Shopping Cart

This tutorial series will teach you the basics of building an ASP.NET Web Forms application using ASP.NET 4.5 and Microsoft Visual Studio Express 2013 for Web. A Visual Studio 2013 project with C# source code is available to accompany this tutorial series.

This tutorial describes the business logic required to add a shopping cart to the Wingtip Toys sample ASP.NET Web Forms application. This tutorial builds on the previous tutorial "Display Data Items and Details" and is part of the Wingtip Toy Store tutorial series. When you've completed this tutorial, the users of your sample app will be able to add, remove, and modify the products in their shopping cart.

What you'll learn:

- How to create a shopping cart for the web application.
- How to enable users to add items to the shopping cart.
- How to add a GridView control to display shopping cart details.
- How to calculate and display the order total.
- How to remove and update items in the shopping cart.
- How to include a shopping cart counter.

Code features in this tutorial:

- Entity Framework Code First
- Data Annotations
- Strongly typed data controls
- Model binding

Creating a Shopping Cart

Earlier in this tutorial series, you added pages and code to view product data from a database. In this tutorial, you'll create a shopping cart to manage the products that users are interested in buying. Users will be able to browse and add items to the shopping cart even if they are not registered or logged in. To manage shopping cart access, you will assign users a unique `ID` using a globally unique identifier (GUID) when the user accesses the shopping cart for the first time. You'll store this `ID` using the ASP.NET Session state.

Note

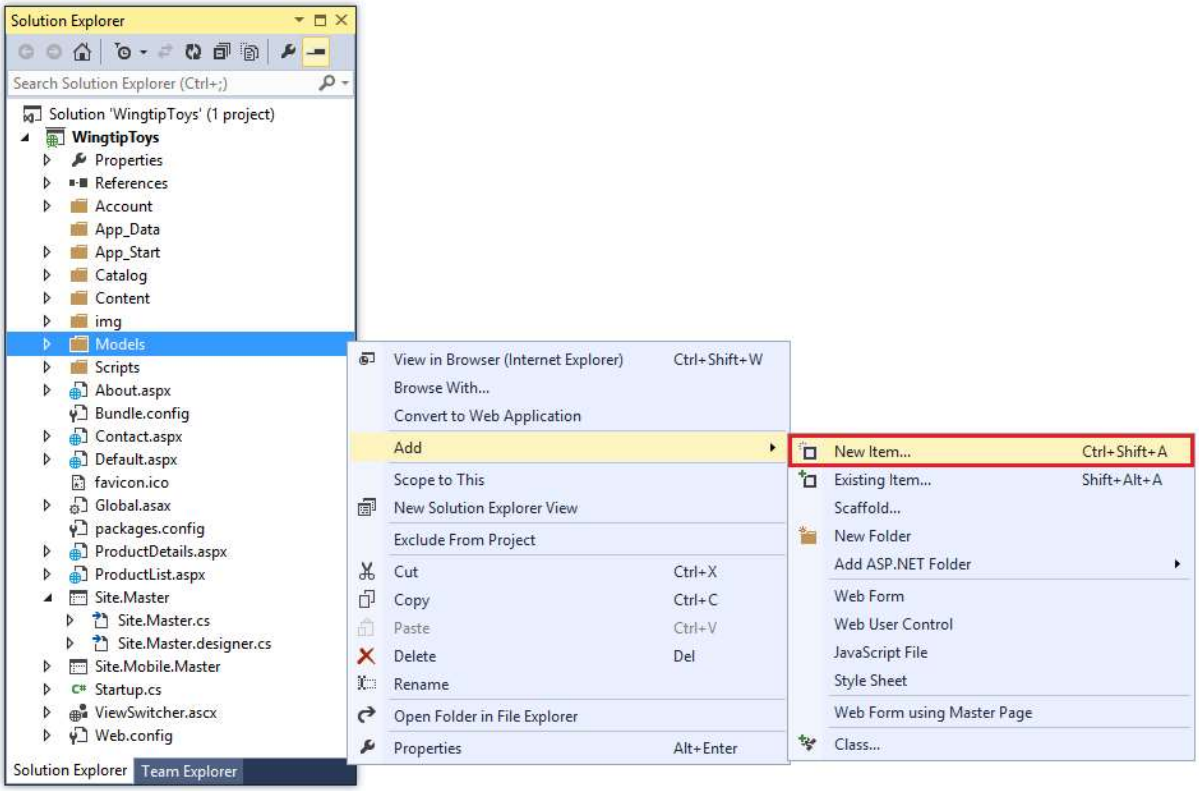
The ASP.NET Session state is a convenient place to store user-specific information which will expire after the user leaves the site. While misuse of session state can have performance implications on larger sites, light use of session state works well for demonstration purposes. The Wingtip Toys sample project shows how to use session state without an external provider, where session state is stored in-process on the web server hosting the site. For larger sites that

provide multiple instances of an application or for sites that run multiple instances of an application on different servers, consider using **Windows Azure Cache Service**. This Cache Service provides a distributed caching service that is external to the web site and solves the problem of using in-process session state. For more information see, How to Use ASP.NET Session State with Windows Azure Web Sites.

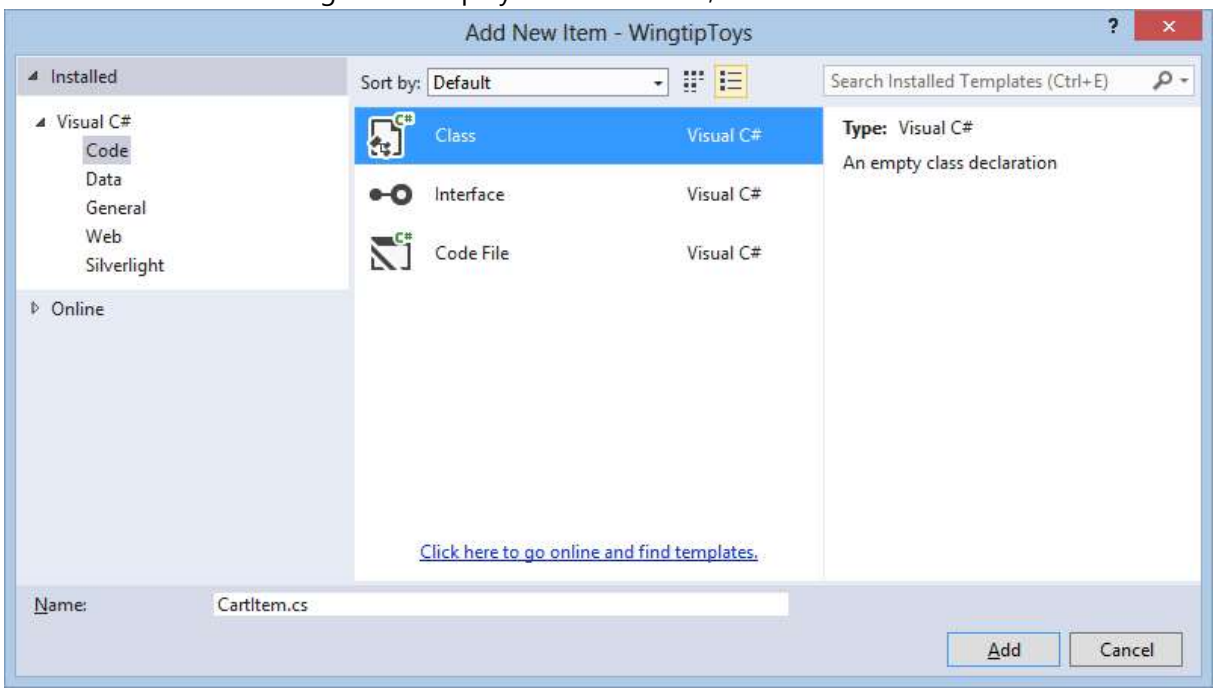
Add CartItem as a Model Class

Earlier in this tutorial series, you defined the schema for the category and product data by creating the `Category` and `Product` classes in the *Models* folder. Now, add a new class to define the schema for the shopping cart. Later in this tutorial, you will add a class to handle data access to the `CartItem` table. This class will provide the business logic to add, remove, and update items in the shopping cart.

- Right-click the ***Models*** folder and select **Add** -> **New Item**.



- The **Add New Item** dialog box is displayed. Select **Code**, and then select **Class**.



- Name this new class *CartItem.cs*.
- Click **Add**.
  The new class file is displayed in the editor.
- Replace the default code with the following code:

```
using System.ComponentModel.DataAnnotations;

namespace WingtipToys.Models
{
  public class CartItem
  {
    [Key]
    public string ItemId { get; set; }

    public string CartId { get; set; }

    public int Quantity { get; set; }

    public System.DateTime DateCreated { get; set; }

    public int ProductId { get; set; }

    public virtual Product Product { get; set; }

  }
}
```

The `CartItem` class contains the schema that will define each product a user adds to the shopping cart. This class is similar to the other schema classes you created earlier in this tutorial series. By convention, Entity Framework Code First expects that the primary key for the `CartItem` table will be either `CartItemId` or `ID`. However, the code overrides the default

behavior by using the data annotation `[Key]` attribute. The `Key` attribute of the ItemId property specifies that the `ItemID` property is the primary key.

