Application Notifications

# Introduction

* 1. All web applications that users consider responsive have one thing in common: they provide appropriate and timely feedback to the user. This feedback can come in many forms, including a save or success message following a completed task, subtle animations in response to a user interface (UI) gesture, a progress message for long-running tasks or input error messages displayed before a page is submitted.

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

* 1. In this chapter you will discover:
  + How to provide unobtrusive user notification messages.
  + How to handle multiple simultaneous notification messages raised by the application.
  + The benefits of encapsulating the display and management of user notification in a single JavaScript object.
  + How to display a helpful global error page.
  + How to set up a global error handler for Ajax requests.
  + Alternatives to modal dialogs for prompting users.
  + How to enable application notifications on the desktop with the Pinned Sites API.
  1. The technologies discussed in this chapter are jQuery UI Widgets and Pinned Sites in Windows® Internet Explorer® 9.
  2. For a comprehensive look at input validation error messages, see Chapter 10, "Data, Caching, and Validation."

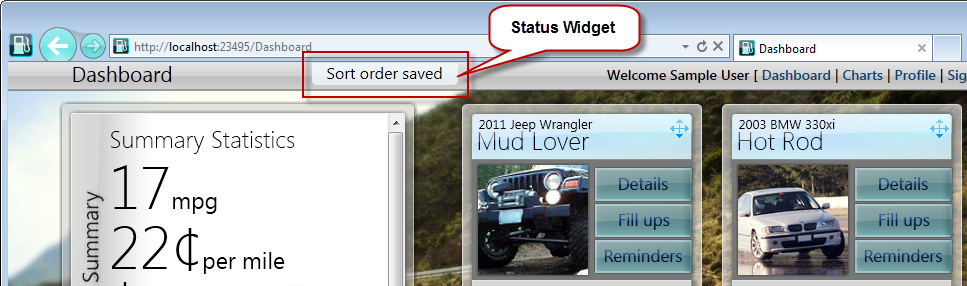
# Notifying the User

* 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 discussed *where* and *how* notification messages would be displayed. We spent time 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 design. Our initial design called for messages and progress bars to be displayed within the boundaries of each jQuery UI widget. After building several prototypes and performing 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 experience than having messages displayed within individual widgets.
  4. Throughout application development, the team tailored application events that triggered user messages based on usability testing. 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 "less is more" principle triumphed, resulting in a good balance of informative messages.
  5. Interactive and engaging applications such as Mileage Stats can execute multiple, asynchronous operations, such as the Dashboard page that loads data for several jQuery UI widgets in addition to the chart widget. 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 nearly 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 message initiator from the rendering object allows both of them to evolve independently and to be tested in isolation.

