Creating a Client Proxy Class

Client proxy classes provide access to business services running in Natural. You can create proxy classes in C# or Visual Basic .NET. The classes support any project types that use these languages. Typically, proxy classes have properties and methods that map to their Natural counterparts.

To create a client proxy class:

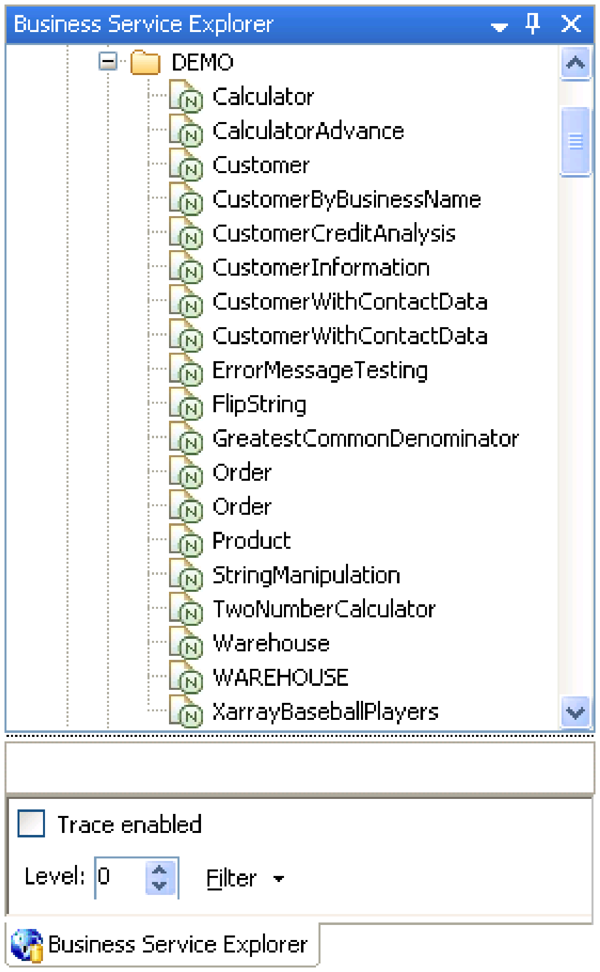
1. Create a C# or Visual Basic .NET project.

   For information, see the Visual Studio documentation.

   The new project is displayed in the Solution Explorer.

2. Select the connection and domain containing the business service in the Business Service Explorer.

   For example:

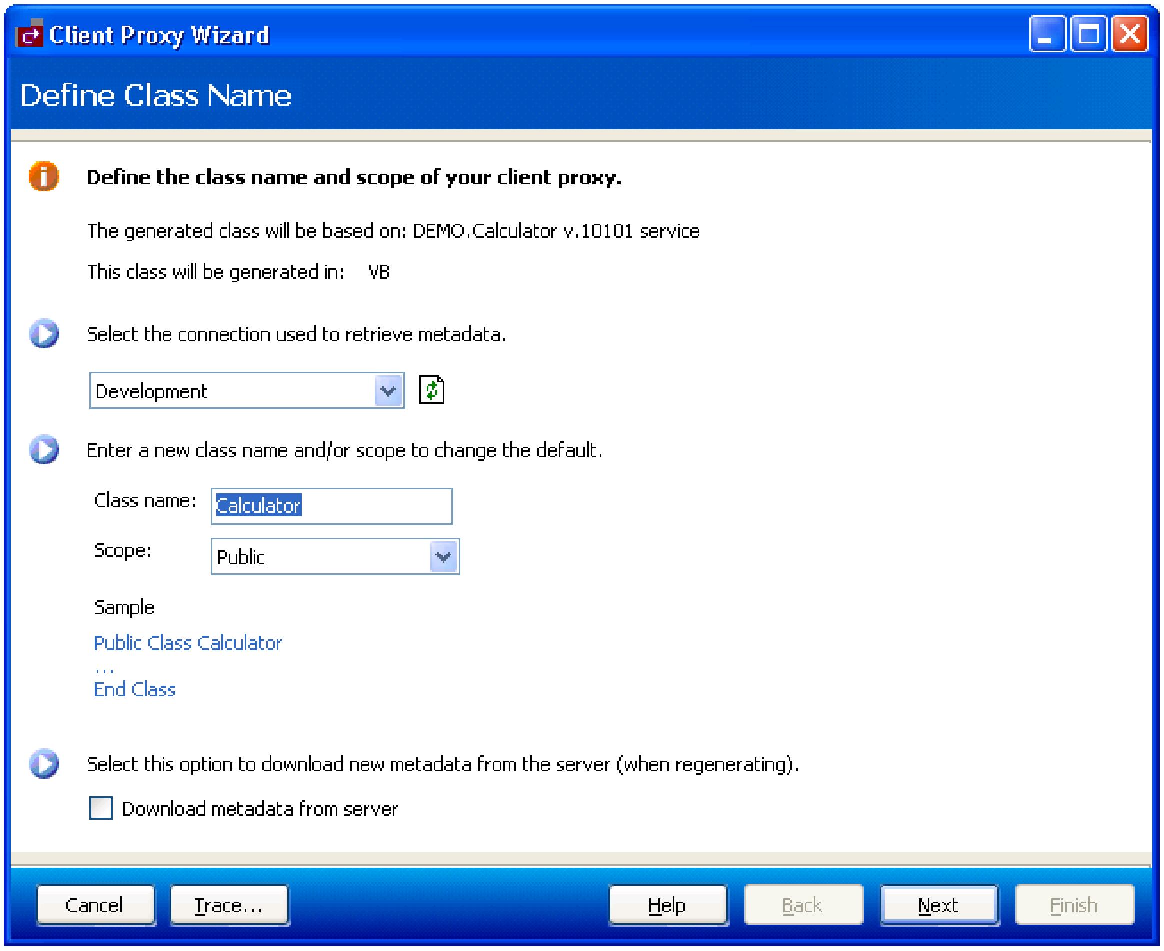


3. Open the context menu for the business service.

4. Select **Generate proxy class** on the submenu.

   Note:
   You can also use the **Find service/create class** option on the submenu to find the business service and then generate the client proxy class.

   The **Define Class Name** panel is displayed. For example:



Define the class name and scope of your client proxy.

The generated class will be based on: DEMO.Calculator v.10101 service

This class will be generated in: VB

Select the connection used to retrieve metadata.

Development

Enter a new class name and/or scope to change the default.

Class name: Calculator

Scope: Public

Sample

Public Class Calculator
...
End Class

Select this option to download new metadata from the server (when regenerating).

