MapsAlive® User Guide

JavaScript API

Last updated March 22, 2016

Table of Contents

JavaScript API

1 Introduction

What is this guide about?
This guide explains how to use the JavaScript API that allows a web page to communicate with an interactive map and vice versa. API stands for Application Programming Interface.

Who this guide is for
This guide is for web developers who are experienced with the JavaScript programming language.

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 Optional Parameters

Some API functions have optional parameters. You can either omit all of the optional parameters, or you can emit one or more from the end. For example, if a function takes one required parameter P1 and three optional parameters P2, P3, and P4, you can provide:

P1, P2, P3, P4
P1, P2, P3
P1, P2
P1

3 Special Parameters

Some API functions take parameters that specify values for colors, opacity, and special effects.

HotspotId
Many API functions take a single hotspot Id or a list of hotspot Ids as a parameter. The Id is a string enclosed in quote. Hotspot Ids are case-insensitive.

Colors
You specify colors using numeric values in the hexadecimal range `0x000000` to `0xffffff`. For example, in the API call below, the values `0x00CC00` and `0xdd1d38` are used to pass line and fill colors to the function.

```
mapsalive.setMarkerShapeAppearance(
       "gallery1, gallery7, gallery19", 0x00CC00, 100, 0xdd1d38, 30, "");
```

Opacity
You specify opacity with an integer value between `0` and `100` that represents a percentage. Zero means invisible (completely transparent) and `100` means solid (completely opaque).

Special Effects
Some API functions take an effects parameter. Effects are described in section 5.

4 API Functions

The API functions are JavaScript functions that a web page can call. The functions are listed in the sections below in alphabetical order.

4.1 changeMarkerNormalShapeAppearance

The changeMarkerNormalShapeAppearance function alters the normal appearance of the shape of one or more markers. It ignores markers that do not have a shape. The function allows you to change the color and opacity of the shape's line and interior fill color or change effects, but it cannot change the shape itself or the thickness of its line.

The change stays in effect until you either call the function again with new parameters or call the restoreMarkerNormalShapeAppearance function to restore the default appearance.

Signature:

```
mapsalive.changeMarkerNormalShapeAppearance(
        hotspotIdList, lineColor, lineAlpha, fillColor, fillAlpha, effects);
```

Example:

```
mapsalive.changeMarkerNormalShapeAppearance (
        "gallery1, gallery7, gallery19", 0x00CC00, 100, 0xdd1d38, 30,"");
```

Parameters:

Parameter	**Description**
`hotspotIdList`	A comma separated list of one or more hotspot Ids. The shape appearance of the marker for each hotspot identified in the list will be altered by the function call. You can also specify "*" to mean all hotspots on the map.
`lineColor`	The color to use for the shape's line.
`lineAlpha`	The opacity to use for the shape's line.
`fillColor`	The color to use for the shape's interior.
`fillAlpha`	The opacity to use for the shape's interior.
`effects`	The effects to use on the shape. For no effects, provide an empty string "".

4.2 changeMarkerSelectedShapeAppearance

The changeMarkerSelectedShapeAppearance function alters the selected appearance of the shape of one or more markers. It ignores markers that do not have a shape. The function allows you to change the color and opacity of the shape's line and interior fill color or change effects, but it cannot change the shape itself or the thickness of its line.

The change stays in effect until you either call the function again with new parameters or call the restoreMarkerSelectedShapeAppearance function to restore the default appearance.

Signature:

```
        mapsalive.changeMarkerSelectedShapeAppearance(
                hotspotIdList, lineColor, lineAlpha, fillColor, fillAlpha, effects);
```

Example:

```
        mapsalive. changeMarkerSelectedShapeAppearance (
                "gallery1, gallery7, gallery19", 0x00CC00, 100, 0xdd1d38, 30,"");
```

Parameters:

Parameter	**Description**
`hotspotIdList`	A comma separated list of one or more hotspot Ids. The shape appearance of the marker for each hotspot identified in the list will be altered by the function call. You can also specify "*" to mean all hotspots on the map.
`lineColor`	The color to use for the shape's line.
`lineAlpha`	The opacity to use for the shape's line.
`fillColor`	The color to use for the shape's interior.
`fillAlpha`	The opacity to use for the shape's interior.
`effects`	The effects to use on the shape. For no effects, provide an empty string "".

4.3 closePopup

The closePopup function closes a pinned popup as though you had clicked its close X. If there is no pinned popup, this function has no effect.

Signature:

```
        mapsalive.closePopup();
```

Example:

```
        mapsalive.closePopup();
```

4.4 drawRoute

The drawRoute function draws a line through a set of hotspot markers. The route is drawn through the center point of each marker. For detailed information about drawing routes, see the *MapsAlive User Guide for Drawing Routes*.

Signature:

```
mapsalive.drawRoute
   (hotspotId, route, lineWidth, lineColor, lineApha, effects);
```

Examples:

```
mapsalive.drawRoute("Route", "H1,H2,H3", 3, 0x00CC00, 50, "Shadow");

mapsalive.drawRoute("Route", "R1", 3, 0x00CC00, 50, "Shadow");
```

Parameters:

Parameter	**Description**
`hotspotId`	A hotspot Id that identifies the route marker to use to draw the route. The hotspot must have its Is Route option checked on the Advanced Hotspot Options screen. If the marker was previously used to draw a different route, the call will erase the prior route before drawing the new route.
`route`	The route parameter must be one of the following: - The name of a route that was imported into MapsAlive from an Excel file. The name must be enclosed in quotes. - A comma-separated list of two or more hotspot Ids enclosed in quotes. The center point of the marker for each hotspot identified in the list will be used as a waypoint along the route. To create a gap in the route, use a semicolon instead of a comma. The semicolon tells the route drawing logic to "pick up the pen" after drawing to the waypoint that precedes the semicolon and to put it down again at the waypoint following the semicolon.
`lineWidth`	*Optional.* The width of the route's line in pixels. Default is 3.
`lineColor`	*Optional*. The color to use for the line. Default is `0xff0000`.
`lineAlpha`	*Optional*. The opacity to use for the line. Default is 100.
`effects`	*Optional*. The effects to use on the line. Default is "shadow". For no effects, provide an empty string "". To learn about effects see section 5 Marker Shape Effects.