The above section provides 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 publish and subscribe (pub/sub) injected into their options object during creation.
  2. Mileage Stats uses a single widget called **status** for displaying messages to the user. The **status** widget subscribes to the Mileage Stats **status** pub/sub message. It also handles the placement and rendering of messages as well as the coordination of 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 easily noticed, easy-to-read, yet unobtrusive user notifications, took time, patience, and usability testing, but 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 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 code snippet below 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. this.options.publish(mstats.events.status, status);
  5. },
  6. 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.
  7. The following functions show how easy it is to initiate the display of a user message:
  + The **\_showDeletingMessage** function is called after the user confirms his intent to delete the vehicle. This message is intended to inform the user that the vehicle deletion has been submitted to the server.
  + The **\_showDeletedMessage** function is called after a successful deletion of the vehicle, informing the user that the deletion was successful.
  + The **\_showDeleteErrorMessage** function is called if an error occurred while deleting the vehicle.
  1. JavaScript
  2. // contained in mstats.vehicle-details.js
  3. \_showDeletingMessage: function () {
  4. this.\_publishStatus({
  5. type: 'saving',
  6. message: 'Deleting the selected vehicle ...',
  7. duration: 5000
  8. });
  9. },
  10. \_showDeletedMessage: function () {
  11. this.\_publishStatus({
  12. type: 'saved',
  13. message: 'Vehicle deleted.',
  14. duration: 5000
  15. });
  16. },
  17. \_showDeleteErrorMessage: function () {
  18. this.\_publishStatus({
  19. type: 'saveError',
  20. message: 'An error occurred deleting the selected vehicle. Please try again.',
  21. duration: 10000
  22. });
  23. }
  24. Each function creates 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.
  25. 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. Mileage Stats uses forms authentication, with a session timeout threshold of 20 minutes. If the session has timed out, the request (Ajax or non-Ajax) is redirected to the page specified by the forms authentication **loginUrl** in the web.config file.
  2. In traditional websites that perform page reloads between pages, it’s common to redirect the user to a sign-in page when their session times out. Applications like Mileage Stats that make heavy use of Ajax calls to retrieve data, perform few full-page reloads. Consequently, if a session timeout occurs, it’s usually during an Ajax request. Let's examine what happens when an Ajax request is redirected because of an authentication session timeout:
  3. Ajax JavaScript Object Notation (JSON) data request initiated.
  4. Forms authentication runtime detects an expired session and redirects the request to the sign-in page.
  5. A parsing error occurs because the Ajax handler is expecting JSON data and not HTML. The HTML is the content of the sign-in page to which the request was redirected.
  6. An Ajax error callback is invoked.
  7. A global Ajax error callback is invoked.
  8. Errors that can occur anywhere in the application can often be handled in a centralized location so that individual objects don't need to repeat the same error handling code. Mileage Stats implements the global **ajaxError** method handler shown below to catch errors occurring during an Ajax request. The primary purpose of this method in Mileage Stats is to identify whether the initiating Ajax request caused a session timeout error and, if so, redirect the user to the sign-in page.
  9. When looking at the code below, "jqXHR.status === 200" appears out of place or incorrect. Remember, this method is only executed when an Ajax error occurs. If the session times out and the request is redirected to the sign-in page, the response status code will be 200 because the redirect succeeded. In addition to checking for the response status code, this method also verifies that the returned HTML contains the sign-in page's title. If both conditions are met, the browser is redirected to the sign-in page.
  10. JavaScript
  11. // contained in mileagestats.js
  12. // setup default error handler for redirects due to session timeout.
  13. $(document).ajaxError(function (ev, jqXHR, settings, errorThrown) {
  14. if ((jqXHR.status === 200)
  15. && (jqXHR.responseText.indexOf('Mileage Stats Sign In') !== -1)) {
  16. window.location.replace(mstats.getRelativeEndpointUrl('/Auth/SignIn'));
  17. }
  18. });
  19. **Note:** If the originating Ajax calling code also implements an error handler, the originating Ajax caller's error handler will be called first, then the above global Ajax error handler will be called.

# Website Error Notification

* 1. ASP.NET provides you with 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 links to continue using the site.
  6. Mileage Stats GenericError.htm page
  7. 

# Prompting Users

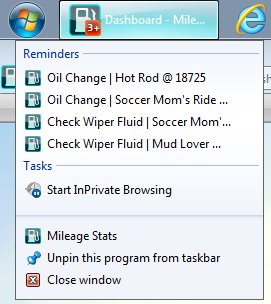
* 1. During the design phase of Project Silk, the team had a goal of not prompting users with modal dialogs. Website UX designers are getting away from modal dialogs that ask the user questions like, "Are you sure?" Instead, designers prefer an undo system, allowing users to undo the previous task. The undo feature also enhances the application by extending undo capabilities to tasks that did not require a confirmation dialog.
  2. Since Mileage Stats is only a sample application, it has limited functionality. A production application could implement this undo feature. The team preferred the undo feature, but other features took priority.

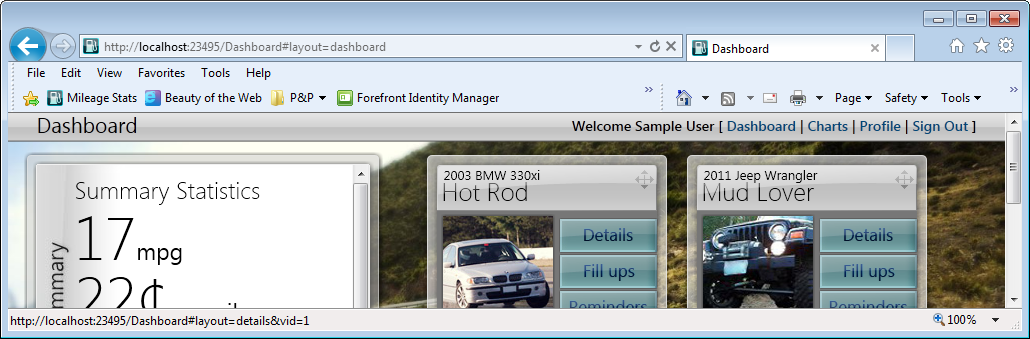
The code below uses the JavaScript **confirm** function to validate the user's request to fulfill a maintenance reminder.