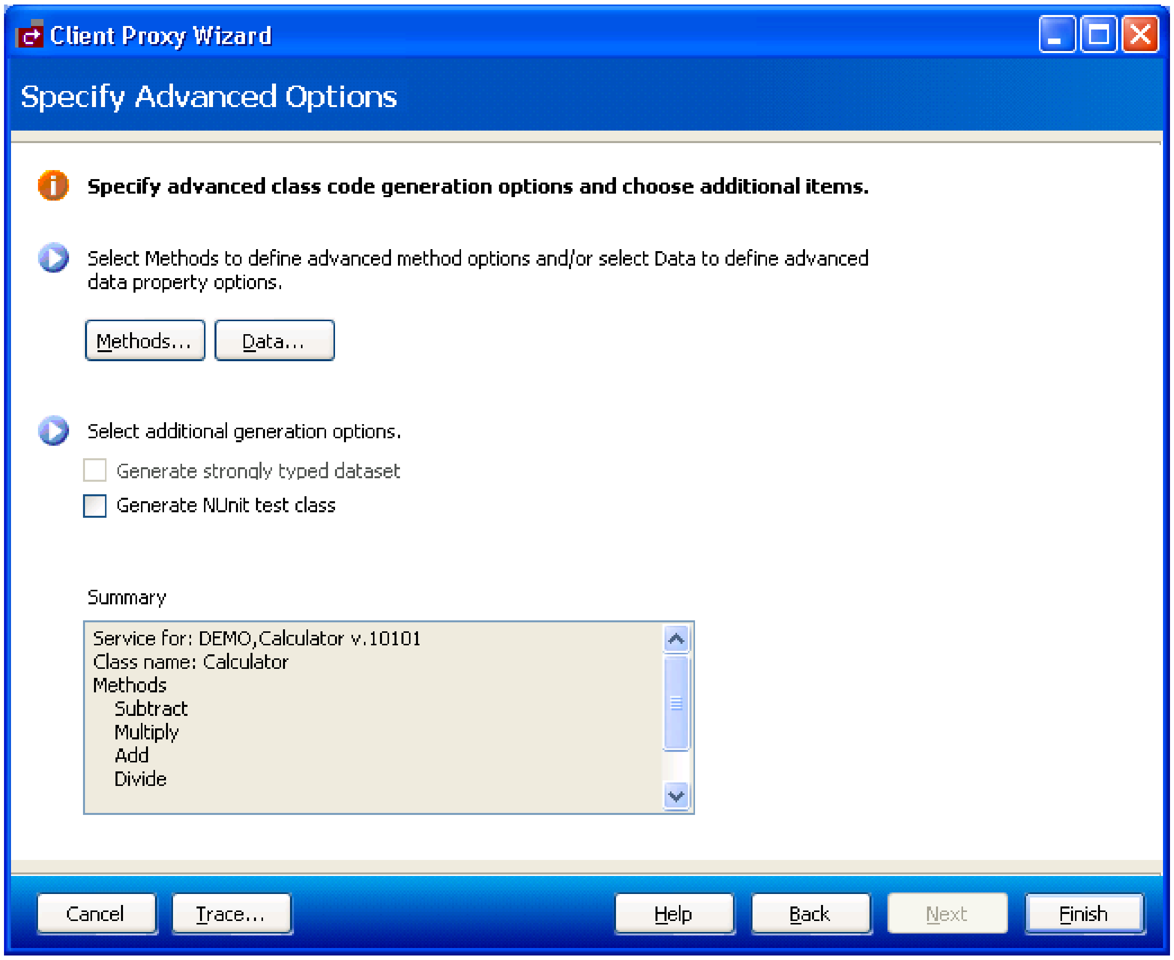
☐ Download metadata from server

   This panel displays the name and version of the business service on which the class is based, as well as the type of class that is generated.

5. Optionally, change the name of the class to be generated and the scope (for example, from public to private).

6. Select **Next** to download the server metadata from the Natural server.

   Natural Business Services retrieves the service metadata and displays the **Specify Advanced Options** panel. For example:



Specify advanced class code generation options and choose additional items.

Select Methods to define advanced method options and/or select Data to define advanced data property options.

Select additional generation options.

- [ ] Generate strongly typed dataset
- [ ] Generate NUnit test class

Summary

```
Service for: DEMO,Calculator v.10101
Class name: Calculator
Methods
   Subtract
   Multiply
   Add
   Divide
```

Use this panel to specify any advanced options for methods or data properties, as well as select additional generation options. Summary provides details about the metadata retrieved from the server.

7. Define advanced options for methods or data properties.

   - For information on defining advanced method options, see Configure the Methods Generated for a Client Proxy Class.

   - For information on defining advanced data property options, see Customize the Fields Generated for a Client Proxy Class.

8. Select additional generation options.

   For example:

   - Select **Generate strongly typed dataset** to increase the usability of your proxy class, as the dataset can be used directly with other .NET framework components (for example, databind to a grid) and .NET keeps track of groups of rows. You can generate a strongly typed dataset for services that were generated using the single-view code generation pattern generated by the Object-Browse-Select-Subp model. The wizard recognizes when this model was used to generate the target subprogram and automatically enables and selects a strongly typed dataset.

- Select **Generate NUnit test class** to generate code into an additional class where you can modify the condition of your test cases. One test case is generated for each service method. Another class, called TestSettings, is also generated into the project. This class contains the user ID, password, and connection settings for the generated tests.

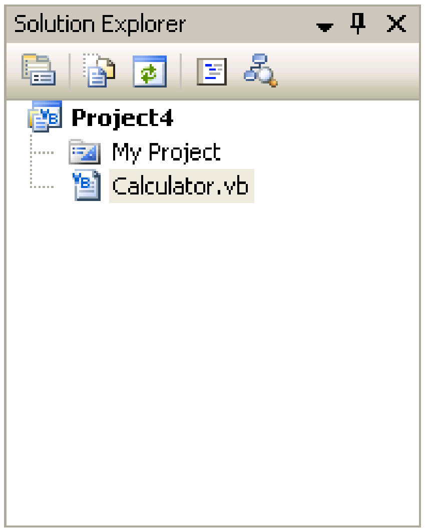
9. Select **Finish** to generate the client proxy class.

