Working with Data in ASP.NET 2.0 :: Paging Report Data in a DataList or Repeater Control

Introduction

Paging and sorting are two very common features when displaying data in an online application. For example, when searching for ASP.NET books at an online bookstore, there may be hundreds of such books, but the report listing the search results lists only ten matches per page. Moreover, the results can be sorted by title, price, page count, author name, and so on. As we discussed in the Paging and Sorting Report Data tutorial, the GridView, DetailsView, and FormView controls all provide built-in paging support that can be enabled at the tick of a checkbox. The GridView also includes sorting support.

Unfortunately, neither the DataList nor Repeater offer automatic paging or sorting support. In this tutorial we'll examine how to add paging support to the DataList or Repeater. We must manually create the paging interface, display the appropriate page of records, and remember the page being visited across postbacks. While this does take more time and code than with the GridView, DetailsView, or FormView, the DataList and Repeater allow for much more flexible paging and data display interfaces.

Note: This tutorial focuses exclusively on paging. In the next tutorial we'll turn our attention to adding sorting capabilities.

Step 1: Adding the Paging and Sorting Tutorial Web Pages

Before we start this tutorial, let's first take a moment to add the ASP.NET pages we'll need for this tutorial and the next one. Start by creating a new folder in the project named `PagingSortingDataListRepeater`. Next, add the following five ASP.NET pages to this folder, having all of them configured to use the master page `Site.master`:

- `Default.aspx`
- `Paging.aspx`
- `Sorting.aspx`
- `SortingWithDefaultPaging.aspx`
- `SortingWithCustomPaging.aspx`

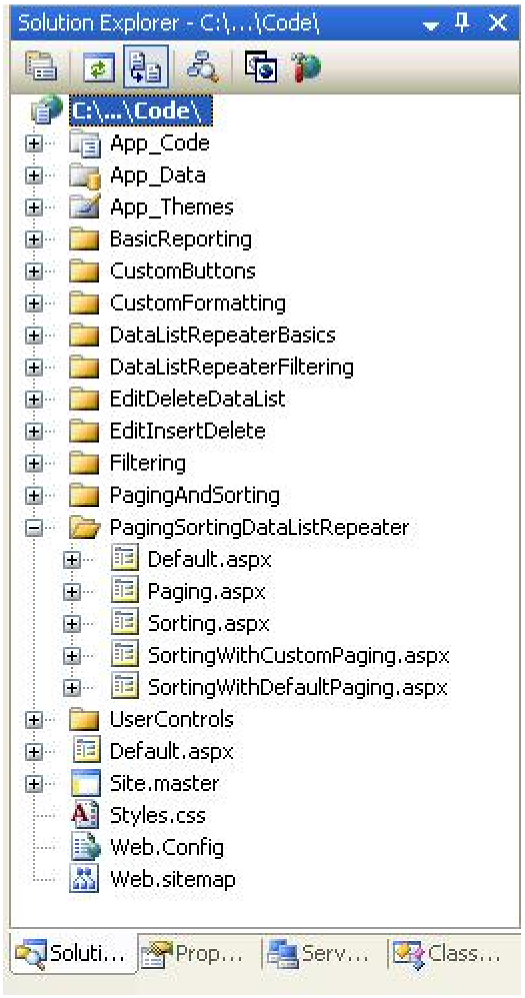


Figure 1: Create a `PagingSortingDataListRepeater` Folder and Add the Tutorial ASP.NET Pages

Next, open the `Default.aspx` page and drag the `SectionLevelTutorialListing.ascx` User Control from the `UserControls` folder onto the Design surface. This User Control, which we created in the Master Pages and Site Navigation tutorial, enumerates the site map and displays those tutorials in the current section in a bulleted list.

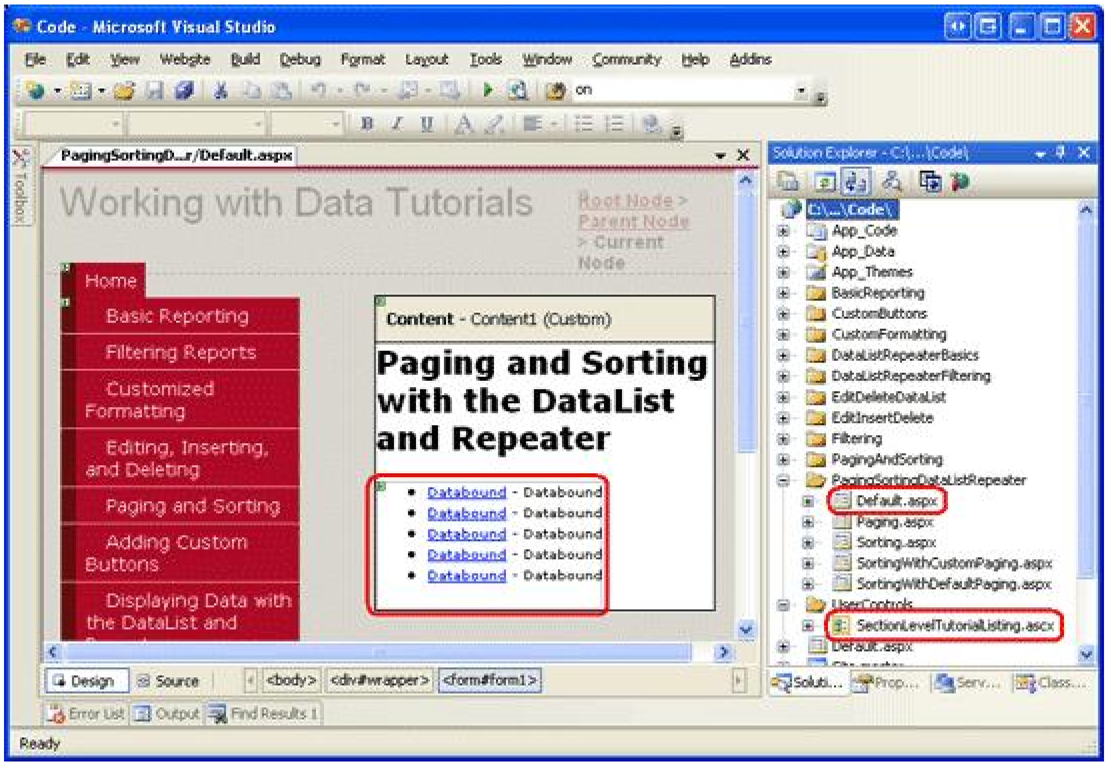

Code - Microsoft Visual Studio
File Edit View Website Build Debug Format Layout Tools Window Community Help Addins
PagingSortingD...r/Default.aspx
Working with Data Tutorials
Root Node > Parent Node > Current Node
Home
Basic Reporting
Filtering Reports
Customized Formatting
Editing, Inserting, and Deleting
Paging and Sorting
Adding Custom Buttons
Displaying Data with the DataList and
Content - Content1 (Custom)
Paging and Sorting with the DataList and Repeater
Databound - Databound
Design
Source
<body> <div#wrapper> <form#form1>
Solution Explorer - C:\...\Code\
C:\...\Code\
App_Code
App_Data
App_Themes
BasicReporting
CustomButtons
CustomFormatting
DataListRepeaterBasics
DataListRepeaterFiltering
EditDeleteDataList
EditInsertDelete
Filtering
PagingAndSorting
PagingSortingDataListRepeater
Default.aspx
Paging.aspx
Sorting.aspx
SortingWithCustomPaging.aspx
SortingWithDefaultPaging.aspx
UserControls
SectionLevelTutorialListing.ascx
Default.aspx
Error List
Output
Find Results 1
Ready

Figure 2: Add the `SectionLevelTutorialListing.ascx` User Control to `Default.aspx`

