# Table of Contents

1. Introduction ........................................................................................................................................... 4
2. Terminology ................................................................................................................................................ 5
3. Live Data Concepts .................................................................................................................................. 6
   3.1 Introduction ........................................................................................................................................ 6
   3.2 Live Data components ......................................................................................................................... 7
   3.3 File locations ........................................................................................................................................ 8
4. Live Data Hotspot .................................................................................................................................... 10
   4.1 Setting a Hotspot’s Live Data option ................................................................................................. 10
   4.2 Parts of Hotspot Content that can be set with Live Data ................................................................. 11
   4.3 Calling a Hotspot’s Messenger Function ......................................................................................... 12
5. Messenger Functions .............................................................................................................................. 12
   5.1 Tour/Server communication ............................................................................................................. 12
   5.2 The Live Data request functions ..................................................................................................... 14
   5.3 Messenger function examples .......................................................................................................... 15
6. Server Script or Data File for Live Data ............................................................................................... 16
7. Web Page for Live Data .......................................................................................................................... 18
8. Hotspot XML Format .............................................................................................................................. 19
   8.1 Overview ............................................................................................................................................ 19
   8.2 Using CDATA or HTML character encoding .................................................................................. 19
   8.3 Hotspot XML elements .................................................................................................................... 20
   8.4 Compatibility with previous versions of MapsAlive ....................................................................... 21
9. Putting it all together .............................................................................................................................. 22
   9.1 Create and publish a Live Data tour .................................................................................................. 22
   9.2 Create a web page that contains your Live Data tour ...................................................................... 22
   9.3 Create a server script or data file to return Live Data .................................................................... 23
   9.4 Put the web page and script or data files on your server ............................................................... 23
   9.5 Test your Live Data web page in browser ...................................................................................... 23
10. Demo files ............................................................................................................................................... 23
| 10.1 | Directory preview ........................................................................................................ 24 |
| 10.2 | Using the `<text>` element ......................................................................................... 24 |
| 10.3 | Styling live data with CSS ......................................................................................... 26 |
| 11   | Customizing the appearance of server errors ............................................................ 26 |
| 11.1 | Default Live Data error message ............................................................................... 26 |
| 11.2 | Custom Live Data error message ............................................................................... 27 |
| 11.3 | Enabling custom error messages ................................................................................ 27 |
| 11.4 | The `maLiveDataSetCustomError` function ............................................................... 28 |
| 11.5 | Reasons that a server error could be reported ............................................................ 28 |
| 11.6 | Using Live Data in Tour Preview ................................................................................ 29 |
| 12   | Requesting Other Information from Your Server ....................................................... 29 |
| 12.1 | Server request functions ......................................................................................... 30 |
| 12.2 | When and where to use the server request functions ............................................... 31 |
| 12.3 | Getting the data that your server sends back ............................................................ 32 |
| 12.4 | XML Format .............................................................................................................. 33 |
| 12.5 | Example .................................................................................................................... 34 |
1 Introduction

What is Live Data?
Live Data refers to hotspot content – text and images – that is obtained from your server in real-time and displayed as the information associated with hotspots in a MapsAlive interactive map. With Live Data, a person using your map sees the latest information available as they move their mouse over hotspots. The data can originate from a file, your database, an RSS feed, or any other source that you make available from your server.

Here are some of the benefits of Live Data:

- Lets you present the latest information without having to edit and publish your map.
- Your interactive map content changes automatically whenever you update data on your server.
- The people who update your data don’t need to have access to MapsAlive.
- You can use any tools you like to create fully formatted HTML for your Live Data content.
- Your data is located in only one place – on your server.

Is it hard to use?
Live Data is an advanced feature, but in its simpler forms, just about anyone can use it. If you can create files containing text or HTML that you want to display from your interactive map, and if you can put those files on your web server, you can probably use Live Data. If you want the data to come from a database or an RSS feed, then you’ll need to have expertise with server technologies such as scripting and query writing. Either way, this document contains a lot of information and examples to help you get started.

Who this guide is for
This guide provides step-by-step information to set up Live Data to work with your server. It is for the person who will be using MapsAlive to create interactive maps and for the person who will be creating files or server-side scripts to return Live Data to those maps. If the data will be coming from your database, your database programmer should read this guide as well.

How to get more information or assistance
If you need more details, a better explanation, or just a little bit of hand-holding, we are here to help. Please email questions to support@mapsalive.com.

You can find other MapsAlive User Guides at http://www.mapsalive.com/LearningCenter.
2 Terminology

MapsAlive
MapsAlive is a web application that runs in your PC or Mac browser with a high-speed internet connection. There is no software to install and you are always using the latest version.

Tour Builder
The MapsAlive Tour Builder is where you create, edit, and publish interactive maps. When you login to your MapsAlive account you can use the Tour Builder to upload map images and photos, enter text, create hotspots, choose layouts, and set the appearance and behavior of your interactive maps.

Tour
A tour is one or more interactive maps and data sheets that are usually related to each other. Many tours might only have one map. Other tours, like a house tour, might have several floor plans together in one tour to let a prospective buyer explore the house. You publish your tour when you are ready for other people to use them.

Map
A map is a component of your tour that serves as a container for a single map image, the map’s hotspots and hotspot content. You upload a map image and add hotspots to the map using the Tour Builder.

Data sheet
A data sheet contains photos, text and other information related to your tour, but it has no map. A data sheet might be used to display introductory text, contact information, specifications, or similar non-interactive content.

Map Image
A map image can be any image such as a diagram, floor plan, geographic map, or even a photograph. You can think of it as a background layer beneath a layer of hotspots. You upload one map image for each map in your tour.

Hotspot
A hotspot is a location on an interactive map that shows content associated with that location when the mouse moves over it, off of it, or clicks it. The size, shape, and appearance of a hotspot are indicated by a marker.

Hotspot Content
Hotspot content is the information associated with a specific hotspot such as photos, text, and videos. When the mouse moves onto or clicks a hotspot, the content displays. A hotspot can also have an action associated with it.

Marker
A marker can be an image (like an arrow), a visible shape (like a rectangle or polygon), or an invisible shape, that marks the presence of a hotspot on a map. Each hotspot on a map has a marker associated with it. The marker can be unique to one hotspot or the marker can be used by several hotspots.

Tiled versus Popup Content
Tiled content means that your hotspot content displays alongside the map – both the content and the map display within the rectangular canvas area. Popup content is hotspot content that displays in a rectangular area that pops up on top of your map when you mouse over a hotspot on the map.

Canvas
Every tour has a canvas area that displays one map or data sheet. For maps with tiled content, the canvas area contains the map image and the hotspot content. For maps with popup content, the canvas area contains only the map image. For data sheets, the canvas area contains the photo and text for the data sheet.
3 Live Data Concepts

This section explains Live Data concepts and points you to other parts of this document where more detailed information can be found.

Sections 3 – 11 of this document explain how to use Live Data to obtain hotspot content from your server, but you can also use Live Data to communicate with your server for other purposes. To learn how, see section 12.

