Table of Content

Introduction

When Visual Studio 2005 is released to the market, one thing that came into our attention was that Visual Studio 2005 introduced a new concept on how to build data source and utilize it in an ASP.NET application. This new concept is to some extend quite remarkable, considering it can do automatic data-binding to .NET data-bound controls with "no code" or very few codes in the Presentation Layer which also means that we can now have a clear separation between Presentation Layer, Business Logic Layer and Data Layer.

Responding to this new concept, Intersoft Solutions as one of the leading company has started to adapt the concept of its entire data-bound components which included WebGrid.NET and WebCombo.NET.

During the research, we found out that it was quite hard and ineffective to perform hierarchical data retrieval using native .NET data source control. Considering this limitation and some other aspects that can be enhanced, Intersoft Solutions decided to develop a new data source control which later called ISDataSource.NET

The main purpose of ISDataSource.NET is to support Hierarchical Data binding for Intersoft data-bound components. However, it is also embedded with many other fascinating features. The following are the list of enhanced feature in ISDataSource.NET:

- Multi table-view support and Hierarchical-relational support
- Connects to DataSet and CustomObject in design time
- Enhanced configuration management
- Enhanced designer (step-by-step wizard & automatic table generation)
- Integration with .NET data-bound controls
- Enhanced caching mechanism
- Integration with Intersoft.NET data-bound controls

Architecture

In simple way, ISDataSource.NET can be described as a collection of tables (ISDataSourceTable) with or without pre-defined schema, which will act as proxies for working with Microsoft.NET data-bound controls and Intersoft data-bound controls.

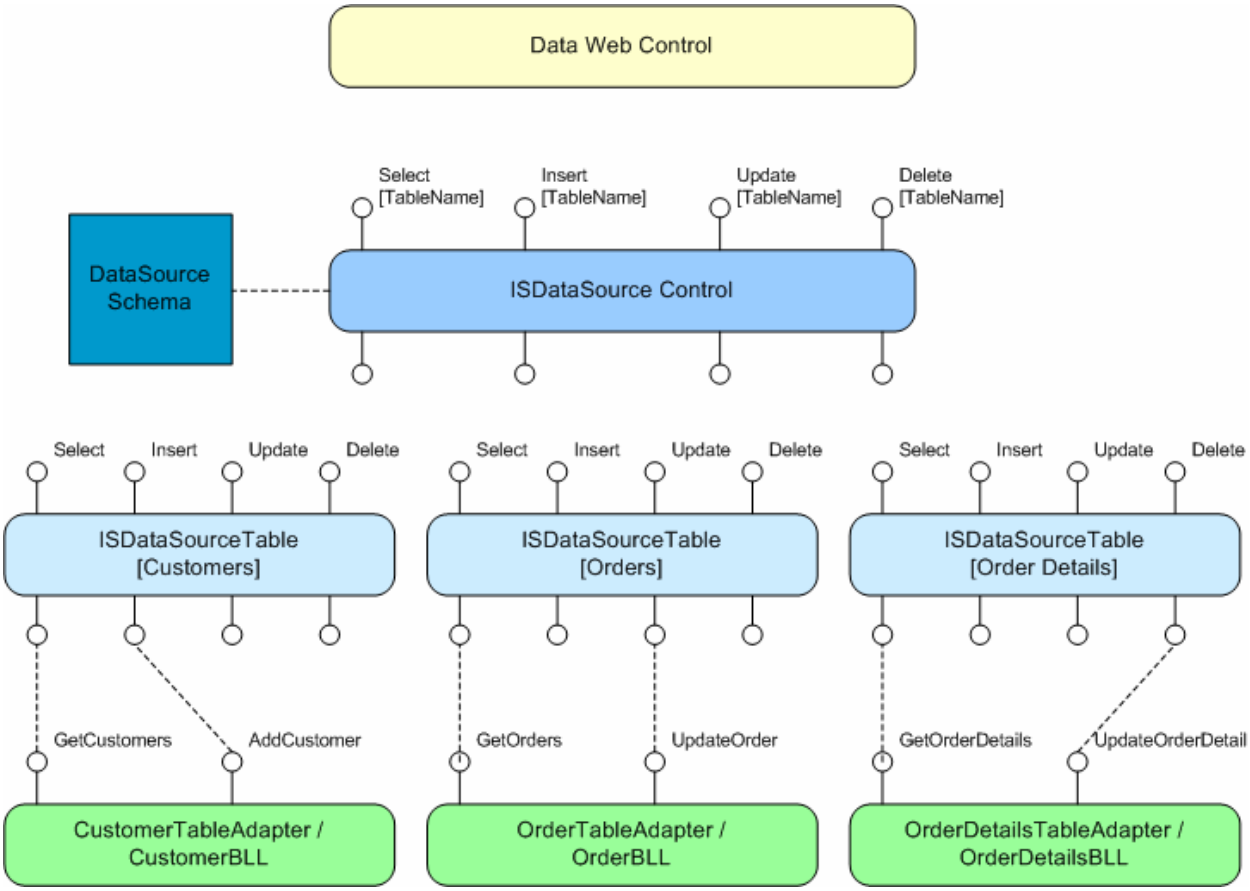

Figure 2.0: Overall design / architecture / concept of ISDataSource.NET

As you have seen from figure 2.0, ISDataSource.NET is used to describe/represent the data structure from your Business Logic Layer or Data Layer itself. Each of the ISDataSourceTable plays a part in determining how the data operations are handled, the same way how ObjectDataSource handle the data operations request.

There are two fundamental things that construct ISDataSource control:

- Collection of Tables (ISDataSourceTable)
  Collection of ISDataSourceTable that hold information on how data is treated (Select/Update/Delete/Insert)

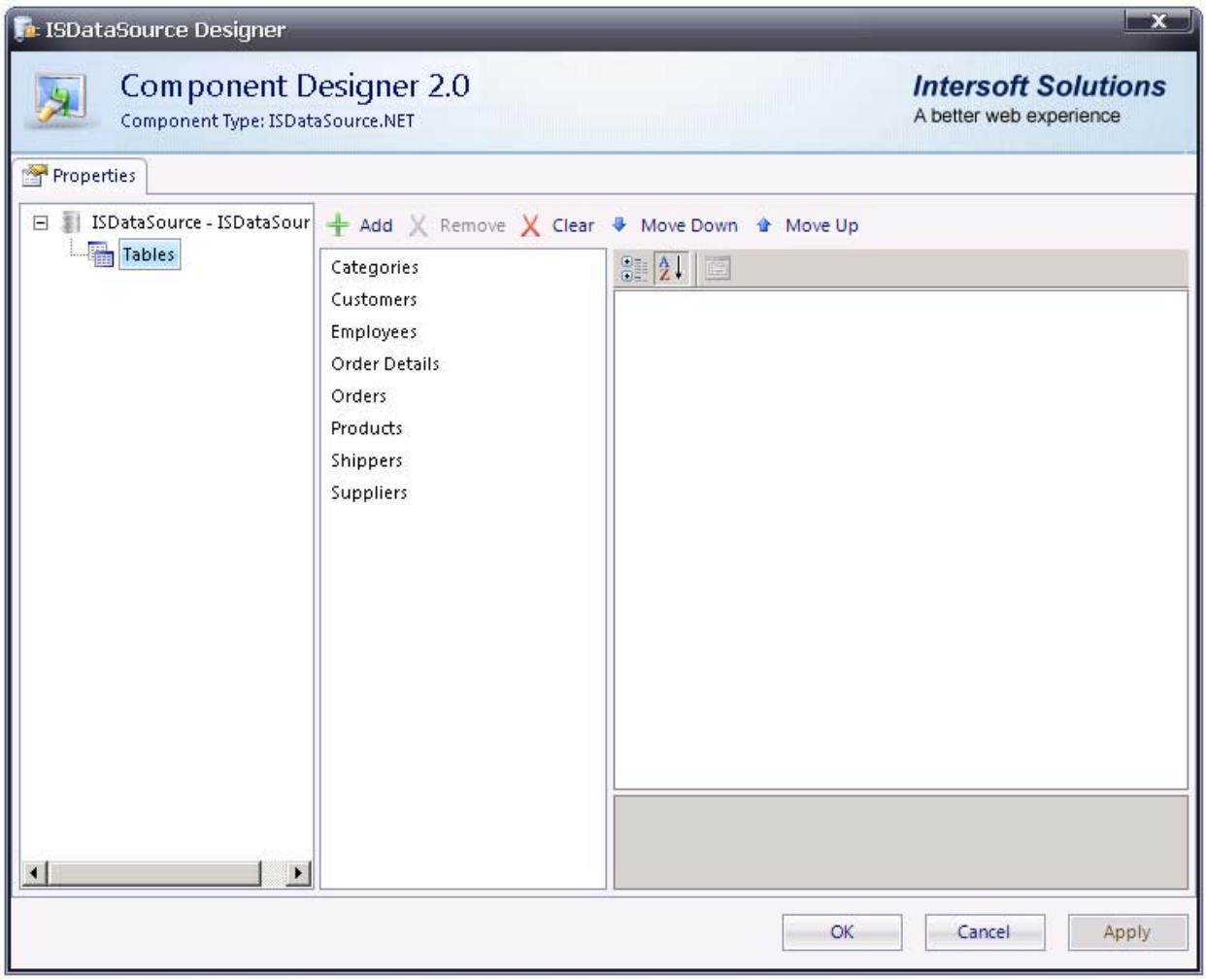

Figure 2.1: Collection of ISDataSource Tables

- DataSourceSchema
  The DataSourceSchema holds information regarding the underlying structure of all tables in ISDataSource control. This structural information is used for hierarchical data retrieval using Intersoft WebGrid.NET control.