In order to have the bulleted list display the paging and sorting tutorials we'll be creating, we need to add them to the site map. Open the `Web.sitemap` file and add the following markup after the "Editing and Deleting with the DataList" site map node markup:

```
<siteMapNode
    url="~/PagingSortingDataListRepeater/Default.aspx"
    title="Paging and Sorting with the DataList and Repeater"
    description="Paging and Sorting the Data in the DataList and Repeater Controls">
    <siteMapNode
        url="~/PagingSortingDataListRepeater/Paging.aspx"
        title="Paging"
        description="Learn how to page through the data shown
                     in the DataList and Repeater controls." />
    <siteMapNode
        url="~/PagingSortingDataListRepeater/Sorting.aspx"
        title="Sorting"
        description="Sort the data displayed in a DataList or
                     Repeater control." />
    <siteMapNode
        url="~/PagingSortingDataListRepeater/SortingWithDefaultPaging.aspx"
        title="Sorting with Default Paging"
        description="Create a DataList or Repeater control that is paged using
                     default paging and can be sorted." />
    <siteMapNode
        url="~/PagingSortingDataListRepeater/SortingWithCustomPaging.aspx"
        title="Sorting with Custom Paging"
        description="Learn how to sort the data displayed in a DataList or
                     Repeater control that uses custom paging." />
</siteMapNode>
```

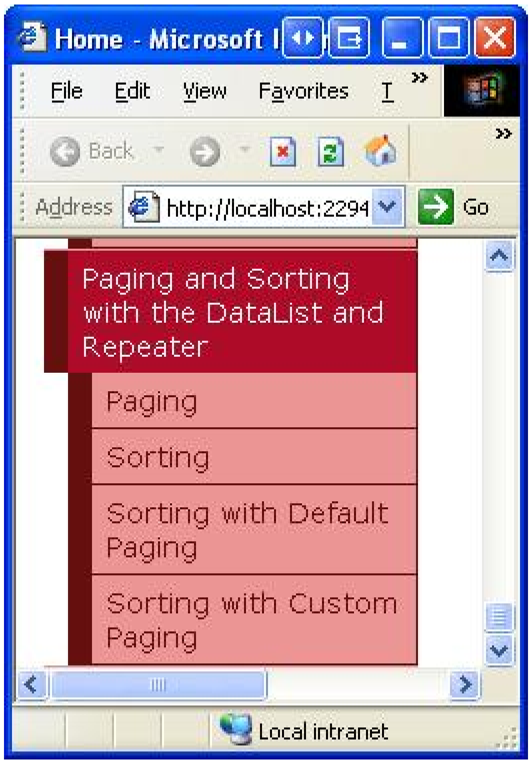

Home - Microsoft I
File
Edit
View
Favorites
T
Back
Address
http://localhost:2294
Go
Paging and Sorting with the DataList and Repeater
Paging
Sorting
Sorting with Default Paging
Sorting with Custom Paging
Local intranet

Figure 3: Update the Site Map to Include the New ASP.NET Pages

A Review of Paging

In previous tutorials we saw how to page through the data in the GridView, DetailsView, and FormView controls. These three controls offer a simple form of paging called *default paging* that can be implemented by simply checking the "Enable Paging" option in the control's smart tag. With default paging, each time a page of data is requested – either on the first page visit or when the user navigates to a different page of data – the GridView, DetailsView, or FormView control re-requests *all* of the data from the ObjectDataSource. It then snips out the particular set of records to display given the requested page index and the number of records to display per page. We discussed default paging in detail in the Paging and Sorting Report Data tutorial.

Since default paging re-requests all records for each page, it is not practical when paging through sufficiently large amounts of data. For example, imagine paging through 50,000 records with a page size of 10. Each time the user moves to a new page, all 50,000 records must be retrieved from the database, even though only ten of them are displayed.

Custom paging solves the performance concerns of default paging by grabbing only the precise subset of records to display on the requested page. When implementing custom paging, we must write the SQL query that will efficiently return just the correct set of records. We saw how to create such a query using SQL Server 2005's new `ROW_NUMBER()` keyword back in the Efficiently Paging Through Large Amounts of Data tutorial.

To implement default paging in the DataList or Repeater controls, we can use the `PagedDataSource` class as a wrapper around the `ProductsDataTable` whose contents are being paged. The `PagedDataSource` class has a `DataSource` property that can be assigned to any enumerable object and `PageSize` and `CurrentPageIndex` properties that indicate how many records to show per page and the current page index. Once these properties have been set, the `PagedDataSource` can be used as the data source of any data Web control. The `PagedDataSource`, when enumerated, will only return the appropriate subset of records of its inner `DataSource` based on the `PageSize` and `CurrentPageIndex` properties. Figure 4 depicts the functionality of the `PagedDataSource` class.

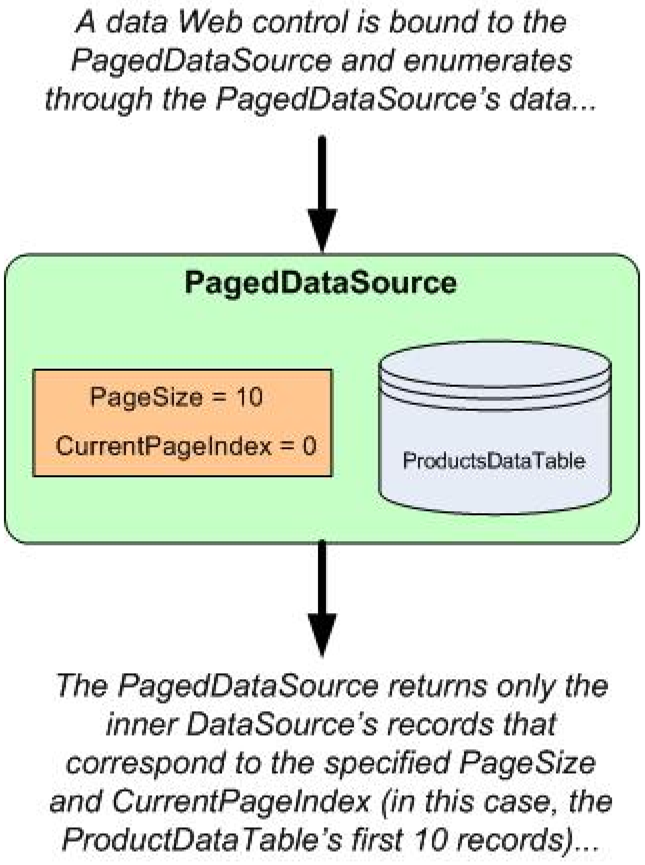

A data Web control is bound to the PagedDataSource and enumerates through the PagedDataSource's data...
PagedDataSource
PageSize = 10
CurrentPageIndex = 0
ProductsDataTable
The PagedDataSource returns only the inner DataSource's records that correspond to the specified PageSize and CurrentPageIndex (in this case, the ProductDataTable's first 10 records)...

Figure 4: The `PagedDataSource` Wraps an Enumerable Object with a Pageable Interface

The `PagedDataSource` object can be created and configured directly from the Business Logic Layer and bound to a DataList or Repeater through an ObjectDataSource, or can be created and configured directly in the ASP.NET page's code-behind class. If the latter approach is used, we must forgo using the ObjectDataSource and instead bind the paged data to the DataList or Repeater programmatically.

The `PagedDataSource` object also has properties to support custom paging. However, we can bypass using a `PagedDataSource` for custom paging because we already have BLL methods in the `ProductsBLL` class designed for custom paging that return the precise records to display.

In this tutorial we'll look at implementing default paging in a DataList by adding a new method to the `ProductsBLL` class that returns an appropriately configured `PagedDataSource` object. In the next tutorial, we'll see how to use custom paging.