3.1 Introduction

When a user moves their mouse over a hotspot's marker on a MapsAlive interactive map, they see the content that is associated with that hotspot. The content is data such as a photo, text, or video. Normally (that is, when not using Live Data) you enter this data directly into MapsAlive by typing text, uploading a photo from your computer, or importing an Excel file. The data you enter is stored in the MapsAlive database and later, when you publish your tour, it is written to files in the tour folder. When the tour is running in a browser and someone mouses over a marker on the map, the data is retrieved from the files and displayed to the user as shown below.

![Figure 1–A tour without Live Data](image)

When a tour uses Live Data, the data is stored on your server instead of in a file the tour. When the tour is running in a browser and someone mouses over a marker on the map, the tour makes a request via the internet to your server to get the data. When your server returns the data, it is displayed to the user as shown below.

![Figure 2–A tour with Live Data](image)

The difference between a tour that uses Live Data and one that does not is where the data is stored. A big advantage to using Live Data is that when the data on your server changes, you don’t have to
change your tour – the next person who uses the tour will see the new data automatically. Without Live Data, you have to edit or re-import your tour’s data when it changes and then republish the tour.

Note that a tour can use mixed data. That is, some hotspots can use Live Data and others not.

### 3.2 Live Data components

This section gives a high level overview of the main components that are required in order to use Live Data. They are listed below and depicted in Figure 3. The numbers in parenthesis correspond to the numbered arrows in the diagram.

- **Live Data Hotspot** – calls your messenger function (1).
- **Messenger Function** – calls your server script (2); receives the data that the server sends back (3); and uses the data to update the content for the hotspot that requested it (4).
- **Server Script** – returns Live Data from your server (3).
- **Web Page** – contains your tour and your messenger function.

![Figure 3 – How Live Data components interact with each other](image)

A brief description of each component follows. Detailed explanations appear later in this document.

**Live Data Hotspot**

A Live Data hotspot is an ordinary MapsAlive hotspot that has the Live Data option selected and has code to call to its messenger function. The code is a single line of JavaScript that specifies the name of the messenger function and provides a parameter to indicate what data is being requested for this hotspot. For example, if the map was an org chart and the hotspot was for an employee, the parameter could be an employee number. *Live Data Hotspots are further explained in section 4.*

**Messenger Function**

A messenger function is a JavaScript function containing just one line of code that calls your server to request the data for the hotspot. In the function call you specify the URL for your server script along with any parameters that your script expects, for example, an employee number. The messenger
function is the “glue” that connects your interactive map with your server. Don’t worry if you don’t know JavaScript – just follow the examples in this guide. Messenger functions are explained in section 5.

Server Script or Data File
A server script (also known as a web service) is a file on your server that can be called from the internet. It is written in a scripting language appropriate for your server. This document provides examples for ASP.NET and PHP server environments. The server script receives Live Data requests from your tour and responds with the requested data.

A data file is just what it sounds like – a file containing data. Live Data supports data in both XML and HTML formats. XML data can contain HTML provided that special HTML characters like ‘<‘ and ‘>‘ are properly escaped (we’ll tell you how in section 8.2). Note that HTML data can be simple text that does not contain any HTML syntax.

You tell MapsAlive whether to use a script or a file in your messenger function. The server script / data file component is explained in section 6.

Web Page
A web page used for Live Data is HTML that is in a file located on your server or is rendered dynamically by your server. You embed your tour inside the web page so that the page contains your tour. The web page can also be used to contain the JavaScript for your messenger function. The web page HTML renders in a browser to present your tour and any other related content to a user. It could be a web page that you already have and now you want to enhance it with an interactive map. All of this will be discussed more later on, but the MapsAlive User Guide for Integrating Interactive Maps explains what it means to embed a tour and how to do it. The web page component is explained in section 7.

3.3 File locations
Each Live Data component has files associated with it. Figure 4 shows two possible configurations.

Figure 4 – Where Live Data component files are located
You can see from the diagrams that both the server script or data file, and the web page that you use for Live Data, must be located on your server. The MapsAlive tour files, however, can either be located on your server as shown on the left, or on our server (or another server) as shown on the right. Which configuration you choose depends on whether or not you want to host the tour on your server. By default, when you publish a tour, it is automatically hosted on our server. To host it on yours, you need to download it and then put it on your server. To learn about the download feature, see the MapsAlive User Guide for Integrating Interactive Maps.

More about the Web Page component
It will be helpful to understand what a MapsAlive tour actually is and what it means for a tour to be “contained” in a web page. A tour is a collection of files in a single folder. The name of the folder is usually the tour number, but that is not a requirement. A tour folder and its files are created when you publish a tour using the MapsAlive Tour Builder. You can run the tour from its published location on tour.mapsalive.com or you can download the tour folder and upload it to your own server. You run a tour by loading one of its page files (page1.htm for example) in a browser.

To use a tour with Live Data, the tour must be contained inside another web page. What does it mean for a tour to be contained in a web page? Clearly the tour’s folder is not physically contained in a web page file. What “contained” means is that the web page contains references to URLs for the tour’s files, specifically its JavaScript and CSS files. These references are made using <script> and <link> tags which you can see in section 7 later in this document.

Now that you know what it means for a tour to be contained in a web page, it should be easier to understand why a tour can be hosted either on your own server or on another server as was shown in Figure 4 above. Since the web page only contains references to the tour’s files, the tour itself can be hosted on any server as long as your server has permission to access it at that location. The tour could not be hosted behind someone else’s firewall or on a non-networked computer, but it can be hosted anywhere that is reachable from your server.

Security restrictions on file locations
The URL of your server script or data file and the URL of your web page must both have the same domain name. This rule is enforced by browsers to implement a security model known as the Same Origin Policy. The policy states that JavaScript code running on a web page may not interact with any resource not originating from the same web site. The policy prevents malicious web coders from creating pages that steal web users’ information or compromise their privacy. This restriction is unlikely to affect you and you don’t need to understand it, but if you want to learn more, do a Google search for “Same Origin Policy.”

What’s important to understand about this security restriction is that

- Your Live Data web page must be located on the same server – your server or someone else’s server, but not on mapsalive.com – as the server script or data file.
• The tour itself (its files) can be located on any server (including mapsalive.com) that the web page has access to.

An alternative to the Web Page component
Strictly speaking you don’t need a separate web page that contains your tour in order to use Live Data. Why? Because a tour’s page file (like page1.htm) is a web page that can serve as the web page component of a Live Data solution. Furthermore, you can code your messenger function inside the JavaScript section of your tour’s Custom HTML thus eliminating the need for a containing web page solely for the purpose of holding JavaScript. Note that even if you don’t choose this option, you can put your messenger function in your tour’s Custom HTML or in a containing web page – it doesn’t matter where it goes as long as the tour can call it. You can read about Custom HTML in the MapsAlive User Guide for the JavaScript API.

If you choose this option, the tour must be hosted on your server to satisfy the security restrictions explained in the previous section. Your configuration would look like diagram on the left in Figure 4 without the separate Web Page.