The `CartId` property specifies the `ID` of the user that is associated with the item to purchase. You'll add code to create this user `ID` when the user accesses the shopping cart. This `ID` will also be stored as an ASP.NET Session variable.

Update the Product Context

In addition to adding the `CartItem` class, you will need to update the database context class that manages the entity classes and that provides data access to the database. To do this, you will add the newly created `CartItem` model class to the `ProductContext` class.

1. In **Solution Explorer**, find and open the *ProductContext.cs* file in the *Models* folder.
2. Add the highlighted code to the *ProductContext.cs* file as follows:

```
using System.Data.Entity;

namespace WingtipToys.Models
{
    public class ProductContext : DbContext
    {
        public ProductContext()
            : base("WingtipToys")
        {
        }

        public DbSet<Category> Categories { get; set; }
        public DbSet<Product> Products { get; set; }
        public DbSet<CartItem> ShoppingCartItems { get; set; }
    }
}
```

As mentioned previously in this tutorial series, the code in the *ProductContext.cs* file adds the `System.Data.Entity` namespace so that you have access to all the core functionality of the Entity Framework. This functionality includes the capability to query, insert, update, and delete data by working with strongly typed objects. The `ProductContext` class adds access to the newly added `CartItem` model class.

Managing the Shopping Cart Business Logic

Next, you'll create the `ShoppingCart` class in a new *Logic* folder. The `ShoppingCart` class handles data access to the `CartItem` table. The class will also include the business logic to add, remove, and update items in the shopping cart.

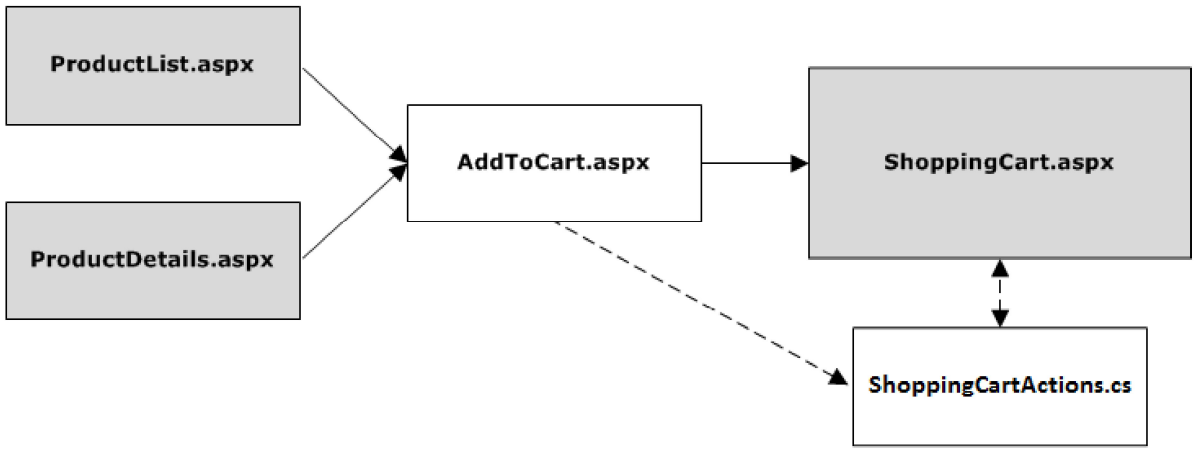
The shopping cart logic that you will add will contain the functionality to manage the following actions:

1. Adding items to the shopping cart
2. Removing items from the shopping cart
3. Getting the shopping cart ID

4. Retrieving items from the shopping cart
5. Totaling the amount of all the shopping cart items
6. Updating the shopping cart data

A shopping cart page (*ShoppingCart.aspx*) and the shopping cart class will be used together to access shopping cart data. The shopping cart page will display all the items the user adds to the shopping cart. Besides the shopping cart page and class, you'll create a page (*AddToCart.aspx*) to add products to the shopping cart. You will also add code to the *ProductList.aspx* page and the *ProductDetails.aspx* page that will provide a link to the *AddToCart.aspx* page, so that the user can add products to the shopping cart.

The following diagram shows the basic process that occurs when the user adds a product to the shopping cart.



When the user clicks the **Add To Cart** link on either the *ProductList.aspx* page or the *ProductDetails.aspx* page, the application will navigate to the *AddToCart.aspx* page and then automatically to the *ShoppingCart.aspx* page. The *AddToCart.aspx* page will add the select product to the shopping cart by calling a method in the ShoppingCart class. The *ShoppingCart.aspx* page will display the products that have been added to the shopping cart.

Creating the Shopping Cart Class

The `ShoppingCart` class will be added to a separate folder in the application so that there will be a clear distinction between the model (Models folder), the pages (root folder) and the logic (Logic folder).

1. In **Solution Explorer**, right-click the **WingtipToys** project and select **Add** -> **New Folder**. Name the new folder *Logic*.
2. Right-click the *Logic* folder and then select **Add** -> **New Item**.
3. Add a new class file named *ShoppingCartActions.cs*.
4. Replace the default code with the following code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using WingtipToys.Models;

namespace WingtipToys.Logic
{
  public class ShoppingCartActions : IDisposable
  {
    public string ShoppingCartId { get; set; }

    private ProductContext _db = new ProductContext();

    public const string CartSessionKey = "CartId";

    public void AddToCart(int id)
    {
      // Retrieve the product from the database.
      ShoppingCartId = GetCartId();

      var cartItem = _db.ShoppingCartItems.SingleOrDefault(
          c => c.CartId == ShoppingCartId
          && c.ProductId == id);
      if (cartItem == null)
      {
        // Create a new cart item if no cart item exists.
        cartItem = new CartItem
        {
          ItemId = Guid.NewGuid().ToString(),
          ProductId = id,
          CartId = ShoppingCartId,
          Product = _db.Products.SingleOrDefault(
           p => p.ProductID == id),
          Quantity = 1,
          DateCreated = DateTime.Now
        };

        _db.ShoppingCartItems.Add(cartItem);
      }
      else
      {
        // If the item does exist in the cart,
        // then add one to the quantity.
        cartItem.Quantity++;
      }
      _db.SaveChanges();
    }

    public void Dispose()
    {
      if (_db != null)
      {
        _db.Dispose();
        _db = null;
      }
    }

    public string GetCartId()
    {
      if (HttpContext.Current.Session[CartSessionKey] == null)
      {
        if (!string.IsNullOrWhiteSpace(HttpContext.Current.User.Identity.Name))
```

```
        {
          HttpContext.Current.Session[CartSessionKey] =
HttpContext.Current.User.Identity.Name;
        }
        else
        {
          // Generate a new random GUID using System.Guid class.
          Guid tempCartId = Guid.NewGuid();
          HttpContext.Current.Session[CartSessionKey] = tempCartId.ToString();
        }
      }
      return HttpContext.Current.Session[CartSessionKey].ToString();
    }

    public List<CartItem> GetCartItems()
    {
      ShoppingCartId = GetCartId();