Step 2: Adding a Default Paging Method in the Business Logic Layer

The `ProductsBLL` class currently has a method for returning all product information – `GetProducts()` – and one for returning a particular subset of products at a starting index – `GetProductsPaged(`*`startRowIndex`*`,` *`maximumRows`*`)`. With default paging, the GridView, DetailsView, and FormView controls all use the `GetProducts()` method to retrieve all products, but then use a `PagedDataSource` internally to display only the correct subset of records. To replicate this functionality with the DataList and Repeater controls, we can create a new method in the BLL that mimics this behavior.

Add a method to the `ProductsBLL` class named `GetProductsAsPagedDataSource` that takes in two integer input parameters:

- `pageIndex` – the index of the page to display, indexed at zero, and
- `pageSize` – the number of records to display per page.

`GetProductsAsPagedDataSource` starts by retrieving *all* of the records from `GetProducts()`. It then creates a `PagedDataSource` object, setting its `CurrentPageIndex` and `PageSize` properties to the values of the passed-in `pageIndex` and `pageSize` parameters. The method concludes by returning this configured `PagedDataSource`:

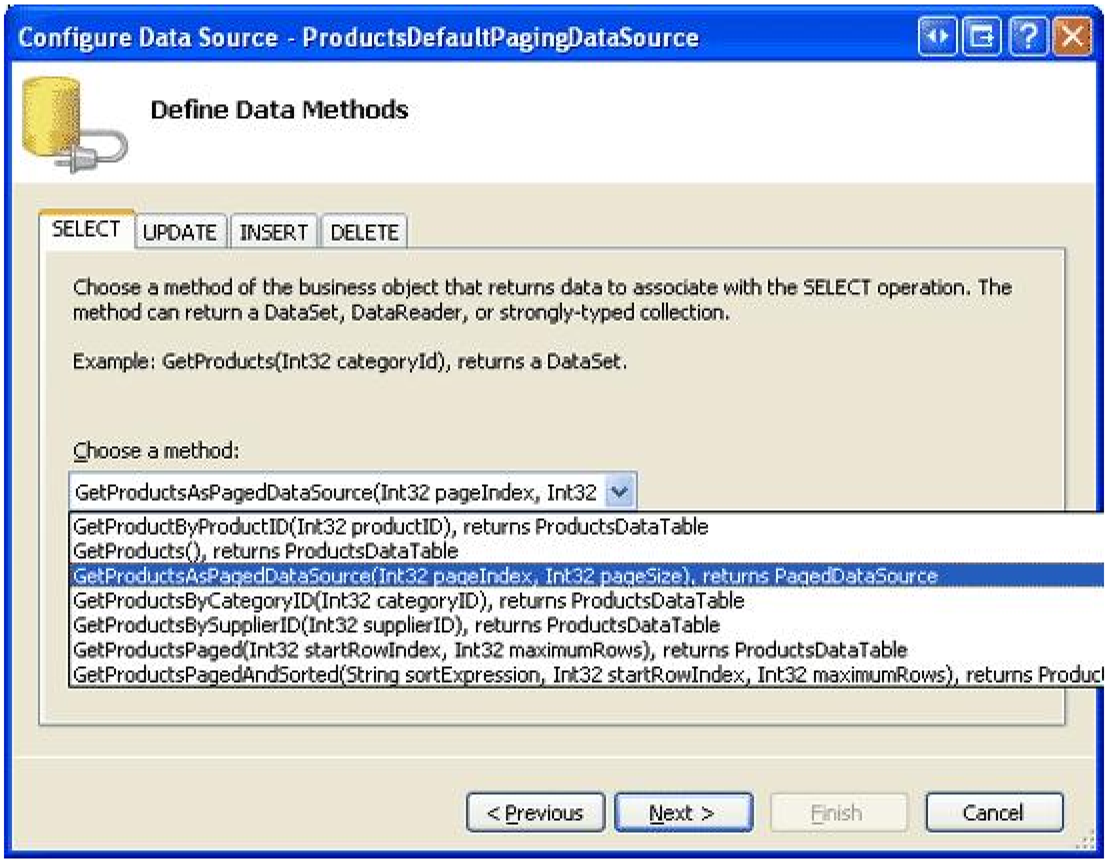
```
<System.ComponentModel.DataObjectMethodAttribute _
    (System.ComponentModel.DataObjectMethodType.Select, False)> _
Public Function GetProductsAsPagedDataSource(ByVal pageIndex As Integer, _
    ByVal pageSize As Integer) As PagedDataSource
    ' Get ALL of the products
    Dim products As Northwind.ProductsDataTable = GetProducts()

    ' Limit the results through a PagedDataSource
    Dim pagedData As New PagedDataSource()
    pagedData.DataSource = products.Rows
    pagedData.AllowPaging = True
    pagedData.CurrentPageIndex = pageIndex
    pagedData.PageSize = pageSize

    Return pagedData
End Function
```

Step 3: Displaying Product Information in a DataList Using Default Paging

With the `GetProductsAsPagedDataSource` method added to the `ProductsBLL` class, we can now create a DataList or Repeater that provides default paging. Start by opening the `Paging.aspx` page in the `PagingSortingDataListRepeater` folder and drag a DataList from the Toolbox onto the Designer, setting the DataList's `ID` property to `ProductsDefaultPaging`. From the DataList's smart tag, create a new ObjectDataSource named `ProductsDefaultPagingDataSource` and configure it so that it retrieves data using the `GetProductsAsPagedDataSource` method.


Configure Data Source - ProductsDefaultPagingDataSource

Define Data Methods

Choose a method of the business object that returns data to associate with the SELECT operation. The method can return a DataSet, DataReader, or strongly-typed collection.

Example: GetProducts(Int32 categoryId), returns a DataSet.

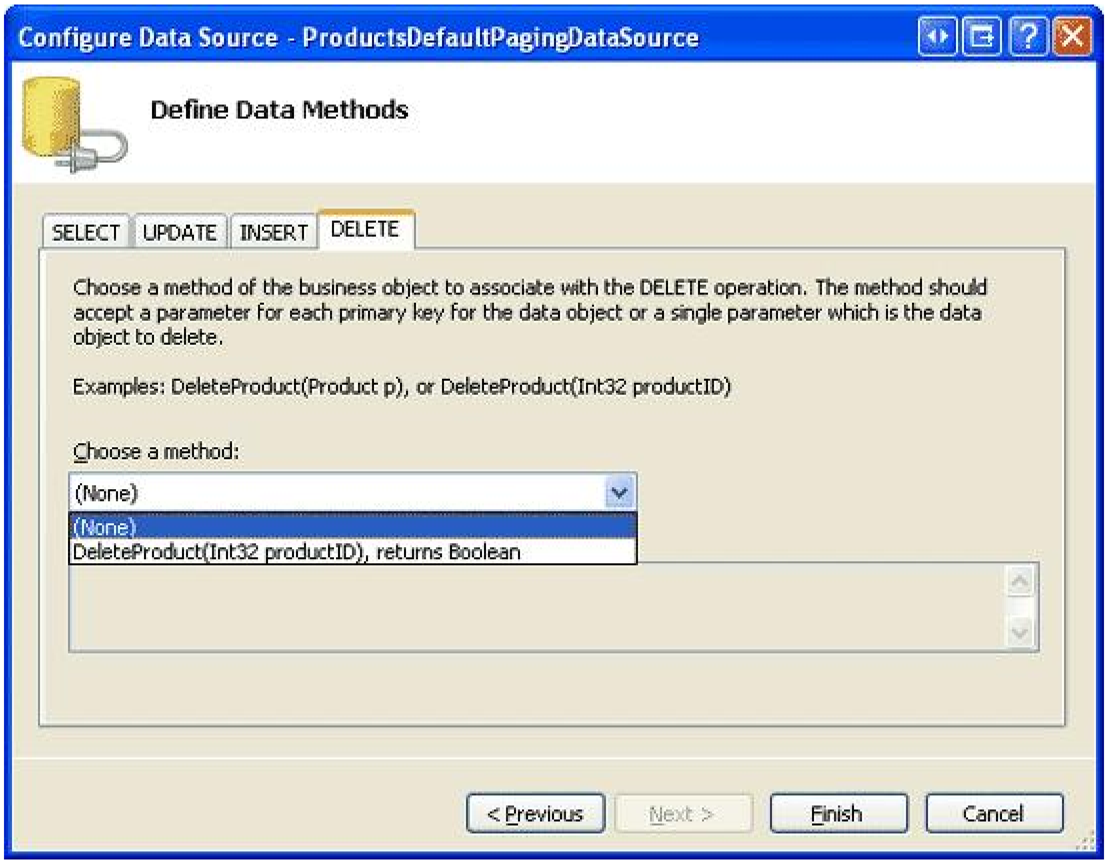
Choose a method:

GetProductsAsPagedDataSource(Int32 pageIndex, Int32

GetProductByProductID(Int32 productID), returns ProductsDataTable
GetProducts(), returns ProductsDataTable
GetProductsAsPagedDataSource(Int32 pageIndex, Int32 pageSize), returns PagedDataSource
GetProductsByCategoryID(Int32 categoryID), returns ProductsDataTable
GetProductsBySupplierID(Int32 supplierID), returns ProductsDataTable
GetProductsPaged(Int32 startRowIndex, Int32 maximumRows), returns ProductsDataTable
GetProductsPagedAndSorted(String sortExpression, Int32 startRowIndex, Int32 maximumRows), returns Product

Figure 5: Create an ObjectDataSource and Configure it to Use the `GetProductsAsPagedDataSource ()` Method

Set the drop-down lists in the UPDATE, INSERT, and DELETE tabs to "(None)".


Configure Data Source - ProductsDefaultPagingDataSource

Define Data Methods

Choose a method of the business object to associate with the DELETE operation. The method should accept a parameter for each primary key for the data object or a single parameter which is the data object to delete.

Examples: DeleteProduct(Product p), or DeleteProduct(Int32 productID)

Choose a method:

(None)

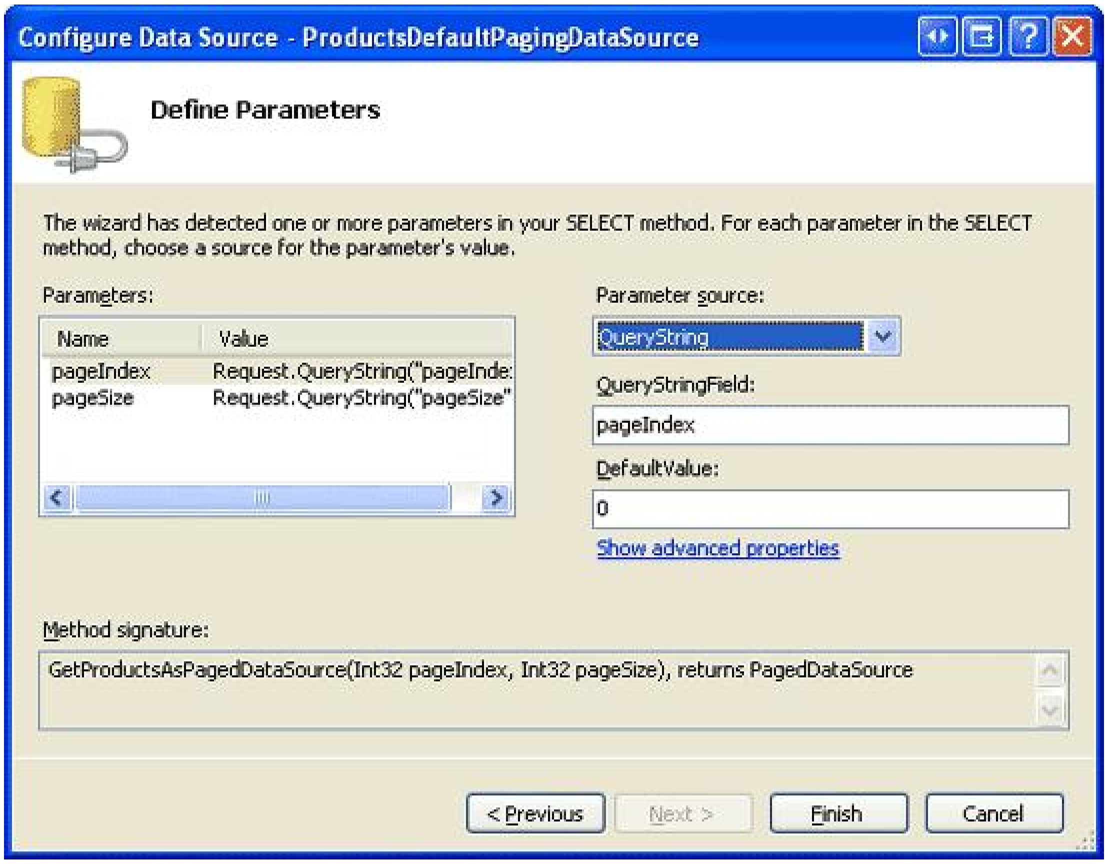
(None)
DeleteProduct(Int32 productID), returns Boolean

Figure 6: Set the Drop-Down Lists in the UPDATE, INSERT, and DELETE tabs to "(None)"

Since the `GetProductsAsPagedDataSource` method expects two input parameters, the wizard prompts us for the source of these parameter values.

The page index and page size values must be remembered across postbacks. They can be stored in view state, persisted to the querystring, stored in session variables, or remembered using some other technique. For this tutorial we'll use the querystring, which has the advantage of allowing a particular page of data to be bookmarked.

In particular, use the querystring fields "pageIndex" and "pageSize" for the `pageIndex` and `pageSize` parameters, respectively (see Figure 7). Take a moment to set the default values for these parameters, as the querystring values won't be present when a user first visits this page. For `pageIndex`, set the default value to 0 (which will show the first page of data) and `pageSize`'s default value to 4.


Configure Data Source - ProductsDefaultPagingDataSource

Define Parameters

The wizard has detected one or more parameters in your SELECT method. For each parameter in the SELECT method, choose a source for the parameter's value.

Parameters:

Name	Value
pageIndex	Request.QueryString("pageInde:
pageSize	Request.QueryString("pageSize"

Parameter source: QueryString

QueryStringField: pageIndex

DefaultValue: 0

Show advanced properties

Method signature:

GetProductsAsPagedDataSource(Int32 pageIndex, Int32 pageSize), returns PagedDataSource

Figure 7: Use the QueryString as the Source for the `pageIndex` and `pageSize` Parameters

After configuring the ObjectDataSource, Visual Studio automatically creates an `ItemTemplate` for the DataList. Customize the `ItemTemplate` so that only the product's name, category, and supplier are shown. Also set the DataList's `RepeatColumns` property to 2, its `Width` to "100%", and its `ItemStyle`'s `Width` to "50%". These width settings will provide equal spacing for the two columns.

