci NOT NULL,

`info` varchar(255) COLLATE utf8\_unicode\_ci NOT NULL,

`icon` varchar(255) COLLATE utf8\_unicode\_ci NOT NULL COMMENT 'Marker icon',

PRIMARY KEY (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8\_unicode\_ci;

<?php

/\* Database configuration \*/

$dbHost = "localhost";

$dbUsername = "root";

$dbPassword = "root";

$dbName = "codexworld";

/\* Create database connection \*/

$db = new mysqli($dbHost, $dbUsername, $dbPassword, $dbName);

/\* Check connection \*/

if ($db->connect\_error) {

die("Connection failed: " . $db->connect\_error);

}

Database Configuration (dbConfig.php)

The **dbConfig.php** file is used to connect and select the database. Specify the database host **($dbHost)**, username **($dbUsername)**, password **($dbPassword)**, and name **($dbName)** as per your MySQL database credentials.

<html>

<head>

<title>Login with Instagram</title>

</head>

<body>

<a href="<?php echo $instURL; ?>">Login with Instagram</a>

</body>

</html>

Fetch Location and Marker Image Info from Database

The latitude and longitude are fetched from the database using PHP and MySQL.

* The first result set is used to generate multiple markers and custom icons information (location, latitude, longitude, and icon image source).
* The second result set is used to generate the content of the info windows.

<?php

/\* Include the database configuration file \*/

require\_once 'dbConfig.php';

/\* Fetch the marker info from the database \*/

$result = $db->query("SELECT \* FROM locations");

/\* Fetch the info-window data from the database \*/

$result2 = $db->query("SELECT \* FROM locations");

?>

Load initMap() function to initialize the Google Map.

<script src="https://maps.googleapis.com/maps/api/js?key=Your\_API\_KEY"></script>

Add Multiple Markers with Custom Images

Load the Google Map JavaScript API and specify an API key in the key parameter.

<script>

function initMap() {

var map;

var bounds = new google.maps.LatLngBounds();

var mapOptions = {

mapTypeId: 'roadmap'

};

/\* Display a map on the web page \*/

map = new google.maps.Map(document.getElementById("mapCanvas"), mapOptions);

map.setTilt(100);

/\* Multiple markers location, latitude, and longitude \*/

var markers = [

<?php if($result->num\_rows > 0){

while($row = $result->fetch\_assoc()){

echo '["'.$row['name'].'", '.$row['latitude'].', '.$row['longitude'].', "'.$row['icon'].'"],';

}

}

?>

];

/\* Info window content \*/

var infoWindowContent = [

<?php if($result2->num\_rows > 0){

while($row = $result2->fetch\_assoc()){ ?>

['<div class="info\_content">' +

'<h3><?php echo $row['name']; ?></h3>' +

'<p><?php echo $row['info']; ?></p>' + '</div>'],

<?php }

}

?>

];

/\* Add multiple markers to map \*/

var infoWindow = new google.maps.InfoWindow(), marker, i;

/\* Place each marker on the map \*/

for( i = 0; i < markers.length; i++ ) {

var position = new google.maps.LatLng(markers[i][1], markers[i][2]);

bounds.extend(position);

marker = new google.maps.Marker({

position: position,

map: map,

icon: markers[i][3],

title: markers[i][0]

});

/\* Add info window to marker \*/

google.maps.event.addListener(marker, 'click', (function(marker, i) {

return function() {

infoWindow.setContent(infoWindowContent[i][0]);

infoWindow.open(map, marker);

}

})(marker, i));

/\* Center the map to fit all markers on the screen \*/

map.fitBounds(bounds);

}

/\* Set zoom level \*/

var boundsListener = google.maps.event.addListener((map), 'bounds\_changed', function(event) {

this.setZoom(14);

google.maps.event.removeListener(boundsListener);

});

}

/\* Load initialize function \*/

google.maps.event.addDomListener(window, 'load', initMap);

</script>

Embed Google Map with Custom Marker Icons

Define an HTML element to embed the Google Map with multiple markers and Info Windows on the web page. Specify the element ID (mapCanvas) in the Google Map object.

<div id="mapCanvas"></div>

CSS Code

Use CSS to set the height and weight of the Google map container.