      return _db.ShoppingCartItems.Where(
          c => c.CartId == ShoppingCartId).ToList();
    }
  }
}
```

The `AddToCart` method enables individual products to be included in the shopping cart based on the product `ID`. The product is added to the cart, or if the cart already contains an item for that product, the quantity is incremented.

The `GetCartId` method returns the cart `ID` for the user. The cart `ID` is used to track the items that a user has in their shopping cart. If the user does not have an existing cart `ID`, a new cart `ID` is created for them. If the user is signed in as a registered user, the cart `ID` is set to their user name. However, if the user is not signed in, the cart `ID` is set to a unique value (a GUID). A GUID ensures that only one cart is created for each user, based on session.

The `GetCartItems` method returns a list of shopping cart items for the user. Later in this tutorial, you will see that model binding is used to display the cart items in the shopping cart using the `GetCartItems` method.

Creating the Add-To-Cart Functionality

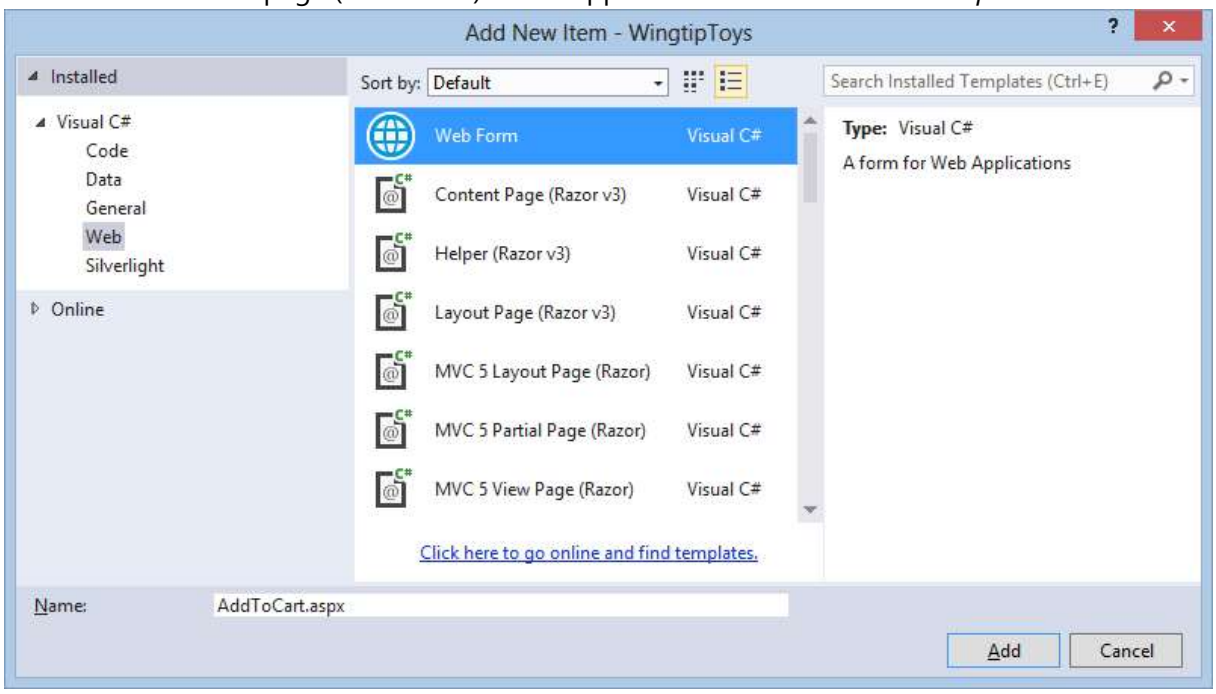
As mentioned earlier, you will create a processing page named *AddToCart.aspx* that will be used to add new products to the shopping cart of the user. This page will call the `AddToCart` method in the `ShoppingCart` class that you just created. The *AddToCart.aspx* page will expect that a product `ID` is passed to it. This product `ID` will be used when calling the `AddToCart` method in the `ShoppingCart` class.

Note

You will be modifying the code-behind (*AddToCart.aspx.cs*) for this page, not the page UI (*AddToCart.aspx*).

To create the Add-To-Cart functionality:

1. In **Solution Explorer**, right-click the **WingtipToys** project, click **Add** -> **New Item**. The **Add New Item** dialog box is displayed.
2. Add a standard new page (Web Form) to the application named *AddToCart.aspx*.



3. In **Solution Explorer**, right-click the *AddToCart.aspx* page and then click **View Code**. The *AddToCart.aspx.cs* code-behind file is opened in the editor.
4. Replace the existing code in the *AddToCart.aspx.cs* code-behind with the following:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Diagnostics;
using WingtipToys.Logic;

namespace WingtipToys
{
  public partial class AddToCart : System.Web.UI.Page
  {
    protected void Page_Load(object sender, EventArgs e)
    {
      string rawId = Request.QueryString["ProductID"];
      int productId;
      if (!String.IsNullOrEmpty(rawId) && int.TryParse(rawId, out productId))
      {
        using (ShoppingCartActions usersShoppingCart = new
ShoppingCartActions())
        {
          usersShoppingCart.AddToCart(Convert.ToInt16(rawId));
        }

      }
      else
```

```
        {
          Debug.Fail("ERROR : We should never get to AddToCart.aspx without a
ProductId.");
          throw new Exception("ERROR : It is illegal to load AddToCart.aspx
without setting a ProductId.");
        }
        Response.Redirect("ShoppingCart.aspx");
      }
    }
}
```

When the *AddToCart.aspx* page is loaded, the product `ID` is retrieved from the query string. Next, an instance of the shopping cart class is created and used to call the `AddToCart` method that you added earlier in this tutorial. The `AddToCart` method, contained in the *ShoppingCartActions.cs* file, includes the logic to add the selected product to the shopping cart or increment the product quantity of the selected product. If the product hasn't been added to the shopping cart, the product is added to the `CartItem` table of the database. If the product has already been added to the shopping cart and the user adds an additional item of the same product, the product quantity is incremented in the `CartItem` table. Finally, the page redirects back to the *ShoppingCart.aspx* page that you'll add in the next step, where the user sees an updated list of items in the cart.

As previously mentioned, a user `ID` is used to identify the products that are associated with a specific user. This `ID` is added to a row in the `CartItem` table each time the user adds a product to the shopping cart.

Creating the Shopping Cart UI

The *ShoppingCart.aspx* page will display the products that the user has added to their shopping cart. It will also provide the ability to add, remove and update items in the shopping cart.

1. In **Solution Explorer**, right-click **WingtipToys**, click **Add** -> **New Item**.
   The **Add New Item** dialog box is displayed.
2. Add a new page (Web Form) that includes a master page by selecting **Web Form using Master Page**. Name the new page *ShoppingCart.aspx*.
3. Select **Site.Master** to attach the master page to the newly created *.aspx* page.
4. In the *ShoppingCart.aspx* page, replace the existing markup with the following markup:

```
<%@ Page Title="" Language="C#" MasterPageFile="~/Site.Master"
AutoEventWireup="true" CodeBehind="ShoppingCart.aspx.cs"
Inherits="WingtipToys.ShoppingCart" %>
<asp:Content ID="Content1" ContentPlaceHolderID="MainContent" runat="server">
    <div id="ShoppingCartTitle" runat="server" class="ContentHead"><h1>Shopping
Cart</h1></div>
    <asp:GridView ID="CartList" runat="server" AutoGenerateColumns="False"
ShowFooter="True" GridLines="Vertical" CellPadding="4"
        ItemType="WingtipToys.Models.CartItem"
SelectMethod="GetShoppingCartItems"
        CssClass="table table-striped table-bordered" >
        <Columns>
```

```
        <asp:BoundField DataField="ProductID" HeaderText="ID"
SortExpression="ProductID" />
        <asp:BoundField DataField="Product.ProductName" HeaderText="Name" />
        <asp:BoundField DataField="Product.UnitPrice" HeaderText="Price (each)"
DataFormatString="{0:c}"/>
        <asp:TemplateField   HeaderText="Quantity">
                <ItemTemplate>
                    <asp:TextBox ID="PurchaseQuantity" Width="40"
runat="server" Text="<%#: Item.Quantity %>"></asp:TextBox>
                </ItemTemplate>
        </asp:TemplateField>
        <asp:TemplateField HeaderText="Item Total">
                <ItemTemplate>
                    <%#: String.Format("{0:c}",
((Convert.ToDouble(Item.Quantity)) *
Convert.ToDouble(Item.Product.UnitPrice)))%>
                </ItemTemplate>
        </asp:TemplateField>
        <asp:TemplateField HeaderText="Remove Item">
                <ItemTemplate>
                    <asp:CheckBox id="Remove" runat="server"></asp:CheckBox>
                </ItemTemplate>
        </asp:TemplateField>
        </Columns>
    </asp:GridView>
    <div>
        <p></p>
        <strong>
            <asp:Label ID="LabelTotalText" runat="server" Text="Order Total:
"></asp:Label>
            <asp:Label ID="lblTotal" runat="server"
EnableViewState="false"></asp:Label>
        </strong>
    </div>
    <br />
</asp:Content>
```

The *ShoppingCart.aspx* page includes a **GridView** control named `CartList`. This control uses model binding to bind the shopping cart data from the database to the **GridView** control. When you set the `ItemType` property of the **GridView** control, the data-binding expression `Item` is available in the markup of the control and the control becomes strongly typed. As mentioned earlier in this tutorial series, you can select details of the `Item` object using IntelliSense. To configure a data control to use model binding to select data, you set the `SelectMethod` property of the control. In the markup above, you set the `SelectMethod` to use the GetShoppingCartItems method which returns a list of `CartItem` objects. The **GridView** data control calls the method at the appropriate time in the page life cycle and automatically binds the returned data. The `GetShoppingCartItems` method must still be added.

Retrieving the Shopping Cart Items

Next, you add code to the *ShoppingCart.aspx.cs* code-behind to retrieve and populate the Shopping Cart UI.

1. In **Solution Explorer**, right-click the *ShoppingCart.aspx* page and then click **View Code**. The *ShoppingCart.aspx.cs* code-behind file is opened in the editor.

- Replace the existing code with the following:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using WingtipToys.Models;
using WingtipToys.Logic;

namespace WingtipToys
{
  public partial class ShoppingCart : System.Web.UI.Page
  {
    protected void Page_Load(object sender, EventArgs e)
    {

    }

    public List<CartItem> GetShoppingCartItems()
    {
      ShoppingCartActions actions = new ShoppingCartActions();
      return actions.GetCartItems();
    }
  }
}
```

As mentioned above, the `GridView` data control calls the `GetShoppingCartItems` method at the appropriate time in the page life cycle and automatically binds the returned data. The `GetShoppingCartItems` method creates an instance of the `ShoppingCartActions` object. Then, the code uses that instance to return the items in the cart by calling the `GetCartItems` method.

Adding Products to the Shopping Cart

When either the *ProductList.aspx* or the *ProductDetails.aspx* page is displayed, the user will be able to add the product to the shopping cart using a link. When they click the link, the application navigates to the processing page named *AddToCart.aspx*. The *AddToCart.aspx* page will call the `AddToCart` method in the `ShoppingCart` class that you added earlier in this tutorial.

Now, you'll add an **Add to Cart** link to both the *ProductList.aspx* page and the *ProductDetails.aspx* page. This link will include the product `ID` that is retrieved from the database.

1. In **Solution Explorer**, find and open the page named *ProductList.aspx*.
2. Add the markup highlighted in yellow to the *ProductList.aspx* page so that the entire page appears as follows:

```
<%@ Page Title="Products" Language="C#" MasterPageFile="~/Site.Master"
AutoEventWireup="true"
         CodeBehind="ProductList.aspx.cs" Inherits="WingtipToys.ProductList" %>
<asp:Content ID="Content1" ContentPlaceHolderID="MainContent" runat="server">
    <section>
        <div>
            <hgroup>
```

```
                <h2><%: Page.Title %></h2>
            </hgroup>

            <asp:ListView ID="productList" runat="server"
                DataKeyNames="ProductID" GroupItemCount="4"
                ItemType="WingtipToys.Models.Product"
SelectMethod="GetProducts">
                <EmptyDataTemplate>
                    <table runat="server">
                        <tr>
                            <td>No data was returned.</td>
                        </tr>
                    </table>
                </EmptyDataTemplate>
                <EmptyItemTemplate>
                    <td runat="server" />
                </EmptyItemTemplate>
                <GroupTemplate>
                    <tr id="itemPlaceholderContainer" runat="server">
                        <td id="itemPlaceholder" runat="server"></td>
                    </tr>
                </GroupTemplate>
                <ItemTemplate>
                    <td runat="server">
                        <table>
                            <tr>
                                <td>
                                    <a
href="ProductDetails.aspx?productID=<%#:Item.ProductID%>">
                                        <img
src="/Catalog/Images/Thumbs/<%#:Item.ImagePath%>"
                                            width="100" height="75"
style="border: solid" /></a>
                                </td>
                            </tr>
                            <tr>
                                <td>
                                    <a
href="ProductDetails.aspx?productID=<%#:Item.ProductID%>">
                                        <span>
                                            <%#:Item.ProductName%>
                                        </span>
                                    </a>
                                    <br />
                                    <span>
                                        <b>Price:
</b><%#:String.Format("{0:c}", Item.UnitPrice)%>
                                    </span>
                                    <br />
                                    <a
href="/AddToCart.aspx?productID=<%#:Item.ProductID %>">
                                        <span class="ProductListItem">
                                            <b>Add To Cart<b>
                                        </span>
                                    </a>
                                </td>
                            </tr>
                            <tr>
                                <td> </td>
                            </tr>
                        </table>
                        </p>
                    </td>
```

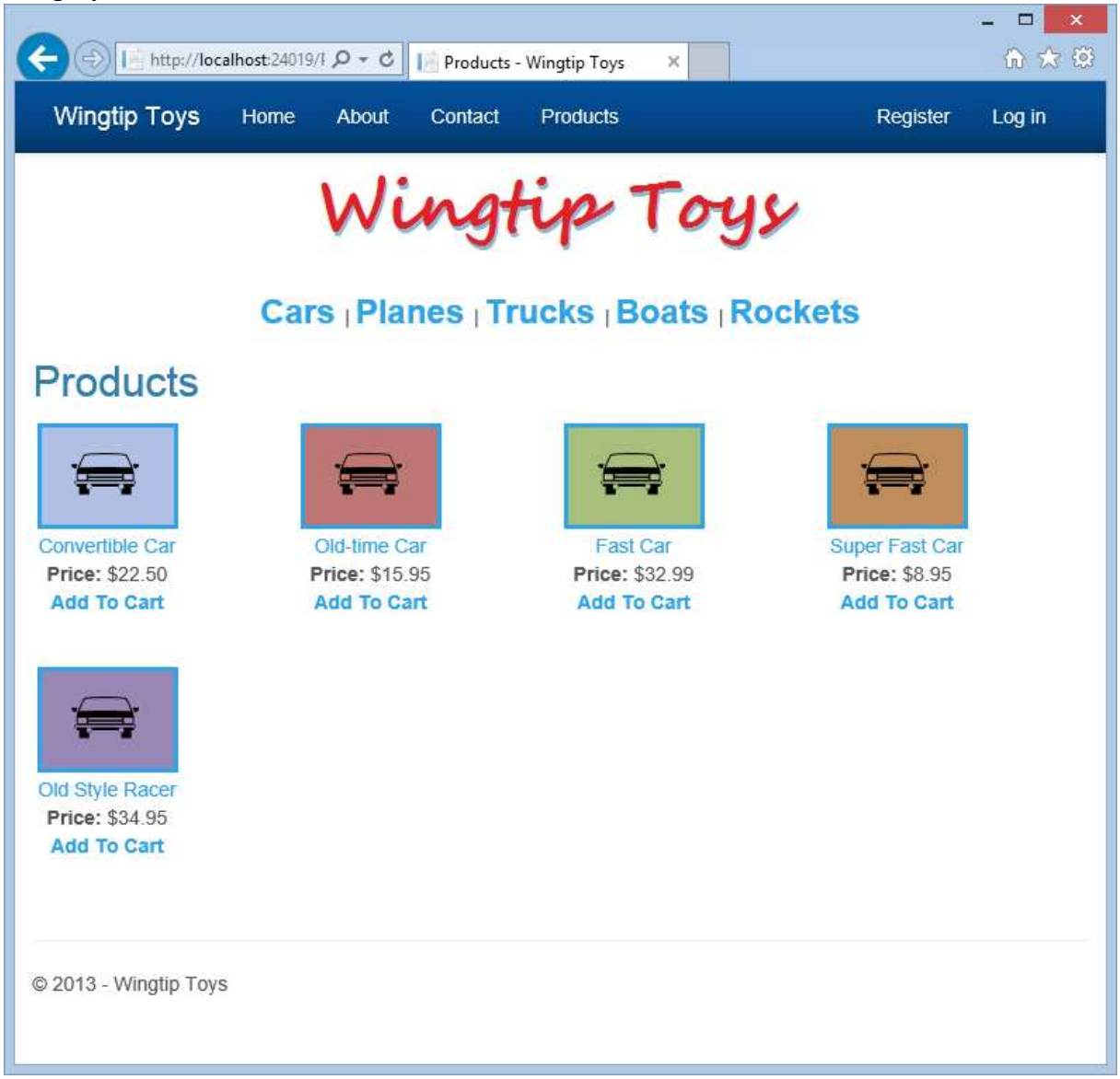
```
                </ItemTemplate>
                <LayoutTemplate>
                    <table runat="server" style="width:100%;">
                        <tbody>
                            <tr runat="server">
                                <td runat="server">
                                    <table id="groupPlaceholderContainer"
runat="server" style="width:100%">
                                        <tr id="groupPlaceholder"
runat="server"></tr>
                                    </table>
                                </td>
                            </tr>
                            <tr runat="server">
                                <td runat="server"></td>
                            </tr>
                            <tr></tr>
                        </tbody>
                    </table>
                </LayoutTemplate>
            </asp:ListView>
        </div>
    </section>
</asp:Content>
```

Testing the Shopping Cart

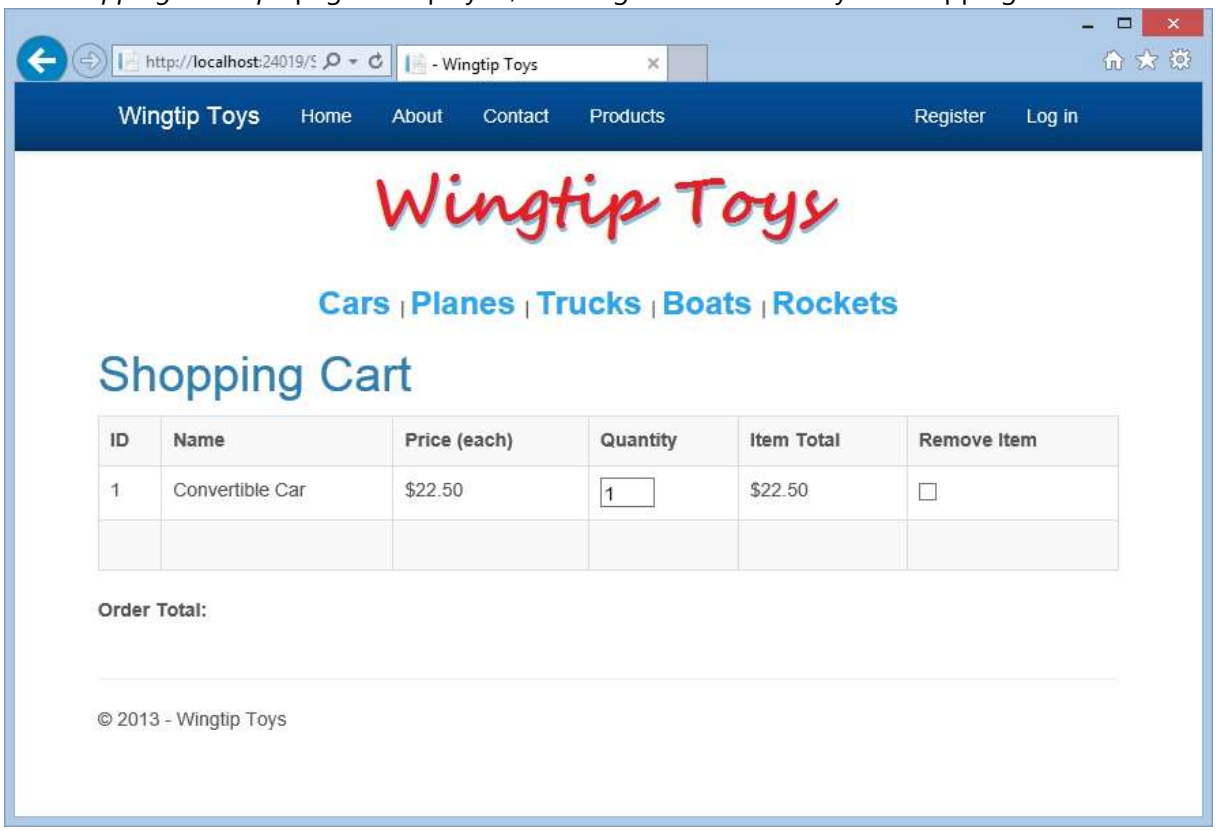
Run the application to see how you add products to the shopping cart.

1. Press **F5** to run the application.
   After the project recreates the database, the browser will open and show the *Default.aspx* page.
2. Select **Cars** from the category navigation menu.
   The *ProductList.aspx* page is displayed showing only products included in the "Cars"

category.


http://localhost:24019/
Products - Wingtip Toys
Wingtip Toys
Home
About
Contact
Products
Register
Log in
Wingtip Toys
Cars | Planes | Trucks | Boats | Rockets
Products
Convertible Car
Price: $22.50
Add To Cart
Old-time Car
Price: $15.95
Add To Cart
Fast Car
Price: $32.99
Add To Cart
Super Fast Car
Price: $8.95
Add To Cart
Old Style Racer
Price: $34.95
Add To Cart
© 2013 - Wingtip Toys

3. Click the **Add to Cart** link next to the first product listed (the convertible car).
   The *ShoppingCart.aspx* page is displayed, showing the selection in your shopping cart.



4. View additional products by selecting **Planes** from the category navigation menu.
5. Click the **Add to Cart** link next to the first product listed.
   The *ShoppingCart.aspx* page is displayed with the additional item.
6. Close the browser.

Calculating and Displaying the Order Total

In addition to adding products to the shopping cart, you will add a `GetTotal` method to the `ShoppingCart` class and display the total order amount in the shopping cart page.

1. In **Solution Explorer**, open the *ShoppingCartActions.cs* file in the *Logic* folder.
2. Add the following `GetTotal` method highlighted in yellow to the `ShoppingCart` class, so that the class appears as follows:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using WingtipToys.Models;

namespace WingtipToys.Logic
{
  public class ShoppingCartActions : IDisposable
  {
```

```
    public string ShoppingCartId { get; set; }

    private ProductContext _db = new ProductContext();

    public const string CartSessionKey = "CartId";

    public void AddToCart(int id)
    {
      // Retrieve the product from the database.
      ShoppingCartId = GetCartId();

      var cartItem = _db.ShoppingCartItems.SingleOrDefault(
          c => c.CartId == ShoppingCartId
          && c.ProductId == id);
      if (cartItem == null)
      {
        // Create a new cart item if no cart item exists.
        cartItem = new CartItem
        {
          ItemId = Guid.NewGuid().ToString(),
          ProductId = id,
          CartId = ShoppingCartId,
          Product = _db.Products.SingleOrDefault(
           p => p.ProductID == id),
          Quantity = 1,
          DateCreated = DateTime.Now
        };

        _db.ShoppingCartItems.Add(cartItem);
      }
      else
      {
        // If the item does exist in the cart,
        // then add one to the quantity.
        cartItem.Quantity++;
      }
      _db.SaveChanges();
    }

    public void Dispose()
    {
      if (_db != null)
      {
        _db.Dispose();
        _db = null;
      }
    }

    public string GetCartId()
    {
      if (HttpContext.Current.Session[CartSessionKey] == null)
      {
        if (!string.IsNullOrWhiteSpace(HttpContext.Current.User.Identity.Name))
        {
          HttpContext.Current.Session[CartSessionKey] =
HttpContext.Current.User.Identity.Name;
        }
        else
        {
          // Generate a new random GUID using System.Guid class.
          Guid tempCartId = Guid.NewGuid();
          HttpContext.Current.Session[CartSessionKey] = tempCartId.ToString();
        }
```

```
        }
        return HttpContext.Current.Session[CartSessionKey].ToString();
    }

    public List<CartItem> GetCartItems()
    {
        ShoppingCartId = GetCartId();

        return _db.ShoppingCartItems.Where(
            c => c.CartId == ShoppingCartId).ToList();
    }

    public decimal GetTotal()
    {
        ShoppingCartId = GetCartId();
        // Multiply product price by quantity of that product to get
        // the current price for each of those products in the cart.
        // Sum all product price totals to get the cart total.
        decimal? total = decimal.Zero;
        total = (decimal?)(from cartItems in _db.ShoppingCartItems
                           where cartItems.CartId == ShoppingCartId
                           select (int?)cartItems.Quantity *
                           cartItems.Product.UnitPrice).Sum();
        return total ?? decimal.Zero;
    }
  }
}
```