   The **Generate Status** window is displayed. All required references and dependencies are included in the project. If a generation error occurred, a message is displayed in the lower portion of the window.

   Note:
   To return to the wizard and make modifications, select **Cancel**.

10. Select **Save** to save the code to your project.

    The client proxy class is listed in the Solution Explorer. For example:



   Note:
   Remember to save your project in Visual Studio.

Configure the Methods Generated for a Client Proxy Class

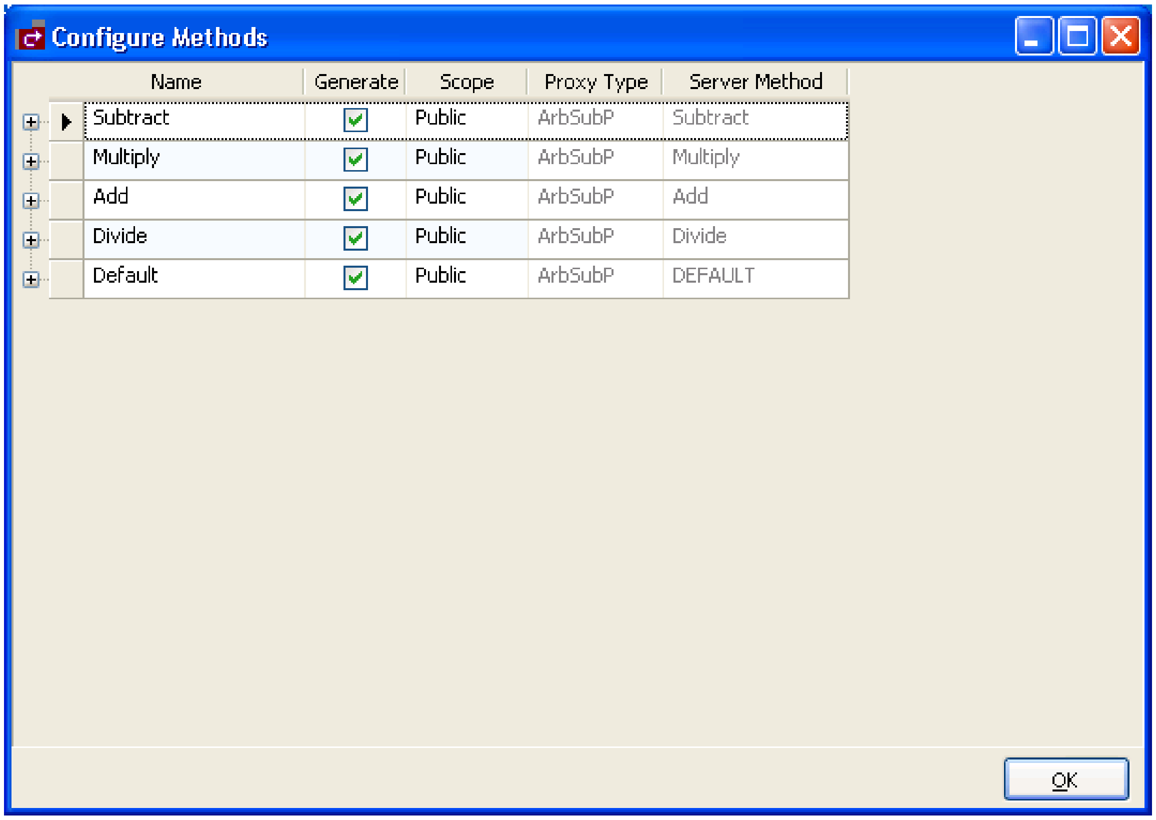
You can customize how methods are generated for the class. The Client Proxy wizard overloads the methods with no parameters. This allows the service to call the same function with different parameters.

Note:
When an empty method is called, instance variables are passed.

To configure the methods generated for a client proxy class:

1. Select **Methods** on the **Specify Advanced Options** panel.

   The **Configure Methods** window is displayed. For example:



Name	Generate	Scope	Proxy Type	Server Method
Subtract	✓	Public	ArbSubP	Subtract
Multiply	✓	Public	ArbSubP	Multiply
Add	✓	Public	ArbSubP	Add
Divide	✓	Public	ArbSubP	Divide
Default	✓	Public	ArbSubP	DEFAULT

   This window provides read-only information about the following:

   - Proxy Type

     Lists the types of Natural code generation patterns (if applicable). For example, a method can invoke the following types:

     - Data maintenance (maint)
     - Data query (browse)
     - Single view query/maintenance (browse-select)
     - Generic wrapper (generic)
     - Customer (arbsub)

   - Server Method

Name of the method on the server.

