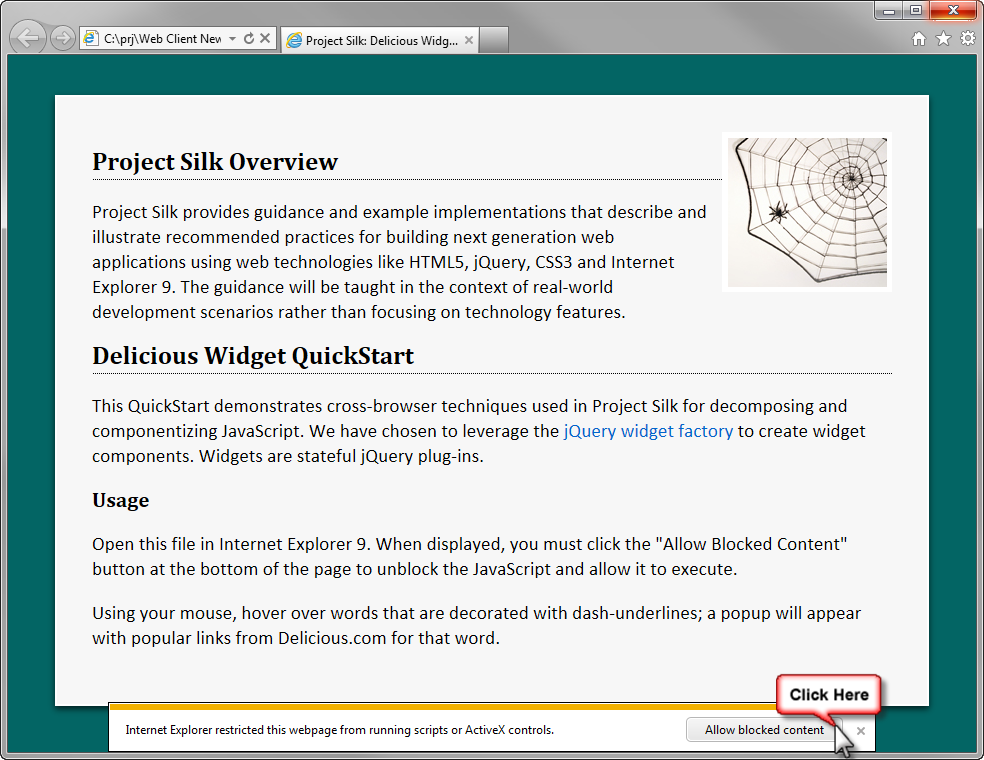
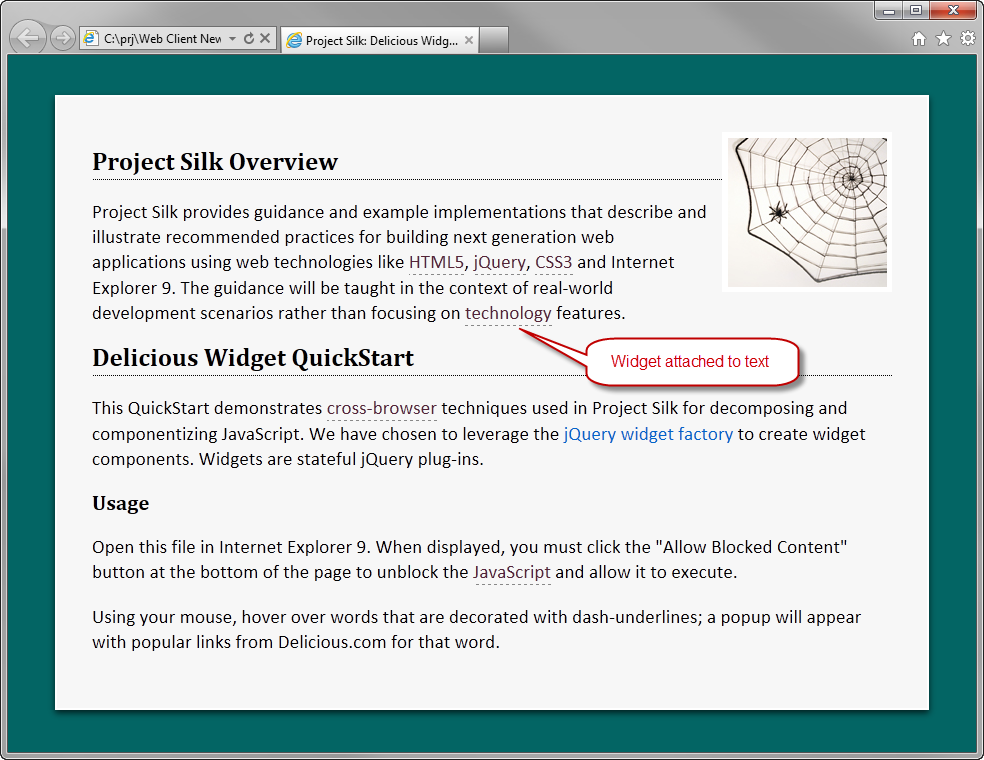