First, the `GetTotal` method gets the ID of the shopping cart for the user. Then the method gets the cart total by multiplying the product price by the product quantity for each product listed in the cart.

Note

The above code uses the nullable type "`int?`". Nullable types can represent all the values of an underlying type, and also as a null value. For more information see, Using Nullable Types.

Modify the Shopping Cart Display

Next you'll modify the code for the *ShoppingCart.aspx* page to call the `GetTotal` method and display that total on the *ShoppingCart.aspx* page when the page loads.

1. In **Solution Explorer**, right-click the *ShoppingCart.aspx* page and select **View Code**.
2. In the *ShoppingCart.aspx.cs* file, update the `Page_Load` handler by adding the following code highlighted in yellow:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using WingtipToys.Models;
using WingtipToys.Logic;

namespace WingtipToys
{
  public partial class ShoppingCart : System.Web.UI.Page
```

```
  {
    protected void Page_Load(object sender, EventArgs e)
    {
      using (ShoppingCartActions usersShoppingCart = new ShoppingCartActions())
      {
        decimal cartTotal = 0;
        cartTotal = usersShoppingCart.GetTotal();
        if (cartTotal > 0)
        {
          // Display Total.
          lblTotal.Text = String.Format("{0:c}", cartTotal);
        }
        else
        {
          LabelTotalText.Text = "";
          lblTotal.Text = "";
          ShoppingCartTitle.InnerText = "Shopping Cart is Empty";
        }
      }
    }