After making these changes, the DataList and ObjectDataSource's markup should look similar to the following:

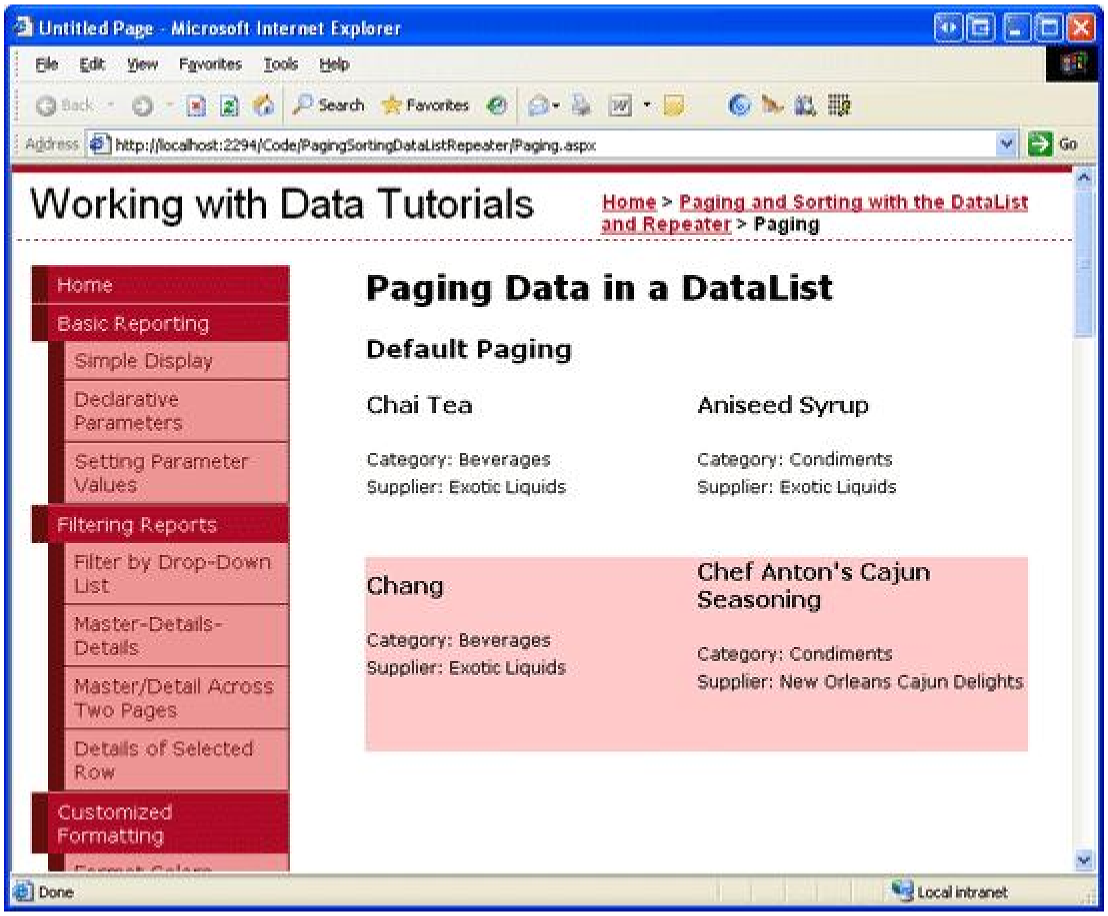
```
<asp:DataList ID="ProductsDefaultPaging" runat="server" Width="100%"
    DataKeyField="ProductID" DataSourceID="ProductsDefaultPagingDataSource"
    RepeatColumns="2" EnableViewState="False">
    <ItemTemplate>
        <h4><asp:Label ID="ProductNameLabel" runat="server"
            Text='<%# Eval("ProductName") %>'></asp:Label></h4>
        Category:
```

```
        <asp:Label ID="CategoryNameLabel" runat="server"
            Text='<%# Eval("CategoryName") %>'></asp:Label><br />
        Supplier:
        <asp:Label ID="SupplierNameLabel" runat="server"
            Text='<%# Eval("SupplierName") %>'></asp:Label><br />
        <br />
        <br />
    </ItemTemplate>
    <ItemStyle Width="50%" />
</asp:DataList>

<asp:ObjectDataSource ID="ProductsDefaultPagingDataSource" runat="server"
    OldValuesParameterFormatString="original_{0}" TypeName="ProductsBLL"
    SelectMethod="GetProductsAsPagedDataSource">
    <SelectParameters>
        <asp:QueryStringParameter DefaultValue="0" Name="pageIndex"
             QueryStringField="pageIndex" Type="Int32" />
        <asp:QueryStringParameter DefaultValue="4" Name="pageSize"
             QueryStringField="pageSize" Type="Int32" />
    </SelectParameters>
</asp:ObjectDataSource>
```

Note: Since we are not performing any update or delete functionality in this tutorial, you may disable the DataList's view state to reduce the rendered page size.

When initially visiting this page through a browser, neither the `pageIndex` nor `pageSize` querystring parameters are provided. Hence, the default values of 0 and 4 are used. As Figure 8 shows, this results in a DataList that displays the first four products.


Untitled Page - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:2294/Code/PagingSortingDataListRepeater/Paging.aspx
Go
Working with Data Tutorials
Home > Paging and Sorting with the DataList and Repeater > Paging
Home
Basic Reporting
Simple Display
Declarative Parameters
Setting Parameter Values
Filtering Reports
Filter by Drop-Down List
Master-Details-Details
Master/Detail Across Two Pages
Details of Selected Row
Customized Formatting
Done
Local intranet

Paging Data in a DataList

Default Paging

Chai Tea

Category: Beverages
Supplier: Exotic Liquids

Aniseed Syrup

Category: Condiments
Supplier: Exotic Liquids

Chang

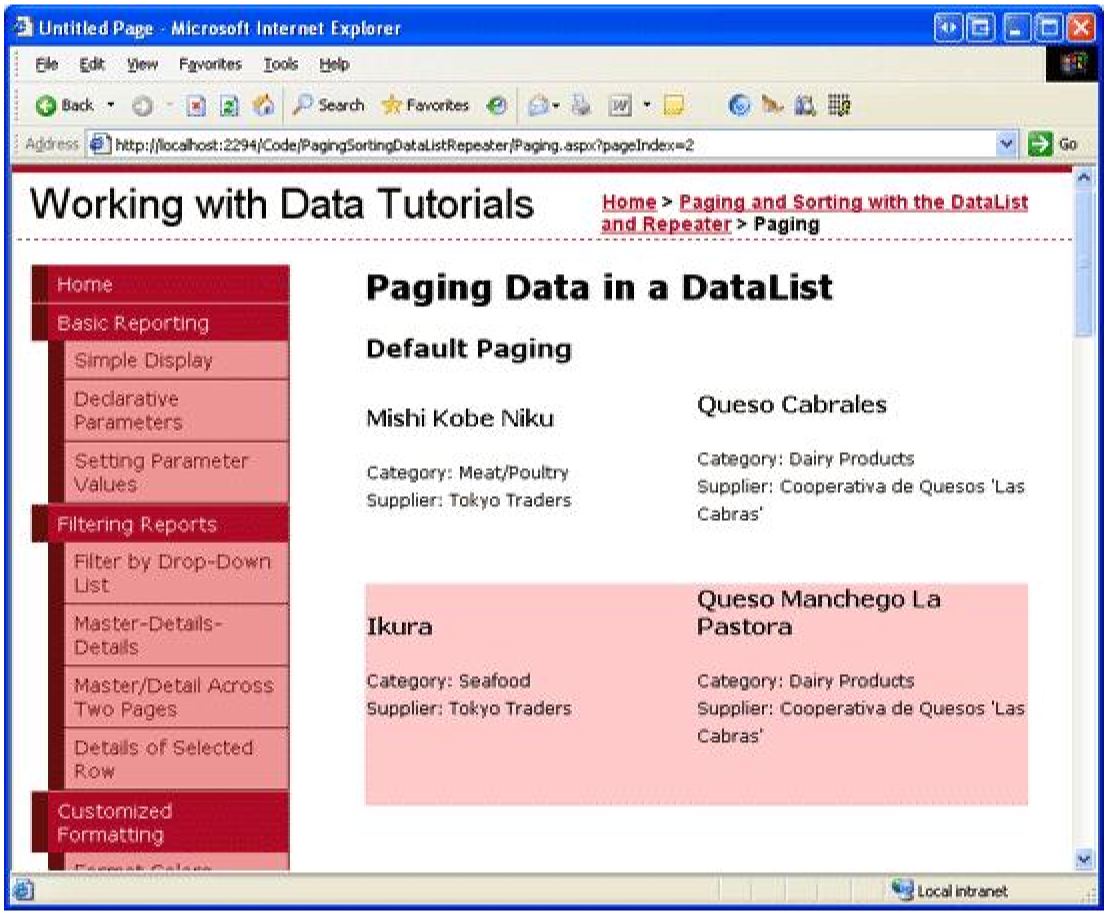
Category: Beverages
Supplier: Exotic Liquids

