files and assembly references that support SignalR to the project.

6. Select **Tools** > **NuGet Package Manager** > **Package Manager Console**.
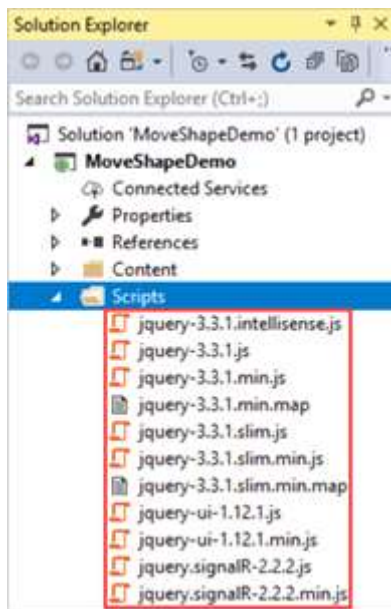
7. In **Package Manager Console**, run this command:

Console

```
Install-Package jQuery.UI.Combined
```

The command installs the jQuery UI library. You use it to animate the shape.

8. In **Solution Explorer**, expand the Scripts node.

Script libraries for jQuery, jQueryUI, and SignalR are visible in the project.

Create the base application

In this section, you create a browser application. The app sends the location of the shape to the server during each mouse move event. The server broadcasts this information to all other connected clients in real time. You learn more about this application in later sections.

1. Open the *MoveShapeHub.cs* file.

2. Replace the code in the *MoveShapeHub.cs* file with this code:

C#

```csharp
using Microsoft.AspNet.SignalR;
using Newtonsoft.Json;

namespace MoveShapeDemo
{
    public class MoveShapeHub : Hub
    {
        public void UpdateModel(ShapeModel clientModel)
        {
            clientModel.LastUpdatedBy = Context.ConnectionId;
            // Update the shape model within our broadcaster

Clients.AllExcept(clientModel.LastUpdatedBy).updateShape(clientModel);
        }
    }
    public class ShapeModel
    {
        // We declare Left and Top as lowercase with
        // JsonProperty to sync the client and server models
        [JsonProperty("left")]
        public double Left { get; set; }
        [JsonProperty("top")]
        public double Top { get; set; }
        // We don't want the client to get the "LastUpdatedBy" property
        [JsonIgnore]
        public string LastUpdatedBy { get; set; }
    }
}
```

3. Save the file.

The `MoveShapeHub` class is an implementation of a SignalR hub. As in the Getting Started with SignalR tutorial, the hub has a method that the clients call directly. In this case, the client sends an object with the new X and Y coordinates of the shape to the server. Those coordinates get broadcasted to all other connected clients. SignalR automatically serializes this object using JSON.

The app sends the `ShapeModel` object to the client. It has members to store the position of the shape. The version of the object on the server also has a member to track which client's data is being stored. This object prevents the server from sending a client's data back to itself. This member uses the `JsonIgnore` attribute to keep the application from serializing the data and sending it back to the client.

Map to the hub when app starts

Next, you set up mapping to the hub when the application starts. In SignalR 2, adding an OWIN startup class creates the mapping.

1. In **Solution Explorer**, right-click the project and select **Add** > **New Item**.

2. In **Add New Item - MoveShapeDemo** select **Installed** > **Visual C#** > **Web** and then select **OWIN Startup Class**.

3. Name the class *Startup* and select **OK**.

4. Replace the default code in the *Startup.cs* file with this code:

```C#
using Microsoft.Owin;
using Owin;

[assembly: OwinStartup(typeof(MoveShapeDemo.Startup))]
namespace MoveShapeDemo
{
    public class Startup
    {
        public void Configuration(IAppBuilder app)
        {
            // Any connection or hub wire up and configuration should go
here
            app.MapSignalR();
        }
    }
}
```

The OWIN startup class calls `MapSignalR` when the app executes the `Configuration` method. The app adds the class to OWIN's startup process using the `OwinStartup` assembly attribute.

Add the client

Add the HTML page for the client.