4.5 flushLiveDataCache

The flushLiveDataCache function flushes the cached data for every hotspot that gets its content from LiveData. Use this function to cause the cache period for every hotspot to expire so that the next time the mouse moves over a hotpot, new data will be requested from the server. This works even if the cache period is set to zero.

Signature:

```
mapsalive.flushLiveDataCache();
```

Example:

```
mapsalive.flushLiveDataCache();
```

This flushLiveDataCache is useful if your map has options that allow different kinds of data to be returned for the same hotspot. For example, on a weather map, mousing over a hotspot might return the forecast showing temperatures in Fahrenheit. Since weather doesn't change frequently, you might use a long cache period. However, if you provide an option that allows the temperature to be shown in Celsius and the user chooses the option, you could call this function to flush the Fahrenheit data so that the server will get called to retrieve Celsius data the next time the mouse moves over a hotspot.

For more information about Live Data see the *MapsAlive User Guide for Live Data.*

4.6 getCurrentHotspot

The getCurrentHotspot function returns an object containing properties for the current hotspot. The current hotspot is the one whose content is showing. If the map uses popups, this function returns the last hotspot that content was shown for. It returns null if no hotspot content has been shown yet as would be the case when a map using popups first loads.

Signature:

```
mapsalive.getCurrentHotspot();
```

Example:

```
var hotspotId = mapsalive.getCurrentHotspot().id;
```

Properties:

Property	**Description**
`id`	The value of the **Hotspot Id** field on the **Edit Hotspot Content** screen.
`title`	The value of the **Title** field on the **Edit Hotspot Content** screen.
`htmlText`	The content from the **Text** field on the **Edit Hotspot Content** screen including HTML tags or an empty string if there is no text content.
`plainText`	The content from the **Text** field on the **Edit Hotspot Content** screen excluding HTML tags or an empty string if there is no text content.
`multimediaText`	The content from the **Multimedia** field on the **Edit Hotspot Content** screen or an empty string if there is no multimedia content.
`imageSrc`	The URL of the photo on the **Edit Hotspot Content** screen or an empty string if there is no photo.

4.7 getCurrentPage

The getCurrentPage function returns an object containing properties for the map, gallery, or data sheet that the tour is currently displaying.

Signature:

```
mapsalive.getCurrentPage();
```

Example:

```
var id = mapsalive.getCurrentPage().id;
```

Properties:

Property	Description
`id`	If the page is a map, the value comes from the **Map Id** field on the **Advanced Map Options** screen. If the page is a gallery, the value comes from the **Gallery Id** field on the **Advanced Gallery Options** screen. If the page is a data sheet, the value comes from the **Data Sheet Id** field on the **Advanced Data Sheet Options** screen.

4.8 getHotspotIdsForCategory

The getHotspotIdsForCategory function returns a list of the hotspot Ids that belong to one or more categories. If more than one category is specified, the function can return either the union (logical OR) or the intersection (logical AND) of the hotspots that belong to those categories. In other words, you can request that it return all hotspot Ids that have any of the categories or to return only the Ids of hotspots that have all of the categories.

Signature:

```
mapsalive.getHotspotIdsForCategory(categoryCodeList, and);
```

Example:

```
var homes = mapsalive.getHotspotIdsForCategory("house, condo");
var palaces = mapsalive.getHotspotIdsForCategory("house, overTenMillion", true);
```

Parameters:

Parameter	Description
`categoryCodeList`	A comma separated list of one or more category codes. If the list contains only an empty string and the value of the *and* parameter is true, the function returns Ids for every hotspot on the map. If the *and* parameter is false, the function returns an empty string. It works this way because when the *and*

	parameter is true, the category code list acts as a filter to restrict what the function returns. When there is no filter, everything gets returned. When the *and* parameter is false, the category code list controls what categories are included in the return value. When the list is empty, nothing gets returned.
and	If omitted or false, the function returns Ids for all hotspots that belong to any of the categories in categoryCodeList. For example, any hotspot that belongs to either the home or condo category. If true, only returns Ids for hotspots that belong to every category in categoryCodeList. For example, only hotspots that belong to both the home category and the overTenMillion category.

4.9 getQueryStringArg

The getQueryStringArg function returns the value of a browser query string parameter. If the parameter specified by the arg parameter is not on the query string, the function returns an empty string.

Signature:

```
mapsalive.getQueryStringArg(arg);
```

Example:

```
var zipcode = mapsalive.getQueryStringArg("zip");
```

Parameters:

Parameter	**Description**
arg	The name of an argument on the browser query string.

4.10 maOnDirectoryEntryClick

This is not an API function per se because you don't call it. However, if your JavaScript has a function with this name, it will get called whenever a hotspot title is clicked in the directory.

The HotspotId for the clicked directory entry is passed as a parameter to the function. If you need to get additional information about the hotspot, you can call getCurrentHotspot.

Example:

```
function maOnDirectoryEntryClick(hotspotId)
{
        // Your code goes here
}
```

4.11 maOnHotspotChanged

This is not an API function per se because you don't call it. However, if your JavaScript has a function with this name, it will get called whenever the current hotspot changes. The current hotspot usually changes when the user moves their mouse over its marker. It can also change when the hotspot is clicked or selected from the directory. Use this function if you need to perform an action every time the current hotspot changes.