Chef Anton's Cajun Seasoning

Category: Condiments
Supplier: New Orleans Cajun Delights

Figure 8: The First Four Products are Listed

Without a paging interface, there's currently no straightforward means for a user to navigate to the second page of data. We'll create a paging interface in step 4. For now, though, paging can only be accomplished by directly specifying the paging criteria in the querystring. For example, to view the second page, change the URL in the browser's Address bar from `Paging.aspx` to `Paging.aspx?pageIndex=2` and hit Enter. This causes the second page of data to be displayed (see Figure 9).


Untitled Page - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:2294/Code/PagingSortingDataListRepeater/Paging.aspx?pageIndex=2
Go
Working with Data Tutorials
Home > Paging and Sorting with the DataList and Repeater > Paging
Home
Basic Reporting
Simple Display
Declarative Parameters
Setting Parameter Values
Filtering Reports
Filter by Drop-Down List
Master-Details-Details
Master/Detail Across Two Pages
Details of Selected Row
Customized Formatting
Local intranet

Paging Data in a DataList

Default Paging

Mishi Kobe Niku
Category: Meat/Poultry
Supplier: Tokyo Traders

Queso Cabrales
Category: Dairy Products
Supplier: Cooperativa de Quesos 'Las Cabras'

Ikura
Category: Seafood
Supplier: Tokyo Traders

Queso Manchego La Pastora
Category: Dairy Products
Supplier: Cooperativa de Quesos 'Las Cabras'

Figure 9: The Second Page of Data is Displayed

Step 4: Creating the Paging Interface

There are a variety of different paging interfaces that can be implemented. The GridView, DetailsView, and FormView controls provide four different interfaces to choose among:

- **Next, Previous** – users can move one page at a time, to either the next or previous one.
- **Next, Previous, First, Last** – in addition to Next and Previous buttons, this interface includes First and Last buttons for moving to the very first or very last page.
- **Numeric** – lists the page numbers in the paging interface, allowing a user to quickly jump to a particular page.
- **Numeric, First, Last** – in addition to the numeric page numbers, includes buttons for moving to the very first or very last page.

For the DataList and Repeater, we are responsible for deciding upon a paging interface and implementing it. This involves creating the needed Web controls in the page and displaying the requested page when a particular paging interface button is clicked. Additionally, certain paging interface controls may need to be disabled. For example, when viewing the first page of data using the Next, Previous, First, Last interface, both the First and Previous buttons would be disabled.

For this tutorial, let's use the Next, Previous, First, Last interface. Add four Button Web controls to the page and set their `ID`s to `FirstPage`, `PrevPage`, `NextPage`, and `LastPage`. Set the `Text` properties to "<< First", "< Prev", "Next >", and "Last >>".

```
<asp:Button runat="server" ID="FirstPage" Text="<< First" />
<asp:Button runat="server" ID="PrevPage" Text="< Prev" />
<asp:Button runat="server" ID="NextPage" Text="Next >" />
<asp:Button runat="server" ID="LastPage" Text="Last >>" />
```

Next, create a `Click` event handler for each of these Buttons. In a moment we'll add the code necessary to display the requested page.

Remembering the Total Number of Records Being Paged Through

Regardless of the paging interface selected, we need to compute and remember the total number of records being paged through. The total row count (in conjunction with the page size) determines how many total pages of data are being paged through, which determines what paging interface controls are added or are enabled. In the Next, Previous, First, Last interface that we are building, the page count is used in two ways:

- To determine whether we are viewing the last page, in which case the Next and Last buttons are disabled.
- If the user clicks the Last button we need to whisk them to the last page, whose index is one less than the page count.

The page count is calculated as the ceiling of the total row count divided by the page size. For example, if we are paging through 79 records with four records per page, then the page count is 20 (the ceiling of 79 / 4). If we are using the Numeric paging interface, this information informs us as to how many numeric page buttons to display; if our paging interface includes Next or Last buttons, the page count is used to determine when to disable the Next or Last buttons.

If the paging interface includes a Last button, it is imperative that the total number of records being paged through be remembered across postbacks so that when the Last button is clicked we can determine the last page index. To facilitate this, create a `TotalRowCount` property in the ASP.NET page's code-behind class that persists its value to view state:

```
Private Property TotalRowCount() As Integer
    Get
        Dim o As Object = ViewState("TotalRowCount")
        If (o Is Nothing) Then
            Return -1
        Else
            Return Convert.ToInt32(o)
        End If
    End Get
    set(Value as Integer)
        ViewState("TotalRowCount") = value
    End Set
End Property
```

In addition to `TotalRowCount`, take a minute to create read-only page-level properties for easily accessing the page index, page size, and page count:

```
Private ReadOnly Property PageIndex() As Integer
    Get
        If (Not String.IsNullOrEmpty(Request.QueryString("pageIndex"))) Then
```

```
            Return Convert.ToInt32(Request.QueryString("pageIndex"))
        Else
            Return 0
        End If
    End Get
End Property

Private ReadOnly Property PageSize() As Integer
    Get
        If (Not String.IsNullOrEmpty(Request.QueryString("pageSize"))) Then
            Return Convert.ToInt32(Request.QueryString("pageSize"))
        Else
            Return 4
        End If
    End Get
End Property

Private ReadOnly Property PageCount() As Integer
    Get
        If TotalRowCount <= 0 OrElse PageSize <= 0 Then
            Return 1
        Else
            Return ((TotalRowCount + PageSize) - 1) / PageSize
        End If
    End Get
End Property
```

Determining the Total Number of Records Being Paged Through

The `PagedDataSource` object returned from the ObjectDataSource's `Select()` method has within it *all* of the product records, even though only a subset of them are displayed in the DataList. The `PagedDataSource`'s `Count` property returns only the number of items that will be displayed in the DataList; the `DataSourceCount` property returns the total number of items within the `PagedDataSource`. Therefore, we need to assign the ASP.NET page's `TotalRowCount` property the value of the `PagedDataSource`'s `DataSourceCount` property.

To accomplish this, create an event handler for the ObjectDataSource's `Selected` event. In the `Selected` event handler we have access to the return value of the ObjectDataSource's `Select()` method – in this case, the `PagedDataSource`.

```
Protected Sub ProductsDefaultPagingDataSource_Selected(ByVal sender As Object, _
    ByVal e As System.Web.UI.WebControls.ObjectDataSourceStatusEventArgs) _
    Handles ProductsDefaultPagingDataSource.Selected
    ' Reference the PagedDataSource bound to the DataList
    Dim pagedData As PagedDataSource = CType(e.ReturnValue, PagedDataSource)

    ' Remember the total number of records being paged through across postbacks
    TotalRowCount = pagedData.DataSourceCount
End Sub
```

Displaying the Requested Page of Data

When the user clicks one of the buttons in the paging interface, we need to display the requested page of data. Since the paging parameters are specified via the querystring, to show the requested page of data use `Response.Redirect(`*`url`*`)` to have the user's browser re-request the `Paging.aspx` page with the appropriate paging parameters. For example, to display the second page of data, we would redirect the user to `Paging.aspx?pageIndex=1`.

