Tutorial: Getting Started with SignalR Asp.net 4.5

This tutorial shows how to use SignalR to create a real-time chat application. You will add SignalR to an empty ASP.NET web application and create an HTML page to send and display messages.

Overview

This tutorial introduces SignalR development by showing how to build a simple browser-based chat application. You will add the SignalR library to an empty ASP.NET web application, create a hub class for sending messages to clients, and create an HTML page that lets users send and receive chat messages. For a similar tutorial that shows how to create a chat application in MVC 4 using an MVC view, see Getting Started with SignalR and MVC 4.

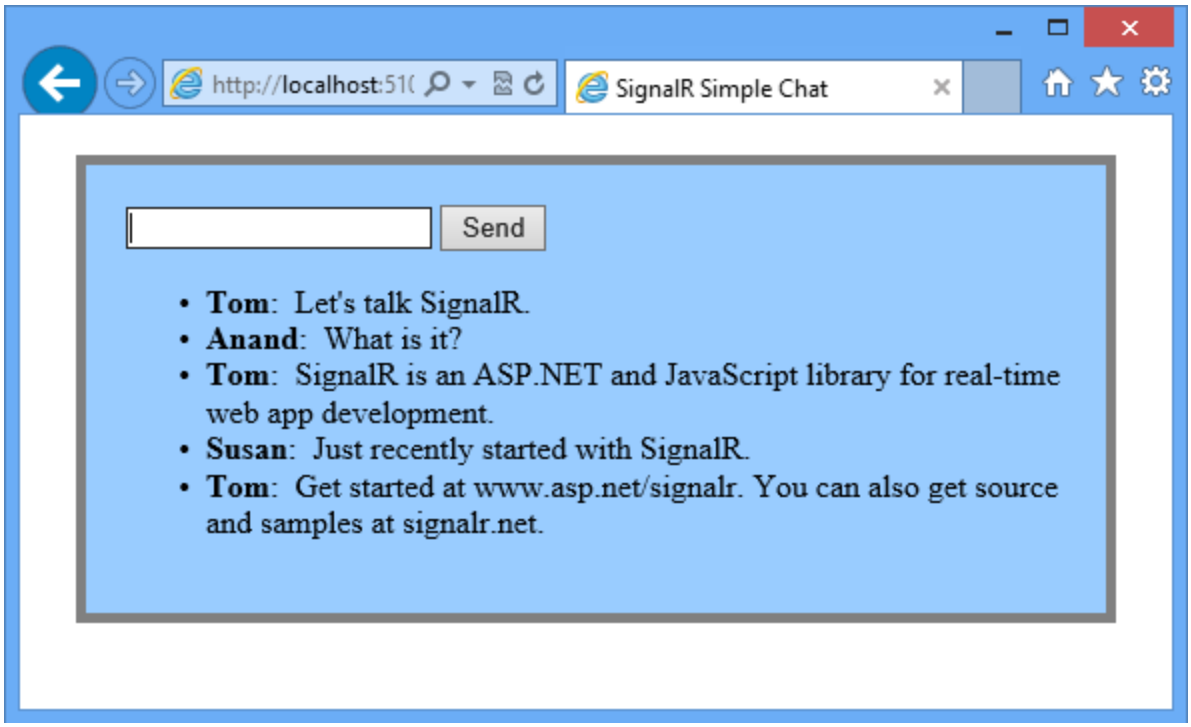
SignalR is an open-source .NET library for building web applications that require live user interaction or real-time data updates. Examples include social applications, multiuser games, business collaboration, and news, weather, or financial update applications. These are often called real-time applications.

SignalR simplifies the process of building real-time applications. It includes an ASP.NET server library and a JavaScript client library to make it easier to manage client-server connections and push content updates to clients. You can add the SignalR library to an existing ASP.NET application to gain real-time functionality.

The tutorial demonstrates the following SignalR development tasks:

- Adding the SignalR library to an ASP.NET web application.
- Creating a hub class to push content to clients.
- Using the SignalR jQuery library in a web page to send messages and display updates from the hub.