The HotspotId for the new hotspot is passed as a parameter to the function. If you need to get additional information about the hotspot, you can call getCurrentHotspot.

If the value of hotspotId is zero, no hotspot is selected. This happens when popups are being used and the mouse moves off of a hotspot. It also happens when a pinned popup is closed.

Example:

```
function maOnHotspotChanged(hotspotId)
{
	// Your code goes here
}
```

4.12 maOnMapLoaded

This is not an API function per se because you don't call it. However, if your JavaScript has a function with this name, it will get called when the map has loaded and is ready to communicate with.

Normally a map loads so quickly that you don't need to code this function; however, if your map has a large number of hotspots, you may need to wait for the map to finish loading before you begin using other API functions. Attempts to communicate with the map via the API before the map has loaded will have no effect and may trigger a JavaScript error.

Example:

```
function maOnMapLoaded()
{
	// Your code goes here
}
```

4.13 maOnPopupClosed

This is not an API function per se because you don't call it. However, if your JavaScript has a function with this name, it will get called whenever a pinned popup is closed. A pinned popup closes when you click its X or its pin icon, when the map is panned or zoomed, and when the directory is displayed. If a pinned popup is showing and you click on another marker that displays a pinned popup, the popup's content changes, but the popup does not close and this function is not called (the maOnHotspotChanged function will be called).

The HotspotId for the closed popup's hotspot is passed as a parameter to the function. If you need to get additional information about the hotspot, you can call getCurrentHotspot.

Example:

```
function maOnPopupClosed(hotspotId)
{
	// Your code goes here
}
```

4.14 maOnSoundManagerReady

This is not an API function per se because you don't call it. However, if your JavaScript has a function with this name, it will get called when the SoundManager library has initialized. If SoundManager is not enabled, this function will never get called. For more information see the SoundManager section of this document.

This function is useful when you want to play a sound when your tour first loads.

Example:

```
function maOnSoundManagerReady()
{
	// Your code goes here
}
```

4.15 mapIsHtml5

The mapIsHtml5 function returns true if the map is being displayed using HTML5 and false if it is being displayed using Flash.

Signature:

```
mapsalive.mapIsHtml5();
```

Example:

```
var isHtml5 = mapsalive.mapIsHtml5();
```

4.16 mapIsTouchDevice

The mapIsTouchDevice function returns true if the map is being displayed using HTML5 and is on a touch device such as an iPad or iPhone and false otherwise. The function always returns false when the map is being displayed using Flash. The function always returns false when the map is being displayed on a desktop computer even if the computer monitor has a touch screen.

Signature:

```
mapsalive mapIsTouchDevice ();
```

Example:

```
var isTouchDevice = mapsalive.mapIsTouchDevice();
```

4.17 playSound

The playSound function plays an audio file that is in MP3 format. If the sound identified by the name parameter is currently playing when this function is called, play will pause. A subsequent call using the same name will cause play to resume. Thus, this function not only causes a sound to start playing, it also acts as a pause and resume toggle. For more information see the SoundManager section of this document. See also `stopSound`.

Signature:

```
mapsalive.playSound(name, url);
```

Example:

```
mapsalive.playSound("Canada", "http://www.mydomain.com/mysounds/OhCanada.mp3");
```

Parameters:

Parameter	Description
`name`	A string that identifies the sound. By giving the sound a name, MapsAlive is able to cache the sound so that it can be replayed more quickly.
`url`	The url of an mp3 audio file that is located on the internet.

4.18 positionMapToShowMarker

The positionMapToShowMarker function pans the map if necessary to ensure that the specified hotspot's marker can be seen. This function only has an effect if the map is zoomed-in and positioned in such a way that the specified marker is outside the part of the map that is visible.

If a popup is open and this function causes the map to pan, the popup will close, the same as it would if you dragged the map with your mouse or if you clicked the pan controls on the map.

If the mouse is over a marker when this function is called, a mouseout event will usually occur because the marker's position will probably change as a result of the map panning. Because of this, use of this function in the JavaScript for a hotspot's Click Action, Mouseover Action, or Mouseout Action can produce unpredictable behavior.

This function has no effect if the map is not zoomable.

Signature:

```
mapsalive.positionMapToShowMarker(hotspotId);
```

Example:

```
mapsalive.positionMapToShowMarker("h1");
```

Parameters:

Parameter	Description
`hotspotId`	A hotspot Id for the marker that is to be made visible.

4.19 restoreMarkerNormalShapeAppearance

The restoreMarkerNormalShapeAppearance function restores the normal appearance of the shape of one or more markers to the appearance as defined in the MapsAlive Tour Builder. Use this function if you have changed the appearance using the changeMarkerNormalShapeAppearance and want to put it back the way it was originally.

Signature:

```
mapsalive.restoreMarkerNormalShapeAppearance(hotspotIdList);
```

Example:

```
mapsalive.restoreMarkerNormalShapeAppearance("gallery1, gallery7, gallery19");
```

Parameters:

Parameter	Description
`hotspotIdList`	A comma separated list of one or more hotspot Ids. The shape appearance of the marker for each hotspot identified in the list will be altered by the function call. You can also specify "*" to mean all hotspots on the map.

4.20 restoreMarkerSelectedShapeAppearance

The restoreMarkerSelectedShapeAppearance function restores the selected appearance of the shape of one or more markers to the appearance as defined in the MapsAlive Tour Builder. Use this function if you have changed the appearance using the changeMarkerSelectedShapeAppearance and want to put it back the way it was originally.

Signature:

```
mapsalive.restoreMarkerSelectedShapeAppearance(hotspotIdList);
```

Example:

```
mapsalive.restoreMarkerSelectedShapeAppearance("gallery1, gallery7, gallery19");
```

Parameters:

Parameter	Description
`hotspotIdList`	A comma separated list of one or more hotspot Ids. The shape appearance of the marker for each hotspot identified in the list will be altered by the function call. You can also specify "*" to mean all hotspots on the map.