Schema	
SchemaName	**dsNorthwind**
SchemaType	DataSet

Figure 2.2a: DataSourceSchema configuration

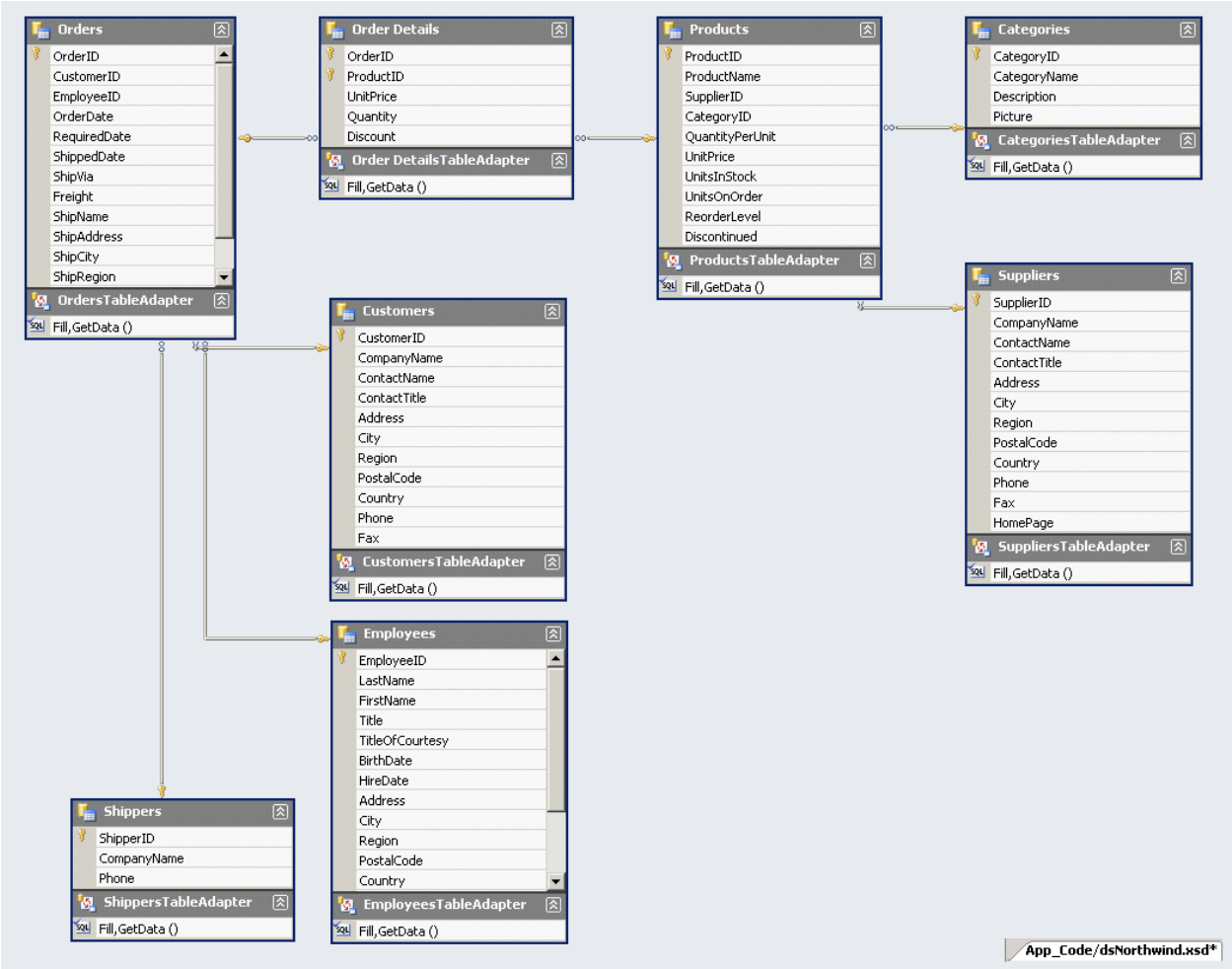

Figure 2.2b: DataSet definition (dsNorthwind.xsd)

The advantages of this architecture are:

- With all tables are collected in one control, you can set a configuration settings from ISDataSource control level or ISDataSourceTable level. Therefore it is easier to manage all the configuration settings for all DataSource that will be used in a page.
- Given the DataSourceSchema information allows the Intersoft WebGrid.NET to perform Retrieve Structure in Design Time and most importantly allow the Intersoft WebGrid.NET to perform Hierarchical binding with no code (codeless Hierarchical data binding)

Organize your data sources in one component

Although ISDataSource.NET is primarily used to perform hierarchical data retrieval, it also supports the standard .NET data source controls binding.

Currently the data source controls that shipped along with Visual Studio 2005 (SqlDataSource, AccessDataSource, and ObjectDataSource) only support one table definition per data source, to some extend this is not a practical approach to do. Considering a scenario where you have a huge and complex table structure and you use lots of table in a single page. You will have lots of data source instances in your designer and it will take times to manage them one by one.

ISDataSource.NET provides a better way to manage your data sources. With only using one control you can organize which tables will be used in the page and how the tables access, retrieve, and modify the data.