The following screen shot shows the chat application running in a browser. Each new user can post comments and see comments added after the user joins the chat.



Sections:

Set up the Project

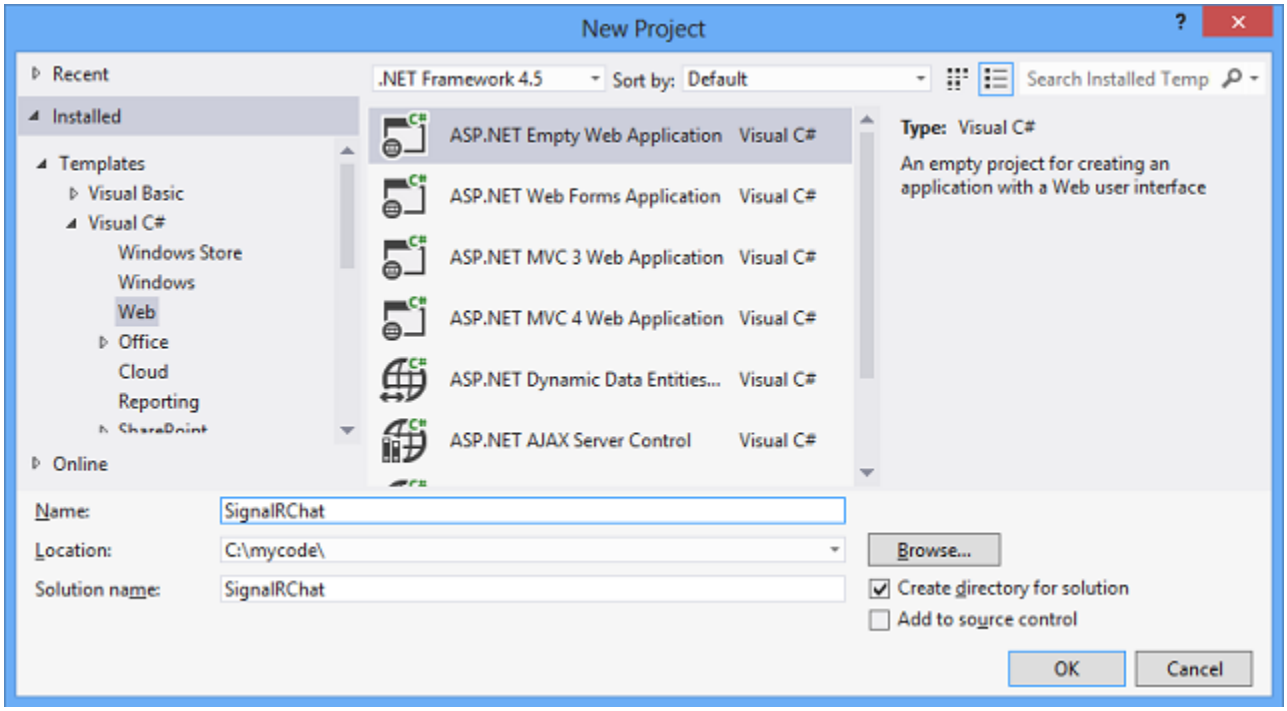
This section shows how to create an empty ASP.NET web application, add SignalR, and create the chat application.

Prerequisites:

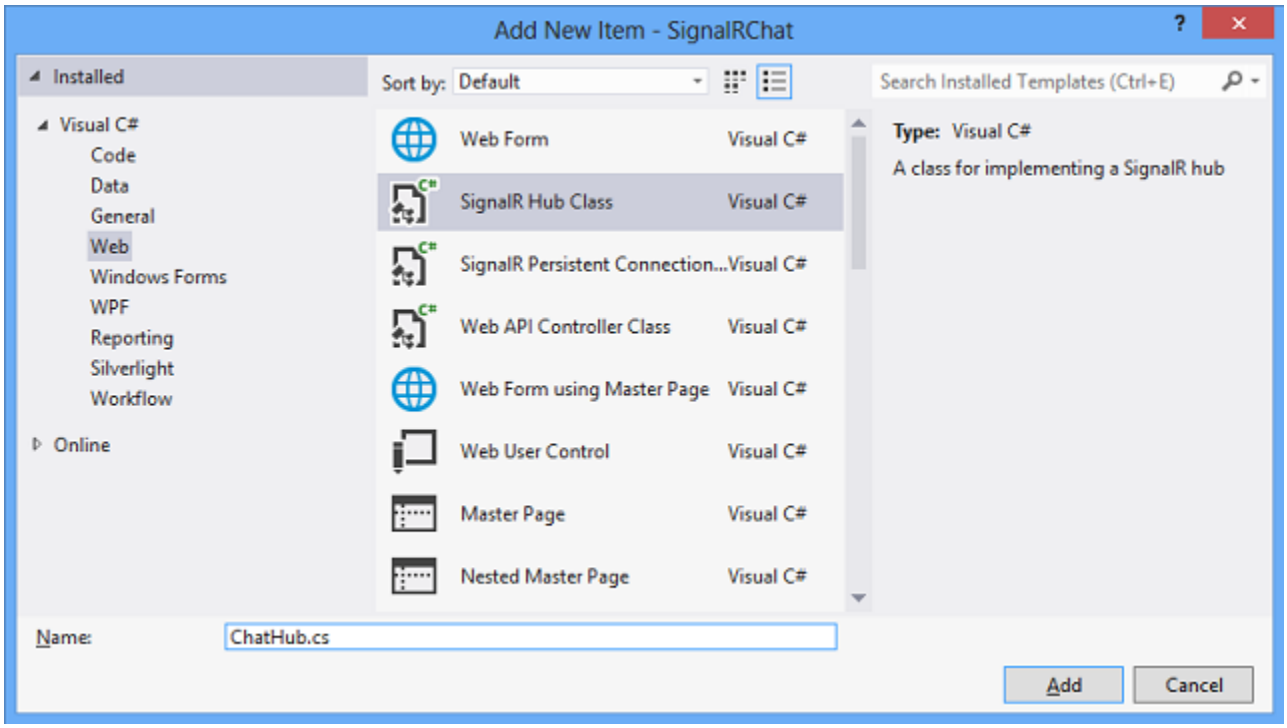
- Visual Studio 2010 SP1 or 2012. If you do not have Visual Studio, see ASP.NET Downloads to get the free Visual Studio 2012 Express Development Tool.
- Microsoft ASP.NET and Web Tools 2012.2. For Visual Studio 2012, this installer adds new ASP.NET features including SignalR templates to Visual Studio. For Visual Studio 2010 SP1, an installer is not available but you can complete the tutorial by installing the SignalR NuGet package as described in the setup steps.

The following steps use Visual Studio 2012 to create an ASP.NET Empty Web Application and add the SignalR library:

1. In Visual Studio create an ASP.NET Empty Web Application.

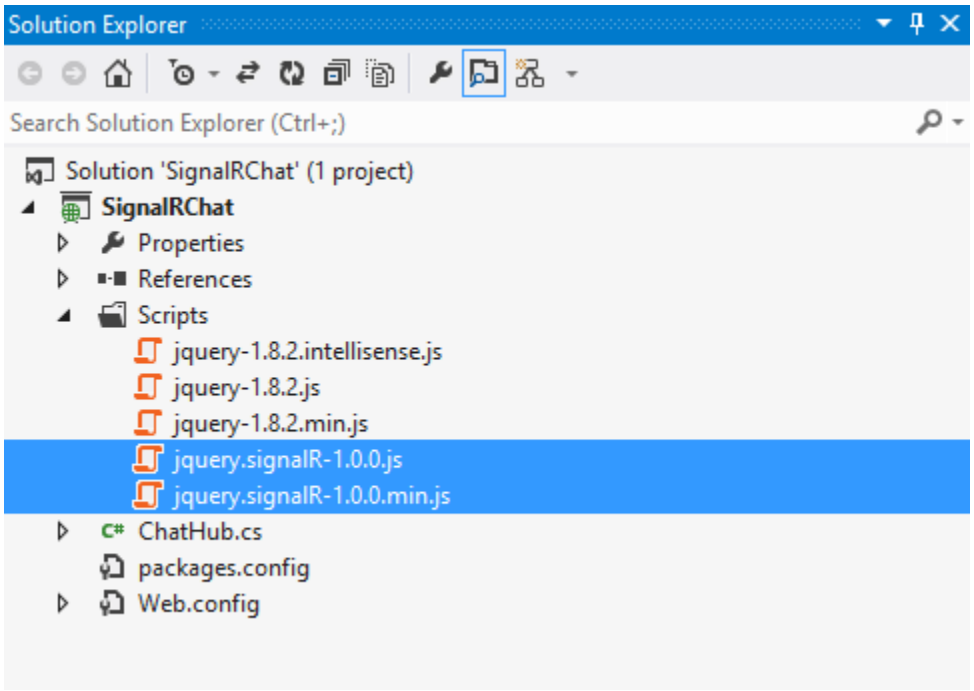


2. In **Solution Explorer**, right-click the project, select **Add | New Item**, and select the **SignalR Hub Class** item. Name the class **ChatHub.cs** and add it to the project. This step creates the **ChatHub** class and adds to the project a set of script files and assembly references that support SignalR.



Note: You can also add SignalR to a project by opening the **Tools | Library Package Manager | Package Manager Console** and running a command: `install-package Microsoft.AspNet.SignalR`. If you use the console to add SignalR, create the SignalR hub class as a separate step after you add SignalR.