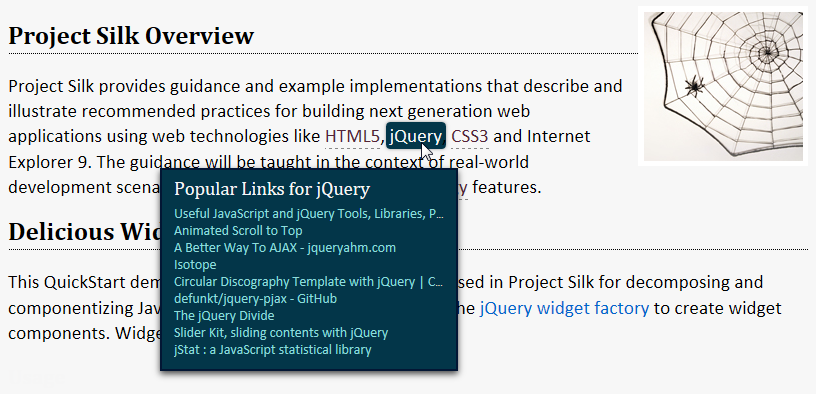
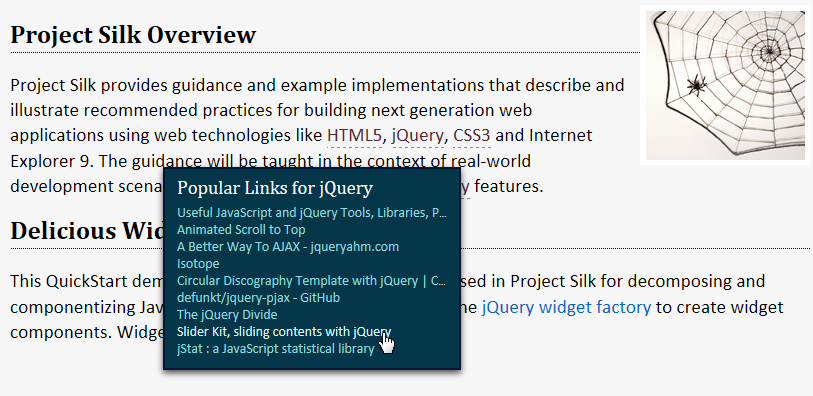
Widget QuickStart