Embed Google Map with Custom Marker Icons

Define an HTML element to embed the Google Map with multiple markers and Info Windows on the web page. Specify the element ID (mapCanvas) in the Google Map object.

#mapCanvas {

width: 100%;

height: 650px;

}

/////////////////

$markers = array();

$sql\_locations1="SELECT DISTINCT uid,latitude,longitude from `location` where uid='$\_REQUEST[locator\_id]' ORDER by `date\_time` DESC limit 1";

$result1 = mysqli\_query($conn,$sql\_locations1);

while($row=mysqli\_fetch\_assoc($result1))

{

$markers[] = array(

'abc',

$row['latitude'],

$row['longitude'],

);

}?>

<div id="map\_canvas" onload="initMap()"></div>

**as you can see 'abc' in array its static value that shows on each markers on map. what i want is user's name from db on markers of their own on the place of that 'abc' static value** **i dont know how to get this db value on that place. below is my function for map**

<script>

function initialize() {

var map;

var bounds = new google.maps.LatLngBounds();

var mapOptions = {

mapTypeId: "roadmap",

center: new google.maps.LatLng(20.5937, 78.9629), // somewhere in the uk BEWARE center is required

zoom: 1,

};

// Display a map on the page

map = new google.maps.Map(document.getElementById("map\_canvas"), mapOptions);

map.setTilt(45);

// Multiple Markers

var markers = <?php echo json\_encode( $markers );?>;

// Display multiple markers on a map

var infoWindow = new google.maps.InfoWindow();

var marker, i;

// Loop through our array of markers & place each one on the map

for (i = 0; i < markers.length; i++) {

var position = new google.maps.LatLng(markers[i][1], markers[i][2]);

bounds.extend(position);

marker = new google.maps.Marker({

position: position,

map: map,

title: markers[i][0]

});

// Allow each marker to have an info window

google.maps.event.addListener(marker, 'click', (function (marker, i) {

return function () {

infoWindow.setContent(infoWindowContent[i][0]);

infoWindow.open(map, marker);

}

})(marker, i));

// Automatically center the map fitting all markers on the screen

map.fitBounds(bounds);

}

//Override our map zoom level once our fitBounds function runs (Make sure it only runs once)

var boundsListener = google.maps.event.addListener((map), 'bounds\_changed', function (event) {

this.setZoom(5);

google.maps.event.removeListener(boundsListener);

});

}

google.maps.event.addDomListener(window, 'load', initialize);

# Using MySQL and PHP with Google Maps

## Overview

This tutorial shows you how to display information from a MySQL database on a Google Map using the Maps JavaScript API. It suits people with intermediate knowledge of MySQL, PHP and XML.

The map in this tutorial displays two kinds of markers to differentiate between the location of restaurants and bars respectively. A database in MySQL stores information about the individual marker locations like the type of place (whether it's a restaurant or a bar), its name, address and geographic coordinates. The map retrieves this information from the database, through an XML file that acts as an intermediary between the database and the map. You can use PHP statements to export marker information from the database to an XML file.

Click a marker on the map below to display an info window with the location name and address. The markers have the labels 'R' or 'B', and they show you the locations of restaurants and bars respectively.

The sample below shows the entire code you need to create this map.

**Tip:** Check out the [Clothing Store Locator solution](https://developers.google.com/maps/solutions/store-locator/clothing-store-locator) to see another example of MySQL and PHP.

var customLabel = {

restaurant: {

label: 'R'

},

bar: {

label: 'B'

}

};

function initMap() {

        var map = new google.maps.Map(document.getElementById('map'), {

          center: new google.maps.LatLng(-33.863276, 151.207977),

          zoom: 12

        });

        var infoWindow = new google.maps.InfoWindow;

// Change this depending on the name of your PHP or XML file

downloadUrl('https://storage.googleapis.com/mapsdevsite/json/mapmarkers2.xml', function(data) {

var xml = data.responseXML;

var markers = xml.documentElement.getElementsByTagName('marker');