    public List<CartItem> GetShoppingCartItems()
    {
      ShoppingCartActions actions = new ShoppingCartActions();
      return actions.GetCartItems();
    }
  }
}
```

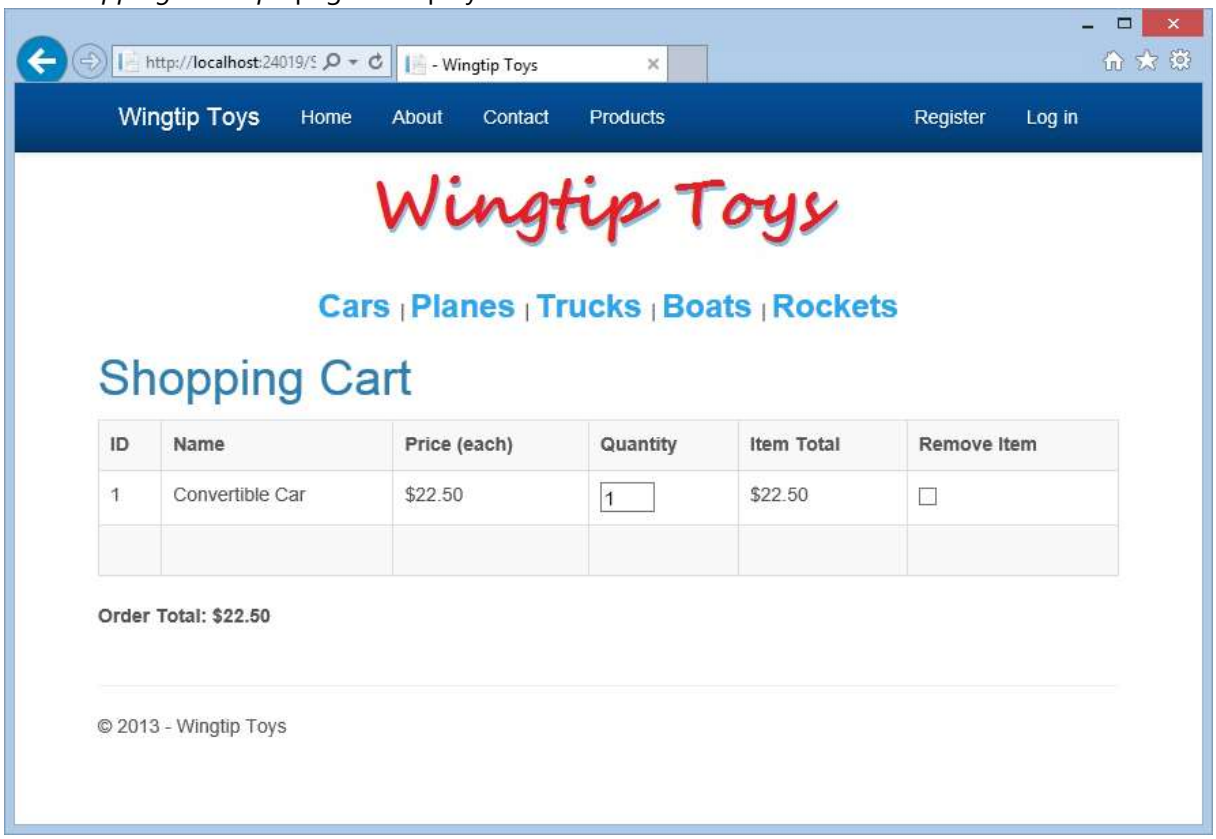
When the *ShoppingCart.aspx* page loads, it loads the shopping cart object and then retrieves the shopping cart total by calling the `GetTotal` method of the `ShoppingCart` class. If the shopping cart is empty, a message to that effect is displayed.

Testing the Shopping Cart Total

Run the application now to see how you can not only add a product to the shopping cart, but you can see the shopping cart total.

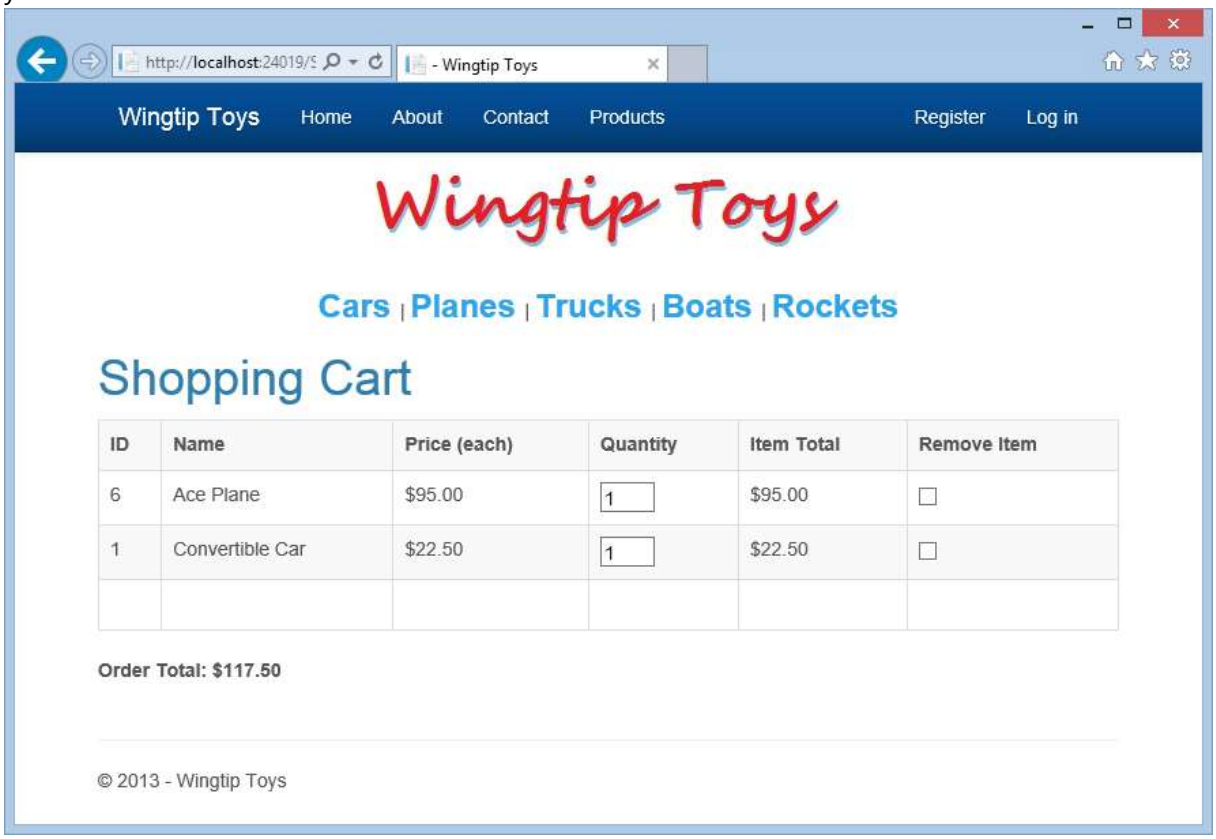
1. Press **F5** to run the application.
   The browser will open and show the *Default.aspx* page.
2. Select **Cars** from the category navigation menu.

3. Click the **Add To Cart** link next to the first product.
   The *ShoppingCart.aspx* page is displayed with the order total.



4. Add some other products (for example, a plane) to the cart.

5. The *ShoppingCart.aspx* page is displayed with an updated total for all the products you've added.



6. Stop the running app by closing the browser window.

Adding Update and Checkout Buttons to the Shopping Cart

To allow the users to modify the shopping cart, you'll add an **Update** button and a **Checkout** button to the shopping cart page. The **Checkout** button is not used until later in this tutorial series.

- In **Solution Explorer**, open the *ShoppingCart.aspx* page in the root of the web application project.
- To add the **Update** button and the **Checkout** button to the *ShoppingCart.aspx* page, add the markup highlighted in yellow to the existing markup, as shown in the following code:

```
<%@ Page Title="" Language="C#" MasterPageFile="~/Site.Master"
AutoEventWireup="true" CodeBehind="ShoppingCart.aspx.cs"
Inherits="WingtipToys.ShoppingCart" %>
<asp:Content ID="Content1" ContentPlaceHolderID="MainContent" runat="server">
    <div id="ShoppingCartTitle" runat="server" class="ContentHead"><h1>Shopping
Cart</h1></div>
    <asp:GridView ID="CartList" runat="server" AutoGenerateColumns="False"
ShowFooter="True" GridLines="Vertical" CellPadding="4"
        ItemType="WingtipToys.Models.CartItem"
SelectMethod="GetShoppingCartItems"
        CssClass="table table-striped table-bordered" >
```

```
        <Columns>
        <asp:BoundField DataField="ProductID" HeaderText="ID"
SortExpression="ProductID" />
        <asp:BoundField DataField="Product.ProductName" HeaderText="Name" />
        <asp:BoundField DataField="Product.UnitPrice" HeaderText="Price (each)"
DataFormatString="{0:c}"/>
        <asp:TemplateField   HeaderText="Quantity">
                <ItemTemplate>
                    <asp:TextBox ID="PurchaseQuantity" Width="40"
runat="server" Text="<%#: Item.Quantity %>"></asp:TextBox>
                </ItemTemplate>
        </asp:TemplateField>
        <asp:TemplateField HeaderText="Item Total">
                <ItemTemplate>
                    <%#: String.Format("{0:c}",
((Convert.ToDouble(Item.Quantity)) *
Convert.ToDouble(Item.Product.UnitPrice)))%>
                </ItemTemplate>
        </asp:TemplateField>
        <asp:TemplateField HeaderText="Remove Item">
                <ItemTemplate>
                    <asp:CheckBox id="Remove" runat="server"></asp:CheckBox>
                </ItemTemplate>
        </asp:TemplateField>
        </Columns>
    </asp:GridView>
    <div>
        <p></p>
        <strong>
            <asp:Label ID="LabelTotalText" runat="server" Text="Order Total:
"></asp:Label>
            <asp:Label ID="lblTotal" runat="server"
EnableViewState="false"></asp:Label>
        </strong>
    </div>
  <br />
    <table>
    <tr>
      <td>
        <asp:Button ID="UpdateBtn" runat="server" Text="Update"
OnClick="UpdateBtn_Click" />
      </td>
      <td>
        <!--Checkout Placeholder -->
      </td>
    </tr>
    </table>
</asp:Content>
```

When the user clicks the **Update** button, the `UpdateBtn_Click` event handler will be called. This event handler will call the code that you'll add in the next step.

Next, you can update the code contained in the *ShoppingCart.aspx.cs* file to loop through the cart items and call the `RemoveItem` and `UpdateItem` methods.

1. In **Solution Explorer**, open the *ShoppingCart.aspx.cs* file in the root of the web application project.
2. Add the following code sections highlighted in yellow to the *ShoppingCart.aspx.cs* file:

```
using System;
```

```
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using WingtipToys.Models;
using WingtipToys.Logic;
using System.Collections.Specialized;
using System.Collections;
using System.Web.ModelBinding;

namespace WingtipToys
{
  public partial class ShoppingCart : System.Web.UI.Page
  {
    protected void Page_Load(object sender, EventArgs e)
    {
      using (ShoppingCartActions usersShoppingCart = new ShoppingCartActions())
      {
        decimal cartTotal = 0;
        cartTotal = usersShoppingCart.GetTotal();
        if (cartTotal > 0)
        {
          // Display Total.
          lblTotal.Text = String.Format("{0:c}", cartTotal);
        }
        else
        {
          LabelTotalText.Text = "";
          lblTotal.Text = "";
          ShoppingCartTitle.InnerText = "Shopping Cart is Empty";
          UpdateBtn.Visible = false;
        }
      }
    }

    public List<CartItem> GetShoppingCartItems()
    {
      ShoppingCartActions actions = new ShoppingCartActions();
      return actions.GetCartItems();
    }

    public List<CartItem> UpdateCartItems()
    {
      using (ShoppingCartActions usersShoppingCart = new ShoppingCartActions())
      {
        String cartId = usersShoppingCart.GetCartId();