* 1. This Widget QuickStart illustrates the way Project Silk uses the [jQuery UI Widget Factory](http://docs.jquery.com/UI/Developer_Guide) to create maintainable widgets that implement client-side behavior.

# Business Scenario

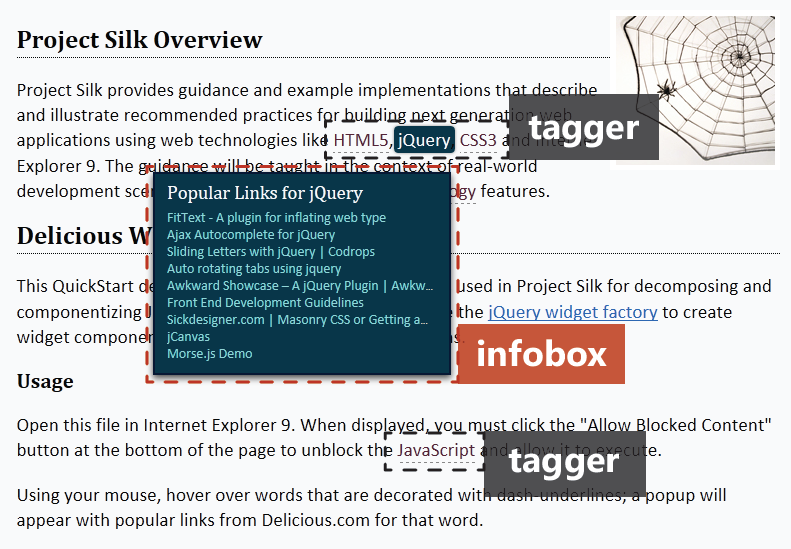
* 1. Our team has been asked to enable cross-browser keyword lookup capabilities in our web pages by hyperlinking select keywords to popular websites. This feature will need to be added dynamically to all company web pages.
  2. Another team has been tasked with tagging the keywords in the web pages. The words will be tagged dynamically, based on server-side business logic driven by agreements with third parties.
  3. The focus of this QuickStart is to enable the client-side behavior for the tagged keywords. When a user hovers over a keyword, the browser will display a pop-up list of popular links for that keyword from the Delicious.com bookmarking service.

# Walkthrough

* 1. To interact with the completed scenario, ensure you have an Internet connection and follow the steps below:
  2. Open the **default.htm** file using Microsoft Internet Explorer 9. After the file's content is displayed, you'll need to click on the **Allow blocked content** button at the bottom of the browser window to enable scripts to run. Blocking active content by default is a security feature of Internet Explorer 9.
     1. Widget QuickStart (default.htm)
     2. 
  3. After allowing blocked content, you'll notice that the keywords are displayed in a new color and have been underlined with a dashed line, as pictured below.
     1. Widget QuickStart after scripts are unblocked
     2. 
  4. Using your mouse, hover over an underlined keyword. A pop-up list with the top-10 most popular links for that keyword will be displayed. Notice that the keyword has been repeated in the title of the pop-up list.
     1. One second after moving your mouse away from the keyword, the pop-up list will close unless your mouse is within the boundaries of the pop-up list.
     2. If the keyword is on the left side of the page, the pop-up list will open to the right of the cursor. If the keyword is on the right side of the page, the pop-up list will open to the left side of the cursor, as in the image below.
        1. Pop-up list for the keyword "jQuery"
        2. 
  5. Move your mouse over the pop-up list. You can now click on a link, which will open in a new browser window.
     1. Links from Delicious.com in the pop-up list
     2. 
  6. Moving your mouse outside the boundaries of the pop-up list will cause the pop-up list to close.

# Conceptual View

This section illustrates the relationship of the jQuery UI widgets to the HTML page. A single **infobox** widget is attached to the page's **body** element. After it's attached, it creates a <div> element and dynamically adds it to the page's <body> element. Additionally, a **tagger** widget is attached to each keyword.

* 1. Relationship of the jQuery UI widgets to the HTML page
  2. 

The HTML below reveals a keyword tagging strategy that takes advantage of HTML5 data attributes. Each of the keywords has been wrapped in a **span** tag with the **data-tag** attribute applied. For this scenario, the keyword wrapping was accomplished on the server side.