Array.prototype.forEach.call(markers, function(markerElem) {

var id = markerElem.getAttribute('id');

var name = markerElem.getAttribute('name');

var address = markerElem.getAttribute('address');

var type = markerElem.getAttribute('type');

var point = new google.maps.LatLng(

parseFloat(markerElem.getAttribute('lat')),

parseFloat(markerElem.getAttribute('lng')));

var infowincontent = document.createElement('div');

var strong = document.createElement('strong');

strong.textContent = name

infowincontent.appendChild(strong);

infowincontent.appendChild(document.createElement('br'));

var text = document.createElement('text');

text.textContent = address

infowincontent.appendChild(text);

var icon = customLabel[type] || {};

var marker = new google.maps.Marker({

map: map,

position: point,

label: icon.label

});

marker.addListener('click', function() {

infoWindow.setContent(infowincontent);

infoWindow.open(map, marker);

});

});

});

}

function downloadUrl(url, callback) {

var request = window.ActiveXObject ?

new ActiveXObject('Microsoft.XMLHTTP') :

new XMLHttpRequest;

request.onreadystatechange = function() {

if (request.readyState == 4) {

request.onreadystatechange = doNothing;

callback(request, request.status);

}

};

request.open('GET', url, true);

request.send(null);

}

function doNothing() {}

<div id="map"></div>

/\* Always set the map height explicitly to define the size of the div

\* element that contains the map. \*/

#map {

height: 100%;

}

/\* Optional: Makes the sample page fill the window. \*/

html, body {

height: 100%;

margin: 0;

padding: 0;

}

<!-- Replace the value of the key parameter with your own API key. -->

<script defer

src="https://maps.googleapis.com/maps/api/js?key=AIzaSyCkUOdZ5y7hMm0yrcCQoCvLwzdM6M8s5qk&callback=initMap">

</script>

## Try it yourself

You can experiment with this code in JSFiddle by clicking the <> icon in the top-right corner of the code window.

<!DOCTYPE html >  
  <head>  
    <meta name="viewport" content="initial-scale=1.0, user-scalable=no" />  
    <meta http-equiv="content-type" content="text/html; charset=UTF-8"/>  
    <title>Using MySQL and PHP with Google Maps</title>  
    <style>  
      /\* Always set the map height explicitly to define the size of the div  
       \* element that contains the map. \*/  
      #map {  
        height: 100%;  
      }  
      /\* Optional: Makes the sample page fill the window. \*/  
      html, body {  
        height: 100%;  
        margin: 0;  
        padding: 0;  
      }  
    </style>  
  </head>  
  
<html>  
  <body>  
    <div id="map"></div>  
  
    <script>  
      var customLabel = {  
        restaurant: {  
          label: 'R'  
        },  
        bar: {  
          label: 'B'  
        }  
      };  
  
        function initMap() {  
        var map = new google.maps.Map(document.getElementById('map'), {  
          center: new google.maps.LatLng(-33.863276, 151.207977),  
          zoom: 12  
        });  
        var infoWindow = new google.maps.InfoWindow;  
  
          // Change this depending on the name of your PHP or XML file  
          downloadUrl('https://storage.googleapis.com/mapsdevsite/json/mapmarkers2.xml', function(data) {  
            var xml = data.responseXML;  
            var markers = xml.documentElement.getElementsByTagName('marker');  
            Array.prototype.forEach.call(markers, function(markerElem) {  
              var id = markerElem.getAttribute('id');  
              var name = markerElem.getAttribute('name');  
              var address = markerElem.getAttribute('address');  
              var type = markerElem.getAttribute('type');  
              var point = new google.maps.LatLng(  
                  parseFloat(markerElem.getAttribute('lat')),  
                  parseFloat(markerElem.getAttribute('lng')));  
  
              var infowincontent = document.createElement('div');  
              var strong = document.createElement('strong');  
              strong.textContent = name  
              infowincontent.appendChild(strong);  
              infowincontent.appendChild(document.createElement('br'));  
  
              var text = document.createElement('text');  
              text.textContent = address  
              infowincontent.appendChild(text);  
              var icon = customLabel[type] || {};  
              var marker = new google.maps.Marker({  
                map: map,  
                position: point,  
                label: icon.label  
              });  
              marker.addListener('click', function() {  
                infoWindow.setContent(infowincontent);  
                infoWindow.open(map, marker);  
              });  
            });  
          });  
        }  
  
