Application Notifications

# Introduction

* 1. Web applications categorized by users as responsive, have one thing in common; they provide appropriate and timely feedback to the user. This feedback comes in many forms: save or success message after completing a task, subtle user interface (UI) animations in response to a UI gesture, loading or progress message for long running tasks, or input error messages that are displayed before a page is submitted.

How the application surfaces the feedback or notifications to the user, is almost as important as the information itself. Intrusive message boxes, modal dialogs, or overlays (floating messages) that require the user to dismiss messages get in the user's way, interrupt workflow, and degrade the overall user experience.

In addition to providing feedback during normal application use, the website must also provide quality feedback when a non-recoverable error occurs. Quality feedback means providing understandable information about what has occurred, along with clear instructions on how to proceed.

## What you will learn in this chapter

* + How to provide unobtrusive user notification messages.
  + Purpose for providing an error page that is consistent with other websites pages.
  + Alternatives to modal dialogs for prompting users.
  + How to enable application notifications on the desktop.
  1. The technologies discussed in this chapter are jQuery UI Widgets and Pinned Sites in Microsoft Internet Explorer 9.
  2. For a comprehensive look at input validation error messages, see Chapter 10, "Data, Caching, and Validation."

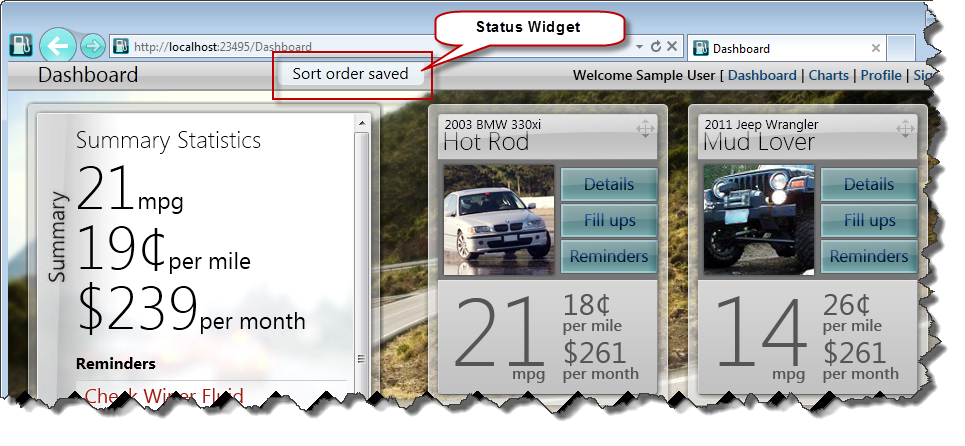
# Notifying the User

* 1. Providing a high quality application notification experience requires careful planning with emphasis on where notifications will be displayed, what events initiate a message, how potential multiple simultaneous messages will be handled, and how to decouple the message originator from the object tasked with displaying the message.
  2. During the design phase of the Mileage Stats application, the Project Silk team iterated on *where* and *how* notification messages would be displayed. We spent time white boarding and prototyping several different notification designs.
  3. Where notification messages are displayed is an essential part of the overall application user experience (UX) and user interface (UI) design. Our initial design called for messages and progress bars to be displayed within the boundaries of each jQuery UI widget. After several prototypes combined with usability testing, the team determined this design was unnecessary because the UI loads very fast, alleviating the need for a loading progress bar in this application. The team decided that displaying user messages in a single location made for a much better UX experience than having messages displayed within individual widgets.
  4. Throughout application development and based on usability testing, the team tailored application events that triggered user messages. Initially, the team displayed messages each time an Ajax request was invoked. This caused the UI to be too busy so we associated a time delay with the message so that it would only display if the request took longer than the time delay. This too, got messy; requiring a good deal of code with little or no value added to the application. In the end, the principle of, "less is more" triumphed, resulting in a good balance of informative messages.
  5. Interactive and engaging applications like Mileage Stats can execute multiple, concurrent, asynchronous operations, such as the Dashboard page loading data for several jQuery UI widgets and the chart plugin. Each of these operations loads data for a region of the UI. Any of these operations is a potential point of failure requiring an error message. It's important that the application notification implementation be able to manage multiple simultaneous or near simultaneous messages.

From an architectural design perspective, it's critical that message initiators not be responsible for determining how to coordinate the display of messages in the UI. Decoupling the initiator and rendering object allows them to evolve independently and to be tested in isolation.