* 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. },
  9. **Note:** The jQuery UI dialog provides an alternative to using the JavaScript **confirm** dialog. If you are leveraging jQuery UI plug-ins, you should consider using the jQuery UI dialog for consistency in your UI.

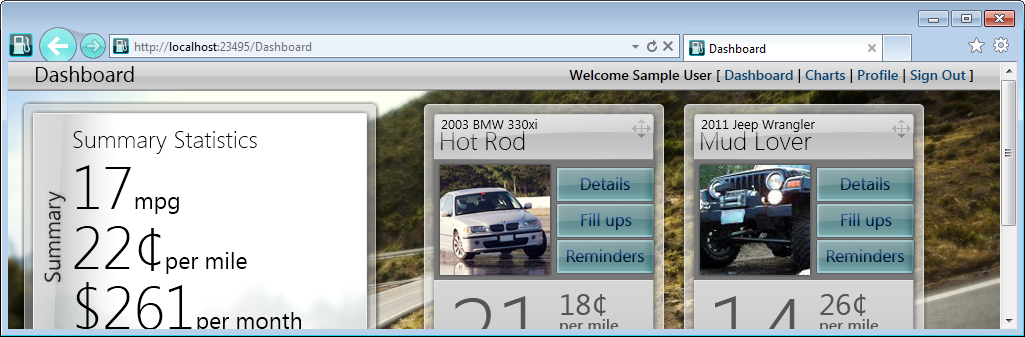
# Desktop Notifications

* 1. Given that modern web applications can provide 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 take advantage of the Microsoft® Windows® 7 taskbar capabilities and, when launched, the browser window is customized specifically for the site. The full Pinned Sites experience requires Internet Explorer 9 running on Windows 7. Windows Vista® provides fewer Pinned Sites features—site pinning, customized reduced chrome, and the disabling of browser add-ons.
  3. Mileage Stats uses Pinned Sites to provide Windows 7 taskbar notifications that indicate whether 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. **Note:** Jump list items will be available whether the site is opened in a browser or not. However, the notification icons are only displayed when the site is opened in the browser.
   2. The two images below contrast Mileage Stats running in a normal browser window and a customized Pinned Sites browser window. The lower image shows the clean, pared 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.
   3. 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 documentation for implementing Pinned Sites in their web applications on MSDN®. The title of this topic is, "Pinned Sites Developer Documentation," and is located at <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 MSDN documentation just mentioned nor cover every line of code pertaining to Pinned Sites. Instead, the Mileage Stats implementation will be explained, enabling you 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 display of 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 on whether or not the user is signed in. This initialization will be described below.

### Feature Detection

* 1. Pinned Sites feature detection is provided by the Internet Explorer 9 **msIsSiteMode** function. Verifying that the page is opened as a pinned site before executing Pinned Site API methods prevents unnecessary JavaScript errors.
  2. The **msIsSiteMode** function returns **true** if the current page is launched as a pinned site; **false** if it is not. The below **isPinned** function wraps the **msIsSiteMode** call and returns **false** if the page is 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 landing page, which is shown below. 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. It will also show and hide the text as the user moves her mouse over or away from the Pinned Sites icon.
  2. **Note:** Developers are not required to implement a draggable site icon as Mileage Stats does to enable site pinning. Providing a draggable icon allows the website more control over the pinning experience.
  3. Without a draggable icon, sites can still be pinned by dragging the tab or the favicon to the taskbar.
  4. Landing page
  5. 
  6. The Pinned Sites JavaScript object is initialized when the above page loads with the below JavaScript function.
  7. CSHTML
  8. // Contained in Index.cshtml
  9. <script>
  10. $(function () {
  11. mstats.pinnedSite.intializePinndedSiteImage();
  12. });
  13. </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 so that the icon appears 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 HTML snippet below shows the required **msPinSite** class applied to the Pinned Sites icon. This class is used by Internet Explorer 9 to enable the user to drag the Pinned Sites icon to the taskbar or Start Menu and pin the site.