        ShoppingCartActions.ShoppingCartUpdates[] cartUpdates = new
ShoppingCartActions.ShoppingCartUpdates[CartList.Rows.Count];
        for (int i = 0; i < CartList.Rows.Count; i++)
        {
          IOrderedDictionary rowValues = new OrderedDictionary();
          rowValues = GetValues(CartList.Rows[i]);
          cartUpdates[i].ProductId = Convert.ToInt32(rowValues["ProductID"]);

          CheckBox cbRemove = new CheckBox();
          cbRemove = (CheckBox)CartList.Rows[i].FindControl("Remove");
          cartUpdates[i].RemoveItem = cbRemove.Checked;

          TextBox quantityTextBox = new TextBox();
          quantityTextBox =
(TextBox)CartList.Rows[i].FindControl("PurchaseQuantity");
```

```
            cartUpdates[i].PurchaseQuantity =
Convert.ToInt16(quantityTextBox.Text.ToString());
          }
          usersShoppingCart.UpdateShoppingCartDatabase(cartId, cartUpdates);
          CartList.DataBind();
          lblTotal.Text = String.Format("{0:c}", usersShoppingCart.GetTotal());
          return usersShoppingCart.GetCartItems();
        }
      }

      public static IOrderedDictionary GetValues(GridViewRow row)
      {
        IOrderedDictionary values = new OrderedDictionary();
        foreach (DataControlFieldCell cell in row.Cells)
        {
          if (cell.Visible)
          {
            // Extract values from the cell.
            cell.ContainingField.ExtractValuesFromCell(values, cell,
row.RowState, true);
          }
        }
        return values;
      }

      protected void UpdateBtn_Click(object sender, EventArgs e)
      {
        UpdateCartItems();
      }
    }
}
```

When the user clicks the **Update** button on the *ShoppingCart.aspx* page, the UpdateCartItems method is called. The UpdateCartItems method gets the updated values for each item in the shopping cart. Then, the UpdateCartItems method calls the `UpdateShoppingCartDatabase` method (added and explained in the next step) to either add or remove items from the shopping cart. Once the database has been updated to reflect the updates to the shopping cart, the **GridView** control is updated on the shopping cart page by calling the `DataBind` method for the **GridView**. Also, the total order amount on the shopping cart page is updated to reflect the updated list of items.

Updating and Removing Shopping Cart Items

On the *ShoppingCart.aspx* page, you can see controls have been added for updating the quantity of an item and removing an item. Now, add the code that will make these controls work.

1. In **Solution Explorer**, open the *ShoppingCartActions.cs* file in the *Logic* folder.

- Add the following code highlighted in yellow to the *ShoppingCartActions.cs* class file:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using WingtipToys.Models;

namespace WingtipToys.Logic
```

```
{
  public class ShoppingCartActions : IDisposable
  {
    public string ShoppingCartId { get; set; }

    private ProductContext _db = new ProductContext();

    public const string CartSessionKey = "CartId";

    public void AddToCart(int id)
    {
      // Retrieve the product from the database.
      ShoppingCartId = GetCartId();

      var cartItem = _db.ShoppingCartItems.SingleOrDefault(
          c => c.CartId == ShoppingCartId
          && c.ProductId == id);
      if (cartItem == null)
      {
        // Create a new cart item if no cart item exists.
        cartItem = new CartItem
        {
          ItemId = Guid.NewGuid().ToString(),
          ProductId = id,
          CartId = ShoppingCartId,
          Product = _db.Products.SingleOrDefault(
           p => p.ProductID == id),
          Quantity = 1,
          DateCreated = DateTime.Now
        };

        _db.ShoppingCartItems.Add(cartItem);
      }
      else
      {
        // If the item does exist in the cart,
        // then add one to the quantity.
        cartItem.Quantity++;
      }
      _db.SaveChanges();
    }

    public void Dispose()
    {
      if (_db != null)
      {
        _db.Dispose();
        _db = null;
      }
    }

    public string GetCartId()
    {
      if (HttpContext.Current.Session[CartSessionKey] == null)
      {
        if (!string.IsNullOrWhiteSpace(HttpContext.Current.User.Identity.Name))
        {
          HttpContext.Current.Session[CartSessionKey] =
HttpContext.Current.User.Identity.Name;
        }
        else
        {
          // Generate a new random GUID using System.Guid class.
```

```
            Guid tempCartId = Guid.NewGuid();
            HttpContext.Current.Session[CartSessionKey] = tempCartId.ToString();
          }
        }
        return HttpContext.Current.Session[CartSessionKey].ToString();
      }

      public List<CartItem> GetCartItems()
      {
        ShoppingCartId = GetCartId();

        return _db.ShoppingCartItems.Where(
            c => c.CartId == ShoppingCartId).ToList();
      }

      public decimal GetTotal()
      {
        ShoppingCartId = GetCartId();
        // Multiply product price by quantity of that product to get
        // the current price for each of those products in the cart.
        // Sum all product price totals to get the cart total.
        decimal? total = decimal.Zero;
        total = (decimal?)(from cartItems in _db.ShoppingCartItems
                           where cartItems.CartId == ShoppingCartId
                           select (int?)cartItems.Quantity *
                           cartItems.Product.UnitPrice).Sum();
        return total ?? decimal.Zero;
      }

      public ShoppingCartActions GetCart(HttpContext context)
      {
        using (var cart = new ShoppingCartActions())
        {
          cart.ShoppingCartId = cart.GetCartId();
          return cart;
        }
      }

      public void UpdateShoppingCartDatabase(String cartId, ShoppingCartUpdates[]
CartItemUpdates)
      {
        using (var db = new WingtipToys.Models.ProductContext())
        {
          try
          {
            int CartItemCount = CartItemUpdates.Count();
            List<CartItem> myCart = GetCartItems();
            foreach (var cartItem in myCart)
            {
              // Iterate through all rows within shopping cart list
              for (int i = 0; i < CartItemCount; i++)
              {
                if (cartItem.Product.ProductID == CartItemUpdates[i].ProductId)
                {
                  if (CartItemUpdates[i].PurchaseQuantity < 1 ||
CartItemUpdates[i].RemoveItem == true)
                  {
                    RemoveItem(cartId, cartItem.ProductId);
                  }
                  else
                  {
                    UpdateItem(cartId, cartItem.ProductId,
CartItemUpdates[i].PurchaseQuantity);
```

```
                }
              }
            }
          }
        }
        catch (Exception exp)
        {
          throw new Exception("ERROR: Unable to Update Cart Database - " +
exp.Message.ToString(), exp);
        }
      }
    }

    public void RemoveItem(string removeCartID, int removeProductID)
    {
      using (var _db = new WingtipToys.Models.ProductContext())
      {
        try
        {
          var myItem = (from c in _db.ShoppingCartItems where c.CartId ==
removeCartID && c.Product.ProductID == removeProductID select
c).FirstOrDefault();
          if (myItem != null)
          {
            // Remove Item.
            _db.ShoppingCartItems.Remove(myItem);
            _db.SaveChanges();
          }
        }
        catch (Exception exp)
        {
          throw new Exception("ERROR: Unable to Remove Cart Item - " +
exp.Message.ToString(), exp);
        }
      }
    }

    public void UpdateItem(string updateCartID, int updateProductID, int
quantity)
    {
      using (var _db = new WingtipToys.Models.ProductContext())
      {
        try
        {
          var myItem = (from c in _db.ShoppingCartItems where c.CartId ==
updateCartID && c.Product.ProductID == updateProductID select
c).FirstOrDefault();
          if (myItem != null)
          {
            myItem.Quantity = quantity;
            _db.SaveChanges();
          }
        }
        catch (Exception exp)
        {
          throw new Exception("ERROR: Unable to Update Cart Item - " +
exp.Message.ToString(), exp);
        }
      }
    }

    public void EmptyCart()
    {
```

```
        ShoppingCartId = GetCartId();
        var cartItems = _db.ShoppingCartItems.Where(
            c => c.CartId == ShoppingCartId);
        foreach (var cartItem in cartItems)
        {
          _db.ShoppingCartItems.Remove(cartItem);
        }
        // Save changes.
        _db.SaveChanges();
      }

      public int GetCount()
      {
        ShoppingCartId = GetCartId();

        // Get the count of each item in the cart and sum them up
        int? count = (from cartItems in _db.ShoppingCartItems
                      where cartItems.CartId == ShoppingCartId
                      select (int?)cartItems.Quantity).Sum();
        // Return 0 if all entries are null
        return count ?? 0;
      }

      public struct ShoppingCartUpdates
      {
        public int ProductId;
        public int PurchaseQuantity;
        public bool RemoveItem;
      }
    }
}
```

The `UpdateShoppingCartDatabase` method, called from the `UpdateCartItems` method on the *ShoppingCart.aspx.cs* page, contains the logic to either update or remove items from the shopping cart. The `UpdateShoppingCartDatabase` method iterates through all the rows within the shopping cart list. If a shopping cart item has been marked to be removed, or the quantity is less than one, the `RemoveItem` method is called. Otherwise, the shopping cart item is checked for updates when the `UpdateItem` method is called. After the shopping cart item has been removed or updated, the database changes are saved.