The above section provides you a glimpse into how the team worked together to maintain the delicate balance of UX, UI and engineering concerns. It's this type of designer-developer collaboration that enabled the team to deliver a successful notification feature.

## Where Notification Messages are Displayed

* 1. Mileage Stats is composed of widgets. The decision to create and use a widget for displaying notification messages is a natural architectural design fit for this application. Widgets have flexible and powerful UI capabilities, provide for encapsulation of behavior, and can have external dependencies like pub/sub injected in their options object on creation.
  2. Mileage Stats uses a single **status** widget for displaying messages to the user. The **status** widget subscribes to the Mileage Stats **status** pub/sub message; handles placement and rendering of messages as well as coordinating multiple simultaneous messages.
  3. Location of the Status Widget
  4. 

The **status** widget is rendered within the **header** widget UI as pictured above. This top, semi-central location was chosen because it's easier for the user to notice the message in this location as opposed to a message area along the bottom of the browser window. The balance of easy to notice, easy to read, yet unobtrusive user notifications, took time, patience, and usability testing. The multiple design iterations were worth the extra investment of time.

## How Notification Messages are Initiated

Mileage Stats notification messages are initiated by widgets and loosely communicated to the **status** widget using the pub/sub JavaScript object. Like other pub/sub messages, the **status** message has an associated payload object that is passed with the message.

* 1. Notification Messages passed using Pub/Sub



The below code snippet is from the **vehicleDetails** widget. The **\_publishStatus** method is responsible for making the pub/sub call. It's called internally by other widget methods to initiate the display of a message.

The **status** argument is the message payload and is forwarded in the **publish** call.

The **publish** method was passed in the widget options object when the widget was created and points to the **pubsub** JavaScript object. The jQuery **isFunction** method verifies that **publish** is a valid JavaScript function object before it's called.

* 1. JavaScript
  2. // Contained in mstats.vehicle-details.js
  3. \_publishStatus: function (status) {
  4. if ($.isFunction(this.options.publish)) {
  5. this.options.publish(mstats.events.status, status);
  6. }
  7. },
  8. The below functions show how easy it is to initiate the display of a user message. As stated earlier, Mileage Stats does not bother the user with data request messages. However, when initiating an Ajax operation such as a save or delete, it's important to keep the user informed by updating the UI as the request proceeds and concludes.
  9. Each of the below functions create an object literal containing a **type**, **message**, and **duration** property. The **type** property is used by the **status** widget to prioritize multiple or overlapping display message requests. The **message** is the text of the message to display and the **duration** is how long the message should display.
  10. The **\_showDeletingMessage** function is called after the user confirms their intent to delete the vehicle. This message is intended to inform the user the vehicle deletion has been submitted to the server.
  11. The **\_showDeletedMessage** function is called after a successful deletion of the vehicle, informing the user that the deletion as successful.
  12. The **\_showDeleteErrorMessage** function is called if an error occurred while deleting the vehicle.
  13. JavaScript
  14. // contained in mstats.vehicle-details.js
  15. \_showDeletingMessage: function () {
  16. this.\_publishStatus({
  17. type: 'saving',
  18. message: 'Deleting the selected vehicle ...',
  19. duration: 5000
  20. });
  21. },
  22. \_showDeletedMessage: function () {
  23. this.\_publishStatus({
  24. type: 'saved',
  25. message: 'Vehicle deleted.',
  26. duration: 5000
  27. });
  28. },
  29. \_showDeleteErrorMessage: function () {
  30. this.\_publishStatus({
  31. type: 'saveError',
  32. message: 'An error occurred deleting the selected vehicle. Please try again.',
  33. duration: 10000
  34. });
  35. }
  36. For detailed information on the inner working of the Mileage Stats pub/sub implementation, see Chapter 8, "Communication."

## How Individual or Multiple Notification Messages are Displayed

* 1. In the following **\_create** method, the **status** widget subscribes to the **status** event. When this event is raised, the **\_statusSubscription** method is invoked.
  2. The **\_statusSubscription** method is responsible for displaying and hiding messages as well as managing multiple simultaneous messages. If a message is being displayed and another message with a higher priority arrives, the higher priority message will be shown.
  3. JavaScript
  4. // contained in mstats.status.js
  5. \_create: function () {
  6. // handle global status events
  7. this.options.subscribe(mstats.events.status, this.\_statusSubscription, this);
  8. },

...