  
  
      function downloadUrl(url, callback) {  
        var request = window.ActiveXObject ?  
            new ActiveXObject('Microsoft.XMLHTTP') :  
            new XMLHttpRequest;  
  
        request.onreadystatechange = function() {  
          if (request.readyState == 4) {  
            request.onreadystatechange = doNothing;  
            callback(request, request.status);  
          }  
        };  
  
        request.open('GET', url, true);  
        request.send(null);  
      }  
  
      function doNothing() {}  
    </script>  
    <script defer  
    src="https://maps.googleapis.com/maps/api/js?key=YOUR\_API\_KEY&callback=initMap">  
    </script>  
  </body>  
</html>

## Getting started

Install, set up and configure a MySQL server with PHP on your machine.

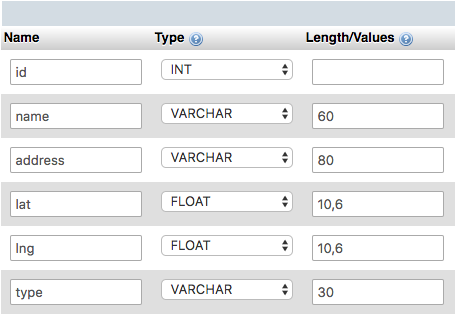
## Creating a table in MySQL

Create a table in MySQL containing attributes of the markers on the map, like the marker id, name, address, lat, lng, and type.

The id attribute serves as the primary key, and the type attribute distinguishes between restaurants and bars.

To keep the storage space for your table at a minimum, you can specify the lat and lng attributes to be floats of size (10,6). This allows the fields to store 6 digits after the decimal, plus up to 4 digits before the decimal.

You can interact with the MySQL database through the phpMyAmin interface. The screenshot below displays the table setup in phpMyAdmin.



You can also use SQL commands to create the table, as in the SQL statement below.

CREATE TABLE `markers` (  
  `id` INT NOT NULL AUTO\_INCREMENT PRIMARY KEY ,  
  `name` VARCHAR( 60 ) NOT NULL ,  
  `address` VARCHAR( 80 ) NOT NULL ,  
  `lat` FLOAT( 10, 6 ) NOT NULL ,  
  `lng` FLOAT( 10, 6 ) NOT NULL ,  
  `type` VARCHAR( 30 ) NOT NULL  
) ENGINE = MYISAM ;

## Populating the table

You can import the marker data into the SQL database using the 'Import' functionality of the phpMyAdmin interface which allows you to import data in various formats.

Below is the marker data for the map in this tutorial, in the .csv format.

1,Love.Fish,"580 Darling Street, Rozelle, NSW",-33.861034,151.171936,restaurant  
2,Young Henrys,"76 Wilford Street, Newtown, NSW",-33.898113,151.174469,bar  
3,Hunter Gatherer,"Greenwood Plaza, 36 Blue St, North Sydney NSW ",-33.840282,151.207474,bar  
4,The Potting Shed,"7A, 2 Huntley Street, Alexandria, NSW",-33.910751,151.194168,bar  
5,Nomad,"16 Foster Street, Surry Hills, NSW",-33.879917,151.210449,bar  
6,Three Blue Ducks,"43 Macpherson Street, Bronte, NSW",-33.906357,151.263763,restaurant  
7,Single Origin Roasters,"60-64 Reservoir Street, Surry Hills, NSW",-33.881123,151.209656,restaurant  
8,Red Lantern,"60 Riley Street, Darlinghurst, NSW",-33.874737,151.215530,restaurant

You can also use the SQL commands below to import the marker data into the SQL table.