4.21 setMapZoomInOut

The setMapZoomInOut function zooms the map in or out by a percentage. The map zoom level will not change if you attempt to zoom in when the map is zoomed in all the way or if you attempt to zoom out when the map is zoomed out all the way.

If a popup is open when this function is called and the function causes the map zoom level to change, the popup will close, the same as it would if you clicked the zoom controls on the map.

If the mouse is over a marker when this function is called, a mouseout event will usually occur because the marker's position will probably change as a result of zooming. Because of this, use of this function in the JavaScript for a hotspot's Click Action, Mouseover Action, or Mouseout Action can produce unpredictable behavior.

This function has no effect if the map is not zoomable.

Signature:

```
	mapsalive.setMapZoomInOut(delta);
```

Example:

```
	mapsalive.setMapZoomInOut(15);    // Zoom in 15%
	mapsAlive.setMapZoomInOut(-10);   // Zoom out 10%
```

Parameters:

Parameter	Description
`delta`	An integer that specifies the amount to zoom in or out. A positive value means zoom in. A negative value means zoom out.

4.22 setMapZoomLevel

The setMapZoomLevel function sets the map's zoom level to a percentage that you specify. If the map is already set to the specified level, the function has no effect.

If a popup is open when this function is called and the function causes the map zoom level to change, the popup will close, the same as it would if you clicked the zoom controls on the map.

If the mouse is over a marker when this function is called, a mouseout event will usually occur because the marker's position will probably change as a result of zooming. Because of this, use of this function in the JavaScript for a hotspot's Click Action, Mouseover Action, or Mouseout Action can produce unpredictable behavior.

This function has no effect if the map is not zoomable.

Signature:

```
mapsalive.setMapZoomLevel(level);
```

Example:

```
mapsalive.setMapZoomLevel(75);    // Zoom the map to 75%
```

Parameters:

Parameter	**Description**
`level`	An integer that specifies the desired zoom level percentage. If the value is larger than 100, the map will be zoomed to 100%. If the value is less than the map's minimum zoom level, the map will be zoomed to its minimum level.

4.23 setMarkerAppearanceNormal

The setMarkerAppearanceNormal function changes the marker's appearance to its normal appearance without actually selecting the marker. To select the marker, use `setMarkerSelected`.

Signature:

```
mapsalive.setMarkerAppearanceNormal(hotspotIdList);
```

Example:

```
mapsalive.setMarkerAppearanceNormal("gallery1");
```

Parameters:

Parameter	**Description**
`hotspotIdList`	A comma separated list of one or more hotspot Ids. The marker for each hotspot identified in the list will have its appearance changed to normal. You can also specify "*" to mean all hotspots on the map.

4.24 setMarkerAppearanceSelected

The setMarkerAppearanceSelected function changes the marker's appearance to its selected appearance without actually selecting the marker. To select the marker, use `setMarkerSelected`.

Use of this function can create unexpected behavior. For example, if you set all markers on a map to appear selected and the user then mouses over one of them, that marker actually becomes selected. If the map uses popups, when the user mouses off of the marker, it will become unselected and revert to its normal appearance. If the map uses a tiled layout, the marker will become unselected and revert to its normal appearance when the user mouses over and selects a different marker.

Signature:

```
mapsalive.setMarkerAppearanceSelected(hotspotIdList);
```

Example:

```
mapsalive.setMarkerAppearanceSelected("gallery1");
```

Parameters:

Parameter	**Description**
`hotspotIdList`	A comma separated list of one or more hotspot Ids. The marker for each hotspot identified in the list will have its appearance changed to selected. You can also specify "*" to mean all hotspots on the map.

4.25 setMarkerBlink

The setMarkerBlink function makes one or more markers blink a specified number of times. The function can also be used to make one or more markers stop blinking.

Signature:

```
mapsalive.setMarkerBlink(hotspotIdList, blinkCount);
```

Example:

```
mapsalive.setMarkerBlink("h1, h2", 10);
```

Parameters:

Parameter	**Description**
`hotspotIdList`	A comma separated list of one or more hotspot Ids. The marker for each hotspot identified in the list will start or stop blinking.
`blinkCount`	Specifies the number of times the markers should blink. The value zero indicates that blinking should stop.

4.26 setMarkerDisabled

The setMarkerDisabled function disables or enables one or more markers. When a marker is disabled it is visible on the map, but does not respond to mouse movements or clicks. When a marker is disabled, it's as though it were just a graphic element of the map image.

Signature:

```
mapsalive.setMarkerDisabled(hotspotIdList, disabled);
```

Example:

```
mapsalive.setMarkerDisabled("gallery1, gallery7, gallery19", true);
```

Parameters:

Parameter	**Description**
`hotspotIdList`	A comma separated list of one or more hotspot Ids. The marker for each hotspot identified in the list will be disabled or enabled.
`disabled`	Specifies whether the function should disable or enable markers. The value true indicates disabled. The value false indicates disabled.

4.27 setMarkerHidden

The setMarkerHidden function hides or shows one or more markers. When a marker is hidden it has no appearance and it does not respond to mouse movements or clicks. When a marker is hidden, it's as though it is not on the map.

Signature:

```
mapsalive.setMarkerHidden(hotspotIdList, visible);
```

Example:

```
mapsalive.setMarkerHidden("gallery1, gallery7, gallery19", false);
```

Parameters:

Parameter	**Description**
`hotspotIdList`	A comma separated list of one or more hotspot Ids. The marker for each hotspot identified in the list will become hidden or visible. You can also specify "*" to mean all hotspots on the map.
`hidden`	Specifies whether the function should make markers hidden or visible. The value true indicates hidden. The value false indicates visible.