1. In **Solution Explorer**, right-click the project and select **Add** > **HTML Page**.

2. Name the page **Default** and select **OK**.

3. In **Solution Explorer**, right-click *Default.html* and select **Set as Start Page**.

4. Replace the default code in the *Default.html* file with this code:

HTML

```html
<!DOCTYPE html>
<html>
<head>
    <title>SignalR MoveShape Demo</title>
    <style>
        #shape {
            width: 100px;
            height: 100px;
            background-color: #FF0000;
        }
    </style>
</head>
<body>
<script src="Scripts/jquery-1.10.2.min.js"></script>
<script src="Scripts/jquery-ui-1.10.4.min.js"></script>
<script src="Scripts/jquery.signalR-2.1.0.js"></script>
<script src="/signalr/hubs"></script>
<script>
 $(function () {
            var moveShapeHub = $.connection.moveShapeHub,
            $shape = $("#shape"),
            shapeModel = {
                left: 0,
                top: 0
            };
            moveShapeHub.client.updateShape = function (model) {
                shapeModel = model;
                $shape.css({ left: model.left, top: model.top });
            };
            $.connection.hub.start().done(function () {
                $shape.draggable({
                    drag: function () {
                        shapeModel = $shape.offset();
                        moveShapeHub.server.updateModel(shapeModel);
                    }
                });
            });
        });
</script>

    <div id="shape" />
</body>
</html>
```

5. In **Solution Explorer**, expand **Scripts**.

   Script libraries for jQuery and SignalR are visible in the project.

   > ⓘ **Important**
   >
   > The package manager installs a later version of the SignalR scripts.

6. Update the script references in the code block to correspond to the versions of the script files in the project.

This HTML and JavaScript code creates a red `div` called `shape`. It enables the shape's dragging behavior using the jQuery library and uses the `drag` event to send the shape's position to the server.

Run the app

You can run the app to se`e it work. When you drag the shape around a browser window, the shape moves in the other browsers too.

1. In the toolbar, turn on **Script Debugging** and then select the play button to run the application in Debug mode.

   A browser window opens with the red shape in the upper-right corner.

2. Copy the page's URL.

3. Open another browser and paste the URL into the address bar.

4. Drag the shape in one of the browser windows. The shape in the other browser window follows.

While the application functions using this method, it's not a recommended programming model. There's no upper limit to the number of messages getting sent. As a result, the clients and server get overwhelmed with messages and performance degrades. Also, the app displays a disjointed animation on the client. This jerky animation happens because the

shape moves instantly by each method. It's better if the shape moves smoothly to each new location. Next, you learn how to fix those issues.

Add the client loop

Sending the location of the shape on every mouse move event creates an unnecessary amount of network traffic. The app needs to throttle the messages from the client.

Use the javascript `setInterval` function to set up a loop that sends new position information to the server at a fixed rate. This loop is a basic representation of a "game loop." It's a repeatedly called function that drives all the functionality of a game.

1. Replace the client code in the *Default.html* file with this code:

HTML

```html
<!DOCTYPE html>
<html>
<head>
<title>SignalR MoveShape Demo</title>
<style>
    #shape {
        width: 100px;
        height: 100px;
        background-color: #FF0000;
    }
</style>
</head>
<body>
<script src="Scripts/jquery-1.10.2.min.js"></script>
<script src="Scripts/jquery-ui-1.10.4.min.js"></script>
<script src="Scripts/jquery.signalR-2.1.0.js"></script>
<script src="/signalr/hubs"></script>
<script>
    $(function () {
        var moveShapeHub = $.connection.moveShapeHub,
            $shape = $("#shape"),
            // Send a maximum of 10 messages per second
            // (mouse movements trigger a lot of messages)
            messageFrequency = 10;
```

```
        messageFrequency = 10,
        // Determine how often to send messages in
        // time to abide by the messageFrequency
        updateRate = 1000 / messageFrequency,
        shapeModel = {
            left: 0,
            top: 0
        },
        moved = false;
    moveShapeHub.client.updateShape = function (model) {
        shapeModel = model;
        $shape.css({ left: model.left, top: model.top });
    };
    $.connection.hub.start().done(function () {
        $shape.draggable({
            drag: function () {
                shapeModel = $shape.offset();
                moved = true;
            }
        });
        // Start the client side server update interval
        setInterval(updateServerModel, updateRate);
    });
    function updateServerModel() {
        // Only update server if we have a new movement
        if (moved) {
            moveShapeHub.server.updateModel(shapeModel);
            moved = false;
        }
    }
});
</script>

<div id="shape" />
</body>
</html>
```

> ⓘ **Important**
>
> You have to replace the script references again. They must match the versions of
> the scripts in the project.