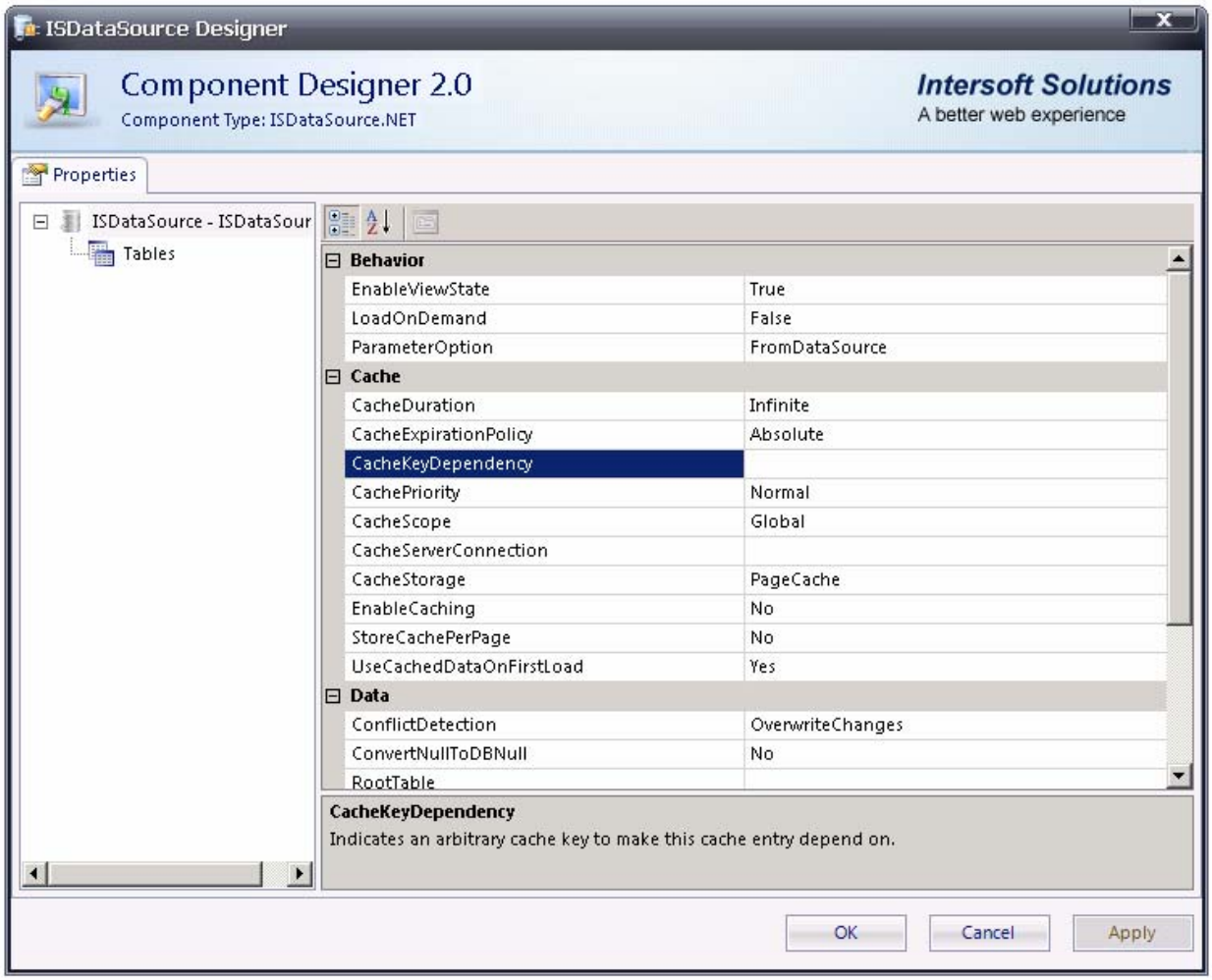

Figure 2.3a: Configuration setting from ISDataSource control level

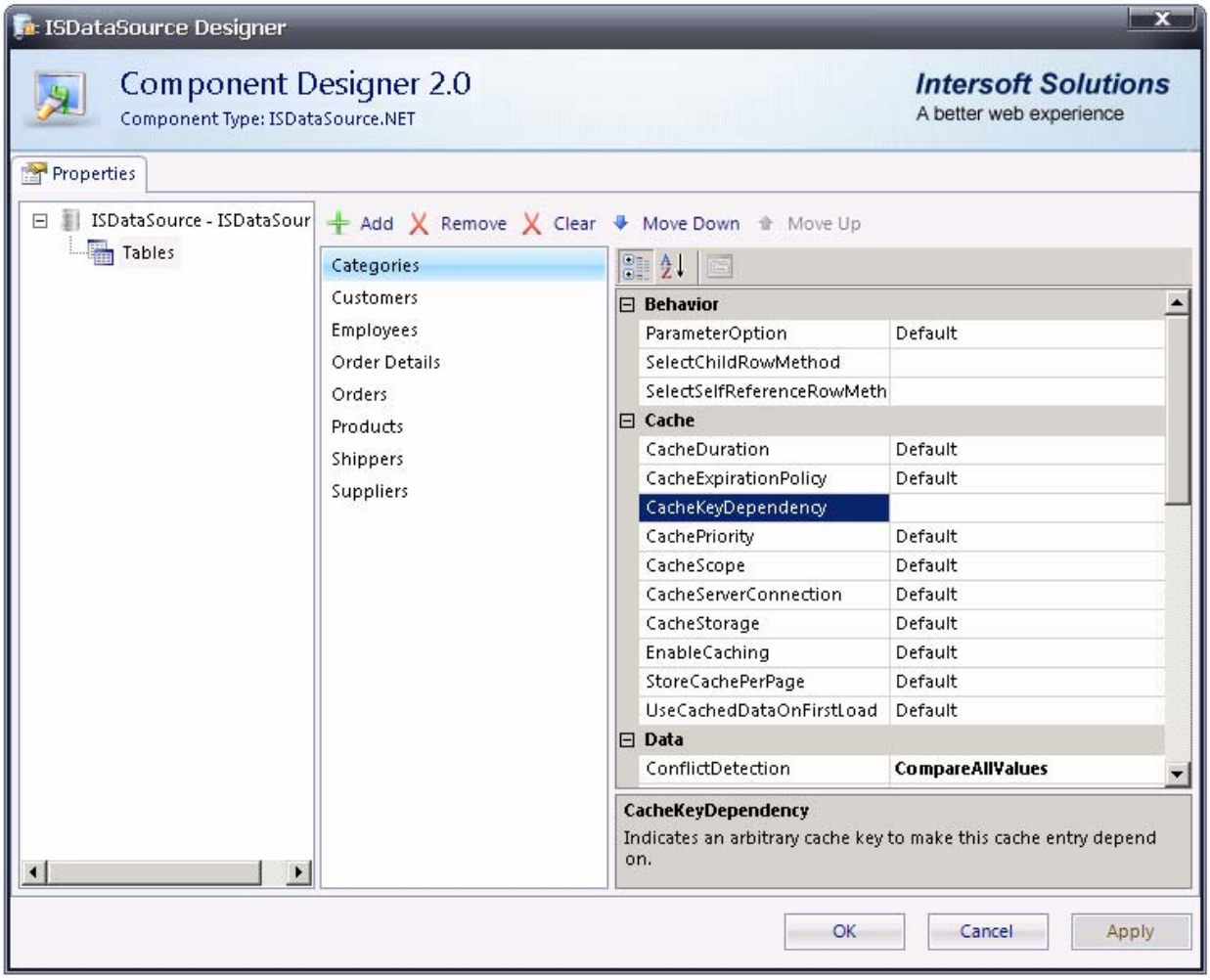

Figure 2.3b: Configuration setting from ISDataSourceTable level

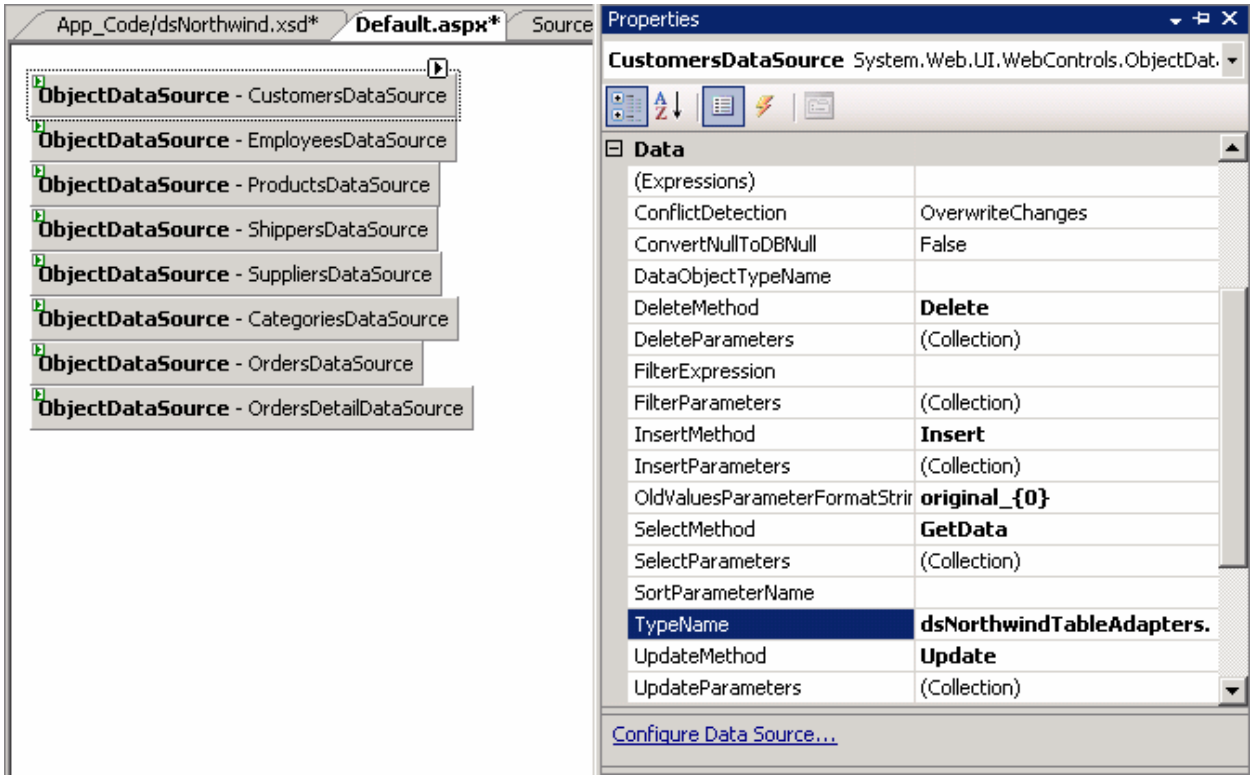

Figure 2.3c: Multiple configurations setting for multiple ObjectDataSource control

With this architecture you can easily add/remove/modify any table that you will use in the page. Moreover, you will be able to control the behavior from ISDataSource level and ISDataSourceTable level more easily.

Integration with .NET data bound controls

Another important feature that ISDataSource.NET provides is the integration to .NET data bound controls. You can select which table to be bounded by utilizing the data member property of .NET data bound controls. If you use Typed DataSet as table definition, by default the .NET data bound controls will generate the column based on data member that you select.

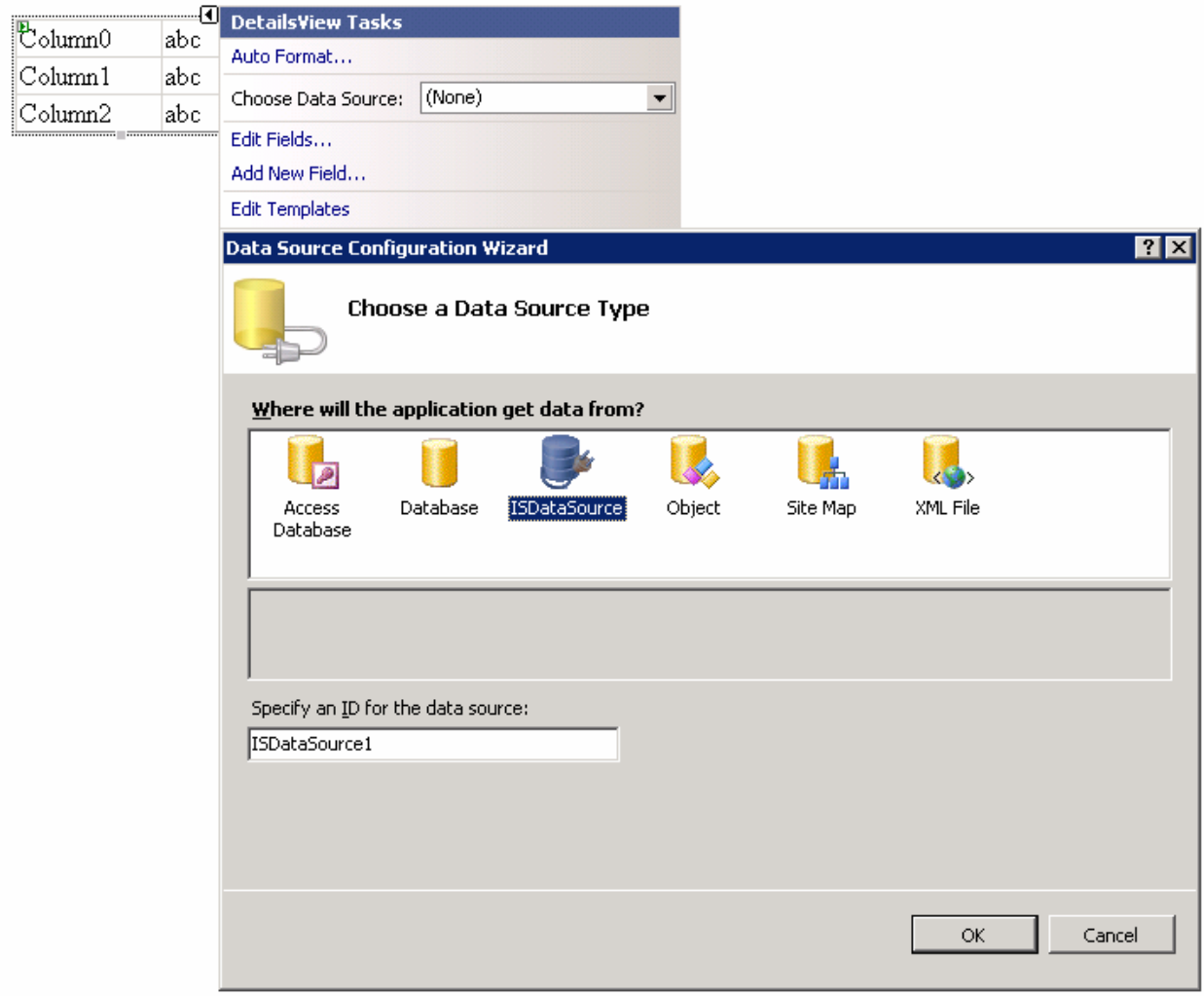

Figure 2.4a: Integration to .NET data-bound controls (Smart Tag)