4.28 setMarkerOnTop

The setMarkerOnTop function moves a marker on top of any other markers that it overlaps.

Signature:

```
mapsalive.setMarkerOnTop(hotspotId);
```

Example:

```
mapsalive.setMarkerOnTop("gallery1");
```

Parameters:

Parameter	**Description**
`hotspotId`	The hotspot Id for the marker that will appear on top of any adjacent markers.

4.29 setMarkerSelected

The setMarkerSelected function selects a marker and displays the content associated it as though the user had selected it either my moving their mouse over the marker or clicking on the marker depending on the marker options. The marker changes to its selected appearance.

Use this function to display a hotspot programmatically without user intervention.

When this function is called, the currently selected marker is deselected and reverts to its normal appearance.

If the map uses popups, calling the function again using the same `hotspotId` hides the popup, but the marker remains selected and continues to display its selected appearance.

- The setMarkerSelected function has no effect if the hotspot's "Show this hotspot when" marker option is set to "never."

- This function should not be called in the JavaScript for a hotspot's Click Action, Mouseover Action, or Mouseout Action. Doing so will cause unpredictable results because calling the function will conflict with the behavior of those mouse actions with regard to what marker is supposed to be selected or deselected.

- If the hotspot's map uses popups and the Mouse Location option is checked (on the **Popup > Popup Behavior** screen), the content will display at the mouse location which might not be near the hotspot's marker since this function is called programmatically. As such, the Mouse Location option should probably be turned off when using this function.

Signature:

```
mapsalive.setMarkerSelected(hotspotId);
```

Example:

```
mapsalive.setMarkerSelected("gallery1");
```

Parameters:

Parameter	Description
`hotspotId`	A hotspot Id for the marker that will become the selected marker and have its content displayed.

4.30 setMarkerStatic

The setMarkerStatic function sets one or more markers as static or not static. When a marker is static, it will not change appearance when the mouse is over it. It will however still respond to mouseover, mouseout, and click events by performing the action associated with those events. When a marker is static, it's as though its normal and selected appearance are the same.

Signature:

```
mapsalive.setMarkerStatic(hotspotIdList, static);
```

Example:

```
mapsalive.setMarkerstatic("H1", true);
```

Parameters:

Parameter	Description
`hotspotIdList`	A comma separated list of one or more hotspot Ids. The marker for each hotspot identified in the list will be set to static or not static.
`static`	Specifies whether the function should make the markers static or not static. The value true indicates static. The value false indicates not static.

4.31 setTourTitle

The setTourTitle function changes the tour's title text. The function has no effect if the tour does not display a title. One way to use it is to dynamically update the title when the user mouses over or clicks a marker.

Signature:

```
mapsalive.setTourTitle(title);
```

Example:

```
mapsalive.setTourTitle("Master Bedroom");
```

Parameters:

Parameter	Description
`title`	A string that specifies the title text.

4.32 stopSound

The stopSound function stops playing the sound that is currently playing as a result of a prior call to the playSound function. For more information see the SoundManager section of this document. See also `playSound`.

Signature:

```
mapsalive.stopSound);
```

Example:

```
mapsalive.stopSound();
```

5 Marker Shape Effects

You can use special effects to make the line and/or fill of shape markers more interesting. The following effects are supported:

- Blend
- Glow
- Shadow

The effects take parameters that specify how the effect should be rendered. Because an effect is used as a single parameter to some API functions, an effect and its own parameters are specified as a string enclosed in quotes.

The string can include more than one effect, each separated by a semicolon, so that you can make effect combinations. For example, you can specify "blend,invert;shadow" to combine the blend and shadow effects.

All parameters are optional; however, any parameters that are specified must appear in the correct sequence. If a parameter is provided after a parameter that is omitted, you must include a comma as a placeholder for the omitted parameter.

The effect name is case-insensitive. For example, you can specify "Glow" or "glow".

5.1 Blend Effect

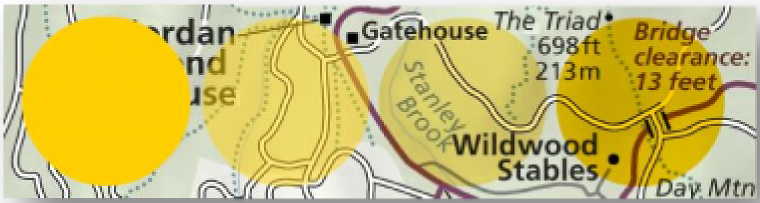
The blend effect combines a shape's semi-transparent line and fill colors with the colors of the underlying map image. How the colors blend is controlled by the mode parameter.

Parameter	Description
`mode`	A mode name that indicates what affect the blend will have. Valid values are: darken, difference, hardlight, invert, lighten, multiply, normal, overlay, screen. The default is multiply. A brief explanation of each mode follows. • **darken** – Replaces only the areas that are lighter than the blend color. Areas darker than the blend color don't change. • **difference** – Subtracts either the blend color from the base color or the base color from the blend color, depending on which has the greater brightness value. The effect is similar to a color negative. • **hardlight** – Multiplies or screens the colors, depending on the blend mode color. The effect is similar to shining a spot light on the object. • **invert** – Inverts the base color. • **lighten** – Replaces only pixels that are darker than the blend color. Areas lighter than the blend color don't change. • **multiply** – Multiplies the base color by the blend color, resulting in darker colors. • **normal** – Applies color normally, with no interaction with the base colors.

	• **overlay** – Multiplies or screens the colors, depending on the base colors. • **screen** – Multiplies the inverse of the blend color by the base color, resulting in a bleaching effect.