This new code adds the `updateServerModel` function. It gets called on a fixed
frequency. The function sends the position data to the server whenever the `moved`
flag indicates that there's new position data to send.

2. Select the play button to start the application

3. Copy the page's URL.

4. Open another browser and paste the URL into the address bar.

5. Drag the shape in one of the browser windows. The shape in the other browser window follows.

Since the app throttles the number of messages that get sent to the server, the animation won't appear as smooth did at first.

Add the server loop

In the current application, messages sent from the server to the client go out as often as they're received. This network traffic presents a similar problem as we see on the client.

The app can send messages more often than they're needed. The connection can become flooded as a result. This section describes how to update the server to add a timer that throttles the rate of the outgoing messages.

1. Replace the contents of `MoveShapeHub.cs` with this code:

```C#
using System;
using System.Threading;
using Microsoft.AspNet.SignalR;
using Newtonsoft.Json;

namespace MoveShapeDemo
{
    public class Broadcaster
    {
        private readonly static Lazy<Broadcaster> _instance =
            new Lazy<Broadcaster>(() => new Broadcaster());
        // We're going to broadcast to all clients a maximum of 25 times
per second
        private readonly TimeSpan BroadcastInterval =
            TimeSpan.FromMilliseconds(40);
        private readonly IHubContext _hubContext;
        private Timer _broadcastLoop;
        private ShapeModel _model;
        private bool _modelUpdated;
        public Broadcaster()
```

```csharp
        {
            // Save our hub context so we can easily use it
            // to send to its connected clients
            _hubContext =
GlobalHost.ConnectionManager.GetHubContext<MoveShapeHub>();
            _model = new ShapeModel();
            _modelUpdated = false;

            // Start the broadcast loop
            _broadcastLoop = new Timer(
                BroadcastShape,
                null,
                BroadcastInterval,
                BroadcastInterval);
        }
        public void BroadcastShape(object state)
        {
            // No need to send anything if our model hasn't changed
            if (_modelUpdated)
            {
                // This is how we can access the Clients property
                // in a static hub method or outside of the hub entirely

_hubContext.Clients.AllExcept(_model.LastUpdatedBy).updateShape(_model);
                _modelUpdated = false;
            }
        }
        public void UpdateShape(ShapeModel clientModel)
        {
            _model = clientModel;
            _modelUpdated = true;
        }
        public static Broadcaster Instance
        {
            get
            {
                return _instance.Value;
            }
        }
    }

    public class MoveShapeHub : Hub
    {
        // Is set via the constructor on each creation
        private Broadcaster _broadcaster;
        public MoveShapeHub()
            : this(Broadcaster.Instance)
        {
        }
        public MoveShapeHub(Broadcaster broadcaster)
        {
            _broadcaster = broadcaster;
```

```csharp
        }
        public void UpdateModel(ShapeModel clientModel)
        {
            clientModel.LastUpdatedBy = Context.ConnectionId;
            // Update the shape model within our broadcaster
            _broadcaster.UpdateShape(clientModel);
        }

    }
    public class ShapeModel
    {
        // We declare Left and Top as lowercase with
        // JsonProperty to sync the client and server models
        [JsonProperty("left")]
        public double Left { get; set; }
        [JsonProperty("top")]
        public double Top { get; set; }
        // We don't want the client to get the "LastUpdatedBy" property
        [JsonIgnore]
        public string LastUpdatedBy { get; set; }
    }

}
```

2. Select the play button to start the application.

3. Copy the page's URL.

4. Open another browser and paste the URL into the address bar.

5. Drag the shape in one of the browser windows.

This code expands the client to add the `Broadcaster` class. The new class throttles the outgoing messages using the `Timer` class from the .NET framework.