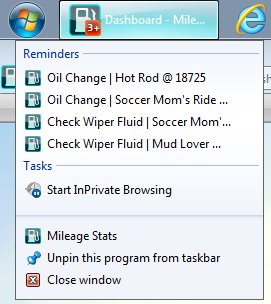
* 1. CSHTML
  2. // Contained in Index.cshtml
  3. <img id="pinnedSiteImage" class="msPinSite" ... />

To call the user's attention to the draggable Pinned Sites icon, the **active** CSS class below 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. The 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 users click 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 invoked 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);

The **initializeData** function saves the **sendRequestFunc** 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 (sendRequestFunc) {
  4. sendRequest = sendRequestFunc;
  5. mstats.pinnedSite.requeryJumpList();
  6. },

The below **requeryJumpList** function is called when the **pinnedSite** object is initialized and called 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.

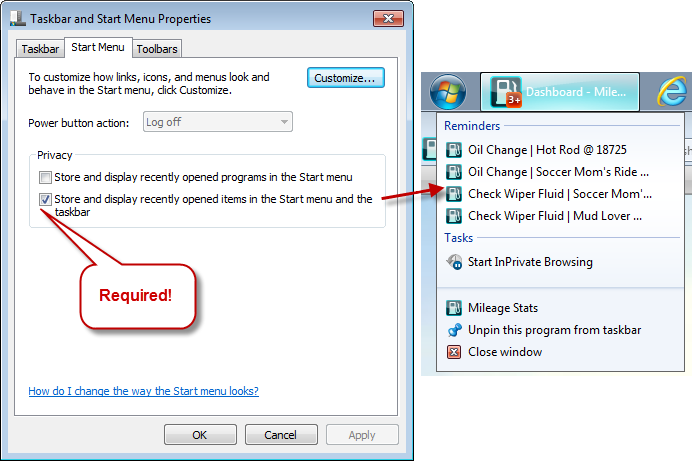
All of the below **msSite** functions are provided by Internet Explorer 9. After using feature detection to determine if the site is pinned, the jump list and overlay icon are cleared, and a new jump list is created.

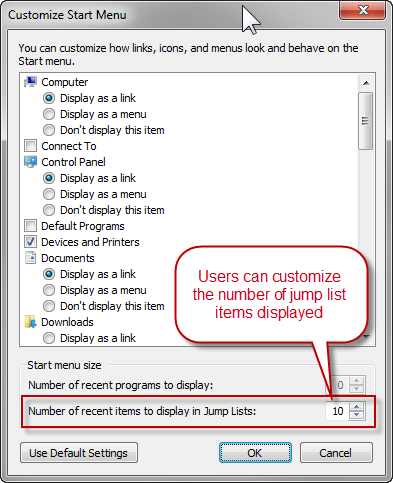
Not shown below is the Ajax call to the data manager to get the array of overdue reminders. If that Ajax request is successful and the **data.Reminders** array has data, a URL will be constructed for each data item and added to the jump list. Next, the appropriate overlay icon is set. Finally, **msSiteModeShowJumpList** 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, you can deliver dynamic desktop notifications in your 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 your website implements the jump list feature, you should provide this information to your users and advise them that the "Store and display recently opened items in the Start Menu and the taskbar" property setting needs to be checked for the jump list items to show up.
  3. Taskbar and Start Menu properties
  4. 
  5. In addition be being able to disable jump list items, users can customize the number of jump list items displayed on their computers. The default value is 10 and can be changed in the Customize Start Menu dialog below. This dialog is opened by clicking the **Customize** button in the Taskbar and Start Menu Properties dialog shown above.
  6. Customizing Start Menu Properties



# Summary

* 1. Providing timely feedback that is uniformly displayed, context sensitive, and understandable to your users without breaking their workflow takes planning by designers and developers alike. Your users will appreciate this extra effort, which results in a polished user experience. By encapsulating the display and management of user notifications in a single JavaScript object, your application will be easier to code, maintain over time, and test. You have also learned about integrating your website with the Windows 7 desktop to provide users with dynamic notifications and jump list items, as well as browsing your site using a customized browser window.

# Further Reading

* 1. For a comprehensive look at input validation error messages, see Chapter 10, "Data, Caching, and Validation."
  2. For detailed information on the inner working of the Mileage Stats pub/sub implementation, see Chapter 8, "Communication."
  3. For more information about the **isFunction** method, see jQuery.isFunction():  
     <http://api.jquery.com/jQuery.isFunction/>
  4. For more information about Pinned Sites, see the Pinned Sites developer documentation:  
     <http://msdn.microsoft.com/en-us/library/gg491731(v=VS.85).aspx>