Perhaps the most useful mode in MapsAlive is multiply. Without the multiply effect, a semi-transparent shape's fill color will tend to wash out the underlying map image, especially text. By using the multiply effect, the shape can highlight and enhance an area of the map image without negatively affecting the map image.

In the image below, all three circles have the same fill color (#ffcf00), but each alters the appearance of the map in a different way. The first circle has 100% fill color opacity without blend. The second circle has 70% opacity without blend. The third circle has 70% opacity with blend multiply. The fourth circle has 100% opacity with blend multiply.



Examples:

```
"blend"
"blend,invert"
```

5.2 Glow Effect

The glow effect adds a glow to the outside of a shape to help it stand out more from the underlying map image.

Parameter	Description
`color`	The color of the glow. The default value is the shape's line color (even if the line thickness is zero and/or the line's opacity is zero).
`opacity`	The opacity value for the color. Valid values are 0 to 100. The default value is 50. Note that the base opacity of the glow itself is the opacity of the shape and therefore the shape and/or its line must be visible in order for the glow to be visible. If the shape and its line are both transparent, there will be no glow.
`blurX`	The amount of horizontal blur. Valid values are 0 to 255. The default value is 10. Directional blur is not supported when the map is rendered in HTML5 – the average of blurX and blurY is used instead.

`blurY`	The amount of vertical blur. Valid values are 0 to 255. The default value is 10. Directional blur is not supported when the map is rendered in HTML5 – the average of blurX and blurY is used instead.

Examples:

```
"glow"
"glow,,50"
"glow,0x00ff00,50,4,4"
```

Tip: if you want a glow around a shape, but want the shape to be invisible, set the shape's fill color to white, 100% opacity, with no line thickness. Also use the blend effect with multiply mode to cause the shape to blend with the background.

5.3 Shadow Effect

The shadow effect adds a drop shadow to a shape to help it stand out more from the underlying map image.

Parameter	Description
`distance`	The offset distance for the shadow, in pixels. The default value is 4.
`angle`	The angle of the shadow. Valid values are 0 to 360 degrees. The default value is 45. To give you a feel for the angle, when zero the shadow appears as if the light is coming from the left side of the marker. An angle of 45 makes the light appear to come from the upper left corner of the marker. An angle of 270 makes the light appear to come from the top of the marker.
`color`	The color of the shadow. The default value is 0x000000.
`opacity`	The opacity value for the shadow color. Valid values are 0 to 100. The default value is 40. Note that the base opacity of the shadow itself is the opacity of the shape and therefore the shape and/or its line must be visible in order for the shadow to be visible. If the shape and its line are both transparent, there will be no shadow.
`blurX`	The amount of horizontal blur. Valid values are 0 to 255. The default value is 10. Directional blur is not supported when the map is rendered in HTML5 – the average of blurX and blurY is used instead.
`blurY`	The amount of vertical blur. Valid values are 0 to 255. The default value is 10. Directional blur is not supported when the map is rendered in HTML5 – the average of blurX and blurY is used instead.

Example:

```
"shadow,5,135,0x748b5d,60,7,7"
```

Effect Combinations

You can combine more than one effect into an effect string. You separate each effect with a semicolon.

Example:

```
"blend,invert;glow,0x00ff00,50,4,4"
```

6 Custom HTML

The MapsAlive Tour Builder has a feature called Custom HTML (requires the Pro Plan) that lets you code HTML, CSS, and JavaScript directly into your tour without having to create a wrapper page. You access the feature by choosing **Tour> Custom HTML** from the menu.

> Custom HTML is an advanced feature intended for people with a good working knowledge of CSS, HTML, and JavaScript. MapsAlive does not validate the code you type. If there are errors, your tour may not display or work correctly. We cannot provide technical support for this feature. If your tour does not work correctly after you have added code on the Custom HTML screen, you can determine what is wrong by backing out your changes one at a time until the tour works again.

6.1 Text Area Fields

The Custom HTML screen contains the five text fields listed below. Each field is explained briefly in the sections that follow.

- CSS
- JavaScript
- Top
- Absolute
- Bottom

CSS

Use the CSS field to include any CSS that you want to use with your tour. Typically, the CSS you type here would be used to style HTML that you put in the Top, Absolute, or Bottom fields.

If you want to style and position the HTML you put in the Absolute field, specify attributes for #maCustomHtmlAbsolute. For example:

```
#maCustomHtmlAbsolute
{
   left:200px;
   top:100px;
   background-color:white;
   border:solid 3px green;
}
```

JavaScript

Use the JavaScript field to code JavaScript that you want to use with your tour. You can code functions to be called when your map first loads or when a user clicks a marker. For example, you can include functions that are called from the Click, Mouseover, or Mouseout actions that you code for a marker on the Hotspot Actions screen.

You can also use the JavaScript field to include an external JavaScript file. See section 6.2 below to learn how.

Top

Use the Top field to include any HTML that you want the browser to render above your tour. You might use this HTML to display a header, graphic, or menu above your tour.

Absolute

Use the Absolute field to include any HTML that you want the browser to display on top of your tour. By default, any HTML entered in this field is absolute positioned relative to the top left corner of the tour and displays on top of the tour. To position your content differently, you can specify the top and left position in the CSS field by coding a #maCustomHtmlAbsolute class (CSS field above).

The Absolute field is useful when you want to display options, a legend, or a navigation mechanism that appears to float over your tour. For example, you could display checkboxes that allow the user to show or hide markers on the map.

Bottom

Use the Bottom field to include any HTML that you want the browser to render below your tour. You might use this field to display a footer or menu below your tour.

6.2 Including an External JavaScript File

If your tour requires a lot of JavaScript, consider putting the code in an external file instead of in the JavaScript field described in the previous section. Or, if some of your JavaScript is specific to the tour and other JavaScript is more general purpose, you can put the tour-specific logic in the JavaScript field and put the general purpose logic in a file.