The `ShoppingCartUpdates` structure is used to hold all the shopping cart items. The `UpdateShoppingCartDatabase` method uses the `ShoppingCartUpdates` structure to determine if any of the items need to be updated or removed.

In the next tutorial, you will use the `EmptyCart` method to clear the shopping cart after purchasing products. But for now, you will use the `GetCount` method that you just added to the *ShoppingCartActions.cs* file to determine how many items are in the shopping cart.

Adding a Shopping Cart Counter

To allow the user to view the total number of items in the shopping cart, you will add a counter to the *Site.Master* page. This counter will also act as a link to the shopping cart.

1. In **Solution Explorer**, open the *Site.Master* page.

- Modify the markup by adding the shopping cart counter link as shown in yellow to the navigation section so it appears as follows:

```
  <ul class="nav navbar-nav">
      <li><a runat="server" href="~/">Home</a></li>
      <li><a runat="server" href="~/About">About</a></li>
      <li><a runat="server" href="~/Contact">Contact</a></li>
      <li><a runat="server" href="~/ProductList">Products</a></li>
      <li><a runat="server" href="~/ShoppingCart"
ID="cartCount"> </a></li>
  </ul>
```

- Next, update the code-behind of the *Site.Master.cs* file by adding the code highlighted in yellow as follows:

```
using System;
using System.Collections.Generic;
using System.Security.Claims;
using System.Security.Principal;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Linq;
using WingtipToys.Models;
using WingtipToys.Logic;

namespace WingtipToys
{
    public partial class SiteMaster : MasterPage
    {
        private const string AntiXsrfTokenKey = "__AntiXsrfToken";
        private const string AntiXsrfUserNameKey = "__AntiXsrfUserName";
        private string _antiXsrfTokenValue;

        protected void Page_Init(object sender, EventArgs e)
        {
            // The code below helps to protect against XSRF attacks
            var requestCookie = Request.Cookies[AntiXsrfTokenKey];
            Guid requestCookieGuidValue;
            if (requestCookie != null && Guid.TryParse(requestCookie.Value, out
requestCookieGuidValue))
            {
                // Use the Anti-XSRF token from the cookie
                _antiXsrfTokenValue = requestCookie.Value;
                Page.ViewStateUserKey = _antiXsrfTokenValue;
            }
            else
            {
                // Generate a new Anti-XSRF token and save to the cookie
                _antiXsrfTokenValue = Guid.NewGuid().ToString("N");
                Page.ViewStateUserKey = _antiXsrfTokenValue;

                var responseCookie = new HttpCookie(AntiXsrfTokenKey)
                {
                    HttpOnly = true,
                    Value = _antiXsrfTokenValue
                };
                if (FormsAuthentication.RequireSSL &&
Request.IsSecureConnection)
```

```
                {
                    responseCookie.Secure = true;
                }
                Response.Cookies.Set(responseCookie);
            }

            Page.PreLoad += master_Page_PreLoad;
        }

        protected void master_Page_PreLoad(object sender, EventArgs e)
        {
            if (!IsPostBack)
            {
                // Set Anti-XSRF token
                ViewState[AntiXsrfTokenKey] = Page.ViewStateUserKey;
                ViewState[AntiXsrfUserNameKey] = Context.User.Identity.Name ??
String.Empty;
            }
            else
            {
                // Validate the Anti-XSRF token
                if ((string)ViewState[AntiXsrfTokenKey] != _antiXsrfTokenValue
                    || (string)ViewState[AntiXsrfUserNameKey] !=
(Context.User.Identity.Name ?? String.Empty))
                {
                    throw new InvalidOperationException("Validation of Anti-
XSRF token failed.");
                }
            }
        }

        protected void Page_Load(object sender, EventArgs e)
        {

        }

        protected void Page_PreRender(object sender, EventArgs e)
        {
          using (ShoppingCartActions usersShoppingCart = new
ShoppingCartActions())
          {
            string cartStr = string.Format("Cart ({0})",
usersShoppingCart.GetCount());
            cartCount.InnerText = cartStr;
          }
        }

        public IQueryable<Category> GetCategories()
        {
          var _db = new WingtipToys.Models.ProductContext();
          IQueryable<Category> query = _db.Categories;
          return query;
        }

        protected void Unnamed_LoggingOut(object sender, LoginCancelEventArgs
e)
        {
            Context.GetOwinContext().Authentication.SignOut();
        }
    }
}
```

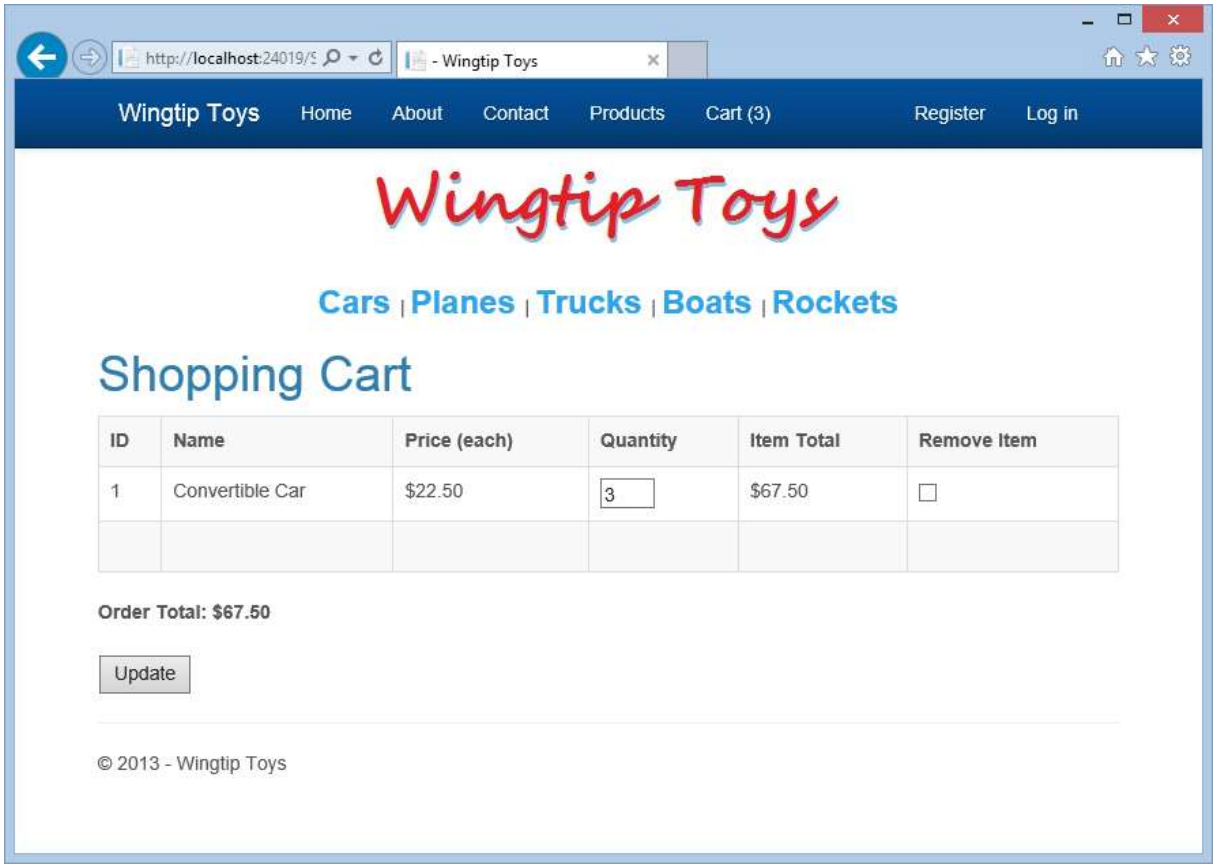
Before the page is rendered as HTML, the `Page_PreRender` event is raised. In the `Page_PreRender` handler, the total count of the shopping cart is determined by calling the `GetCount` method. The returned value is added to the `cartCount` span included in the markup of the *Site.Master* page. The `<span>` tags enables the inner elements to be properly rendered. When any page of the site is displayed, the shopping cart total will be displayed. The user can also click the shopping cart total to display the shopping cart.

Testing the Completed Shopping Cart

You can run the application now to see how you can add, delete, and update items in the shopping cart. The shopping cart total will reflect the total cost of all items in the shopping cart.

1. Press **F5** to run the application.
   The browser opens and shows the *Default.aspx* page.
2. Select **Cars** from the category navigation menu.
3. Click the **Add To Cart** link next to the first product.
   The *ShoppingCart.aspx* page is displayed with the order total.
4. Select **Planes** from the category navigation menu.
5. Click the **Add To Cart** link next to the first product.
6. Set the quantity of the first item in the shopping cart to 3 and select the **Remove Item** check box of the second item.

7. Click the **Update** button to update the shopping cart page and display the new order total.



Summary

In this tutorial, you have created a shopping cart for the Wingtip Toys Web Forms sample application. During this tutorial you have used Entity Framework Code First, data annotations, strongly typed data controls, and model binding.

The shopping cart supports adding, deleting, and updating items that the user has selected for purchase. In addition to implementing the shopping cart functionality, you have learned how to display shopping cart items in a **GridView** control and calculate the order total.

Addition Information

- ASP.NET Session State Overview