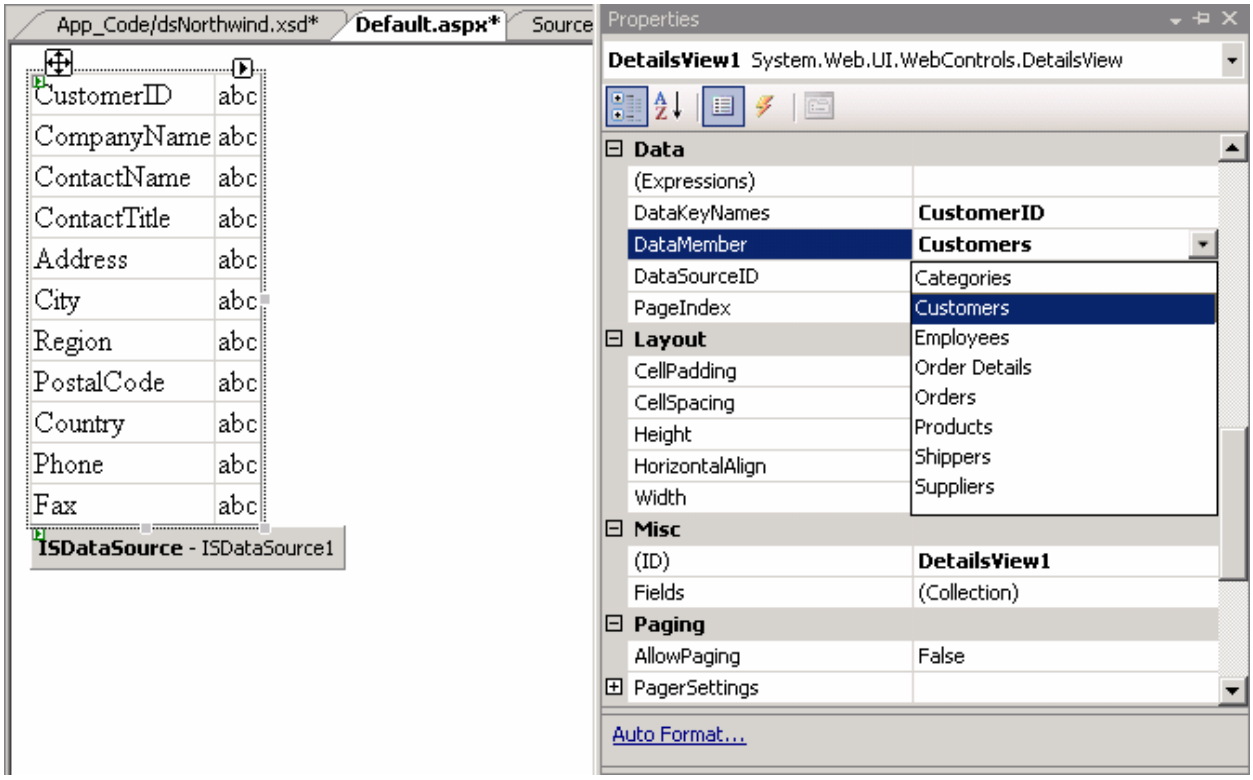

Figure 2.4b: Integration to .NET data-bound controls (Data Member)

Processing data using ISDataSource.NET

ISDataSource.NET performs data operations through its ISDataSourceTable collection. When working with a data-bound control you need to determine which ISDataSourceTable from ISDataSourceTable collection to be bounded to the data-bound control. You can do this through DataMember property.

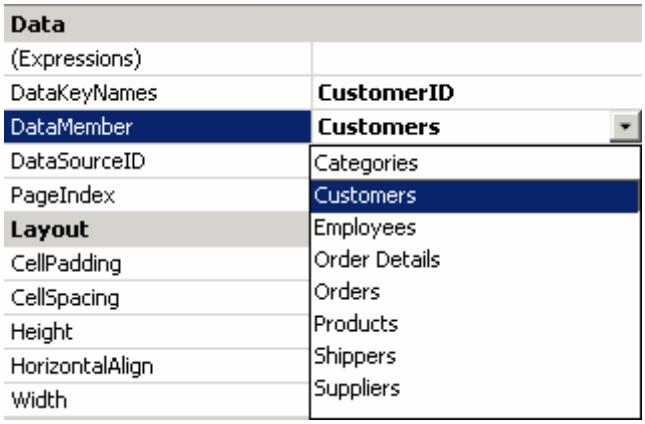

Figure 2.5: Assigning DataMember for GridView control

To retrieve data from a business object, set the corresponding ISDataSourceTable's SelectMethod property with the name of the method that retrieve the data. If the method does not return an IEnumerable or DataSet object, the object is wrapped by runtime in an IEnumerable collection.

If the method signature has parameters, you can add Parameter objects to the SelectParameters collection and bind them to the values that you want to pass to the method that is specified in ISDataSourceTable's SelectMethod method. The parameter must match the names and the types of the parameters in the method signature in order for ISDataSourceTable to use the parameter.

Walkthrough: Binding to Data using ISDataSource.NET

Prerequisites
DataSet Component in App_Code.
(e.g, dsNorthwind.xsd)

Steps

- Drag ISDataSource control from toolbox to the designer surface.
- Configure Data Source for the newly created ISDataSource.
- Select DataSet for the SchemaType.
- Click Schema Name dropdown and select dsNorthWind from the list.
- In the next dialog, click AutoGenerate button which is located at the most bottom.
- Click Finish.
- Drag GridView / DetailsView / FormView or others data-bound controls
- Set the DataSourceID to ISDataSource1.
- Set the DataMember
- You are all set. Run the page in browser.

ISDataSource Settings:

```
<ISDataSource:ISDataSource ID="ISDataSource1" runat="server"
   SchemaName="dsNorthwind">
   <Tables>
      <ISDataSource:ISDataSourceTable TableName="Customers"
         OldValuesParameterFormatString="Original_{0}"
         SelectMethod="GetData"
         TypeName="dsNorthwind.CustomersTableAdapter">
   </Tables>
</ISDataSource:ISDataSource>
```

Walkthrough: Binding Generic Collection using ISDataSource.NET

Prerequisites
DataSet Component in App_Code.
(e.g, Generic.Customers and Generic.CustomerDataAccess)

Steps

- Drag ISDataSource control from toolbox to the designer surface.
- Configure Data Source for the newly created ISDataSource.
- Select CustomObject for the SchemaType.
- In the next dialog, add new table with name "Customers"
- Select BusinessObject = Generic.CustomerDataAccess.
- Select SelectMethod = GetCustomers
- Click Finish.
- Drag GridView / DetailsView / FormView or others data-bound controls
- Set the DataSourceID to ISDataSource1.
- Set the DataMember
- You are all set. Run the page in browser.

ISDataSource Settings:

```
<ISDataSource:ISDataSource ID="ISDataSource1" runat="server"
   SchemaType="CustomObject">
   <Tables>
      <ISDataSource:ISDataSourceTable
         DataObjectTypeName="GenericObject.Customer" DeleteMethod="Delete"
         InsertMethod="Insert"
         OldValuesParameterFormatString="Original_{0}"
         SelectMethod="GetCustomers" TableName="Customers"
         TypeName="GenericObject.CustomerDataAccess" UpdateMethod="Update">
         <UpdateParameters>
            <asp:Parameter Name="customer" Type="Object" />
         </UpdateParameters>
         <InsertParameters>
            <asp:Parameter Name="customer" Type="Object" />
         </InsertParameters>
         <DeleteParameters>
            <asp:Parameter Name="customer" Type="Object" />
         </DeleteParameters>
      </ISDataSource:ISDataSourceTable>
   </Tables>
</ISDataSource:ISDataSource>
```

Based on the capabilities of the business object that the ISDataSource control works with, you can perform data operations such as updates, inserts, and deletes. In order to perform these data operations, you need to set the appropriate method name and any associated parameters for the operation that you want to perform.

For example, for an update operation, set the ISDataSourceTable's UpdateMethod property to the name of business object method that perform updates and add any required parameters to the UpdateParameters collection.

If the ISDataSource control is associated with a data-bound control, the parameters are added by the data-bound control. In this case, you need to ensure that the parameters name of the method match the field names in the data-bound control. The update is performed when the Update method is called, either explicitly by your code or automatically by a data-bound control. The same general pattern is followed for Delete and Insert operations.

Important Note: Business objects are assumed to perform these types of data operations one record at a time, rather than batched.