INSERT INTO `markers` (`id`, `name`, `address`, `lat`, `lng`, `type`) VALUES ('1', 'Love.Fish', '580 Darling Street, Rozelle, NSW', '-33.861034', '151.171936', 'restaurant');  
INSERT INTO `markers` (`id`, `name`, `address`, `lat`, `lng`, `type`) VALUES ('2', 'Young Henrys', '76 Wilford Street, Newtown, NSW', '-33.898113', '151.174469', 'bar');  
INSERT INTO `markers` (`id`, `name`, `address`, `lat`, `lng`, `type`) VALUES ('3', 'Hunter Gatherer', 'Greenwood Plaza, 36 Blue St, North Sydney NSW', '-33.840282', '151.207474', 'bar');  
INSERT INTO `markers` (`id`, `name`, `address`, `lat`, `lng`, `type`) VALUES ('4', 'The Potting Shed', '7A, 2 Huntley Street, Alexandria, NSW', '-33.910751', '151.194168', 'bar');  
INSERT INTO `markers` (`id`, `name`, `address`, `lat`, `lng`, `type`) VALUES ('5', 'Nomad', '16 Foster Street, Surry Hills, NSW', '-33.879917', '151.210449', 'bar');  
INSERT INTO `markers` (`id`, `name`, `address`, `lat`, `lng`, `type`) VALUES ('6', 'Three Blue Ducks', '43 Macpherson Street, Bronte, NSW', '-33.906357', '151.263763', 'restaurant');  
INSERT INTO `markers` (`id`, `name`, `address`, `lat`, `lng`, `type`) VALUES ('7', 'Single Origin Roasters', '60-64 Reservoir Street, Surry Hills, NSW', '-33.881123', '151.209656', 'restaurant');  
INSERT INTO `markers` (`id`, `name`, `address`, `lat`, `lng`, `type`) VALUES ('8', 'Red Lantern', '60 Riley Street, Darlinghurst, NSW', '-33.874737', '151.215530', 'restaurant');

## Outputting data as XML using PHP

At this point, you should have a table named markers containing the map marker data. This section shows you how to export the table data from the SQL database in an XML format, using PHP statements. The map can use the XML file to retrieve the marker data through asynchronous JavaScript calls.

Using an XML file as an intermediary between your database and your Google map allows for faster initial page load, and a more flexible map application. It makes debugging easier as you can independently verify the XML output from the database, and the JavaScript parsing of the XML. You can also run the map entirely based on static XML files only, and not use the MySQL database.

If you have never used PHP to connect to a MySQL database, visit [php.net](http://www.php.net/) and read up on mysql\_connect, mysql\_select\_db, my\_sql\_query, and mysql\_error.

You can use any of the three different techniques below to output the SQL table data as XML with PHP.

* [Using PHP's DOM XML functions to output XML](https://developers.google.com/maps/documentation/javascript/mysql-to-maps#domxml)
* [Using PHP's echo to output XML](https://developers.google.com/maps/documentation/javascript/mysql-to-maps#echoxml)
* [Using PHP's DOM functions to output XML](https://developers.google.com/maps/documentation/javascript/mysql-to-maps#domfunctions)

When using a public browser to access a database using PHP files, it's important to ensure that your database credentials are secure. You can do this by putting your database connection information in a separate PHP file to that of the main PHP code. The file with your credentials should look like the one below, but containing your own database information.

<?php  
$username="username";  
$password="password";  
$database="username-databaseName";  
?>

### Using PHP's DOM XML functions to output XML

The DOM XML functions of PHP take care of subtleties such as escaping special entities in the XML, and make it easy to create XML with more complex structures. You can use DOM XML functions to create XML nodes, append child nodes, and output an XML document to the screen. To determine if your server's PHP has the DOM XML functionality enabled, check your configuration or try initializing a domxml\_new\_doc().

Below is the PHP file that connects to the MySQL database, and dumps the XML to the browser.

<?php  
require("phpsqlajax\_dbinfo.php");  
  
// Start XML file, create parent node  
$doc = domxml\_new\_doc("1.0");  
$node = $doc->create\_element("markers");  
$parnode = $doc->append\_child($node);  
  
// Opens a connection to a MySQL server  
$connection=mysql\_connect ('localhost', $username, $password);  
if (!$connection) {  
  die('Not connected : ' . mysql\_error());  
}  
  
// Set the active MySQL database  
$db\_selected = mysql\_select\_db($database, $connection);  
if (!$db\_selected) {  
  die ('Can\'t use db : ' . mysql\_error());  
}  
  