To tell MapsAlive that your tour needs to reference an external JavaScript file, you specify the URL of the file as the first line of the JavaScript field like in this example:

```
//#include http://www.yourwebsite.com/yourscript.js
```

It is very important that you specify the include syntax exactly as shown above. The first 10 characters must be `//#include` followed by a space followed by the URL without quotes. Although the syntax is for a JavaScript comment, when MapsAlive builds your tour it will parse the line and emit a <script> tag to include the URL in the tour. If you look at the Code Snippets section of the Tour Preview screen you will see that tag. If you are embedding the tour in another web page, you will need to code the <script> tag yourself (or copy/paste it from Code Snippets).

The include file must be located on a server that is accessible from your tour.

Including an external JavaScript file is an advanced feature that you should only use if you have experience with using JavaScript that is referenced in a separate file.

6.3 How Custom HTML Works

The HTML, CSS, and JavaScript that you code in the Custom HTML fields gets built into your tour. If you archive and restore the tour, or download the tour to host on your own server, the code will be there without you having to keep track of or maintain separate files. Note, however, that if your interactive map application requires large amounts of additional HTML, CSS, or JavaScript, you probably will want to maintain it in separate files. The Custom HTML feature is intended for small to moderate amounts of code.

To try a working example of a tour that uses Custom HTML to display a map legend click this link http://samples.mapsalive.com/20022 or type it into your browser.

You can learn how this example was created you can read the Population Map Tutorial at http://docs.mapsalive.com/Tutorials/PopulationMap/MapsAliveTutorial_PopulationMap.pdf.

6.4 Using Custom CSS to Override MapsAlive Styles

MapsAlive uses CSS to control the appearance of your tours. If you are comfortable coding CSS, you can override the styling of certain HTML elements of a tour page to achieve the look you need. This section lists some of the CSS classes you can override.

This information is intended for use by experienced web developers who know how to write CSS. Be very careful when making changes. If you experience problems, undo your changes and add them back one at a time until you determine which one is causing the issue.

.maPopup

The .maPopup class controls the appearance of a popup's corners and drop shadow. You can override these attributes.

- -moz-border-radius
- border-radius
- -moz-box-shadow
- -webkit-box-shadow
- box-shadow

#maHotspotImage

The #maHotspotImage class controls the appearance of corners of the image within a popup. You can override these attributes:

- -moz-border-radius
- border-radius

#maHotspotMediaArea
The #maHotspotMediaArea class can be used to alter the styling of the <div> that is used to contain the hotspot's image or video. Be very careful when coding CSS for this class and be sure to test your changes with all browsers that your tour will run on.

#maHotspotText
The #maHotspotText class controls the appearance of the descriptive text that displays for a hotspot. By adding your own CSS for this class, you can set the appearance of the text for all of your hotspots without having to modify each using the text editor on the Edit Hotspot Content screen. For example, you could use this class to make all of your text appear in green italics.

#maHotspotTitle
The #maHotspotTitle class controls the appearance of the title text that displays for a hotspot. You can use this class in a similar way as described above for #maHotspotText.

#maTextArea
The #maTextArea class can be used to alter the styling of the <div> that contains the hotspot title and text. For example, you could use this class to make all of your hotspot titles and text appear centered.

Classes that control the appearance of the Directory

These are some of the classes that control the appearance of the directory elements. Many of the directory elements can be controlled on the **Tour > Directory Options** screen in the Tour Builder – the classes listed below allow for additional customization.

If you are also targeting mobile devices or touch screens you can change both the standard class and the touch class as listed below. The classes that end with the word Touch apply when a tour is running on a mobile device or when you have chosen the **Use Touch User Interface for Desktop Browser** option on the **Tour Manager** screen.

We have documented these classes to give you more flexibility in styling the directory, but be aware that changing some attributes may adversely affect the overall layout of the directory elements in general. Make these changes at your own risk.

.maDir and **.maDirTouch**
The .maDir and .maDirTouch classes control the overall appearance of the directory title area. This class can be used to control attributes like the font-family, padding or height of this area. Be aware that changing the height may cause the directory title area to not fit within the tour title bar, but can be useful if you are positioning the directory yourself. Some attributes such as font-size are overridden by classes below.

.maDirTitle and **.maDirTitleTouch**
The .maDirTitle and .maDirTitleTouch classes control the title text in the directory title area. Use these to control the font size, style and weight of the text or to modify the alignment. You can set the actual title text and title text color on the Directory Options screen, but use this class to format it.

.maDirSearchLabel and **.maDirSearchLabelTouch**
The .maDirSearchLabel and .maDirSearchLabelTouch classes control the search label text if you are using the search feature. You can set the actual text and label color on the Directory Options screen, but use this class to format it.

.maDirSearchBox and **.maDirSearchBoxTouch**
The .maDirSearchBox and .maDirSearchBoxTouch classes control the text box where you type a search term. You might use this class to adjust the width, font-size or height of this box, but be aware that changing the size of the box may adversely affect the layout of the directory in general.

.maDirStatusLine and **.maDirStatusLineTouch**
The .maDirStatusLine and .maDirStatusLineTouch classes control the status row that displays below the directory title area and above the directory entries themselves. You can set the text color and background color of the status line on the Directory Options screen.

.maDirBody and **.maDirBodyTouch**
The .maDirBody and .maDirBodyTouch classes control the overall appearance of the area of the directory below the directory title area. This is the area that contains the status line and the directory entries themselves. You can set the border color, width and background color of this area on the Directory options screen, but if you need to make other adjustments that apply to the whole area use this class.