To facilitate this, create a `RedirectUser(`*`sendUserToPageIndex`*`)` method that redirects the user to `Paging.aspx?pageIndex=`*`sendUserToPageIndex`*. Then, call this method from the four Button `Click` event handlers. In the `FirstPage Click` event handler, call `RedirectUser(0)`, to send them to the first page; in the `PrevPage Click` event handler, use `PageIndex - 1` as the page index; and so on.

```
Protected Sub FirstPage_Click(ByVal sender As Object, ByVal e As EventArgs) _
    Handles FirstPage.Click
    ' Send the user to the first page
    RedirectUser(0)
End Sub

Protected Sub PrevPage_Click(ByVal sender As Object, ByVal e As EventArgs) _
    Handles PrevPage.Click
    ' Send the user to the previous page
    RedirectUser(PageIndex - 1)
End Sub

Protected Sub NextPage_Click(ByVal sender As Object, ByVal e As EventArgs) _
    Handles NextPage.Click
    ' Send the user to the next page
    RedirectUser(PageIndex + 1)
End Sub

Protected Sub LastPage_Click(ByVal sender As Object, ByVal e As EventArgs) _
    Handles LastPage.Click
    ' Send the user to the last page
    RedirectUser(PageCount - 1)
End Sub

Private Sub RedirectUser(ByVal sendUserToPageIndex As Integer)
    ' Send the user to the requested page
    Response.Redirect(String.Format("Paging.aspx?pageIndex={0}&pageSize={1}", _
        sendUserToPageIndex, PageSize))
End Sub
```

With the `Click` event handlers complete, the DataList's records can be paged through by clicking the buttons. Take a moment to try it out!

Disabling Paging Interface Controls

Currently, all four buttons are enabled regardless of the page being viewed. However, we want to disable the First and Previous buttons when showing the first page of data, and the Next and Last buttons when showing the last page. The `PagedDataSource` object returned by the ObjectDataSource's `Select()` method has properties – `IsFirstPage` and `IsLastPage` – that we can examine to determine if we are viewing the first or last page of data.

Add the following to the ObjectDataSource's `Selected` event handler:

```
' Configure the paging interface based on the data in the PagedDataSource
FirstPage.Enabled = Not pagedData.IsFirstPage
PrevPage.Enabled = Not pagedData.IsFirstPage
NextPage.Enabled = Not pagedData.IsLastPage
LastPage.Enabled = Not pagedData.IsLastPage
```

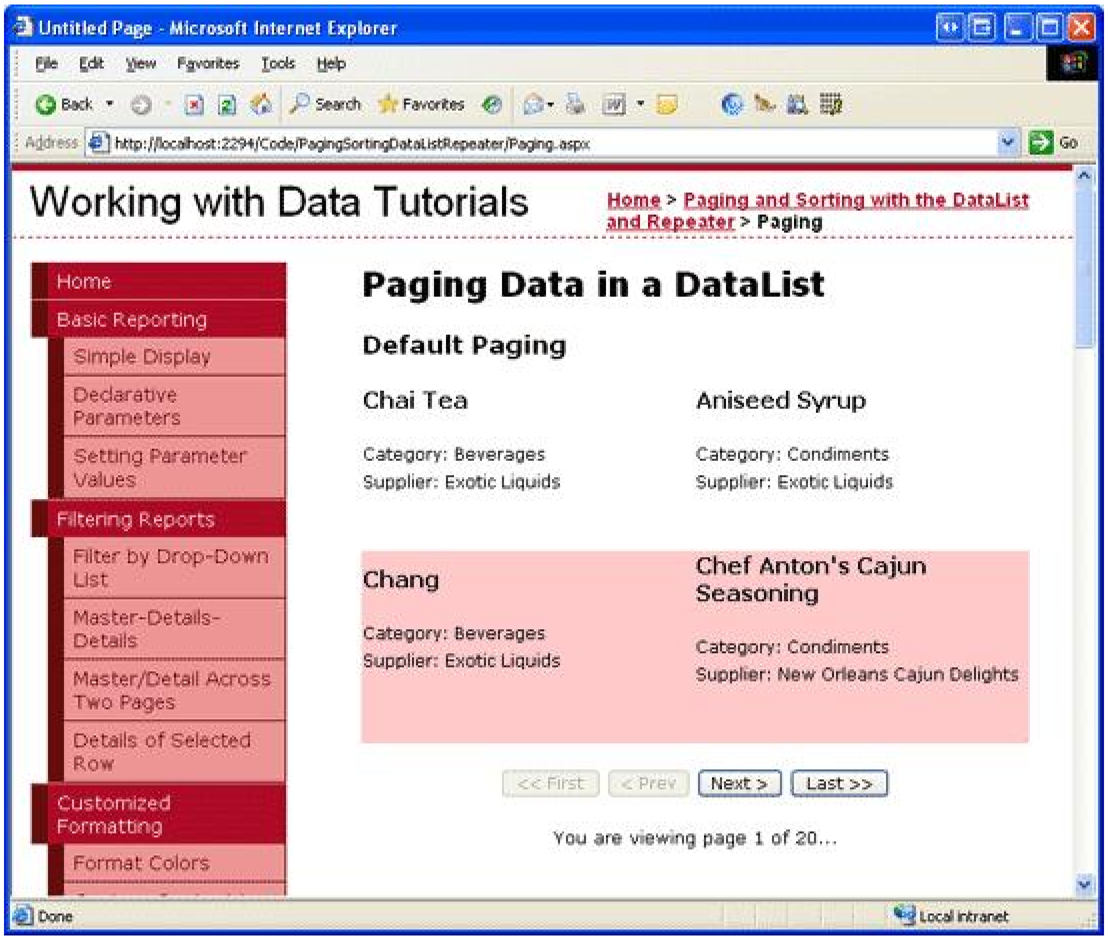
With this addition, the First and Previous buttons will be disabled when viewing the first page, while the Next and Last buttons will be disabled when viewing the last page.

Let's complete the paging interface by informing the user what page they're currently viewing and how many total pages exist. Add a Label Web control to the page and set its `ID` property to `CurrentPageNumber`. Set its `Text` property in the ObjectDataSource's Selected event handler such that it includes the current page being viewed

(`PageIndex + 1`) and the total number of pages (`PageCount`).

```
' Display the current page being viewed...
CurrentPageNumber.Text = String.Format("You are viewing page {0} of {1}...", _
    PageIndex + 1, PageCount)
```

Figure 10 shows `Paging.aspx` when first visited. Since the querystring is empty, the DataList defaults to showing the first four products; the First and Previous buttons are disabled. Clicking Next displays the next four records (see Figure 11); the First and Previous buttons are now enabled.