4 Live Data Hotspot
A hotspot that uses Live Data differs from ordinary hotspot in two ways:

1. It has its Live Data option checked
2. It has a messenger function to request the hotspot’s data from your server

The sections that follow go into detail about using Live Data hotspots.

4.1 Setting a Hotspot’s Live Data option
To specify that a hotspot uses Live Data, you check the Use Live Data box on the Advanced Hotspot Options screen in the MapsAlive Tour Builder. You also type a messenger function call in the text box. You do this for each hotspot that uses Live Data. In the figure below, the messenger function is called getEmployee and its parameter is the employee number for a specific hotspot.

![Figure 5 – Live Data option on the Advanced Hotspot Options screen](image-url)
If your tour has a lot of hotspots, you can quickly choose the Live Data option and provide the messenger function and parameters for all of your hotspots using an import file. You can read about import in the MapsAlive User Guide for Content Management.

Not all hotspots in a tour have to use Live Data. Hotspots that don’t have their Live Data box checked will use the data that you enter manually on the Edit Hotspot Content screen or import from a file.

4.2 Parts of Hotspot Content that can be set with Live Data

When you set a hotspot’s Live Data option, you are telling MapsAlive that when the mouse moves over this hotspot’s marker, the hotspot should get its information from the server instead of from the tour itself. The following parts of a hotspot’s content can be set from Live Data. Each part is identified in Figure 6.

- Text area – can contain any HTML to display images, text, links and more
- Directory preview image (only supported for data provided as XML)
- Directory preview text (only supported for data provided as XML)

![Figure 6 – Parts of a hotspot’s content that can be set with Live Data](image)

In Figure 6, the user is viewing the hotspot content for Kate Bush-White while mousing over the directory entry for Billy Cordan, the Eastern Region Manager. Live Data is being used to retrieve the images and text for both the directory preview and the hotspot’s content.

You can put any kind of HTML in the text area to display formatted text, images, tables, form fields etc. As you’ll learn later, you can style your HTML using CSS. For the directory preview, you provide a URL for the image and plain text or HTML for the text.
4.3 Calling a Hotspot’s Messenger Function

Figure 5 above showed an example of a Live Data call to a messenger function named `getEmployee`. In the example, the function takes a parameter “1234” that will be passed to the server to indicate which employee the data is being requested for. The tour will call this messenger function automatically when the user mouses over the marker for that hotspot. Note that whether or not the parameter must be enclosed in quotes depends on whether or not your server is expecting string or numeric data.

Every hotspot in your tour can call the same messenger function, or different hotspots can call different messenger functions. For example, a hotspot that displays employee data would call a different messenger function than a hotspot that displays information about stock prices.

5 Messenger Functions

A messenger function acts as a liaison between your tour and your server. It is called a messenger function because it is used to send a message to your server (a request for data) and get a message back (the data that was requested). Since a tour can’t possibly know how to communicate with your server, you must tell it how by specifying your server script URL and parameter names. You do this inside the messenger function as shown in the example below.

```javascript
function getEmployee(id)
{
    maLiveDataRequestHotspotXml(60, "getemployee.php", "employeeId", id);
}
```

Figure 7 – A messenger function example

As you can see, the messenger function itself doesn’t do much. It’s really just a stand-in or proxy for a Live Data function that knows how to talk to servers. The messenger function is very important, however, because it allows many Live Data hotspots to call a single piece of code that has knowledge of how to talk to your server. If the way you call your server changes (for example, if your server’s URL changes), all you need to do is edit the messenger function – you don’t need to change every hotspot that uses the messenger function. The `maLiveDataRequestHotspotXml` function will be explained in the next section.

5.1 Tour/Server communication

When you check the Use Live Data box for a hotspot in your tour, MapsAlive automatically includes a JavaScript library named `livedata.js` when it creates your tour. This library has two hotspot request functions that you use for getting Live Data for a hotspot. The two functions are `maLiveDataRequestHotspotXml` and `maLiveDataRequestHotspotHtml`. Before we continue, we want to emphasize that these function names are case-sensitive. That means that you must type them with the
upper and lower case letters exactly as shown (so don’t be tempted to end them with XML or HTML instead of Xm1 or Htm1).

Whether you use XML or HTML depends on what makes most sense for you on your server. Note, however, that if you want to use Live Data for the directory preview and text values, you must use XML as was pointed out in section 4.2 above.

The hotspot request functions do the work of allowing your tour to communicate with your server using what is called an HTTP request (sometimes referred to as AJAX). Which of the two functions you use depends on whether your server will be returning XML or HTML. The round-trip process of tour/server communication is done for you automatically. You don’t need to know anything about HTTP requests and you don’t have to write any JavaScript except to for the one-line messenger function.

As a note for advanced users, you can look at the livedata.js file to see how it works, but do not attempt to call any functions except those that are documented in this user guide. The other functions should be considered private and their names and parameters could change at any time.

Figure 8 below shows how a Live Data hotspot gets its data from the server when a user moves their mouse over the marker for an employee on an interactive org chart map. This example shows data returned as XML, but it works similarly for data returned as HTML.

The numbered steps in the diagram above are explained here:

1. A person using your tour moves their mouse over the marker for employee 1234.
2. The hotspot calls your messenger function getEmployee passing “1234” as a parameter.
3. The messenger functions calls maLiveDataRequestHotspotXml passing your server URL and parameters.
4. The livedata.js library receives the call, sets up an HTTP request, and sends it to your server.

Figure 8 – Round trip from a tour to the server and back again
5. Your server receives the request, gets the data for employee 1234, and formats it as HTML that you want displayed as your employee content.

6. Your server constructs Hotspot XML containing the HTML from step 5 and returns it as its response to the HTTP request. The server encodes special characters in the HTML so they won’t conflict with XML characters (see section 8.2 to learn how to do this).

7. The livedata.js library receives the server response, extracts and decodes the HTML from the XML, and updates the hotspot with the HTML for employee 1234.

### 5.2 The Live Data request functions

In the previous sections you saw how `maLiveDataRequestHotspotXml` and `maLiveDataRequestHotspotHtml` are at the heart of messenger functions and tour/server communication. This section explains how to use these very powerful function.

The signature for `maLiveDataRequestHotspotXml` is:

```js
maLiveDataRequestHotspotXml(cachePeriod, url, parameter-list)
```

The signature for `maLiveDataRequestHotspotHtml` is:

```js
maLiveDataRequestHotspotHtml(cachePeriod, url, parameter-list)
```

Both functions take the same parameters which are described in the table below:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| cachePeriod    | The cache period is the number of seconds that must elapse between Live Data calls to your server for any given hotspot in a tour. The minimum number of seconds you can specify is 1 second. Specify a value of 1 or greater for data that could change on your server while someone is viewing your tour. Specify zero for data that rarely or never changes such as employee information. Zero tells MapsAlive to only call your server once during the time someone is using a map (if they leave the map and come back, the server will be called again).