* 1. HTML
  2. <!-- Contained in default.htm -->
  3. <!DOCTYPE html>
  4. <html>
  5. <head ...>
  6. <body>
  7. <div id="container">
  8. <img src="projectsilk.png" />
  9. <h1>Project Silk Overview</h1>
  10. <p>
  11. Project Silk provides guidance and example implementations
  12. that describe and illustrate recommended practices for
  13. building next generation web applications using web
  14. technologies like <span data-tag>HTML5</span>,
  15. <span data-tag>jQuery</span>, <span data-tag>CSS3</span>
  16. and Internet Explorer 9. The guidance will be taught in
  17. the context of real-world development scenarios rather
  18. than focusing on <span data-tag>technology</span>
  19. features.</p>

# Attaching Widgets

Once created, the widget is attached to an HTML element and its **options** can be set.

* 1. JavaScript
  2. // Contained in startup.js
  3. (function ($) {
  4. var infobox = $('body').infobox({
  5. dataUrl: 'http://feeds.delicious.com/v2/json/popular/'
  6. });
  7. $('span[data-tag]').tagger({
  8. activated: function (event, data) {
  9. infobox.infobox('displayTagLinks', event, data.name);
  10. },
  11. deactivated: function () {
  12. infobox.infobox('hideTagLinks');
  13. }
  14. });
  15. } (jQuery));

The code above demonstrates the **infobox** widget being attached to the **body** element. The **dataUrl** **option** value will be used when performing popular keyword link lookups.

The jQuery selector **span[data-tag]** returns a jQuery wrapped set that contains all **span** tags with a **data-tag** attribute. A **tagger** widget will be attached to each of the **span** tags in the returned collection. The **tagger** widget has **activated** and **deactivated** options that are used as callbacks. These callbacks are used to handle events raised when the mouse hovers over the tag.

# Widget Initialization

When a widget is created (attached), the jQuery UI widget factory will call the private method **\_create**. This method provides the developer an opportunity to perform widget setup actions. Examples include building and injecting markup, adding CSS classes, binding events, and so forth.

* 1. JavaScript
  2. // Contained in jquery.qs.infobox.js
  3. \_create: function () {
  4. var that = this,
  5. name = that.name;
  6. that.infoboxElement = $('<div class="qs-infobox" />');
  7. that.infoboxElement.appendTo('body')
  8. .bind('mouseenter.' + name, function () {
  9. mouseOverBox = true;
  10. })
  11. .bind('mouseleave.' + name, function () {
  12. mouseOverBox = false;
  13. that.hideTagLinks();
  14. });
  15. },

1. The code snippet above first creates a variable for **this** called **that** within the closure, so the widget can be referenced within the **mouseenter** and **mouseleave** event handlers.
2. Recall that the **infobox** widget is attached to the **body** element. The element **div.qs-infobox** will contain the UI for this widget. It is stored in **that.infoboxElement**, attached to the **body** element, and bound to some events. The **name** variable holds the name of the widget and is appended to the name of the event it's binding to. This is a recommended practice when using jQuery; the reasons why will be explained later in the QuickStart.
   1. **Note**: Most of the time, widgets are attached to the element that they will control; however, there are times when a widget will need to create additional elements.

In the above **\_create** function, the **infobox** widget creates a **div** to hold the list of links. The default.htm HTML page could have been modified to include the **div** in the first place, making it unnecessary for the widget to add an additional structure. However, the code was written this way to illustrate a widget adding UI elements to an existing HTML structure.

# Widget Interactions

* 1. An interesting challenge in this scenario is giving the user enough time to click the links without showing the pop-up list longer than needed. The implementation requires coordination between the two widgets.