3. In **Solution Explorer** expand the Scripts node. Script libraries for jQuery and SignalR are visible in the project.

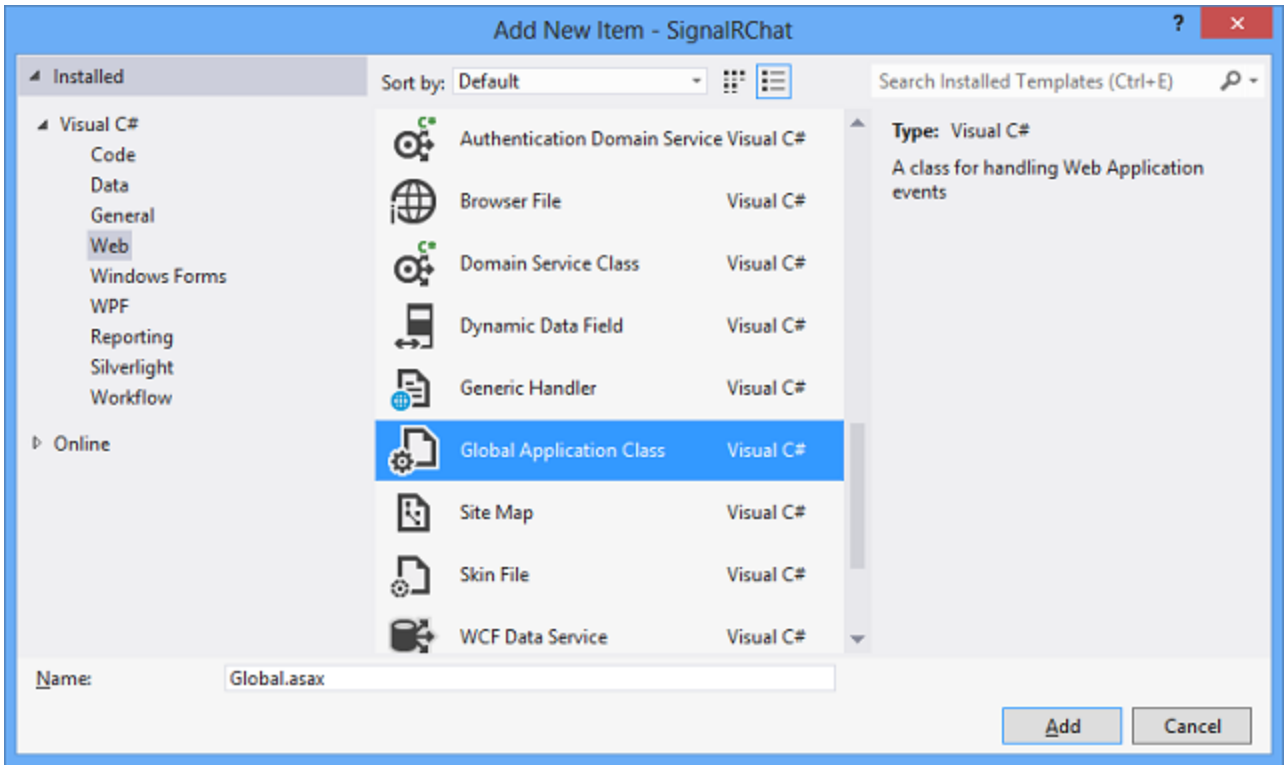


4. Replace the code in the new **ChatHub** class with the following code.

```
using System;

using System.Web;

using Microsoft.AspNet.SignalR;

namespace SignalRChat

{
```

```
11.     public class ChatHub : Hub
12.     {
13.         public void Send(string name, string message)
14.         {
15.             // Call the broadcastMessage method to update clients.
16.             Clients.All.broadcastMessage(name, message);
17.         }
18.     }
19. }
```

20. In **Solution Explorer**, right-click the project, then click **Add | New Item**. In the **Add New Item** dialog, select **Global Application Class** and click **Add**.



21. Add the following `using` statements after the provided `using` statements in the Global.asax.cs class.

```
using System.Web.Routing;

using Microsoft.AspNet.SignalR;
```

24. Add the following line of code in the `Application_Start` method of the Global class to register the default route for SignalR hubs.

```
// Register the default hubs route: ~/signalr/hubs

RouteTable.Routes.MapHubs();
```

27. In **Solution Explorer**, right-click the project, then click **Add | New Item**. In the **Add New Item** dialog, select Html Page and click **Add**.

28. In **Solution Explorer**, right-click the HTML page you just created and click **Set as Start Page**.

29. Replace the default code in the HTML page with the following code.

```
<!DOCTYPE html>
<html>
<head>
    <title>SignalR Simple Chat</title>
    <style type="text/css">
        .container {
            background-color: #99CCFF;
            border: thick solid #808080;
            padding: 20px;
            margin: 20px;
        }
    </style>
</head>
<body>
    <div class="container">
        <input type="text" id="message" />
```

```
46.         <input type="button" id="sendmessage" value="Send" />
47.         <input type="hidden" id="displayname" />
48.         <ul id="discussion">
49.         </ul>
50.     </div>
51.     <!--Script references. -->
52.     <!--Reference the jQuery library. -->
53.     <script src="/Scripts/jquery-1.8.2.min.js" ></script>
54.     <!--Reference the SignalR library. -->
55.     <script src="/Scripts/jquery.signalR-1.0.0.js"></script>
56.     <!--Reference the autogenerated SignalR hub script. -->
57.     <script src="/signalr/hubs"></script>
58.     <!--Add script to update the page and send messages.-->
59.     <script type="text/javascript">
60.         $(function () {
61.             // Declare a proxy to reference the hub.
```