2. Modify the following default settings:

Setting	Description
Name	Method name used for the class.
Generate	If this option is selected, the corresponding method will be generated for the class.
Scope	Scope for the method. Valid values are: public, private, or friend.
Input	If this option is selected, the corresponding parameter group will be sent on input calls.
Output	The corresponding parameter group will be sent on output calls.
Overrides	Override indicators. Field overrides are based on the Natural field name and allow you to set conditions when invoking the method. The override indicators are: ● FieldName Name of a field in a parameter grouping with a field override. ● Value Value set for the field when the corresponding method is invoked.

3. Select **OK**.

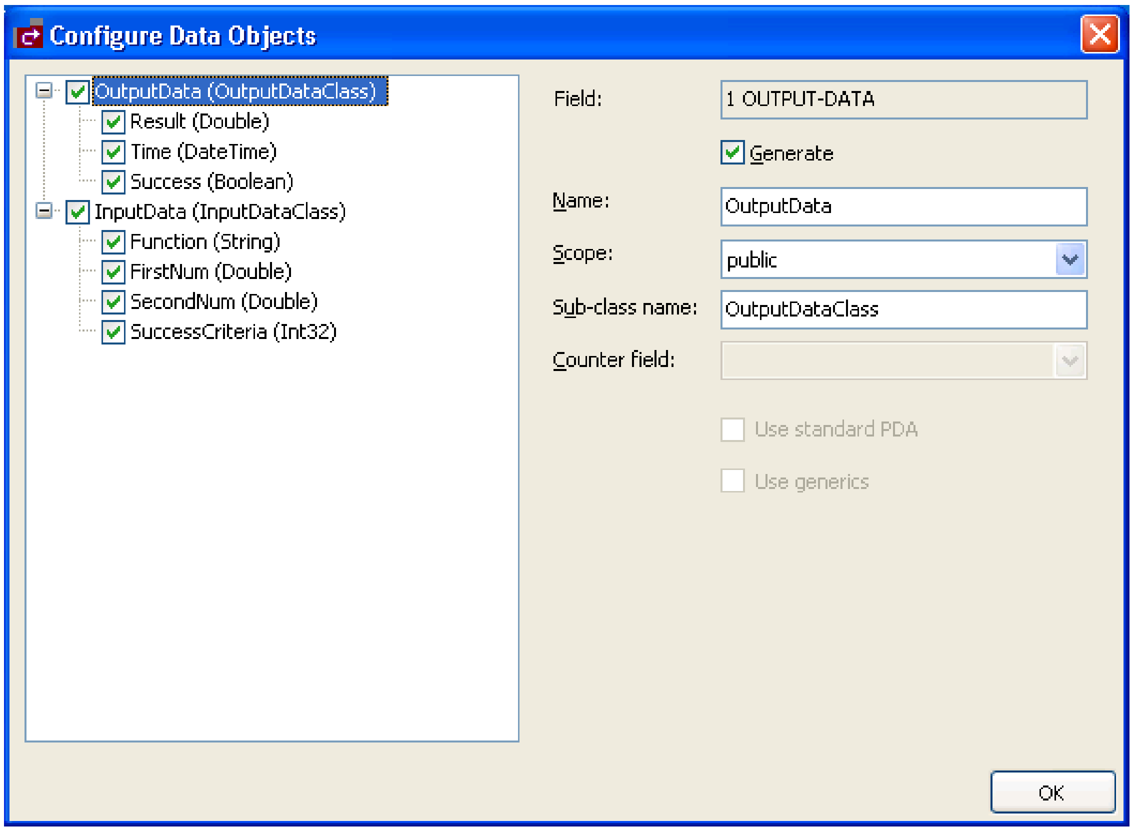
Customize the Fields Generated for a Client Proxy Class

You can customize how fields are generated for a client proxy class.