// Select all the rows in the markers table  
$query = "SELECT \* FROM markers WHERE 1";  
$result = mysql\_query($query);  
if (!$result) {  
  die('Invalid query: ' . mysql\_error());  
}  
  
header("Content-type: text/xml");  
  
// Iterate through the rows, adding XML nodes for each  
while ($row = @mysql\_fetch\_assoc($result)){  
  // Add to XML document node  
  $node = $doc->create\_element("marker");  
  $newnode = $parnode->append\_child($node);  
  
  $newnode->set\_attribute("id", $row['id']);  
  $newnode->set\_attribute("name", $row['name']);  
  $newnode->set\_attribute("address", $row['address']);  
  $newnode->set\_attribute("lat", $row['lat']);  
  $newnode->set\_attribute("lng", $row['lng']);  
  $newnode->set\_attribute("type", $row['type']);  
}  
  
$xmlfile = $doc->dump\_mem();  
echo $xmlfile;  
  
?>

In the file above, the PHP code first initializes a new XML document and creates the "markers" parent node. It then connects to the database, executes a SELECT \* (select all) query on the markers table, and iterates through the results. For each row in the table (each location), the code creates a new XML node with the row attributes as XML attributes, and appends it to the parent node. The last part of code then dumps the XML to the browser screen.

**Note:**If your database contains international characters or you otherwise need to force a UTF-8 output, you can use [utf8\_encode](http://us2.php.net/manual/en/function.utf8-encode.php) on the data output.

### Using PHP's echo to output XML

If you don't have access to PHP's dom\_xml functions, then you can output the XML with the echo function. Use a helper function (e.g. parseToXML) when using just the echo function to correctly encode a few special entities (<,>,",') to be XML friendly.

Below is the PHP file that connects to the MySQL database, and dumps the XML to the browser.

<?php  
require("phpsqlajax\_dbinfo.php");  
  
function parseToXML($htmlStr)  
{  
$xmlStr=str\_replace('<','&lt;',$htmlStr);  
$xmlStr=str\_replace('>','&gt;',$xmlStr);  
$xmlStr=str\_replace('"','&quot;',$xmlStr);  
$xmlStr=str\_replace("'",'&#39;',$xmlStr);  
$xmlStr=str\_replace("&",'&amp;',$xmlStr);  
return $xmlStr;  
}  
  
// Opens a connection to a MySQL server  
$connection=mysqli\_connect ('localhost', $username, $password);  
if (!$connection) {  
  die('Not connected : ' . mysqli\_error());  
}  
  
// Set the active MySQL database  
$db\_selected = mysqli\_select\_db($database, $connection);  
if (!$db\_selected) {  
  die ('Can\'t use db : ' . mysqli\_error());  
}  
  
// Select all the rows in the markers table  
$query = "SELECT \* FROM markers WHERE 1";  
$result = mysqli\_query($query);  
if (!$result) {  
  die('Invalid query: ' . mysqli\_error());  
}  
  
header("Content-type: text/xml");  
  
// Start XML file, echo parent node  
echo "<?xml version='1.0' ?>";  
echo '<markers>';  
$ind=0;  
// Iterate through the rows, printing XML nodes for each  
while ($row = @mysqli\_fetch\_assoc($result)){  
  // Add to XML document node  
  echo '<marker ';  
  echo 'id="' . $row['id'] . '" ';  
  echo 'name="' . parseToXML($row['name']) . '" ';  
  echo 'address="' . parseToXML($row['address']) . '" ';  
  echo 'lat="' . $row['lat'] . '" ';  
  echo 'lng="' . $row['lng'] . '" ';  
  echo 'type="' . $row['type'] . '" ';  
  echo '/>';  
  $ind = $ind + 1;  
}  
  
// End XML file  
echo '</markers>';  
  
?>

The code above connects to the database and executes the SELECT \* (select all) query on the markers table. It then echoes the parent markers node, and iterates through the query results. The code then echoes the XML node for the marker in each row of the table (for each location). This sends the name and address fields through the parseToXML function first, in case there are any special entities in them. The script ends by echoing out the closing markers tag.

**Note:** You can use [utf8\_encode](http://us2.php.net/manual/en/function.utf8-encode.php) on the XML data output if your database contains international characters, or if you need to force an UTF-8 output.