* 1. \_statusSubscription: function (status) {
  2. var that = this;
  3. status.priority = that.\_getPriority(status);
  4. // cancel displaying the current message if its priority is lower than
  5. // the new message. (the lower the int the higher priority)
  6. if (that.currentStatus && (status.priority < that.currentStatus.priority)) {
  7. clearTimeout(that.currentStatus.timer);
  8. }
  9. that.currentStatus = status;
  10. that.element.text(status.message).show();
  11. // set the message for the duration
  12. that.currentStatus.timer = setTimeout(function () {
  13. that.element.fadeOut();
  14. that.currentStatus = null;
  15. }, status.duration || that.options.duration);
  16. },

## User Session Timeout Notification

* 1. Errors that can occur anywhere in the application can often be handled in a centralized location so that individual components don't need to repeat the same error handling code.
  2. Mileage Stats implements a global **ajaxError** method handler; if an error occurs during an Ajax request, the below method will be invoked.
  3. JavaScript
  4. // contained in mileagestats.js
  5. // setup default error handler for redirects due to session timeout.
  6. $(document).ajaxError(function (ev, jqXHR, settings, errorThrown) {
  7. if ((jqXHR.status === 200)
  8. && (settings.dataType === 'json')
  9. && (errorThrown.message.indexOf('Invalid JSON') == -1)) {
  10. window.location.replace(mstats.getRelativeEndpointUrl('/Auth/SignIn'));
  11. }
  12. });
  13. **Note:** If the originating Ajax calling code also implemented an error handler, the originating Ajax caller's error handler will be called first, then the above global Ajax error handler.
  14. The primary purpose of this method is to identify if the initiating Ajax request caused a session timeout error and if so, redirect to the Sign In page. Mileage Stats uses forms authentication, with a session timeout threshold of 20 minutes. If the session times out, the request will be redirected to the page specified by the forms authentication **loginUrl**. Since our request is an Ajax request and an error occurred, the Ajax request will be redirected to the **loginUrl** but the browser UI won't be. The above function detects this condition and redirects the browser to the Sign In page.

# Website Error Notification

* 1. ASP.NET provides developers the ability to specify a default error page for their website that the ASP.NET runtime will redirect to when an unhandled exception occurs. This error page is configured in the web.config file **customErrors** section.
  2. C#
  3. // Contained in web.config
  4. <customErrors defaultRedirect="GenericError.htm" mode="RemoteOnly" />
  5. The error page should look and feel like it is part of the website, contain a brief explanation of why the user has been redirected to this page, and provide users with links to try and continue using the site.
  6. Mileage Stats GenericError.htm page
  7. 

# Prompting Users

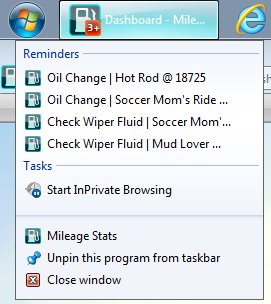
* 1. During the design phase of Project Silk, the team had a UX design goal of, not prompting users with modal dialogs. Website UX designers are getting away from modal dialogs that ask the user questions like, "Are you sure?" Instead, they prefer to implement an undo system, allowing the user to undo the last task as opposed to confirming they want to perform the task. The undo feature could also enhance the application by extending undo capabilities to tasks that did not require a confirmation dialog.
  2. Our initial design was to provide a state field on each record in the database, which could be queried against, allowing previous actions to be viewed and undone. Unfortunately, this feature was cut so other higher priority features could be implemented.

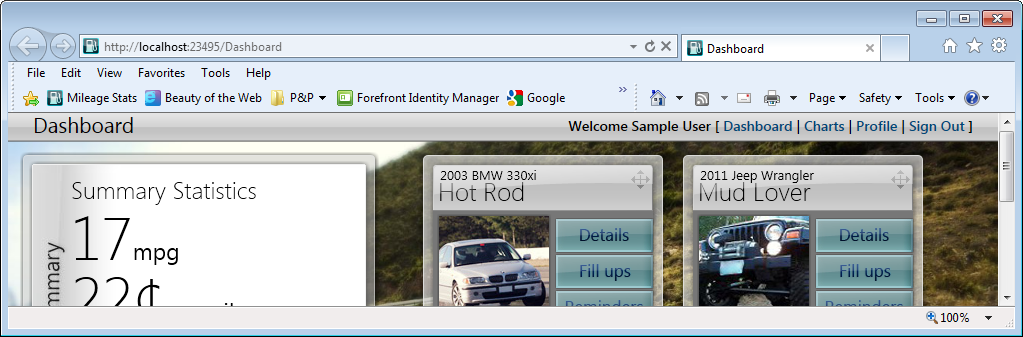
The below code utilizes the JavaScript **confirm** function to validate the users request to fulfill a maintenance reminder.