Walkthrough: Modifying Data using ISDataSource.NET

Prerequisites
DataSet Component in App_Code.
(e.g, dsNorthwind.xsd)

Steps

- Drag ISDataSource control from toolbox to the designer surface.
- Configure Data Source for the newly created ISDataSource.
- Select DataSet for the SchemaType.
- Click Schema Name dropdown and select dsNorthWind from the list.
- In the next dialog, click AutoGenerate button which is located at the most bottom.
- Click Finish.
- Drag GridView / DetailsView / FormView or others data-bound controls
- Set the DataSourceID to ISDataSource1.
- Set the DataMember
- Enable the Inserting / Deleting / Updating from the data-bound controls
- You are all set. Run the page in browser.

ISDataSource Settings:

```
<ISDataSource:ISDataSource ID="ISDataSource1" runat="server"
   SchemaName="dsNorthwind">
   <Tables>
      <ISDataSource:ISDataSourceTable ConflictDetection="CompareAllValues"
         DeleteMethod="Delete" InsertMethod="Insert"
         OldValuesParameterFormatString="Original_{0}"
         SelectMethod="GetData" TableName="Customers"
         TypeName="dsNorthwind.CustomersTableAdapter"
         UpdateMethod="Update">
         <UpdateParameters>
            <asp:Parameter Name="CustomerID" Type="String" />
            <asp:Parameter Name="CompanyName" Type="String" />
            <asp:Parameter Name="ContactName" Type="String" />
            <asp:Parameter Name="ContactTitle" Type="String" />
            <asp:Parameter Name="Address" Type="String" />
            <asp:Parameter Name="City" Type="String" />
```

```
            <asp:Parameter Name="Region" Type="String" />
            <asp:Parameter Name="PostalCode" Type="String" />
            <asp:Parameter Name="Country" Type="String" />
            <asp:Parameter Name="Phone" Type="String" />
            <asp:Parameter Name="Fax" Type="String" />
            <asp:Parameter Name="Original_CustomerID" Type="String" />
            <asp:Parameter Name="Original_CompanyName" Type="String" />
            <asp:Parameter Name="Original_ContactName" Type="String" />
            <asp:Parameter Name="Original_ContactTitle" Type="String" />
            <asp:Parameter Name="Original_Address" Type="String" />
            <asp:Parameter Name="Original_City" Type="String" />
            <asp:Parameter Name="Original_Region" Type="String" />
            <asp:Parameter Name="Original_PostalCode" Type="String" />
            <asp:Parameter Name="Original_Country" Type="String" />
            <asp:Parameter Name="Original_Phone" Type="String" />
            <asp:Parameter Name="Original_Fax" Type="String" />
         </UpdateParameters>
         <InsertParameters>
            <asp:Parameter Name="CustomerID" Type="String" />
            <asp:Parameter Name="CompanyName" Type="String" />
            <asp:Parameter Name="ContactName" Type="String" />
            <asp:Parameter Name="ContactTitle" Type="String" />
            <asp:Parameter Name="Address" Type="String" />
            <asp:Parameter Name="City" Type="String" />
            <asp:Parameter Name="Region" Type="String" />
            <asp:Parameter Name="PostalCode" Type="String" />
            <asp:Parameter Name="Country" Type="String" />
            <asp:Parameter Name="Phone" Type="String" />
            <asp:Parameter Name="Fax" Type="String" />
         </InsertParameters>
         <DeleteParameters>
            <asp:Parameter Name="Original_CustomerID" Type="String" />
            <asp:Parameter Name="Original_CompanyName" Type="String" />
            <asp:Parameter Name="Original_ContactName" Type="String" />
            <asp:Parameter Name="Original_ContactTitle" Type="String" />
            <asp:Parameter Name="Original_Address" Type="String" />
            <asp:Parameter Name="Original_City" Type="String" />
            <asp:Parameter Name="Original_Region" Type="String" />
            <asp:Parameter Name="Original_PostalCode" Type="String" />
            <asp:Parameter Name="Original_Country" Type="String" />
            <asp:Parameter Name="Original_Phone" Type="String" />
            <asp:Parameter Name="Original_Fax" Type="String" />
         </DeleteParameters>
      </ISDataSource:ISDataSourceTable>
   </Tables>
</ISDataSource:ISDataSource>
```

Process Diagram / Sequence

For better understanding about how data-operations is processed take a look at figure below.

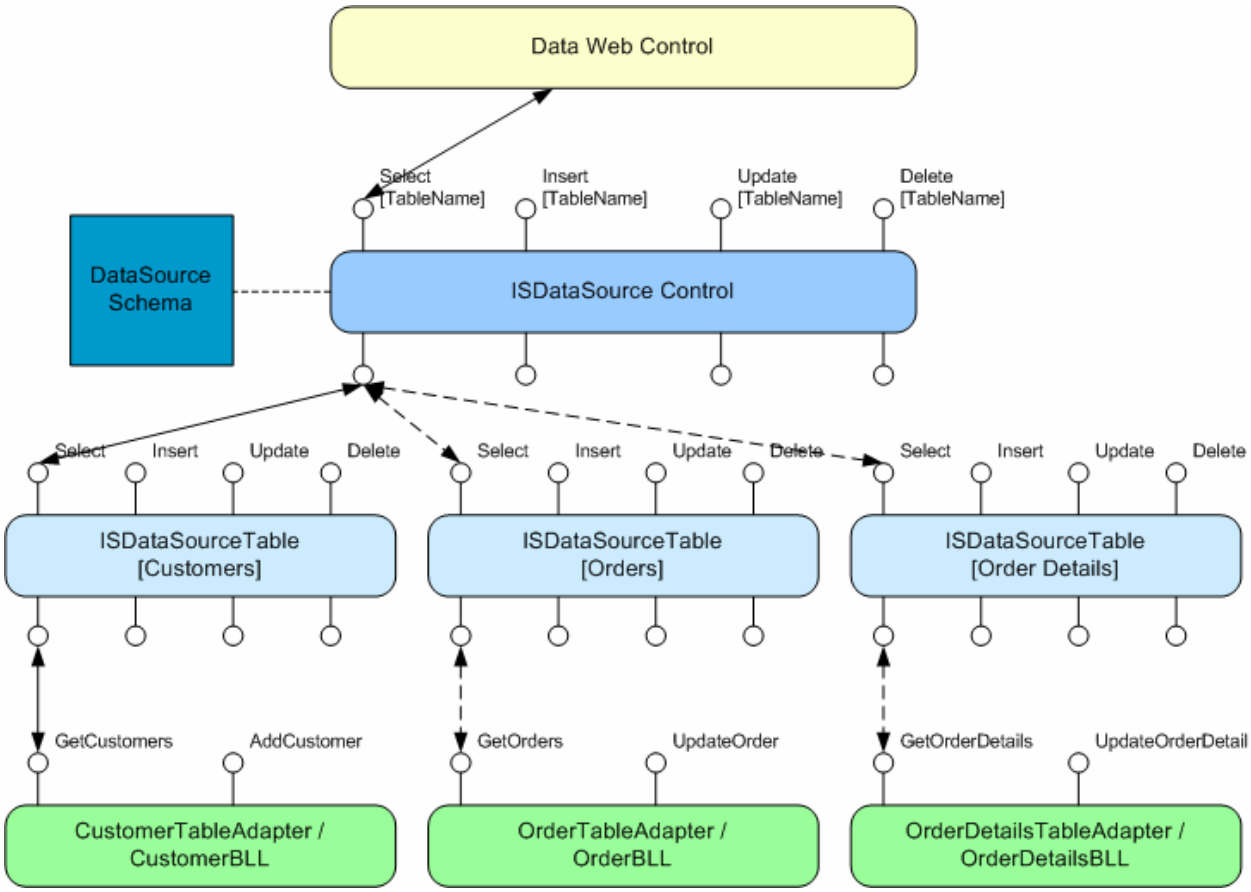

Figure 2.6: Processing data operations

When a request is occurred, Data Web Control will provide TableName information along with the request call. ISDataSource control will access respective ISDataSourceTable through TableName property and perform the request.

For example if user requests to retrieve data for Customers data the sequence of processes will be:

DataWebControl ➔ Select [Customers] ➔ ISDataSourceControl
ISDataSourceControl ➔ Select [Customers] ➔ ISDataSourceTable [Customers]
ISDataSourceTable [Customers] ➔ GetCustomers ➔ CustomerTableAdapter / CustomerBLL

And after that processes finished the data is sent back to DataWebControl following the same sequence

CustomerTableAdapter ➔ GetCustomers ➔ ISDataSourceTable [Customers]
ISDataSourceTable [Customers] ➔ Select [Customers] ➔ ISDataSourceControl
ISDataSourceControl ➔ Customers Data ➔ DataWebControl

The same sequence of processes happens when you perform another data operation processes (insert/update/delete).

Parameter Option

By default in .NET data-bound controls will pass all textbox fields to DataSource control as InputParameters. In some extend this can be quite troublesome since we need to create a DataOperation method signature (Insert/Update/Delete) that match the InputParameters given.

ISDataSource provide a flag for WebGrid.NET Enterprise 5.0 called ParameterOption. This flag is used to determine whether WebGrid.NET will pass all available fields as InputParameters or merge data from available fields (updated data) and original data retrieved previously as InputParameters.

Consider following scenario:

In Schema Definition we have:
Table: Customers
Fields: CustomerID, ContactName, CompanyName, ContactTitle, Address, City, Country, Region.