When your tour requests data from your server, the tour keeps a copy of the data in its cache. If the elapsed time between two requests for the same hotspot is less than or equal to the value of cachePeriod, or if cachePeriod is set to zero, the tour will display the cached data for that hotspot instead of calling the server. Otherwise, the tour will make a new call to the server to get data.

You can force the cache period for all hotspots on a map to expire immediately by calling the MapsAlive JavaScript API function `flushLiveDataCache`.

<table>
<thead>
<tr>
<th>url</th>
<th>The url specifies how to call your server script that returns Live Data. Since</th>
</tr>
</thead>
</table>


the web page that is making this call must be on the same server as the script, the URL can be a relative reference, that is, it does not need to specify the full path to the script. For example it could be just `getEmployee.php` instead of `http://www.myserver.com/getEmployee.php`. You can however use a fully qualified URL if you like.

| parameter-list | The parameter-list is a variable length list of name/value pairs separated by commas. They are used to tell your server script what data is being requested. If you don't need to pass any parameters you can omit the list. |

### 5.3 Messenger function examples

To better illustrate how a Live Data request function is used, study the following code samples.

**Example requesting XML using a PHP server script**

Below is a messenger function that calls `maLiveDataRequestHotspotXml` to make a request to a PHP server. It specifies 60 seconds for the cache period, a url of "demo.php", and one name/value pair for the parameter-list.

```php
function getEmployee(cityName)
{
    maLiveDataRequestHotspotXml(60, "demo.php", "employeeCity", cityName);
}
```

Here is how the messenger function above might be called from a hotspot:

```php
getEmployee("newyork");
```

For those familiar with how server parameters are passed from a browser using query strings, the following syntax shows the query string equivalent of the call shown above.

```php
demo.php?employeeCity=newyork
```

**Example requesting HTML using an ASP.NET server script**

Below is a messenger function that calls `maLiveDataRequestHotspotHtml` to make a request to an ASP.NET server. It specifies 10 seconds for the cache period, a url of "stocks.asmx/GetPrice", and two name/value pairs for the parameter-list.

```php
function getStockQuote(ticker, exchange)
{
    maLiveDataRequestHotspotHtml(10, "stocks.asmx/GetPrice", "tick", ticker, "ex", exchange);
}
```

Here is how the messenger function above might be called from a hotspot:

```php
getStockQuote("ATT", "NYSE");
```
Below is the query string equivalent of the call shown above.

stocks.asmx/GetPrice?tick=ATT&ex=NYSE

**Example requesting HTML from a file on any kind of web server**

Below is a messenger function that calls `maLiveDataRequestHotspotHtml` to make a request to any kind of web server to get HTML containing information about a house for sale.

Here is how the messenger function above might be called from a hotspot:

```javascript
getListingInfo(789);
```

When the number 789 is passed as the propertyId, the function calls the server specifying 0 seconds for the cache period, a url of "listing789.htm", and no parameters.

```javascript
function getListingInfo(propertyId)
{
    var url = "listing" + propertyId + "htm";
    maLiveDataRequestHotspotHtml(0, url);
}
```

This last example passes no parameters which should always be the case when requesting Live Data from an HTML file (listing789.htm) or an XML file (for example listing789.xml). Unlike a server script, a data file has no logic – you simply get back whatever data the file contains. While more limiting than a server script, it is still a powerful mechanism. In the example, the file name is constructed from a property Id with the idea that the server hosts one HTML file for each house for sale. If you had an interactive map showing the locations of houses for sale, you could have the listing information display when the user mouses over a house. If the data for a house changes – its price for example – you could simply edit its file on the server and the next person to mouse over that house would see the new price.

### 6 Server Script or Data File for Live Data

You can skip this section if you are going to provide your data in an HTML or XML file. Note that if you are just getting started with Live Data, returning your data in a file is a great way to see how it works before you do any server-side programming.

A server script (also known as a web service) is a file on your server that can be called from the internet. It is written in a scripting language appropriate for your server. The script receives a Live Data request from your map and responds with the requested data. Simple examples in ASP.NET and PHP are shown below. If you are using another scripting language such as classic ASP or Java, you can use the samples shown here as a guide for coding your own scripts in those languages.

**ASP.NET Example**

The code below returns the value of the parameter that was passed to it along with the time.
[WebMethod]
public XmlDocument Hello(string name)
{
    // Make some data.
    string data = "Hello " + name + " at " + DateTime.Now.ToString();

    // Put the data inside the hotspot XML.
    string xml = string.Format("<hotspot><text>{0}</text></hotspot>", data);

    // Return the XML to the browser.
    XmlDocument xmlDoc = new XmlDocument();
    xmlDoc.LoadXml(xml);
    return xmlDoc;
}

Figure 9 – ASP.NET web service in C#

PHP Example
This is the PHP version of the example above. It also returns its parameter value along with the time.

```php
<?php
// Get the name parameter.
$name = isset($_REQUEST["name"]) ? $_REQUEST["name"] : "no-name";

// Make some data.
$data = "Hello " . $name . " at " . date("H:i:s");

// Put the data inside the hotspot XML.
$xml = "<hotspot><text>" . $data . "</text></hotspot>";

// Send the XML to the browser.
header("Content-Type: text/xml");
echo "<?xml version="1.0" encoding="utf-8"?>";
echo $xml
?>
```

Figure 10 – PHP server script

Output
The two scripts above return a greeting and time of day. If you call either script and pass “world” as the name parameter, they will respond with the following XML:

```xml
<?xml version="1.0" encoding="utf-8" ?>
<hotspot>
   <text>Hello world at 09:18:43</text>
</hotspot>
```

Figure 11 – XML returned from server script
7 Web Page for Live Data

A web page used for Live Data acts as a container for your tour and its messenger function. For an explanation of what it means for a tour to be contained in a web page, see section 3.3.

You don’t have to create your Live Data web page by hand because all the code you need is provided for you in the Code Snippets section of the MapsAlive Tour Preview screen. Of course, in real life your web page will contain more than just a tour. It will probably have navigation, graphics, and other content, but you can still paste the relevant code snippets into your HTML to minimize what you have to write.

When a tour has at least one hotspot that uses Live Data, the code snippets will contain a messenger function placeholder. If you don’t see it in Tour Preview, make sure you have a Live Data hotspot and that you are previewing the map that contains the hotspot.

