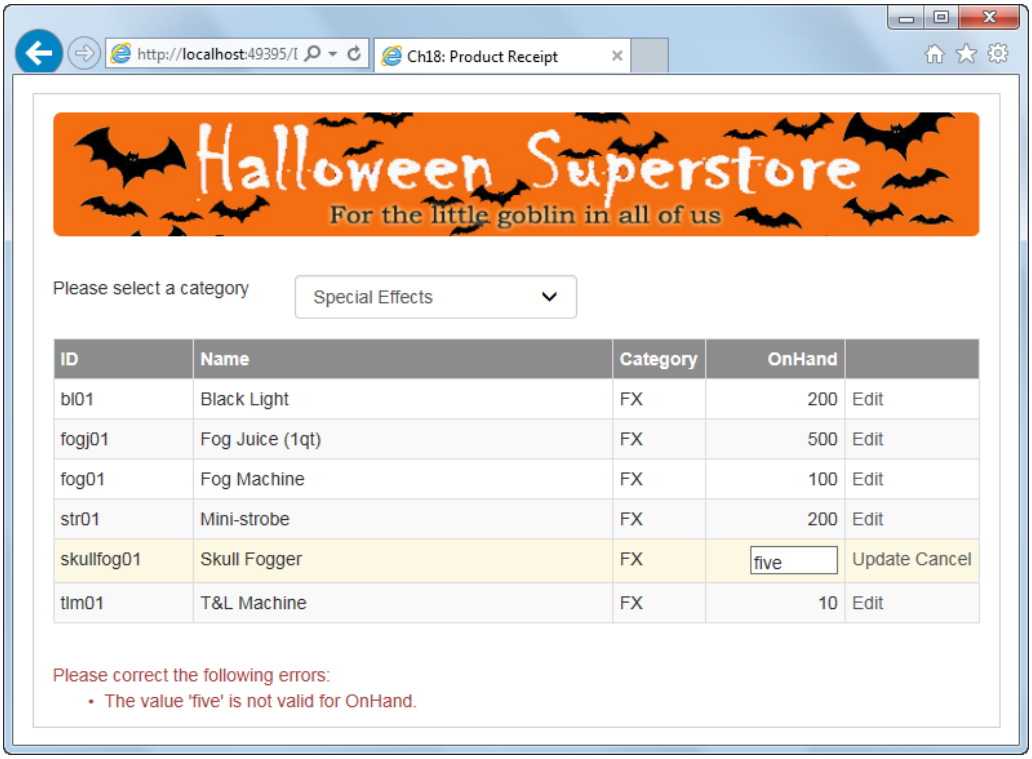
Modify the Product Receipt application

In this exercise, you'll create a version of the Product Receipt application that uses model binding with an Entity Framework Entity Data Model. In this version of the application, the user will select a product category from a drop-down list to display all the products in that category in the GridView control.



Add an Entity Data Model to the application

1. Open the provided application in your project directory. This application contains the starting page with the validation summary and label controls for error messages, along with the database, image, and style sheets used by the page.
2. Add an Entity Data Model named Halloween to the Models folder using the Entity Data Model Wizard. Choose the option that creates the model from an existing database and lets you work with it in the Entity Designer. Then, add the Products and Categories tables to the model.
3. Review the objects in the Designer and the files that have been added to the Models folder. When you're done, build the solution.

Bind category data to the drop-down list

Since IntelliSense doesn't always work with a drop-down list, you'll use the GridView control in the following steps to generate the code you need for the drop-down list.

4. In the aspx file, add an ItemType property to the GridView control and select the Product object from the list. Then, add a SelectMethod property and select the Create New Method option that IntelliSense provides.
5. In the code-behind file, add a using directive for the Models folder. Then, adjust the method stub so it doesn't use a fully qualified name for the Product object.
6. Make a copy of the method stub, change the name of that copy to ddlCategory_GetData, and change its return type to IQueryable<Category>.
7. Create a variable that stores a new instance of the HalloweenEntities object context. Create this variable at the class level so it can be used by all methods on the page.
8. Add a LINQ query to the ddlCategory_GetData method that retrieves all categories and sorts them by LongName.

9. Back in the aspx file, set the SelectMethod property of the drop-down list to ddlCategory_GetData. Then, set the DataValueField property to CategoryID and the DataTextField property to LongName. Finally, run the application to make sure the data binding for the drop-down list is working correctly.

Bind filtered product data to the GridView control

10. In the code-behind file, add a using directive for the System.Web.ModelBinding namespace. Then, find the GetData method for the GridView that you generated in step 4. Add a parameter and a model binding attribute to the method so the method accepts the value of the category drop-down list.
11. Add a LINQ query to the method that returns all products in the selected category, sorted by Name. Provide for the category ID being null by setting it to the category ID of the first item in Categories collection of the object context. Be sure to sort the collection by LongName so the order is the same as the order in the drop-down list.
12. Run and test the application to make sure it's working correctly.

Add a templated field to the GridView that shows the product's category

13. In the aspx file, add a templated field to the Columns element of the GridView control following the Name column and set its HeaderText property to "Category".
14. Add Item and EditItem templates to the field, and add a label control to each template. Use the Item keyword and databinding syntax for to set the Text property of the labels to the ShortName property of the product's Category property.
15. In the code-behind file, adjust the GetData method for the GridView so it retrieves each product's category data using eager loading. Then, run and test the application.

Add update functionality to the GridView

16. In the aspx file, add an UpdateMethod property to the GridView and select the Create New Method option.
17. In the code-behind file, adjust the method stub so it doesn't use fully qualified names and so its parameter has the same name and type as the field that's identified in the DataKeyNames property of the GridView control.
18. Add code that uses the parameter that's passed to the method to retrieve the selected product from the object context and store it in a variable named item. Then, add code that saves changes to the object context if the model state is valid.
19. Back in the aspx file, find the ValidationSummary control and set its ShowModelStateErrors property to true.
20. Run the application and test whether you can update the OnHand amount for a product. Also test what happens when you enter invalid data, like the string value shown above.

Once working, publish the project to your web180odd directory. Submit the URL to your work in the submission point in Lesson 13.