By default when we create a DataSet we will have methods with signature as follows: (OverwriteChanges applied)

Insert(CustomerID, ContactName, CompanyName, ContactTitle, Address, City, Country, Region)
Delete(OriginalCustomerID)
Update(CustomerID, ContactName, CompanyName, ContactTitle, Address, City, Country, Region, OriginalCustomerID)

Now let say we bound this ISDataSource to GridView but we do not want to bind all the columns, let say only CustomerID, ContactName, CompanyName. At this situation we can't use the auto-generated method from DataSet to update a row since the parameters supplied from GridView will not be the same as in method signature.

Update(CustomerID, ContactName, CompanyName, OriginalCustomerID)
Vs
Update(CustomerID, ContactName, CompanyName, ContactTitle, Address, City, Country, Region, OriginalCustomerID)

This kind of issue can be resolved using ParameterOption = "FromDataSource", this flag will told WebGrid to merge all the input parameters from given DataSource with the updated fields. Using this approach you can reuse the auto-generated method from DataSet nicely.

If you prefer the default way you can also set the ParameterOption = "FromFields" so WebGrid will only pass available fields as input parameters.

Sorting and Filtering Data

The ISDataSourceTable can sort and filter data that is retrieved by the SelectMethod property if the data is returned as a DataSet, DataView or DataTable object.

You can set the FilterExpression property to a filtering expression using a format string syntax and bind values in the expression to parameters that are specified in the FilterParameters collection.

Furthermore you can also set the SortExpression property which will act as secondary sorting expression after data-bound control SortExpression if any.

How-to: Filter data using FilterExpression and FilterParameters

You can use a FilterExpression to filter the data retrieved using ISDataSource control. You can either set the static filter expression e.g. FilterExpression = "ProductName LIKE 'A%' or use a filter parameters as in the sample below.

```
<ISDataSource:ISDataSourceTable
   FilterExpression="ProductName Like '{0}%'"
   SelectMethod="GetData"
   TableName="Products"
   TypeName="dsNorthwindTableAdapters.ProductsTableAdapter">
   <FilterParameters>
      <asp:ControlParameter PropertyName="Text" Name="ProductName"
         ControlID="TextBox1">
      </asp:ControlParameter>
   </FilterParameters>
</ISDataSource:ISDataSourceTable>
```

Notice the {0} symbol , this will be replaced by FilterParameters value at index 0. So in this case it will look at TextBox1's Text property.

How-to: Using SortExpression

SortingExpression is used to sort the data retrieved by ISDataSourceControl. The SortExpression in ISDataSource will always act as secondary sorting after the data-bound controls sorting mechanism.

```
<ISDataSource:ISDataSourceTable
   SortExpression="City DESC AND Country ASC"
   SelectMethod="Customers"
   TableName="Products"
   TypeName="dsNorthwindTableAdapters.CustomersTableAdapter">
</ISDataSource:ISDataSourceTable>
```

Hierarchical Data Retrieval

In Hierarchical data retrieval ISDataSource will traverse through all child tables in specified DataSourceSchema property. Starting from the given RootTable, ISDataSource will go to each child tables and perform the data retrieval on each child tables. In the end ISDataSource will generate a DataSet that holds all the data and the hierarchical relationship.

The DataSourceSchema only used to determine which table in the hierarchical schema to be populated. The process to populate each table is the same as Flat Data Retrieval. You need to provide SelectMethod for each ISDataSourceTable that represent the child tables.

You can also bind to a Hierarchical Custom Objects by providing a custom schema definition. You can create the custom schema definition by implementing IHierarchical and IObjectRelations interfaces.

Walkthrough: Binding to Hierarchical Data using ISDataSource.NET

Prerequisites
DataSet Component in App_Code.
(e.g, dsNorthwind.xsd)

Steps

- Drag ISDataSource control from toolbox to the designer surface.
- Configure Data Source for the newly created ISDataSource.
- Select DataSet for the SchemaType.
- Click Schema Name dropdown and select dsNorthWind from the list.
- In the next dialog, click AutoGenerate button which is located at the most bottom.
- Click Finish.
- Drag WebGrid control from toolbox to the designer surface.
- Set the DataSourceID to ISDataSource1.
- In the Connect to DataSource dialog, select Retrieve Hierarchical Structure, then click OK.
- You are all set. Run the page in browser.

Load on Demand Data Retrieval

One of important key features in WebGrid.NET is the ability to load data on demand. ISDataSource also designed to supports all load on demand scenarios in WebGrid.NET and also in WebCombo.NET.

There are three types of load on demand in general:

1. Virtual Page load on demand (Custom VirtualLoad)
2. Hierarchical load on demand
3. Self-Reference load on demand

Working with designer

ISDataSource.NET package also comes with a wizard designer. The designer is adapted from ObjectDataSource wizard designer with some additional features and implementation that supports the overall architecture.

Automatic table generation

One of the valuable features in ISDataSource.NET architecture is the ability to generate / predefined all tables' definition based on the given schema (DataSet schema). Thus with only three clicks you can have a structured data source completes with all the table definition on how to process the data.

Step by step wizard

ISDataSource.NET designer is divided into three sections:

- Schema definition
- Method / Table definition
- Parameter definition

Schema definition

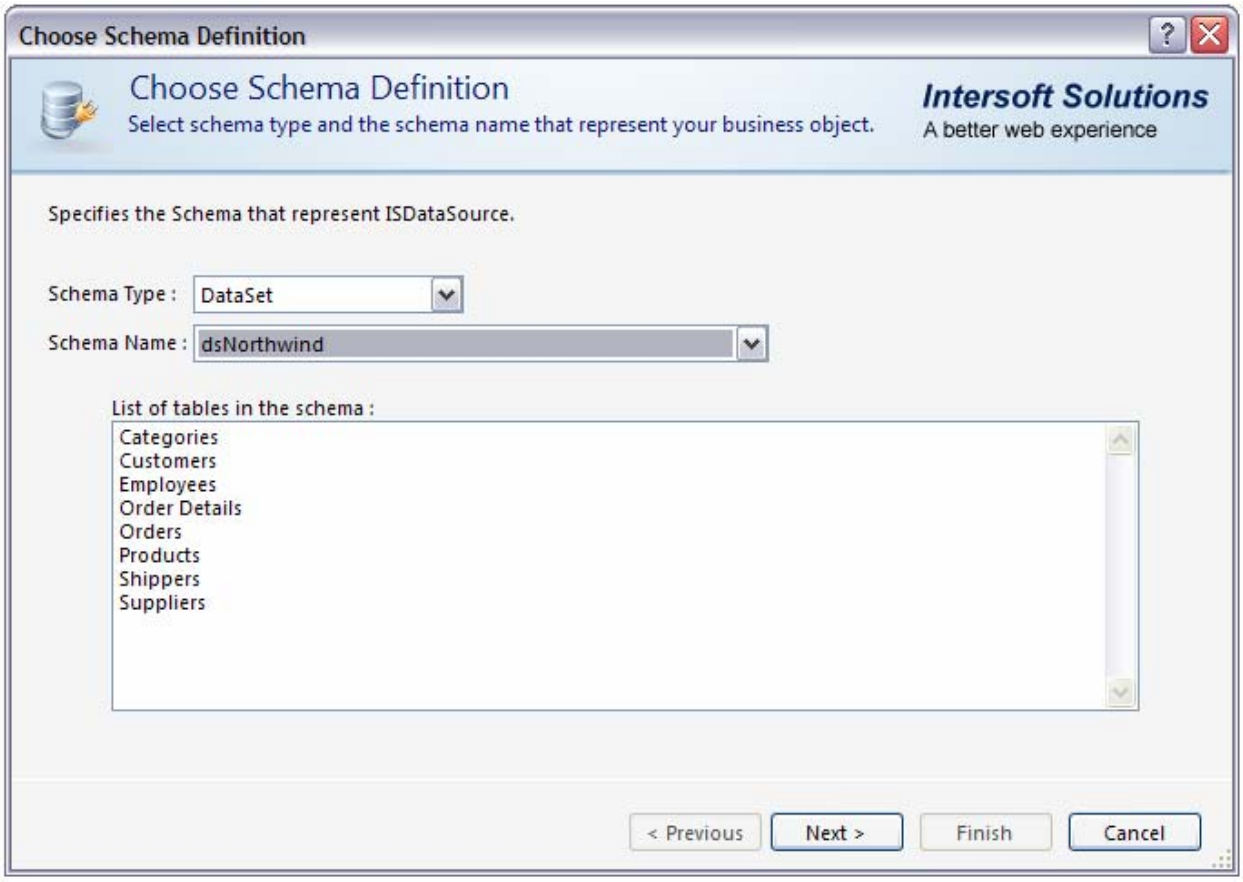

Figure 3.0: Schema definition

This section is designed to help you decide which schema will be used. When you choose the SchemaType to DataSet, all predefined Typed DataSet will be listed in the SchemaName. You can observe what tables are listed in the Schema from the Table List below it.