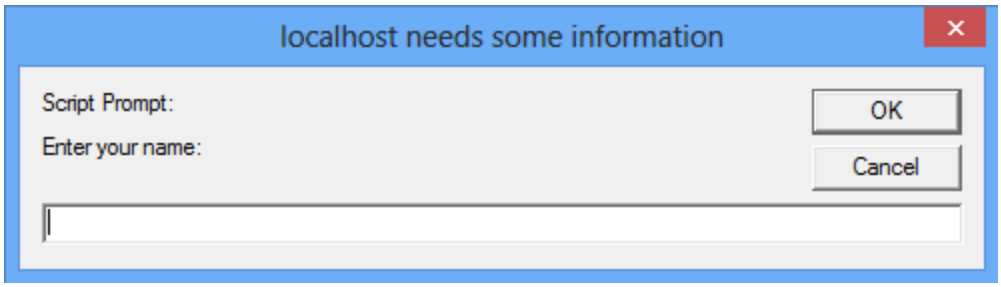
```
62.             var chat = $.connection.chatHub;
63.             // Create a function that the hub can call to broadcast messages.
64.             chat.client.broadcastMessage = function (name, message) {
65.                 // Html encode display name and message.
66.                 var encodedName = $('<div />').text(name).html();
67.                 var encodedMsg = $('<div />').text(message).html();
68.                 // Add the message to the page.
69.                 $('#discussion').append('<li><strong>' + encodedName
70.                     + '</strong>:  ' + encodedMsg + '</li>');
71.             };
72.             // Get the user name and store it to prepend to messages.
73.             $('#displayname').val(prompt('Enter your name:', ''));
74.             // Set initial focus to message input box.
75.             $('#message').focus();
76.             // Start the connection.
77.             $.connection.hub.start().done(function () {
```

```
78.                 $('#sendmessage').click(function () {
79.                     // Call the Send method on the hub.
80.                     chat.server.send($('#displayname').val(),
   $('#message').val());
81.                     // Clear text box and reset focus for next comment.
82.                     $('#message').val('').focus();
83.                 });
84.             });
85.         });
86.     </script>
87. </body>
88. </html>
```

89. **Save All** for the project.