Untitled Page - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:2294/Code/PagingSortingDataListRepeater/Paging.aspx
Go
Working with Data Tutorials
Home > Paging and Sorting with the DataList and Repeater > Paging
Home
Basic Reporting
Simple Display
Declarative Parameters
Setting Parameter Values
Filtering Reports
Filter by Drop-Down List
Master-Details-Details
Master/Detail Across Two Pages
Details of Selected Row
Customized Formatting
Format Colors
<< First
< Prev
Next >
Last >>
Done
Local intranet

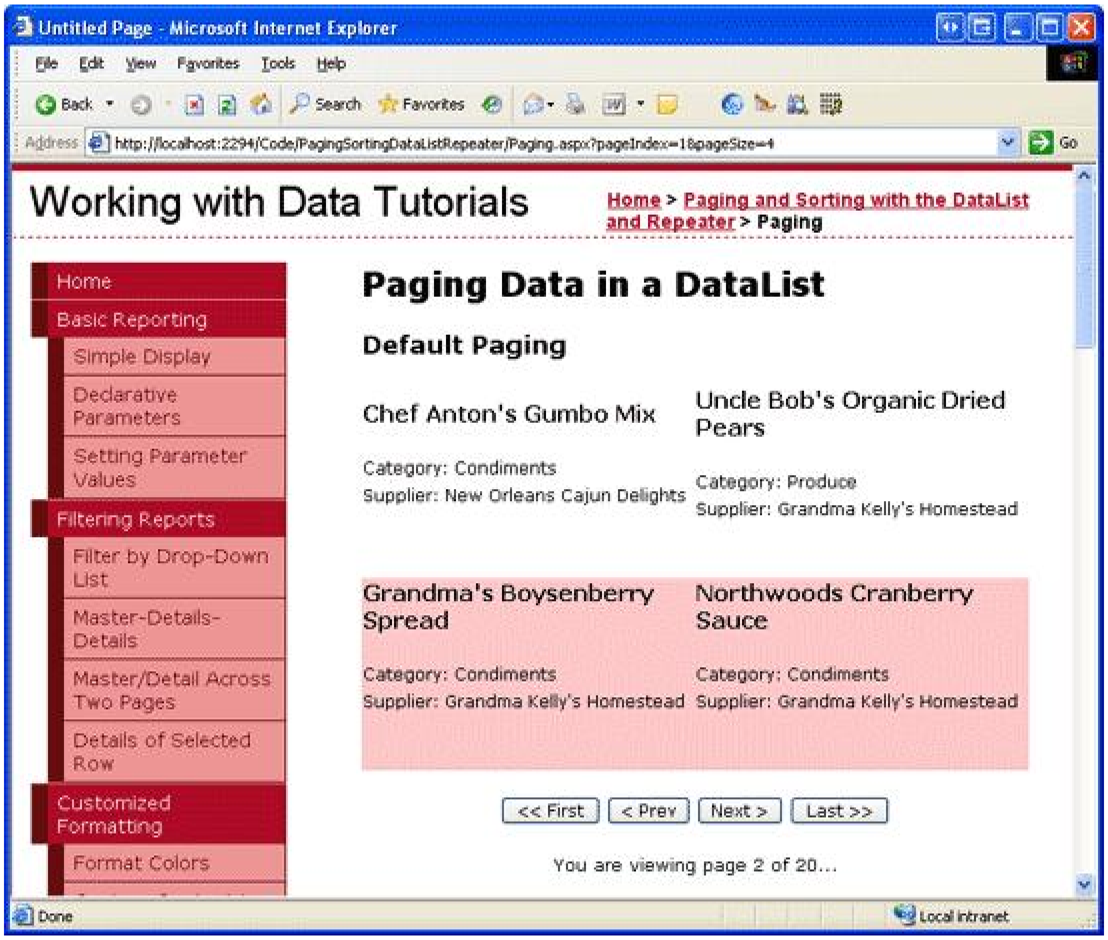
Paging Data in a DataList

Default Paging

Chai Tea Category: Beverages Supplier: Exotic Liquids	Aniseed Syrup Category: Condiments Supplier: Exotic Liquids
Chang Category: Beverages Supplier: Exotic Liquids	Chef Anton's Cajun Seasoning Category: Condiments Supplier: New Orleans Cajun Delights

You are viewing page 1 of 20...

Figure 10: The First Page of Data is Displayed


Untitled Page - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:2294/Code/PagingSortingDataListRepeater/Paging.aspx?pageIndex=1&pageSize=4
Go
Working with Data Tutorials
Home > Paging and Sorting with the DataList and Repeater > Paging
Home
Basic Reporting
Simple Display
Declarative Parameters
Setting Parameter Values
Filtering Reports
Filter by Drop-Down List
Master-Details-Details
Master/Detail Across Two Pages
Details of Selected Row
Customized Formatting
Format Colors
<< First
< Prev
Next >
Last >>
Done
Local intranet

Paging Data in a DataList

Default Paging

Chef Anton's Gumbo Mix

Category: Condiments
Supplier: New Orleans Cajun Delights

Uncle Bob's Organic Dried Pears

Category: Produce
Supplier: Grandma Kelly's Homestead

Grandma's Boysenberry Spread

Category: Condiments
Supplier: Grandma Kelly's Homestead

Northwoods Cranberry Sauce

Category: Condiments
Supplier: Grandma Kelly's Homestead

You are viewing page 2 of 20...

Figure 11: The Second Page of Data is Displayed

Note: The paging interface can be further enhanced by allowing the user to specify how many pages to view per page. For example, a DropDownList could be added listing page size options like 5, 10, 25, 50, and All. Upon selecting a page size, the user would need to be redirected back to `Paging.aspx?pageIndex=0&pageSize=`*`selectedPageSize`*. I leave implementing this enhancement as an exercise for the reader.

Using Custom Paging

The DataList pages through its data using the inefficient default paging technique. When paging through sufficiently large amounts of data, it is imperative that custom paging be used. Although the implementation details differ slightly, the concepts behind implementing custom paging in a DataList are the same as with default paging. With custom paging, use the `ProductBLL` class's `GetProductsPaged` method (instead of `GetProductsAsPagedDataSource`). As discussed in the Efficiently Paging Through Large Amounts of Data tutorial, `GetProductsPaged` must be passed the start row index and maximum number of rows to return. These parameters can be maintained through the querystring just like the `pageIndex` and `pageSize` parameters used in default paging.

Since there's no `PagedDataSource` with custom paging, alternative techniques must be used to determine the total number of records being paged through and whether we're displaying the first or last page of data. The `TotalNumberOfProducts()` method in the `ProductsBLL` class returns the total number of products being paged through. To determine if the first page of data is being viewed, examine the start row index – if it is zero, then the first page is being viewed. The last page is being viewed if the start row index plus the maximum rows to return is greater than or equal to the total number of records being paged through.

We'll explore implementing custom paging in greater detail in the next tutorial.

Summary

While neither the DataList nor Repeater offers the out of the box paging support found in the GridView, DetailsView, and FormView controls, such functionality can be added with minimal effort. The easiest way to implement default paging is to wrap the entire set of products within a `PagedDataSource` and then bind the `PagedDataSource` to the DataList or Repeater. In this tutorial we added the `GetProductsAsPagedDataSource` method to the `ProductsBLL` class to return the `PagedDataSource`. The `ProductsBLL` class already contains the methods needed for custom paging – `GetProductsPaged` and `TotalNumberOfProducts`.

Along with retrieving either the precise set of records to display for custom paging or all records in a `PagedDataSource` for default paging, we also need to manually add the paging interface. For this tutorial we created a Next, Previous, First, Last interface with four Button Web controls. Also, a Label control displaying the current page number and total number of pages was added.

In the next tutorial we'll see how to add sorting support to the DataList and Repeater. We'll also see how to create a DataList that can be both paged and sorted (with examples using default and custom paging).

Happy Programming!

About the Author

Scott Mitchell, author of six ASP/ASP.NET books and founder of 4GuysFromRolla.com, has been working with Microsoft Web technologies since 1998. Scott works as an independent consultant, trainer, and writer, recently completing his latest book, Sams Teach Yourself ASP.NET 2.0 in 24 Hours. He can be reached at mitchell@4guysfromrolla.com or via his blog, which can be found at ScottOnWriting.NET.

Special Thanks To…

This tutorial series was reviewed by many helpful reviewers. Lead reviewers for this tutorial were Liz Shulok, Ken Pespisa, and Bernadette Leigh. Interested in reviewing my upcoming MSDN articles? If so, drop me a line at mitchell@4guysfromrolla.com.