## Mouse Entering a Keyword Span

* 1. When the mouse enters the keyword span, the **mouseenter** event handler in the **tagger** widget is invoked. The **name** being appended to the event name is the name of the widget and is used as a namespace for the event binding. This is a recommended practice. Any string can be used as the namespace, but using the name of the widget allows you to tap into a feature of the widget factory described later in the QuickStart.
  2. JavaScript
  3. // Contained in jquery.qs.tagger.js
  4. .bind('mouseenter.' + name, function (event) {
  5. clearTimeout(timer);
  6. that.\_trigger('activated', event, {name: tag});

})

* 1. The **clearTimeout** call uses the **timer** variable, which is defined outside of the widget prototype and set in the handler for **mouseleave**, discussed in the next section. This means there will be only one timer created and shared among all instances of the **tagger** widget.
  2. The next line raises the **tagactivated** event. It doesn't raise the **taggeractivated** event because the widget sets the **widgetEventPrefix** property, as shown in the next code snippet. It also doesn't raise the **activated** event as you may have suspected because the widget factory changes the name of raised events by prepending the name of the widget to the name of the event being triggered.
  3. JavaScript

// Contained in jquery.qs.tagger.js

$.widget('qs.tagger', {

* 1. widgetEventPrefix: 'tag',
  2. options: {
  3. When the **tagactivated** event is raised, the **displayTagLinks** method is called on the **infobox** widget. As you will notice from having a look at **jquery.qs.infobox.js**, it never binds to this event. Doing so would create a dependency between the widgets. A better option is to follow a recommended pattern and take advantage of a related jQuery UI feature. It is recommended that a widget provide callback options for all of the events it raises.
  4. JavaScript

// Contained in jquery.qs.tagger.js

options: {

* 1. activated: null,
  2. deactivated: null
  3. },

The jQuery UI widget factory will automatically call any option with the same name as the event being raised. This feature allows the event handlers to be associated by setting the value of the option. The QuickStart does this in the startup file.

* 1. JavaScript
  2. // Contained in startup.js
  3. $('span[data-tag]').tagger({
  4. activated: function (event, data) {
  5. infobox.infobox('displayTagLinks', event, data.name);
  6. },
  7. deactivated: function () {
  8. infobox.infobox('hideTagLinks');
  9. }
  10. });
  11. This approach is also a nice way to avoid having to know if the event is called **tagactivated** or **taggeractivated** or something else. The **displayTagLinks** method accepts a browser event and the name to look up. The first part of the method sets up enclosed variables to be used in the second part of the method.
  12. JavaScript
  13. // Contained in jquery.qs.infobox.js
  14. displayTagLinks: function (event, tagName) {
  15. var i,
  16. html,
  17. that = this,
  18. options = that.options,
  19. elem = that.infoboxElement,
  20. top = event.pageY + offsetY,
  21. left = event.pageX + offsetX,
  22. url = options.dataUrl + tagName + '?count=' + options.maxItems,
  23. displayResult = function () {
  24. elem.html(html);
  25. elem.css({top: top, left: left});
  26. elem.show();
  27. };
  29. if (event.pageX > window.screenWidth / 2) {
  30. left = event.pageX + leftSideAdjustment;
  31. }
  32. After the closure is prepared, **left** is adjusted in case the tag is on the right-hand side of the page. The second part of the **displayTagLinks** method is an Ajax call to the **url**, constructed above, for the Delicious bookmarking service.
  33. JavaScript
  34. // Contained in jquery.qs.infobox.js
  35. $.ajax({
  36. url: url,
  37. dataType: 'jsonp',
  38. success: function (data) {
  39. if (data != null) {
  40. html = '<h1>Popular Links for ' + tagName + '</h1><ul>';
  41. for (i = 0; i < data.length - 1; i += 1) {
  42. html += '<li><a href="' +
  43. data[i].u +
  44. '" target="\_blank">' +
  45. data[i].d + '</a></li>';
  46. }
  47. html += '</ul>';
  48. } else {
  49. html = '<h1>Data Error</h1><p>[snipped]</p>';
  50. }
  51. displayResult();
  52. },
  53. error: function (jqXHR, textStatus, errorThrown) {
  54. html = '<h1>Ajax Error</h1>' +
  55. '<p>The Ajax call returned the following error: ' +
  56. jqXHR.statusText + '.</p>';
  57. displayResult();
  58. }

});

* 1. The local **displayResult** function is scoped only to the **displayTagLinks** method since it was needed for both **success** and **error** conditions and nowhere else. This is the method that applies the result to the element for the user to see.