### Using PHP's DOM functions to output XML

Below is the PHP file that connects to the MySQL database, and dumps the XML to the browser.

<?php  
  
require("phpsqlajax\_dbinfo.php");  
  
// Start XML file, create parent node  
  
$dom = new DOMDocument("1.0");  
$node = $dom->createElement("markers");  
$parnode = $dom->appendChild($node);  
  
// Opens a connection to a MySQL server  
  
$connection=mysql\_connect ('localhost', $username, $password);  
if (!$connection) {  die('Not connected : ' . mysql\_error());}  
  
// Set the active MySQL database  
  
$db\_selected = mysql\_select\_db($database, $connection);  
if (!$db\_selected) {  
  die ('Can\'t use db : ' . mysql\_error());  
}  
  
// Select all the rows in the markers table  
  
$query = "SELECT \* FROM markers WHERE 1";  
$result = mysql\_query($query);  
if (!$result) {  
  die('Invalid query: ' . mysql\_error());  
}  
  
header("Content-type: text/xml");  
  
// Iterate through the rows, adding XML nodes for each  
  
while ($row = @mysql\_fetch\_assoc($result)){  
  // Add to XML document node  
  $node = $dom->createElement("marker");  
  $newnode = $parnode->appendChild($node);  
  $newnode->setAttribute("id",$row['id']);  
  $newnode->setAttribute("name",$row['name']);  
  $newnode->setAttribute("address", $row['address']);  
  $newnode->setAttribute("lat", $row['lat']);  
  $newnode->setAttribute("lng", $row['lng']);  
  $newnode->setAttribute("type", $row['type']);  
}  
  
echo $dom->saveXML();  
  
?>

The code above initializes a new XML document and creates the "markers" parent node. It then connects to the database, executes a SELECT \* (select all) query on the markers table, and iterates through the results. The code then creates a XML node for each row in the table (for each location), with the row attributes as XML attributes, and appends it to the parent node. It then dumps the output XML to the browser screen.

**Note:**If your database contains international characters, or you need to force a UTF-8 output, you can use [utf8\_encode](http://us2.php.net/manual/en/function.utf8-encode.php) on the data output.

### Checking that XML output works

To confirm that the PHP script is producing valid XML, call the php script file you created from your browser to see the XML output as below.

<markers>  
<marker id="1" name="Love.Fish" address="580 Darling Street, Rozelle, NSW" lat="-33.861034" lng="151.171936" type="restaurant"/>  
<marker id="2" name="Young Henrys" address="76 Wilford Street, Newtown, NSW" lat="-33.898113" lng="151.174469" type="bar"/>  
<marker id="3" name="Hunter Gatherer" address="Greenwood Plaza, 36 Blue St, North Sydney NSW" lat="-33.840282" lng="151.207474" type="bar"/>  
<marker id="4" name="The Potting Shed" address="7A, 2 Huntley Street, Alexandria, NSW" lat="-33.910751" lng="151.194168" type="bar"/>  
<marker id="5" name="Nomad" address="16 Foster Street, Surry Hills, NSW" lat="-33.879917" lng="151.210449" type="bar"/>  
<marker id="6" name="Three Blue Ducks" address="43 Macpherson Street, Bronte, NSW" lat="-33.906357" lng="151.263763" type="restaurant"/>  
<marker id="7" name="Single Origin Roasters" address="60-64 Reservoir Street, Surry Hills, NSW" lat="-33.881123" lng="151.209656" type="restaurant"/>  
<marker id="8" name="Red Lantern" address="60 Riley Street, Darlinghurst, NSW" lat="-33.874737" lng="151.215530" type="restaurant"/>  
</markers>

If your browser isn't displaying the marker data from your database as an XML output, try debugging by removing the line in the file that sets the header to the text/xml content type. This line may cause your browser to try to parse XML and make it difficult to see your debugging messages.

## Creating the map

This section shows you how to develop the map example in this tutorial using JavaScript, and the output XML file. Read the documentation to learn more about the [basics of creating a Google map](https://developers.google.com/maps/documentation/javascript/adding-a-google-map).

Create a new file in a text editor and save it as index.html. Read the sections that follow to understand the code that you can add to this file.