.maDirLevel1 and **.maDirLevel1Touch**
The .maDirLevel1 and .maDirLevel1Touch classes control the attributes of the Map names or Category titles in the directory when the grouping is set to anything other than *None*. You can choose the color for these on the Directory Options screen, but use this class to set other attributes or override the font size specified in the .maDirBody or .maDirBodyTouch classes.

.maDirLevel2 and **.maDirLevel2Touch**
The .maDirLevel2 and .maDirLevel2Touch classes control the attributes of the Map names or Category titles at the second level in the directory when the directory grouping is set to either *By category, then map* or *By map, then category*.

.maDirLevel Count and **.maDirLevelCountTouch**
The .maDirLevelCount and .maDirLevelCountTouch classes control the text that shows how many entries there are for a map or a category. You can set the color of the count text on the Directory Options screen.

.maDirEntry and **.maDirEntryTouch**
The .maDirEntry and .maDirEntry Touch classes control the text links of the hotspot names in the directory. You can set the color of the entry text on the Directory Options screen.

.maDirEntrySearchResult and **.maDirEntrySearchResultTouch**
The .maDirEntrySearchResult and .maDirEntrySearchResultTouch classes control the search results area that displays below the title bar when you type a search. You can set the color of the results text and background on the Directory Options screen.

.maInstructionsTitle and **.maInstructions**
The .maInstructionsTitle and .maInstructions classes control the appearance of the instructions title and text respectively. Overriding these classes only works when the map is displayed using HTML5. It has no effect when the map is displayed using Flash.

7 SoundManager

MapsAlive lets you play MP3 audio using SoundManager 2 from Scott Schiller. SoundManager is a JavaScript library that utilizes Flash to play sounds. To learn more about SoundManager, visit http://www.schillmania.com/projects/soundmanager2.

The MapsAlive JavaScript API serves as a wrapper around SoundManager to make it as easy as possible for you to play sounds from your interactive maps. In fact, there are only two MapsAlive API functions: `playSound` and `stopSound`.

> The MP3 files containing your sounds must be located at a URL somewhere on the internet. MapsAlive does not provide a way for you to upload or store audio files.

7.1 How to play a sound

Here's all that is required to play a sound when the mouse moves over a hotspot on your map:

1. Choose **Tour** > **Tour Manager** in the menu and check the **Enable SoundManager** box.
2. Choose **Hotpot** > **Hotspot Actions** in the menu.
3. Choose **JavaScript** for the **Mouseover Action**.
4. For the JavaScript, code a call to **mapsalive.playSound** as described in section 4.13 above.

Example: `mapsalive.playSound("hello", "http://samples.mapsalive.com/audio/hello.mp3");`

If you want the sound to stop playing when the mouse moves off of the hotspot, call mapsalive.stopSound from the hotspot's mouseout action.

> In order to use the playSound and stopSound functions, you must have checked the **Enable SoundManager** option on the **Tour Manager** screen of the Tour Builder.

7.2 Including the SoundManager library in your tour

When you check the Enable SoundManager box on the Tour Manager screen, you are telling MapsAlive to include the SoundManager library with your tour files. The library consists of a JavaScript file named `soundmanager2-nodebug-jsmin.js` and a Flash file named `soundmanager2.swf`.

If you forget to check the Enable SoundManager box, you'll get an error message when you call `playSound` or `stopSound`.

7.3 Embedding a tour that uses SoundManager

Embedding Using an Iframe

You can embed your tour using an iframe by copying the single line of HTML from section 4 of the Code Snippets on the Tour Preview screen.

Direct Embedding

If you want to directly embed your tour, look at the HTML in section 5 of the Code Snippets on the Tour Preview screen. When using SoundManager, there are two extra `<script>` tags that include the SoundManager library. You can copy the entire snippet and paste it into your web page. If your web page already embeds your tour and now you are adding sound, you must add the two lines of code to your web page.

IMPORTANT: If you are directly embedding a tour in a web page, the web page and the SoundManager library files must be hosted on the same domain otherwise sounds will not play.

There are two ways to ensure that the web page and SoundManager library files are on the same domain. You must do one or the other:

1. Download the tour and upload it to the same location as your web page, or

2. Upload the SoundManager library files to the same location as your web page. You can get the two files by downloading your tour and extracting them from the download zip file. The two files are `soundmanager2-nodebug-jsmin.js` and `soundmanager2.swf`.

To learn about downloading a tour, see the *MapsAlive User Guide for Integrating Interactive Maps.*

Note that the restriction above that the web page and SoundManager library files are together is due to how Flash security works with SoundManager. If the web page and files are not on the same domain, a JavaScript error will occur due to a cross-domain Flash security violation. To learn more about this, see the forums at http://www.schillmania.com/projects/soundmanager2.

7.4 Pausing and resuming play

You can pause the playing of a sound by calling `playSound` again for the sound that is currently playing. Calling `playSound` for the sound that is currently paused will cause play to resume. Thus, `playSound` not only causes a sound to start playing, it also acts as a pause/resume toggle.

7.5 Playing a sound when your tour loads

If you want to have a sound play when your tour first loads, you can call the `playSound` function from the `maOnSoundManagerReady` function (see section 4.134.12 above) as shown in the example below.

```
function maOnSoundManagerReady()
{
   mapsalive.playSound("magical", "http://samples.mapsalive.com/audio/magical.mp3");
}
```

Note that that any calls you make to `playSound` or `stopSound` before SoundManager is loaded and ready will be ignored.

7.6 Other way to play audio

MapsAlive includes SoundManager for your convenience, but it’s not the only way to play audio from your interactive map. You can use any sound library that you like, but you’ll need to write your own HTML and JavaScript to utilize it.

7.7 Disclaimer

MapsAlive includes API support for SoundManager as a convenience to you, but SoundManager is not our product. We are happy to answer your questions if you are having difficulties, but we cannot offer technical support for SoundManager itself.