Below is the code snippet that was automatically created by Tour Preview for tour 1234:

```html
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>
<body>
<h3>This is a MapsAlive tour embedded directly into a web page</h3>
<link rel="stylesheet" type="text/css" href="http://tour.mapsalive.com/1234/page1.css" />
<script type="text/javascript" src="http://tour.mapsalive.com/1234/mapsalive.js"></script>
<script type="text/javascript" src="http://tour.mapsalive.com/1234/livedata.js"></script>
<script type="text/javascript">
// Your Live Data messenger function goes here or in the JavaScript section of Custom HTML function yourFunctionName(arg1)
{
  maLiveDataRequestHotspotXml(60, 'your-web-service-url', 'arg1name', arg1);
}
</script>
<script type="text/javascript">maClient.path="http://tour.mapsalive.com/1234/";</script>
<script type="text/javascript" src="http://tour.mapsalive.com/1234/page1.js"></script>
<p>This is the rest of the page</p>
</body>
</html>

Figure 12 – Code snippet for Live Data web page

Notice the messenger function. In this example we have highlighted in green the little bit of text that you need to change to specify your server script’s URL and parameters. If you are returning your data as HTML instead of XML, use the maLiveDataRequestHotspotHtml function instead of the maLiveDataRequestHotspotXml function. See section 5.3 above for example code.

Note that you can put your messenger function in the JavaScript section of your tour’s Custom HTML instead of coding it where shown above. We’ll talk about this a little more in section 9.2 when we put
together a working Live Data example. You can read about Custom HTML in the MapsAlive User Guide for the JavaScript API.

8 Hotspot XML Format

You can skip this section if you are returning your data from you server as HTML by calling maLiveDataRequestHotspotHtml.

8.1 Overview

This section describes the XML that your server must return in response to a Live Data request made using the maLiveDataRequestHotspotXml function. We call it Hotspot XML just to give it a name that we can talk about, but it’s just ordinary XML and it is very simple.

The format of Hotspot XML is shown below:

```
<hotspot>
  <text>plain text or HTML</text>
  <previewImage>an image URL</previewImage>
  <previewText>plain text or HTML</previewText>
  <serverError>an error message</serverError>
</hotspot>
```

Every element of the Hotspot XML is optional. Your server only needs to emit the elements for parts of your hotspot content that you want to use Live Data for. For example, if you only want to update a hotspot’s text area, the following XML would be all that you need:

```
<hotspot>
  <text>This is text to be displayed in the hotspot’s text area.</text>
</hotspot>
```

Other sections of this document contain many more examples of Hotspot XML. Section 8.3 explains each Hotspot XML element individually.

8.2 Using CDATA or HTML character encoding

The HTML you use inside the <text> or <previewText> elements must be treated so that it will not be parsed as XML by the XML parser. This is a requirement because characters like ‘<’ and ‘&’ are illegal in XML elements. ‘<’ will generate an error because the parser interprets it as the start of a new element. ‘&’ will generate an error because the parser interprets it as the start of a character entity.

**CDATA**

You can use a CDATA section within Hotspot XML elements to tell the XML parser to ignore the content in that section. A CDATA section starts with “<![CDATA[“ and ends with ”]]>“. Here is an example of using CDATA to embed HTML inside your Hotspot XML:
Note in the example above that the CDATA section immediately follows <text> without a line break. With a line break, some browsers will not recognize the data, so code as shown to be safe.

**Html Encoding**

If you are generating XML programmatically from a server script, you can use CDATA or you can encode the HTML that appears within Hotspot XML elements. Encoding converts a character like `<` to the character entity `&lt;`. You can encode in ASP.NET using the `HttpUtility.HtmlEncode` method. In PHP you can do it using the `htmlspecialchars` function. For very small amounts of hand-written XML you can encode by hand by typing `&lt;`, `&gt;`, and `&amp;` instead of `<`, `>`, and `&` respectively.

If your Live Data hotspots are not displaying the content from your XML, verify that the XML is valid. An invalid character or a tag that is not closed properly will prevent the data from being displayed. An easy way to validate your XML is to put it in a file and then open the file in a browser. Most browsers will point out an XML error to you.

### 8.3 Hotspot XML elements

The table below describes the elements you can use in the Hotspot XML that your server script returns.

<table>
<thead>
<tr>
<th>XML Element</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;text&gt;</code></td>
<td>Use the <code>&lt;text&gt;</code> element to set the content of a hotspot’s text area. Its value can be unformatted text or HTML. Be sure that HTML is in a CDATA section or is encoded so that HTML characters like <code>&lt;</code> appear as <code>&amp;lt;</code>.</td>
</tr>
<tr>
<td></td>
<td>Examples:</td>
</tr>
<tr>
<td></td>
<td>&lt;text&gt;Welcome to Chicago.&lt;/text&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;text&gt;&lt;![CDATA[&lt;img src='/images.Chicago.jpg' /&amp;gt]]&gt;&lt;/text&gt;</td>
</tr>
<tr>
<td>Tag</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>&lt;previewImage&gt;</td>
<td>Use the <code>&lt;previewImage&gt;</code> element if your tour uses a directory and you want preview text to appear when the mouse moves over an item in the directory. When Live Data is not used for a hotspot, MapsAlive uses the hotspot’s text for the preview text; however, when Live Data is used, you have to provide the preview text. It can be plain text or HTML. Example: <code>&lt;previewText&gt;The windy city&lt;/previewText&gt;</code></td>
</tr>
<tr>
<td>&lt;previewText&gt;</td>
<td>Use the <code>&lt;previewText&gt;</code> element if your tour uses a directory and you want preview text to appear when the mouse moves over an item in the directory. When Live Data is not used for a hotspot, MapsAlive uses the hotspot’s text for the preview text; however, when Live Data is used, you have to provide the preview text. It can be plain text or HTML. Example: <code>&lt;previewText&gt;The windy city&lt;/previewText&gt;</code></td>
</tr>
<tr>
<td>&lt;serverError&gt;</td>
<td>Use the <code>&lt;serverError&gt;</code> element if you detect an error on your server while responding to a LiveData request. Use a plain text error message as the element’s value. When the XML contains a <code>&lt;serverError&gt;</code> element, MapsAlive will ignore any other elements and display a Live Data error which contains your server error message. Example: <code>&lt;serverError&gt;No data found.&lt;/serverError&gt;</code></td>
</tr>
</tbody>
</table>

**Note when using `<previewText>`**: The preview text is requested from the server when you mouse over an item in the directory and when you mouse over the corresponding hotspot’s marker on the map. If you use the directory’s search feature, only text that has already been requested will be searched. As such, search may not be very useful. Because of this, we recommend that you don’t show the search box in the directory when using Live Data.

### 8.4 Compatibility with previous versions of MapsAlive

**Hotpot XML**

In previous versions of MapsAlive, Live Data requests expected to receive XML from your server that was referred to as Slide XML. The format of Hotspot XML is identical to Slide XML except that the `<hotspot>` element was called `<slide>`. So, in previous versions your server might have returned XML like this:

```
<slide>
  <text>plain text or HTML</text>
  <previewImage>an image URL</previewImage>
  <previewText>plain text or HTML</previewText>
  <serverError>an error message</serverError>
</slide>
```

The new and preferred format is:

```
<hotspot>
  <text>plain text or HTML</text>
  <previewImage>an image URL</previewImage>
  <previewText>plain text or HTML</previewText>
  <serverError>an error message</serverError>
</hotspot>
```
For compatibility, MapsAlive will recognize either the <hotspot> element or the <slide> element.

Function  maLiveDataSendRequest
In earlier versions of Live Data, you requested XML using a function named maLiveDataSendRequest. The equivalent function is now called maLiveDataRequestHotspotXml, but the old name is still supported to prevent existing tours from breaking. Note also that the old function used the POST method when making an HTTP request. The new method uses GET instead. If for some reason your application requires POST, you should continue to use the old function name, but also please contact support@mapsalive.com so that we understand your need. If you don’t know what POST and GET mean, this change probably won’t affect you.

9  Putting it all together
This section outlines the steps for creating an interactive map that uses Live Data. Each will be explained in detail in the sections that follow.

- Create and publish a tour with hotspots that have the Live Data option checked (section 9.1).
- Create a web page that contains the tour and its messenger function (section 9.2).
- Create a server script or data file that the tour can use to get its Live Data (section 9.3).
- Put the web page and the script or data file on your server (section 9.4).
- Open the web page in a browser to test that Live Data is working (section 9.5).

9.1  Create and publish a Live Data tour
You can make any hotspot use Live Data by choosing Hotspot > Advanced Hotspot Options in the menu and checking the Use Live Data box. Then type a messenger function in the text box that appears. See section 4.1 for more information.

Note that if you click Tour Preview and mouse over the marker for a Live Data hotspot, you’ll see a message indicating that Live Data does not work in Tour Preview. This is normal.

9.2  Create a web page that contains your Live Data tour
A tour that uses Live Data must be contained in a web page as was explained in section 7. To create that web page, follow these steps:

1. In Tour Preview, click Show Code Snippets.
2. The HTML you need appears as shown in Figure 12.
3. Open a new file using Notepad or other text editor.
4. Copy the HTML from the snippet and paste it into your editor.
5. Locate the messenger function in the HTML and edit its name and the parameters for the Live Data request function as necessary for your server script or data file. See section 5.3 for examples.

9.3 Create a server script or data file to return Live Data
If you Live Data will be coming from an HTML or XML file, all you need to do is create the file. If you will be creating a server script you can use the code shown in Figure 9 and Figure 10 as an example. Test your server script by running it directly from a browser. Type its name and parameters in the browser’s address area and make sure that the script returns the XML or HTML that you expect.

Note that using a data file is a quick and easy way to get started. Once you have that working, you can replace the data file with a server script.

9.4 Put the web page and script or data files on your server
Now that you have a web page and a script or data file, you need to put them on your web server in the same domain as explained in section 3.3 above. If you don’t know how to copy files to your server, you’ll need to talk to your web administrator.

9.5 Test your Live Data web page in browser
Open a browser and type the URL for the web page on your server that contains your Live Data tour. When you mouse over the markers for the Live Data hotspots you should see the text or HTML that your server returned appearing in the hotspot’s text area. If MapsAlive detects an error while trying to communicate with your server, it will display a message in the text area.

If you are not seeing Live Data, here are some things to check:

1. Be sure that the layout has a text area. If there is no text area, there will be no place for Live Data to appear. These layouts don’t have a text area: 25, 26, 31, 32, 37, and 38.
2. Be sure that both your web page and your script or data file are on the same server in the same domain as explained in section 3.3. The tour itself can reside on any server.
3. Make sure the hotspots have their “Use Live Data” option checked and that their messenger function is specified to correctly call your server script.
4. Test that your server script is returning Hotspot XML by running the script in a browser.
5. Verify that the XML you are returning is well formed – if it contains even one error, Live Data won’t display.

10 Demo files
The previous section took you through the process of creating a simple tour that uses Live Data. In the real world, your tour, web page, and server scripts would be more substantial. In this section we’ll
This section explains how to use the Hotspot XML elements <previewImage> and <previewText>. You can read about these elements in section 8.
The hotspot content shown above was created in demo1 by returning HTML from the server in the <text> element. The HTML is shown below, but you can also see it by running the server script directly:

```
http://livedata.mapsalive.com/demo/demoservice.asmx/GetLocation?city=newyork
```

The decoded XML for the hotspot in Figure 14 is shown in Figure 15 below. Notice that some of the HTML tags have class attributes. The complete styling of the hotspot content in Figure 14 depends on these attributes combined with use of a CSS style sheet that will be discussed in section 10.3.

```
<?xml version="1.0" encoding="utf-8" ?>
<hotspot>
  <text>
    <div class="slideContentArea">
      <div class="slideTitle">
        Empire State Building<br>350 5th Ave.<br>
        New York, NY 10018
      </div>
      <img src="images/location-newyork.jpg" style="height:80px;">
      <table class="employeeTable">
        <tr>
          <td><img src="images/employee-newyork.jpg" style="height:40px;"></td>
          <td>Billy François Cordan<br>Regional Manager</td>
        </tr>
      </table>
      <a href="mailto:bordan@livedatacorp.com">bordan@livedatacorp.com</a>
    </div>
  </text>
</hotspot>
```

**Figure 15 – XML with decoded HTML for the Location hotspot**

**Note:** When using the <text> element for all hotspot data, your tour should use a content layout that does not have an image area. The text-only layout for popups is #39. For a tiled layout it’s #29.
10.3 Styling live data with CSS
Web developers tend to rely heavily on CSS and so does MapsAlive. When you publish a tour, MapsAlive automatically constructs a CSS style sheet for each of its pages. You can see the MapsAlive CSS file being referenced in the web page file shown in section 7. The actual line of code is:

```
<link rel="stylesheet" type="text/css" href="http://tour.mapsalive.com/1234/page1.css" />
```

You can create your own CSS file and put it in the same folder as your web page. You can use your CSS by adding another `<link>` tag (right after the tag shown above) like this:

```
<link rel="stylesheet" type="text/css" href="mycustom.css" />
```

The hotspot content shown earlier in Figure 14 demonstrates the effectiveness of this technique. The HTML for the hotspot (see Figure 15) contains a `<table>` tag with a class attribute named `employeeTable`. The definition of that class is located in the file `demo1.css`. It looks like this:

```
.employeeTable
{
    margin: 12px auto 4px auto;
    font-weight: bold;
    font-style: italic;
    border: solid 2px white;
    text-align: left;
    background-color: #b9090b;
    color: white;
}
```

Of course the demo1 server script could have styled the table HTML in-line and emitted it as part of the `<text>` element, but that would have been more work. Even worse, it would mean that a programmer would need to change the server code whenever style changes were required. By using CSS, the HTML that you generate within your Hotspot XML can be kept fairly simple. It only needs to provide overall structure. This simplicity means less work for programmers who are doing the server-side work to generate the XML and it gives more control to graphic designers by letting them work with CSS.

11 Customizing the appearance of server errors
This section explains how you can customize the appearance of error messages that can appear in a tour if an error occurs while Live Data is trying to communicate with your server.

11.1 Default Live Data error message
If an error occurs while trying to obtain Live Data from your server, the tour will display a message instead of the hotspot content. By default, the message looks similar to this:
11.2 Custom Live Data error message
You can customize the error to change its text and background colors and provide your own error message at the top. Below is an example of a custom error:

![Custom Live Data error message example]

You can also choose to hide the details of the error so that the user does not see them:

![Custom Live Data error message example without details]

11.3 Enabling custom error messages
To enable custom error message, the web page that contains your tour and messenger function, must call a MapsAlive JavaScript function called `maLiveDataSetCustomError` (explained in the next section). You can put the call in the same section of your page as your messenger function. Here is an example.