Method / Table definition

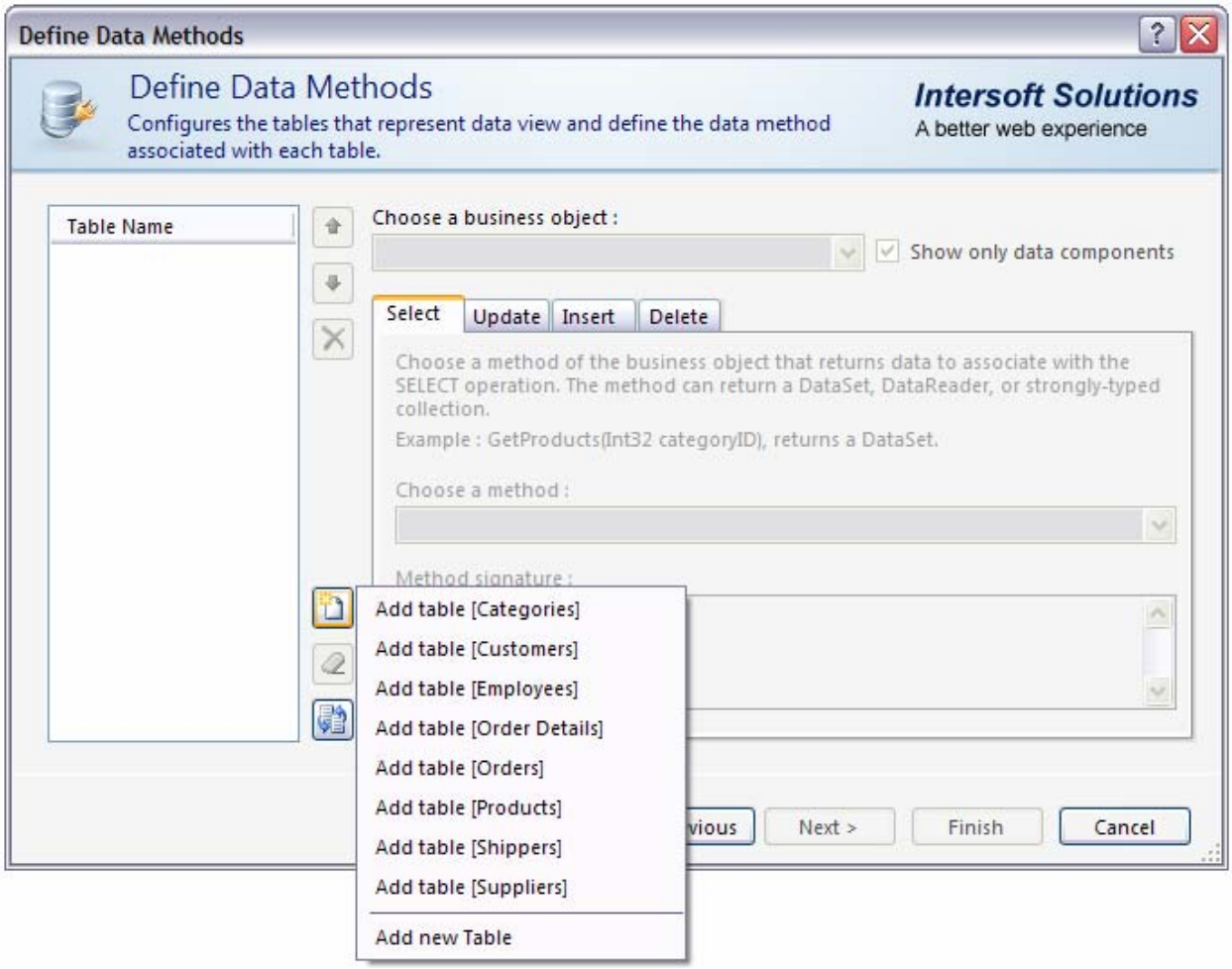

Figure 3.1a: Method / Table definition (Add new ISDataSourceTable)

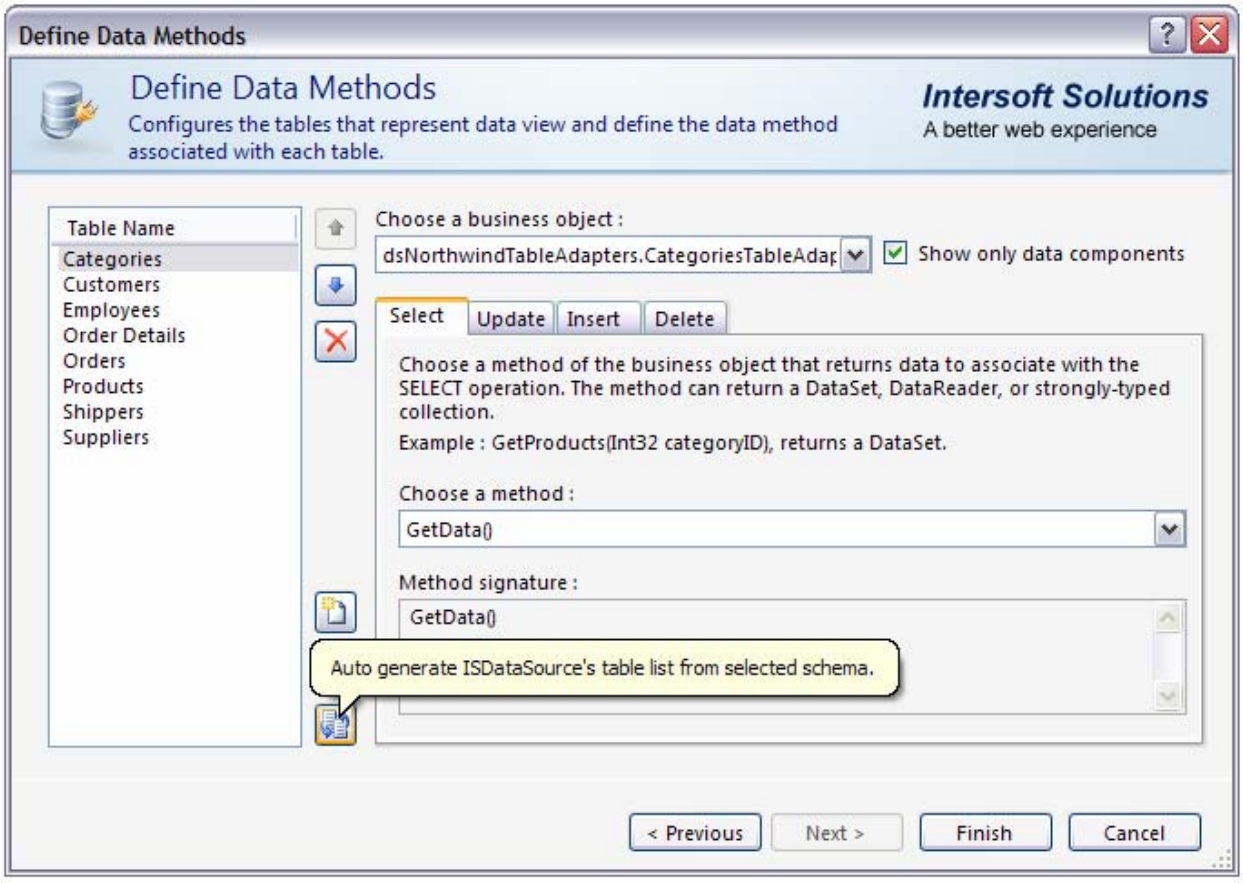

Figure 3.1b: Method / Table definition (Auto generate ISDataSourceTables)

This section is designed to define all your table definition. You can add / delete / modify any table using this designer. If you specify a schema in Schema definition section, you can use the Generate button to automatically generate all table definitions exist in the Schema.

Ultimately you need to specify the Select Method for each Table that belongs to the ISDataSource.

Parameter definition

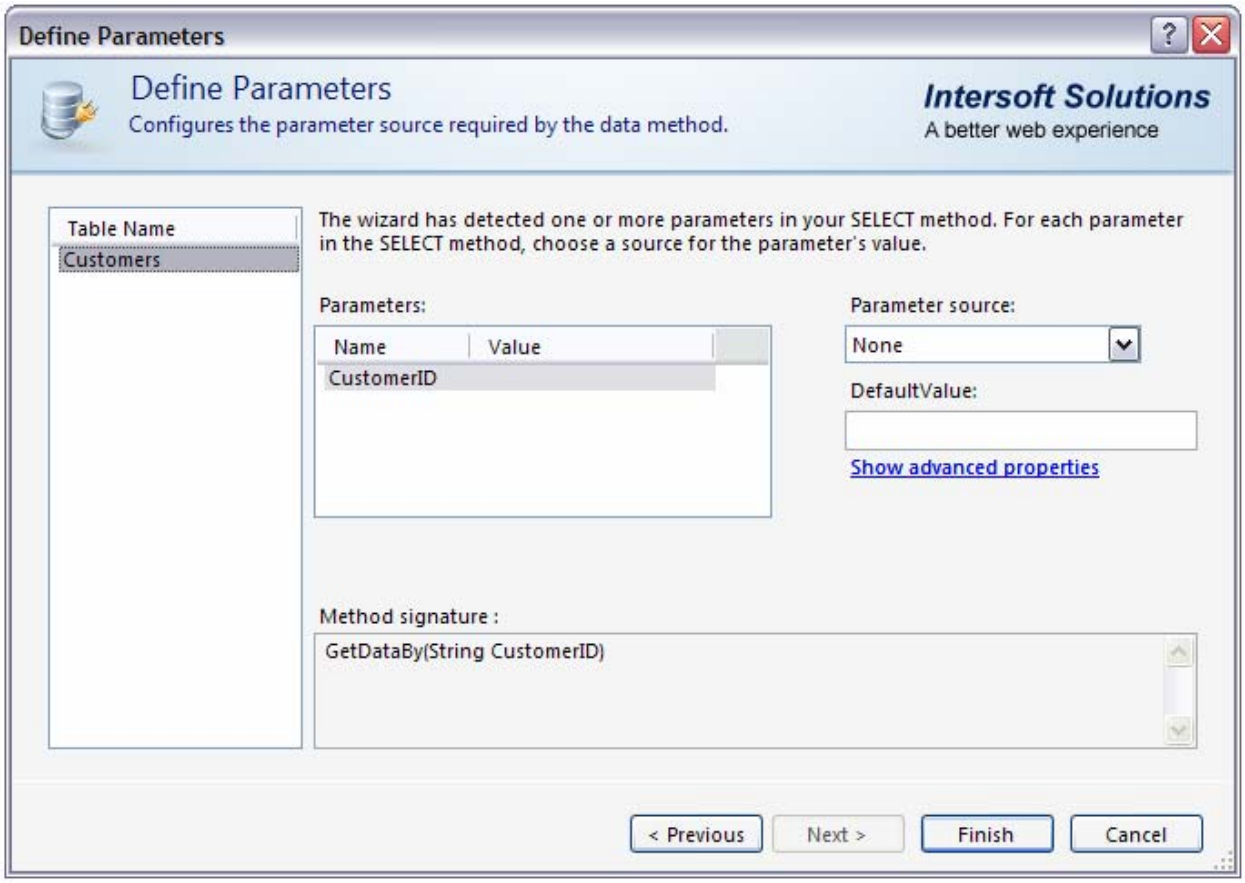

Figure 3.2: Parameter definition

This section is used to define how the SelectMethod's parameter retrieves the value.