* 1. JavaScript
  2. // contained in mstats.reminders.js
  3. fulfillReminder: function (fulfillmentUrl) {
  4. var shouldfulfill = confirm('Are you sure you want to fulfill this reminder?');
  5. if (shouldfulfill) {
  6. this.\_fulfillReminder(fulfillmentUrl);
  7. }
  8. },

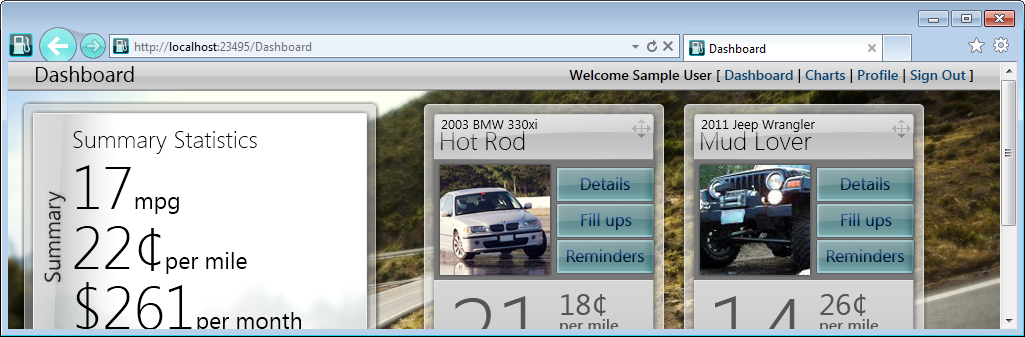
# Desktop Notifications

* 1. Given that modern web applications can have excellent user experiences that rival desktop applications, the team wanted to take the next logical step and integrate the Mileage Stats application with the user's desktop to provide appropriate dynamic user notifications. This integration was made possible by the Internet Explorer 9 Pinned Site API.
  2. Websites that implement the Pinned Site API can feel more like a native Windows application. They can leverage Windows 7 taskbar capabilities and when launch, the browser window is customized specifically for the site. Pinned Sites requires Internet Explorer 9 running on Windows 7.
  3. Mileage Stats uses Pinned Sites to provide Windows 7 taskbar notifications that indicate the user has one or more overdue maintenance reminders. In addition, a dynamic jump list provides a direct link to each overdue maintenance reminder.
  4. Mileage Stats taskbar integration

1. 
   1. The two below images contrast Mileage Stats running in a normal browser window and a customized Pinned Sites browser window. The lower image shows the clean, slimmed-down browser window with potentially distracting browser features removed from view, allowing the user to focus on the application features. Applications run in the customized browser window when they are launched from a taskbar or start menu Pinned Sites icon.
   2. Mileage Stats without using Pinned Sites



* 1. Mileage Stats using Pinned Sites



In addition to a cleaner browser window, Pinned Sites also allows the developer to customize the color of the browser back and forward buttons and displays the website favicon to the left of the back button. This favicon is also a link to the website home page.

## Implementing Pinned Sites

* 1. Microsoft provides developers excellent documentation for implementing Pinned Sites in their web applications on MSDN. The title of this topic is, "Pinned Sites Developer Documentation" located here <http://msdn.microsoft.com/en-us/library/gg491731(v=VS.85).aspx>.

## Pinned Sites in Mileage Stats

* 1. The following sections will not attempt to duplicate the above MSDN documentation nor cover every line of code pertaining to Pinned Sites. Instead, the core Mileage Stats implementation will be explained, enabling developers to understand pieces, requirements, capabilities, and value of the Pinned Sites API.
  2. The Pinned Sites implementation in Mileage Stats includes feature detection, site pinning, dynamic jump list updating, and displaying notification icons. These features are encapsulated in the **mstats.pinnedSite** JavaScript object that is contained in the mstats.pinnedsite.js file. The **pinnedStite** object is initialized differently depending if the user is sign in or not, this initialization will be described below.