Run the Sample

1. Press F5 to run the project in debug mode. The HTML page loads in a browser instance and prompts for a user name.

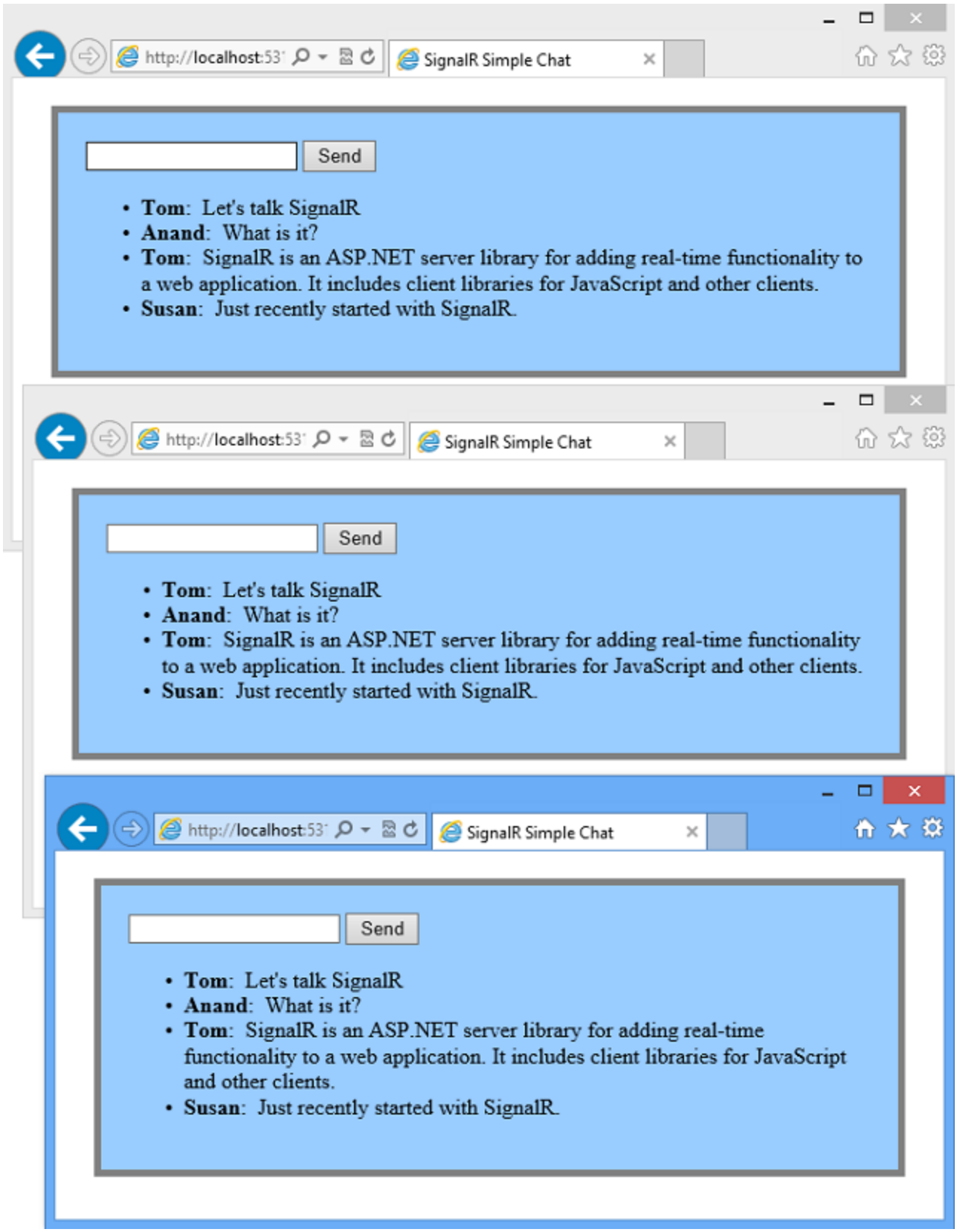


2. Enter a user name.

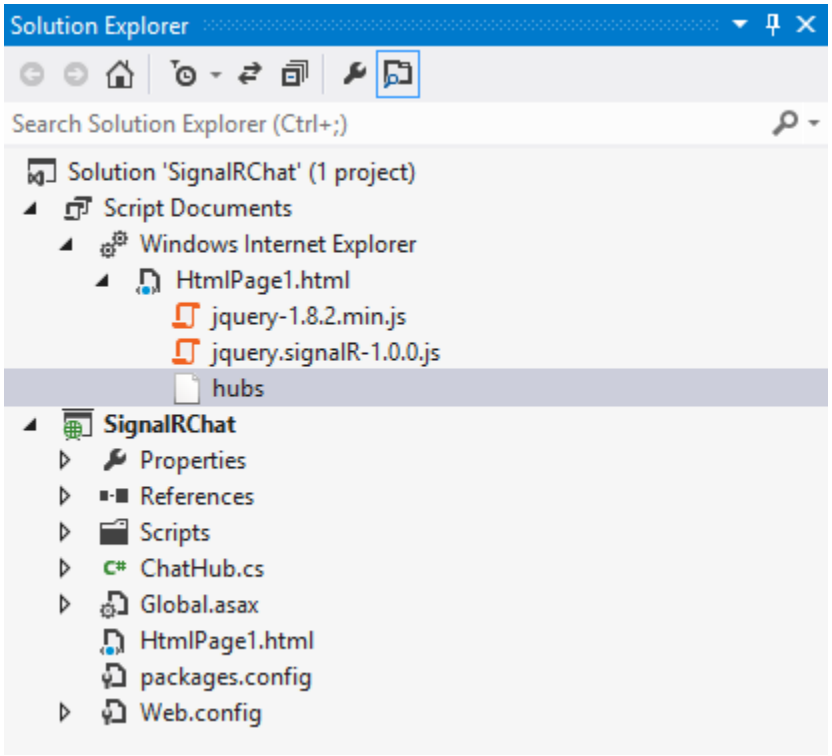
3. Copy the URL from the address line of the browser and use it to open two more browser instances. In each browser instance, enter a unique user name.
4. In each browser instance, add a comment and click **Send**. The comments should display in all browser instances.

   Note: This simple chat application does not maintain the discussion context on the server. The hub broadcasts comments to all current users. Users who join the chat later will see messages added from the time they join.