```html
<!-- Your Live Data messenger function(s) go here -->
function getCityData(arg1)
{
    maLiveDataRequestHotspotXml(5, 'xdemo1Service.php', 'city', arg1);
}
var errorMessage = "<div style='font-size:14px;color:red;font-weight:bold;'>" +
   "An unexpected server error occurred.</div>" +
   "Please report to " +
   "<a href='mailto:support@yourcompany.com'>support</a> " +
   "or call 800-555-5555.";

maLiveDataSetCustomError(errorMessage, "black", "#FAFAD2", true);
</script>

Figure 19 – Example of a call to maLiveDataSetCustomError

The example above is the code that was used to create the message displayed in Figure 17 above. The error without details shown in Figure 18 was created with the same code except that the last parameter to maLiveDataSetCustomError was false instead of true.

11.4 The maLiveDataSetCustomError function

To customize the error message, the web page that embeds your MapsAlive tour can call a livedata.js function named maLiveDataSetCustomError. Its signature is:

    maLiveDataSetCustomError (message, color, backgroundColor, showDetails)

The parameters for maLiveDataSetCustomError are described in the table below:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>message</td>
<td>Plain text or HTML that will be displayed at the top of the error.</td>
</tr>
<tr>
<td>color</td>
<td>The color parameter lets you set the error’s text color. You can specify a color name such as “blue” or a hex value like “#0000ff”.</td>
</tr>
<tr>
<td>backgroundColor</td>
<td>The backgroundColor parameter lets you set the error’s background color. You can specify a color name such as “blue” or a hex value like “#0000ff”.</td>
</tr>
<tr>
<td>showDetails</td>
<td>By default, the error shows detailed information about what went wrong. If you don’t want your users to see this information, pass false as the showDetails parameter. To show the details, pass true. Even if you don’t want users to see details, you might want to show them while developing your Live Data tour to help you quickly identify the cause of errors.</td>
</tr>
</tbody>
</table>

11.5 Reasons that a server error could be reported

A server error can occur for one of the following reasons:

- An error response was returned from the server such as 404 (url not found). This can happen when the url passed to maLiveDataRequestHotspotXml is not specified correctly. An example of a 404 error being reported is shown in Figure 16. It occurred because demo2Service.php was
mistyped as xdemo2Service.php.

- The XML returned from the server by your script contained a `<serverError>` element. This will only occur if your server-side logic detects an error and deliberately reports it in the XML it returns to your tour.

### 11.6 Using Live Data in Tour Preview

Note that Live Data does not work when viewing a tour in the MapsAlive Tour Preview. When you preview Live Data hotspots, a message appears similar to the one shown in the figure below. This is normal and can’t be helped since MapsAlive Tour Preview cannot make calls to your server due to internet rules for AJAX calls from one domain (www.mapsalive.com) to another (your server’s domain). See section 3.3 Security restrictions on file locations for more information about these rules.

The message you see in Tour Preview looks like the one shown below in Figure 20. Note that if you use `mALiveDataSetCustomError` to customize your error message, the customization will not appear in Tour Preview.

![Figure 20](image)

**Figure 20 – Normal message when viewing a Live Data hotspot in Tour Preview**

### 12 Requesting Other Information from Your Server

In addition to having your hotspot content come from your server, you may want to have your interactive map communicate with your server for other purposes. The Live Data feature lets you make requests to your server for either XML or HTML and provides an easy way for you to get back the data that your server sends in response.

As an example, suppose you have a map showing cubicles in an office layout and you use a color coding scheme to indicate which ones are vacant and which ones are occupied by different departments such as marketing or human resources. You can set things up so that the color used for each cube is set based on occupancy data on your server. Whenever the map loads, it can make a request to your server to get the colors for each office and then apply them to the hotspot marker for each office. This way you never need to update the map itself when office occupants change – the map keeps itself up to date automatically.

You could even have your map call your server on a regular basis to check the status of objects depicted on the map. For example, suppose the map displays sensors for temperature. The map can periodically
query the server – say every five seconds – to ask if any sensor has a reading above a safe threshold. If one does, the logic can change the color of that sensor’s marker or even make the marker blink.

The following sections explain how you request and get back data from your server.

12.1 Server request functions

The MapsAlive Live Data library supports two functions that you can call to ask your server for information and receive back a response. They are `maLiveDataRequestXml` and `maLiveDataRequestHtml`.

The signature for `maLiveDataRequestXml` is:

```
maLiveDataRequestXml(requestId, url, parameter-list)
```

The signature for `maLiveDataRequestHtml` is:

```
maLiveDataRequestHtml(requestId, url, parameter-list)
```

The parameters are the same for both functions and they are described in the table below:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>requestId</td>
<td>A quoted string that you provide to identify the request so that you’ll know what the data is when your server sends it back to your map. The requestId can be anything you like and if you don’t have a use for it, just pass &quot;&quot; (an empty string).</td>
</tr>
<tr>
<td>url</td>
<td>The url specifies how to call your server script that returns Live Data. Since the web page that is making this call must be on the same server as the script, the url can be a relative reference, that is, it does not need to specify the full path to the script. For example it could be just <code>getEmployee.php</code> instead of <code>http://www.myserver.com/getEmployee.php</code>. This parameter works the same way it does for the <code>maLiveDataRequestHotspotXml</code> and <code>maLiveDataRequestHotpotHtml</code> functions.</td>
</tr>
<tr>
<td>parameter-list</td>
<td>The parameter-list is a variable length list of name/value pairs separated by commas. They are used to tell your server script what data is being requested. This parameter works the same way it does for the <code>maLiveDataRequestHotspotXml</code> and <code>maLiveDataRequestHotpotHtml</code> functions.</td>
</tr>
</tbody>
</table>

Examples

Below is a server request function that calls `maLiveDataRequestHotpotXml` specifying "cubes" as the request Id, "webservice.asmx/GetCubicleStatus" as the url, and no parameters.

```
maLiveDataRequestXml("cubes", "webservice.asmx/GetCubicleStatus");
```

For other examples that will help you understand how to call your server, see section 5.3 above.
In order to use the server request functions, your map must include the livedata.js library. The library is included automatically whenever at least one hotspot on the map uses Live Data. If you want to use the server request functions, but don’t need Live Data for any of your hotspots, simply create a dummy hotspot that uses Live Data, but don’t put the hotspot on your map.

12.2 When and where to use the server request functions
You can call either of the server request functions whenever and however you like, but this section present three common scenarios. The examples shown probably won’t make much sense until the next section where we explain how you get the data that comes back from the server. What you need to understand right now is that server requests are asynchronous. That means that you make the request, but you don’t get the data back right away. Instead, you get notified when it has arrived. In practice, assuming a fast internet connection, the data comes back instantly, but it is not returned by the request function – you have to get it using a different function as will be explained in section 12.3.