### Feature Detection

* 1. Pinned Sites feature detection is provided by the Internet Explorer 9 **msIsSiteMode** function. Verifying the page was opened as a pinned site before attempting to execute Pinned Site API methods prevents unnecessary JavaScript errors.
  2. The **msIsSiteMode** function returns **true** if the current page was launched as a pinned site, otherwise **false**. The below **isPinned** function wraps the **msIsSiteMode** call and returns **false** if the page was not launched as a pinned site, or the browser is not Internet Explorer 9.
  3. JavaScript
  4. // Contained in mstats.pinnedsite.js
  5. isPinned: function () {
  6. try {
  7. return window.external.msIsSiteMode();
  8. }
  9. catch (e) {
  10. return false;
  11. }
  12. }

### Enabling Website Pinning

* 1. Unauthenticated users visiting the site are directed to the below landing page. This page allows users to sign in, pin the site, and view the Mileage Stats video (not pictured). The Pinned Sites icon will glow when it is draggable, allowing the user to pin the site to the taskbar or start menu. The callout text displays for 5 seconds when the page loads then hides. It will also show and hide as the user moves their mouse over or away from the Pinned Sites icon.
  2. Landing Page
  3. 
  4. The Pinned Sites JavaScript object is initialized when the above page loads by the below JavaScript function.
  5. CSHTML
  6. // Contained in Index.cshtml
  7. <script>
  8. $(function () {
  9. mstats.pinnedSite.intializePinndedSiteImage();
  10. });
  11. </script>

If the browser is Internet Explorer 9 and the website is not currently pinned, the **intializePinndedSiteImage** method will attach appropriate event handlers for hiding and showing the callout text. It also adds the **active** CSS class to the Pinned Sites icon, to cause the icon to appear to glow.

* 1. JavaScript
  2. // Contained in mstats.pinnedsite.js
  3. intializePinndedSiteImage: function () {
  4. try {
  5. // Do not enable site pinning for non-Internet Explorer 9 browsers
  6. // Do not show the callout if the site is already pinned
  7. if (!(document.documentMode === undefined || mstats.pinnedSite.isPinned())) {
  8. $('#pinnedSiteImage')
  9. .bind('mousedown mouseout', mstats.pinnedSite.hideCallout)
  10. .bind('mouseover', mstats.pinnedSite.showCallout)
  11. .addClass('active');
  12. $('#pinnedSiteCallout').show();
  13. setTimeout(mstats.pinnedSite.hideCallout, 5000);
  14. }
  15. }
  16. catch (e) {
  17. // Fail silently. Pinned Site API not supported.
  18. }
  19. },

The below HTML snippet, shows the required **msPinSite** class applied to the Pinned Sites icon. This class is used by Internet Explorer 9 to enable this icon to be dragged to the taskbar or start menu to pin the site.

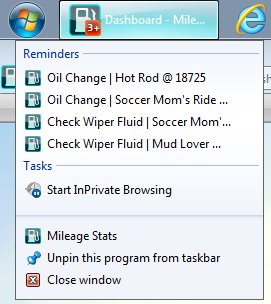
* 1. CSHTML
  2. // Contained in Index.cshtml
  3. <img id="pinnedSiteImage" class="msPinSite" … />

To call the users attention to the draggable Pinned Sites icon, the below **active** CSS class, adds an attractive outer glow to it.

* 1. CSS
  2. // Contained in static.css
  3. #pinnedSiteImage.active
  4. {
  5. cursor: pointer;
  6. box-shadow: 0px 0px 15px #6Dffff, inset 0px 0px 10px #6Dffff;
  7. border-radius: 12px;
  8. }
  9. User can pin a website by dragging the Pinned Sites icon, browser tab, or favicon to the taskbar or start menu. Internet Explorer 9 integrates with the Windows shell to accomplish the pinning.

### Dynamic Jump List Updating and Notification Icons

* 1. Mileage Stats uses the jump list and notification icons to notify users of overdue maintenance reminders. When the user clicks on the jump list entry, they will be taken to that reminder. The notification overlay icon displays 1, 2, 3, or 3+ to provide a taskbar indication of outstanding reminders.
  2. Jump list and notification icon



On the initial page load after the user authenticates, the client-side widgets and JavaScript objects are bootstrapped by code in the mileagestats.js file. The **pinnedSite** object is initialized by passing it a delegate to the data manager's **sendRequest** method.

* 1. JavaScript
  2. // Contained in mileagestats.js
  3. mstats.pinnedSite.intializeData(mstats.dataManager.sendRequest);

**initializeData** saves the **sendRequestDelegate** in the **sendRequest** property for future calls to the data manager by the **requeryJumpList** function.