   The following screen shot shows the chat application running in three browser instances, all of which are updated when one instance sends a message:


http://localhost:53
SignalR Simple Chat
Send
Tom: Let's talk SignalR
Anand: What is it?
Tom: SignalR is an ASP.NET server library for adding real-time functionality to a web application. It includes client libraries for JavaScript and other clients.
Susan: Just recently started with SignalR.
http://localhost:53
SignalR Simple Chat
Send
Tom: Let's talk SignalR
Anand: What is it?
Tom: SignalR is an ASP.NET server library for adding real-time functionality to a web application. It includes client libraries for JavaScript and other clients.
Susan: Just recently started with SignalR.
http://localhost:53
SignalR Simple Chat
Send
Tom: Let's talk SignalR
Anand: What is it?
Tom: SignalR is an ASP.NET server library for adding real-time functionality to a web application. It includes client libraries for JavaScript and other clients.
Susan: Just recently started with SignalR.

5. In **Solution Explorer**, inspect the **Script Documents** node for the running application. There is a script file named **hubs** that the SignalR library dynamically generates at runtime. This file manages the communication between jQuery script and server-side code.



Examine the Code

The SignalR chat application demonstrates two basic SignalR development tasks: creating a hub as the main coordination object on the server, and using the SignalR jQuery library to send and receive messages.

SignalR Hubs

In the code sample the **ChatHub** class derives from the **Microsoft.AspNet.SignalR.Hub** class. Deriving from the **Hub** class is a useful way to build a SignalR application. You can create public methods on your hub class and then access those methods by calling them from jQuery scripts in a web page.

In the chat code, clients call the **ChatHub.Send** method to send a new message. The hub in turn sends the message to all clients by calling **Clients.All.broadcastMessage**.

The **Send** method demonstrates several hub concepts :

- Declare public methods on a hub so that clients can call them.

- Use the **Microsoft.AspNet.SignalR.Hub.Clients** dynamic property to access all clients connected to this hub.
- Call a jQuery function on the client (such as the `broadcastMessage` function) to update clients.

```
public class ChatHub : Hub
{
    public void Send(string name, string message)
    {
        // Call the broadcastMessage method to update clients.
        Clients.All.broadcastMessage(name, message);
    }
}
```

SignalR and jQuery

The HTML page in the code sample shows how to use the SignalR jQuery library to communicate with a SignalR hub. The essential tasks in the code are declaring a proxy to reference the hub, declaring a function that the server can call to push content to clients, and starting a connection to send messages to the hub.

The following code declares a proxy for a hub.

```
var chat = $.connection.chatHub;
```

Note: In jQuery the reference to the server class and its members is in camel case. The code sample references the C# **ChatHub** class in jQuery as **chatHub**.

The following code is how you create a callback function in the script. The hub class on the server calls this function to push content updates to each client. The two lines that HTML encode the content before displaying it are optional and show a simple way to prevent script injection.

```
chat.client.broadcastMessage = function (name, message) {

    // Html encode display name and message.

    var encodedName = $('<div />').text(name).html();

    var encodedMsg = $('<div />').text(message).html();

    // Add the message to the page.

    $('#discussion').append('<li><strong>' + encodedName

        + '</strong>:  ' + encodedMsg + '</li>');

};
```

The following code shows how to open a connection with the hub. The code starts the connection and then passes it a function to handle the click event on the **Send** button in the HTML page.

Note: This approach insures that the connection is established before the event handler executes.

```
$.connection.hub.start().done(function () {

    $('#sendmessage').click(function () {

        // Call the Send method on the hub.

        chat.server.send($('#displayname').val(), $('#message').val());

        // Clear text box and reset focus for next comment.
```

```
            $('#message').val('').focus();

        });

    });
```

Next Steps

You learned that SignalR is a framework for building real-time web applications. You also learned several SignalR development tasks: how to add SignalR to an ASP.NET application, how to create a hub class, and how to send and receive messages from the hub.

You can make the sample application in this tutorial or other SignalR applications available over the Internet by deploying them to a hosting provider. Microsoft offers free web hosting for up to 10 web sites in a free Windows Azure trial account. For a walkthrough on how to deploy the sample SignalR application, see Publish the SignalR Getting Started Sample as a Windows Azure Web Site. For detailed information about how to deploy a Visual Studio web project to a Windows Azure Web Site, see Deploying an ASP.NET Application to a Windows Azure Web Site.

Reference : http://www.asp.net/signalr/overview/getting-started/tutorial-getting-started-with-signalr