### Loading the XML file

To load the XML file into the page, you can take advantage of the browser-provided [XMLHttpRequest](http://www.w3.org/TR/2009/WD-XMLHttpRequest-20090820/) object. This object allows you retrieve a file that resides on the same domain as the requesting webpage, and is the basis of "AJAX" programming.

Define your own function for loading the file, and call it downloadUrl(). The function takes two parameters:

1. url specifies the path to either your XML file or to the PHP script that generates the file, depending on whether you want to dynamically update the XML file when your database changes. This map in this tutorial calls a static XML file for the marker data.  
   It is usually easiest to have this XML file reside in the same directory as the HTML file so that you can just refer to it by filename.
2. callback indicates the function that the script calls when the XML returns to the JavaScript.

The code below shows you the function declaration.

function downloadUrl(url,callback) {  
 var request = window.ActiveXObject ?  
     new ActiveXObject('Microsoft.XMLHTTP') :  
     new XMLHttpRequest;  
  
 request.onreadystatechange = function() {  
   if (request.readyState == 4) {  
     request.onreadystatechange = doNothing;  
     callback(request, request.status);  
   }  
 };  
  
 request.open('GET', url, true);  
 request.send(null);  
}

**Note:**Since the XMLHttpRequest is asynchronous, the callback function initiates the downloadURL function based on the size of the XML file. The bigger your XML file, the longer it may take. For this reason, it would be best not to put any code after the downloadUrl that relies on the markers inside the callback function. Instead, such code can be put inside the callback function.

Now that you have defined the function, you can call it from your code, passing in the name of your PHP file and callback function. The map in this tutorial calls a static XML file for the marker data, as in the code below.

downloadUrl('https://storage.googleapis.com/mapsdevsite/json/mapmarkers2.xml', function(data) {  
  var xml = data.responseXML;  
  var markers = xml.documentElement.getElementsByTagName('marker');  
  Array.prototype.forEach.call(markers, function(markerElem) {  
    var id = markerElem.getAttribute('id');  
    var name = markerElem.getAttribute('name');  
    var address = markerElem.getAttribute('address');  
    var type = markerElem.getAttribute('type');  
    var point = new google.maps.LatLng(  
        parseFloat(markerElem.getAttribute('lat')),  
        parseFloat(markerElem.getAttribute('lng')));  
  
    var infowincontent = document.createElement('div');  
    var strong = document.createElement('strong');  
    strong.textContent = name  
    infowincontent.appendChild(strong);  
    infowincontent.appendChild(document.createElement('br'));  
  
    var text = document.createElement('text');  
    text.textContent = address  
    infowincontent.appendChild(text);  
    var icon = customLabel[type] || {};  
    var marker = new google.maps.Marker({  
      map: map,  
      position: point,  
      label: icon.label  
    });

### Creating custom markers

The code below shows you how to add custom labels for your markers by first creating an associative array. This associates your labels with the marker type strings: restaurant or bar. This makes the labels easy to reference when you create markers from the XML file.

var customLabel = {  
  restaurant: {  
    label: 'R'  
  },  
  bar: {  
    label: 'B'  
  }  
};

### Creating markers and info windows

When creating markers, you can retrieve the appropriate marker labels by using the type as a key for the customLabel associated array. Forward the .label property as a google.maps.Marker constructor option.

Next, create the HTML that you want to show up in the info window by concatenating the name, address, and HTML tags to emphasize the name.

If you store HTML-formatted descriptions in your database, you may have to deal with escaping HTML entities and be bound to that HTML output. By waiting until you have retrieved each attribute separately in the JavaScript, you are free to play around with the HTML on the client side and can quickly preview new formatting.

After constructing the HTML string, the code below adds an event listener to the marker which displays an info window on click.

marker.addListener('click', function() {  
  infoWindow.setContent(infowincontent);  
  infoWindow.open(map, marker);  
});

### Putting it all together

Open the index.html file in a browser. When the page loads, the initMap function sets up the map and then calls the downloadUrl function. This function iterates through all the marker elements, and retrieves the name, address, type, and latLng attributes for each marker element.  
The code then creates markers, adds the markers to the map, and binds an info window to each marker to display a description on click.

## More information