* 1. JavaScript
  2. // Contained in mstats.pinnedsite.js
  3. intializeData: function (sendRequestDelegate) {
  4. sendRequest = sendRequestDelegate;
  5. mstats.pinnedSite.requeryJumpList();
  6. },

The below **requeryJumpList** function is called when the **pinnedSite** object is initialized and externally by the **layoutManager** widget when a reminder is fulfilled. It's the layout manager's call that initializes the dynamic updating of the jump list and notification icon.

* 1. **Note:** Only the essential lines of code that demonstrate the loading of the jump list and updating of the notification icon are listed below.

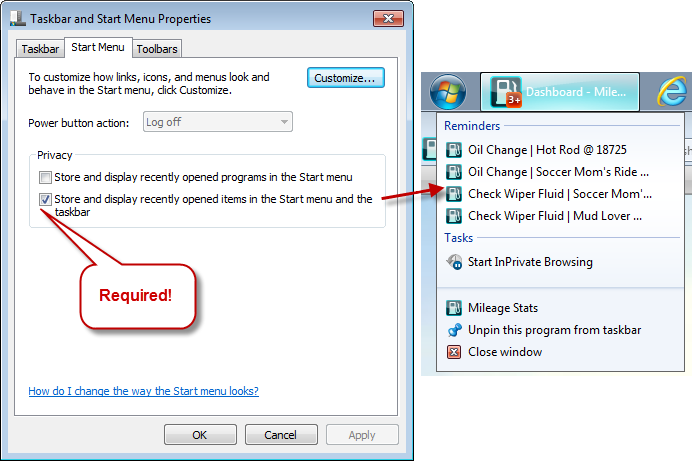
All of the below **msSite** functions are provided by Internet Explorer 9. After using feature detection to determine if the site is pinned, the jump list and overlay icon are cleared, and a new jump list is created.

Not shown below is the Ajax call to the data manager to get the array of overdue reminders. If that Ajax request is successful and the **data.Reminders** array has data, a URL will be constructed for each data item and added to the jump list. Next the appropriate overlay icon is set. Finally **msSiteModeShowJumpList** is called to update the jump list.

* 1. JavaScript
  2. // Contained in mstats.pinnedsite.js
  3. requeryJumpList: function () {
  4. try {
  5. if (mstats.pinnedSite.isPinned()) {
  6. ...
  7. var g\_ext = window.external,
  8. ...
  9. g\_ext.msSiteModeClearJumpList();
  10. g\_ext.msSiteModeCreateJumpList("Reminders");
  11. g\_ext.msSiteModeClearIconOverlay();
  12. if (data.Reminders) {
  13. for (i = 0; i < data.Reminders.length; i += 1) {
  14. reminderUrl = mstats.getRelativeEndpointUrl('/reminder/details/' +
  15. data.Reminders[i].Reminder.ReminderId.toString());
  16. g\_ext.msSiteModeAddJumpListItem(data.Reminders[i].FullTitle, reminderUrl,
  17. "./favicon.ico", "self");
  18. }
  19. if (data.Reminders.length > 0) {
  20. iconOverlayUrl = '/content/overlay-' + data.Reminders.length + '.ico';
  21. iconOverlayMessage = 'You have ' + data.Reminders.length.toString() +
  22. ' maintenance tasks that are ready to be accomplished.';
  23. if (data.Reminders.length > 3) {
  24. iconOverlayUrl = '/content/overlay-3plus.ico';
  25. }
  26. g\_ext.msSiteModeSetIconOverlay(iconOverlayUrl, iconOverlayMessage);
  27. }
  28. }
  29. g\_ext.msSiteModeShowJumpList();
  30. ...

1. The above code demonstrates that with a small investment, developers can deliver dynamic desktop notifications in their websites.

## Requirement for Jump List Items to Appear

* 1. The Windows 7 taskbar jump list items can be disabled by your users, preventing them from displaying even though the website has been pinned to the taskbar.
  2. If you website implements the jump list feature, you should provide this information to your visitors and advise them that the, "Store and display recently opened items in the Start menu and the taskbar" property setting will disable the jump list items if set to false.
  3. Taskbar and Start Menu Properties
  4. 

# Summary

* 1. TODO

# Further Reading

* 1. jQuery: isFunction():  
     <http://api.jquery.com/jQuery.isFunction/>
  2. Pinned Sites Developer Documentation:  
     <http://msdn.microsoft.com/en-us/library/gg491731(v=VS.85).aspx>