It's good to learn that the hub itself is transitory. It's created every time it's needed. So the app creates the `Broadcaster` as a singleton. It uses lazy initialization to defer the `Broadcaster`'s creation until it's needed. That guarantees that the app creates the first hub instance completely before starting the timer.

The call to the clients' `UpdateShape` function is then moved out of the hub's `UpdateModel` method. It's no longer called immediately whenever the app receives incoming messages. Instead, the app sends the messages to the clients at a rate of 25 calls per second. The process is managed by the `_broadcastLoop` timer from within the `Broadcaster` class.

Lastly, instead of calling the client method from the hub directly, the `Broadcaster` class

needs to get a reference to the currently operating `_hubContext` hub. It gets the reference with the `GlobalHost`.

Add smooth animation

The application is almost finished, but we could make one more improvement. The app moves the shape on the client in response to server messages. Instead of setting the position of the shape to the new location given by the server, use the JQuery UI library's `animate` function. It can move the shape smoothly between its current and new position.

1. Update the client's `updateShape` method in the *Default.html* file to look like the highlighted code:

   HTML

   ```
   <!DOCTYPE html>
   <html>
   <head>
       <title>SignalR MoveShape Demo</title>
       <style>
           #shape {
               width: 100px;
               height: 100px;
               background-color: #FF0000;
           }
       </style>
   </head>
   <body>
   <script src="Scripts/jquery-1.10.2.min.js"></script>
   <script src="Scripts/jquery-ui-1.10.4.min.js"></script>
   <script src="Scripts/jquery.signalR-2.1.0.js"></script>
   <script src="/signalr/hubs"></script>
   <script>
           $(function () {
               var moveShapeHub = $.connection.moveShapeHub,
                   $shape = $("#shape"),
                   // Send a maximum of 10 messages per second
                   // (mouse movements trigger a lot of messages)
                   messageFrequency = 10,
                   // Determine how often to send messages in
                   // time to abide by the messageFrequency
                   updateRate = 1000 / messageFrequency,
                   shapeModel = {
                       left: 0,
                       top: 0
                   },
   ```

```
                moved = false;




                moveShapeHub.client.updateShape = function (model) {
                    shapeModel = model;
                    // Gradually move the shape towards the new location (in-
terpolate)
                    // The updateRate is used as the duration because by the
time
                    // we get to the next location we want to be at the
"last" location
                    // We also clear the animation queue so that we start a
new
                    // animation and don't lag behind.
                    $shape.animate(shapeModel, { duration: updateRate, queue:
false });
                };
                $.connection.hub.start().done(function () {
                    $shape.draggable({
                        drag: function () {
                            shapeModel = $shape.offset();
                            moved = true;
                        }
                    });
                    // Start the client side server update interval
                    setInterval(updateServerModel, updateRate);
                });
                function updateServerModel() {
                    // Only update server if we have a new movement
                    if (moved) {
                        moveShapeHub.server.updateModel(shapeModel);
                        moved = false;
                    }
                }
            });
    </script>

        <div id="shape" />
    </body>
    </html>
```

2. Select the play button to start the application.

3. Copy the page's URL.

4. Open another browser and paste the URL into the address bar.

4. Open another browser and paste the URL into the address bar.

5. Drag the shape in one of the browser windows.

The movement of the shape in the other window appears less jerky. The app interpolates its movement over time rather than being set once per incoming message.

This code moves the shape from the old location to the new one. The server gives the position of the shape over the course of the animation interval. In this case, that's 100 milliseconds. The app clears any previous animation running on the shape before the new animation starts.

Get the code

[Download Completed Project](#)

Additional resources

The communication paradigm you just learned about is useful for developing online games and other simulations, like [the ShootR game created with SignalR](#).

For more about SignalR, see the following resources:

- [SignalR Project](#)

- [SignalR GitHub and Samples](#)

- [SignalR Wiki](#)

Next steps

In this tutorial, you:

- ✔ Set up the project
- ✔ Created the base application
- ✔ Mapped to the hub when app starts
- ✔ Added the client
- ✔ Ran the app
- ✔ Added the client loop
- ✔ Added the server loop

✓ Added smooth animation

Advance to the next article to learn how to create a web application that uses ASP.NET
SignalR 2 to provide server broadcast functionality.

SignalR 2 and server broadcasting