An asynchronous request is like ordering a meal at a take-out restaurant. You tell someone what you want (make a request) and that person gives you an order number (a request identifier), but you don’t get your food immediately – you have to wait. While you are waiting you can do other things like read a newspaper or make a phone call. When your food is ready, you are notified by hearing your order number on a loud speaker and you take your food. In contrast, a synchronous request would be like buying a sandwich from a vending machine. You put your money in, press a button, and the food comes out and you take it – you don’t do anything else between the time you press the button and the time you get your sandwich. The thing you made the request to (the machine) is the same thing that provides the response.

Don’t worry if asynchronous requests seem confusing. For now, look at these common scenarios for requesting data from your server and in section 12.3 will explain how you actually get the data.

Request data when the map first loads
You can use the maOnMapLoaded handler of the MapsAlive JavaScript API to be notified when your map has loaded and is ready to use (see the MapsAlive User Guide for the JavaScript API for information about this handler). That’s a good time to request data that you might need to alter the appearance of some of the map’s markers. Here is an example;

```javascript
function maOnMapLoaded()
{
    maLiveDataRequestXml("cubes", "webservice.asmx/GetCubicleStatus");
}
```

Request data from a hotspot’s click, mouseover, or mouseout handler
You can use a hotspot’s action handler as a way to trigger a call to your server to request data. You could, for example, create a button hotspot that you can click when you want to call your server to get XML or HTML. You set click, mouseover, and mouseout handlers on the Hotspot Actions screen.
A hotspot handler can call one of the server request functions directly, or it could call one of your functions which in turn calls the server request function. In general, it’s a good idea to wrap server request functions in your own function so that the server-specific information like its URL and parameters are isolated to just one place. That way if something changes – like the server URL – you only have to edit that function instead of all the hotspot handlers that call it.

**Request data at a regular interval using a timer**

If you need to get data from your server at regular intervals, you can call the JavaScript `setInterval` function (refer to your favorite JavaScript book to learn about it). Here is an example of `setInterval` being called when the map loads. The interval is set for 2 seconds (2,000 milliseconds) which causes a function named `getCubeInfo` to get called at that frequency. Each time `getHotspotStates` is called, it makes a request to your server for data. Note that `getCubeInfo` is a function that you write – you can call it anything you like – whereas `maOnMapLoaded` is part of the MapsAlive JavaScript API. If you want to use it you must name it exactly as shown.

```javascript
function getCubeInfo
{
    maLiveDataRequestXml("cubes", "webservice.asmx/GetCubicleStatus");
}

function maOnMapLoaded()
{
    setInterval(getCubeInfo, 2000);
}
```

12.3 **Getting the data that your server sends back**

The previous section told you how to request data from your server. This section explains how you actually get the data. Recall from section 12.2 that requests are asynchronous and that when the data is returned from the server you get notified. The way you get notified is by providing a handler named `maOnLiveDataResponse`. When the data comes back from the server, MapsAlive calls the handler and passes the data to it. This is analogous to your meal being ready at the take-out window and your order number being announced on the loud speaker.

The signature for `maOnLiveDataResponse` is:

```
maOnLiveDataResponse(requestId, dataId, data, xml)
```

The parameters for `maOnLiveDataResponse` are described in the table below:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>requestId</td>
<td>A string that you provided when you made the request. If your code only makes one kind of request, you won’t need to examine this value, but if you are making different requests for different kinds of data, you can use the value to determine which kind of request the data is for. For example, one request might be to get information about hotspots and another might be to...</td>
</tr>
</tbody>
</table>
get a message that will be displayed to the user.

| dataId              | A string that identifies the data that was returned from the server. If the value is the integer zero, a server error occurred while retrieving this data. Note that the proper way to test for zero is to use the Identity operator like this:
|                    | `if (dataId === 0)`
|                    | If you use the Equality operator (==) instead, you won’t be able to distinguish between a string "0" and the integer 0.
|                    | If your code only requests one data item at a time, you might not need to use this value, but if you ask for multiple data items in the same request, the dataId will tell you which data item is being returned on this call to `maOnLiveDataResponse`. For example, you might request that the server return the temperature of every temperature sensor hotspot that is over 100 degrees. If the response from the server included three hotspots, the `maOnLiveDataResponse` handler would get called once for each, and the dataId would be the hotspot Id for one of the sensors. The temperature would be in the data parameter.

| data                | This is the actual data returned for the request. If the request was for HTML, this will be the HTML. If the request was for XML, this will be the text content of the <data> element (see section 12.4).

| xml                 | If the request was for HTML, this value will be null. If the request was for XML, this will be the XML for the <data> element (see section 12.4). If you are requesting XML, or if you are requesting XML and only want the text content from the data parameter, you can omit xml from the function signature.

|                    | If you want to use data returned as XML, it’s up to you to write the JavaScript code that can parse XML and obtain the values from its elements and attributes. See your JavaScript documentation to learn how.

### 12.4 XML Format

You can skip this section if you are only requesting data in HTML format.

When you request XML from your server, your server must return the data in a format that MapsAlive recognizes. The format is very simple – the XML must contain zero or more <data> elements. Here are some examples:

**Example of five <data> elements**

In this example there is an outer tag named <hotspots> but it could be called anything you like. MapsAlive only looks for <data> elements with a single attribute named “id” which is optional.

```
<hotspots>
  <data id="H1">2</data>
  <data id="H2">1</data>
```
Example of one <data> element
This is a message from your server</data>

Example of one <data> element containing nested XML
In this example the <data> element has child elements.

<first>George</first>
<last>Smith</last>
<title>President</title>

12.5 Example
This last section provides an example of using the maOnLiveDataResponse function to get data from your server. The example assumes that two different kinds of server requests are made. One request is to get text to be displayed as the tour’s title. The other request is to get a list of hotspots that need to have their color changed based on color codes provided by the server.

Note that the same handler is used for both kinds of requests because there is only ever one maOnLiveDataResponse handler, but it gets called each time data is returned from the server. If the server returns multiple items for the same request, the handler will get called once for each item.

function maOnLiveDataResponse(requestId, dataId, data)
{
    if (dataId === 0)
    {
        // The server responded with an error.
        return;
    }

    if (requestId == "title")
    {
        // The request was to get information to display as the tour title.
        mapsalive.setTourTitle(data);
        return;
    }

    // There was no error and the request was not for the title so assume
    // it was to set the color of a hotspot. The data parameter contains a code
    // to indicate which color to use. The value of dataId is the hotspotId
    // of the marker that needs its color changed.
    var color;

    switch (data)
    {

case "1": color = 0xff0000; break;
case "2": color = 0x00ff00; break;
case "3": color = 0x0000ff; break;
case "4": color = 0x000000; break;
case "5": color = 0xffffff; break;
case "6": color = 0x770000; break;
case "7": color = 0x007700; break;
case "9": color = 0x000077; break;
default: color = 0x777777; break;

// Change fill color of the marker identified by dataId.
mapsalive.changeMarkerNormalShapeAppearance(dataId, 0x00CC00, 100, color, 100, "");

Figure 21 – JavaScript to handle the response from a server request