## Mouse Leaving a Keyword Span

* 1. When the mouse leaves the tag's **span**, a similar coordination occurs. The **tagger** widget has a namespaced event bound to the span's **mouseleave** event.
  2. JavaScript
  3. // Contained in jquery.qs.tagger.js
  4. .bind('mouseleave.' + name, function () {
  5. timer = setTimeout(function () {
  6. that.\_trigger('deactivated');
  7. }, hideAfter);
  8. });
  9. The **timer** is set to raise the **tagdeactivated** event after 1000 milliseconds, which is the value of **hideAfter**.
  10. When the **tagger** widget was applied to the span elements, a function was supplied to the **deactivated** callback, as you also saw earlier in the QuickStart.
  11. JavaScript
  12. // Contained in startup.js
  13. $('span[data-tag]').tagger({
  14. activated: function (event, data) {
  15. infobox.infobox('displayTagLinks', event, data.name);
  16. },
  17. deactivated: function () {
  18. infobox.infobox('hideTagLinks');
  19. }
  20. });
  21. The function invokes the **hideTagLinks** method on the **infobox** widget.
  22. JavaScript

// Contained in jquery.qs.infobox.js

hideTagLinks: function () {

* 1. !mouseOverBox && this.infoboxElement.hide();
  2. },
  3. The **infobox** is only hidden if the mouse is not over it. Effectively, the 1000 ms delay provides the user time to move the mouse from the keywords to the links.

## Mouse Entering the Pop-up List

* 1. Internally, the **infobox** widget uses the **mouseOverBox** variable to maintain state indicating whether or not the mouse is over the pop-up list. This variable is defined in the closure created by the self-executing anonymous function wrapping the file.
  2. JavaScript
  3. // Contained in jquery.qs.infobox.js

(function ($) {

* 1. var offsetX = 20,
  2. offsetY = 20,
  3. mouseOverBox = false,
  4. leftSideAdjustment = -270;
  5. $.widget('qs.infobox', {
  6. When the mouse enters the **infobox**, **mouseOverBox** is set to **true**.
  7. JavaScript
  8. // Contained in jquery.qs.infobox.js: \_create
  9. .bind('mouseenter.' + name, function () {
  10. mouseOverBox = true;

})

## Mouse Leaving the Pop-up List

* 1. When the mouse leaves the pop-up list, **mouseOverBox** is set to **false** and **hideTagLinks** is invoked.
  2. JavaScript
  3. // Contained in infobox.js
  4. .bind('mouseleave.' + name, function () {
  5. mouseOverBox = false;
  6. that.hideTagLinks();
  7. });
  8. hideTagLinks: function () {
  9. !mouseOverBox && this.infoboxElement.hide();
  10. },

# Further Reading

* 1. You may find the following links useful in your investigation of the jQuery UI widget factory:
  + [jQuery UI API Developer Guide](http://jqueryui.com/docs/Developer_Guide)
  + [Widget factory](http://wiki.jqueryui.com/w/page/12138135/Widget-factory) on the jQuery UI wiki
  + [Tips for Developing jQuery UI 1.8 Widgets](http://www.erichynds.com/jquery/tips-for-developing-jquery-ui-widgets/) on Eric Hynds' blog
  + [Understanding jQuery UI widgets: A tutorial](http://bililite.com/blog/understanding-jquery-ui-widgets-a-tutorial/) on bililite.com