Automatic conflict detection behavior

When you finished configuring ISDataSource through Step by Step Wizard above, the wizard will perform one last intelligent checking to determine the conflict detection behavior.

If your ISDataSource table configuration used a table-schema definition that perform optimistic concurrency, the wizard will directly change the ISDataSourceTable's conflict detection property to CompareAllValues, or else it will use the default OverwriteChanges.

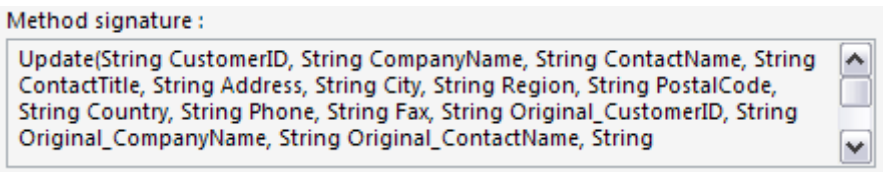

Figure 3.3: Method signature

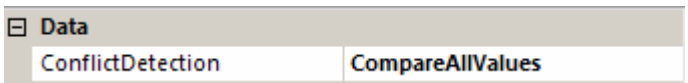

Figure 3.4: Automatically change conflict detection setting by reading method signature

Working with Typed DataSet

This section will demonstrate how to bind ISDataSource to data-bound controls using Typed-DataSet.

First of all you need to create the Typed-DataSet first using Visual Studio 2005 DataSet designer. These are the steps:

1. Add New Item and select DataSet templates
2. Specify your connection string, and create your DataSet structures in the given designer and configure how the data-operation is executed.

After you have a TypeDataSet ready, you may continue creating an ISDataSource and configure it to work in our data-bound controls.

1. Drag and drop ISDataSource component from Visual Studio 2005 toolbox.
2. Select "configure data source"
    - right click the component and select configure data source
3. A wizard-designer will be opened.
4. In "Choose Schema Definition" panel:

Let the SchemaType value as default (DataSet), and select the SchemaName from the dropdown. Upon SchemaName selection you will see the list of tables that belong to that DataSet / Schema.

5. Click Next
6. In "Define DataMethods" panel:

Click the generate table to automatically populates all the table definitions from the given schema you specified before.
(Note: you can also manually add/remove/manage the table definition using the same way as ObjectDataSource process)

7. Click Finish

Now you have an ISDataSource object, you can start binding it to data-bound controls.

1. Drag one of data-bound controls from ToolBox. (e.g. GridView).
2. Specify its "DataSourceID" property
3. By default it will take the first table in ISDataSource to auto-generate the GridView structure. You can change this by specifying its "DataMember" property

Up until this point you have managed to bind a data-bound control with minimal code especially in UI layer.

Caching Architecture

ISDataSource.NET also support data caching. While data is cached, call to any ISDataSourceTables' Select method will retrieve the data from the cache rather than from business object that associated with it.

ISDataSource.NET provides a more flexible caching configuration. The following are the list of settings that you can configure:

- CacheStorage
- CacheScope
- CachePriority
- StoreCachePerPage
- CacheKeyDependency
- CacheDuration & CacheExpirationPolicy

Cache storage

ISDataSource.NET provides three media (PageCache, Session, and FileServer) to store the Cache Data, each of them have their own unique characteristic that you might want to explore first before deciding which media you will use.

Page Cache

Page Cache stores the Cache Data in the server's memory. The cache is regulated by a combination of the CacheDuration, CacheExpirationPolicy and CachePriority setting.

The CacheDuration indicates the number of seconds that the cache storing data before the cache entry is discarded. On the other hand, the CacheExpirationPolicy describes the behavior of the cache in order to release the memory. If the CacheExpirationPolicy is set to Absolute, the cache data will be hold in memory for - at most - the amount of time that is specified by the CacheDuration. The data might be released before the duration time if the memory is needed. If the CacheExpirationPolicy is set to Sliding, the cache expires when there is no activity for a time that is equal to the CacheDuration.

One more important setting for PageCache setting is CachePriority. When the Web server hosting an ASP.NET application runs low on memory, the PageCache object selectively purges items to free system memory. When an item is added to PageCache, you can assign relative priority / CachePriority compared to other items stored in the cache. Items to which you assign higher priority values are less likely to be deleted from the cache when the server is processing a large number of requests, whereas items to which you assign lower priority values are more likely to be deleted.

Session

Session stores the Cache Data in Session Object. The cache is expired when session is timed out or Session.Abandon() method is called. In order to extend the cache time, simply increase the Session's timeout value. Session is best used for scenarios when data is scoped based per user/session and when on-demand data retrieval implementation is required.

You can also implement SqlServer mode to store the Session's data for larger applications that work in WebFarm or clustering.

File Server

File Server stores the Cache Data in a server (physical hard-disk). Using the unique-patterned key generated by ISDataSource.NET the data will be paired and stored in an actual server/machine. You can configure how the cached data is removed from Fileserver through ISDataSource or you can perform it manually from the Fileserver itself.

Some benefits of using Fileserver cache are:

- Highly scalable. You can choose a network drive to store the caches which enable multiple web servers to access the state.
- Decent performance. With more than dozen of high performances hard drives, you can choose a near-memory speed with high random access and sequential read-write hard drive to store the caches.
- Easily extensible. Unlike memory (RAM), you can easily extend a hard drive's capacity with larger one when it becomes insufficient.

Features of Fileserver:

- Automatic cleaning. When the cache for the viewstate has expired, the cache will be automatically deleted.
- Easily configurable for application-wide through web.config.
- Allow customization on cache policy and expiration duration.

To save the cached data to FileServer you need to specify the CacheServerConnection. The following list the parameters that can be used in CacheServerConnection:

- path. Specify the path of file server storage
- expireduration. Specify the time in minutes how long should a cache stay in the storage.
- expirationpolicy. Specify the expiration policy of the cache. The valid values are Sliding, Absolute.

Each parameter should be separated using ";" (semicolon) character.

How-to: Using file server for an instance of control.

The following steps demonstrate how to use file server to store the cache of ISDataSource.NET control:

1. Set CacheStorage to FileServer
2. Set CacheServerConnection to "path=C:\CacheStorage"

Note: The specified path (e.g. C:\CacheStorage) requires read and write permission for aspnep_wp worker process (in IIS 5.x) or iis_wpg (in IIS6+)

How-to: Configure file server to use shared network drive

The following steps demonstrate how to use a shared network drive for file server :

1. Set CacheStorage to FileServer
2. Set CacheServerConnection to "path=\\CacheServer\CacheStorage"
3. Configure the identity of the account that has access to the server and network resources in web.config

Sample web.config setting for identity:

```
<system.web>
    <identity impersonate="true" username="CacheServer\CacheAccount"
            password="CachePassword" />
```

</system.web>

Cache Scope

Another cache configuration setting that available from ISDataSource.NET is CacheScope. You can define the scope of the cache, whether it is available to global user or specific user (session).

Global

Global cache data allows all users that access the application share the same data. It is usually used for common data that rarely updated and used by all users.

Per User

Per User cache data will create a new instance for each user that accessing the application. The data is exclusive to each user.

Store Cache per Page

StoreCachePerPage is another additional setting that available from ISDataSource.NET. This feature allows you to decide whether to distinguish the data between pages.

Cache Key Dependency

Last but not least is CacheKeyDependency, this settings is treated as a key wild card to identifies the cache if needed.

Cache Key Pattern

A unique cache entry is created using combination of all these configurations.

Follows are the Cache Key Pattern:
{SessionID} : {PageName} : KeyDependency : ViewName : Duration : ExpirationPolicy : TypeName : SelectMethod+ Params ++ : startRowIndex : maximumRows

The field under {} is optional.
If the CacheScope is set to Global, the key will not have {SessionID}
If the StoreCachePerPage is set to No, the key will not have {PageName}
The other information is essential to create the Cache Key to differentiate the Cache Data.

From the pattern above you can see that multiple ISDataSourceTable that share the same characteristic can share the same cached data. Therefore carefully planning the cache configuration brings a better approach to optimize the performance your application.

Use cached on first load

Besides the configuration that used to determine the unique cache key above, one more vital settings that ISDataSource.NET cache architecture provide is the ability to determine whether ISDataSource.NET will use the cache data on first load or always look for actual data from database.

Event Handlers

ISDataSource.NET provides the same Event Handlers as in .NET ObjectDataSource. Those event handlers are:

- Deleted
- Deleting
- Filtering
- Inserted
- Inserting
- Selected
- Selecting
- Updated
- Updating

Any data operation from any ISDataSourceTable will call the same associated event handlers. In order to distinguish which ISDataSourceTable actually performing the data operation, each of event arguments have is added with a property called ViewName.