To customize the fields generated for a client proxy class:

1. Select **Data** on the **Specify Advanced Options** panel.

   The **Configure Data Objects** window is displayed. For example:



- [✓] OutputData (OutputDataClass)
  - [✓] Result (Double)
  - [✓] Time (DateTime)
  - [✓] Success (Boolean)
- [✓] InputData (InputDataClass)
  - [✓] Function (String)
  - [✓] FirstNum (Double)
  - [✓] SecondNum (Double)
  - [✓] SuccessCriteria (Int32)

Field: 1 OUTPUT-DATA

[✓] Generate

Name: OutputData

Scope: public

Sub-class name: OutputDataClass

Counter field:

[ ] Use standard PDA

[ ] Use generics

The tree on the left lists the parameter groups, array groups, and structures for each class, as well as the name of the class in brackets. The fields derived for each group or structure are displayed with the field type in brackets.

Note:
For redefined fields, you can either select the base field or one of the redefined fields.

2. Modify the following default settings:

Setting	Description
Field	Natural field name.
Generate	If this option is selected, the corresponding property will be generated for the class. **Note:** You can set overrides at the method level for fields that are not generated.
Name	Property name used for the field.
Scope	Scope for the property. Valid values are: public, private, or friend.
Sub-class name	Name of the class that will implement the child properties for parameter groups, array groups, and structures.
Counter field	Name of the field used to determine how many instances of an array are used for the business service. This field is used in conjunction with the Use generics field (see below).
Use standard PDA	If this option is selected, certain parameter groups will use the standard PDAs at runtime (for example, the CDPDA-M error message PDA). Properties will not be generated for these parameters groups and runtime versions of the standard PDAs will be used instead.
Use generics	If this option is selected, a Generic collection of up to 20 array items will be used for one-dimensional arrays (i.e., a collection of <Field Type> values). To determine how many instances of the array are used, you can specify a Counter field (see above).

3. Select **OK**.

Items Generated

Depending on the options specified for the class you are generating, the Client Proxy wizard generates the following items into your project:

Item	Description
ClassName	Generated proxy class.
ParameterGroupName	Generated metadata definition for a parameter group, which is saved in a child resx resource file.
*ClassName*Test	Class containing generated NUnit tests.
TestSettings	Class containing settings used to run NUnit tests in this project.
*ClassName*DatasetBase	Schema describing the dataset. It is only generated for services that have the Generate strongly typed dataset option selected on the **Specify Advanced Options** panel for the Client Proxy wizard.

Regenerate a Client Proxy Class

Proxy classes can be run through the code generator more than once (called regeneration). The wizard saves specification data at the bottom of the class (see the `Specs DO NOT DELETE` region).

To regenerate a client proxy class:

1. Open the context menu for the class in the Solution Explorer.
2. Select one of the following options on the submenu:

Option	Description
Show Wizard	Displays the **Client Proxy** wizard panel. Edit the panel as desired and select **Finish**.
Regenerate	Regenerates the client proxy without displaying the wizard panel. New metadata is downloaded from the server before regeneration. All of your previous settings are preserved (such as customizations to methods and properties). **Tip:** To regenerate multiple client proxy classes, select them in the Solution Explorer, open the context menu, and select **Regenerate**.

Example of a Client Proxy Class

This section contains an example of a class generated by the Client Proxy wizard. The example uses the Order service, version 020101, in the DEMO domain. The following topics are covered:

- Output Generated
- Example of Using the Order Client Proxy Class

Output Generated

The following items were generated for the client proxy class for the Order business service:

- Child resx File
- Client Proxy Class
- Client Proxy Test
- Client Proxy Dataset Base

Child resx File

This resource file stores PDA definitions for a class and creates the corresponding NaturalDataArea objects.

Client Proxy Class

Called Order.vb, this is the main class that invokes the remote Natural Business Services methods. This class includes:

Item	Description
Constructor	Contains a parameter called IRemoteCaller, which communicates with the server.
Methods	Each method for the class corresponds to a method in the business service. For each method that included the Generate Strongly Typed Dataset option (selected on the **Specify Advanced Options** panel for the Client Proxy wizard), a corresponding method is generated in the class to accept a dataset as a parameter.

Item	Description
Data	Each level 1 field from a PDA used in a business service becomes a property of the class. Each group within a PDA becomes a sub-class. If the Generate Strongly Typed Dataset option was selected on the **Specify Advanced Options** panel for the Client Proxy wizard, an additional property is generated (called RowDataDataset). For example, if the following definition is specified:

```
01 Group1
  02 Group2
    03 Field1 (A10)
```

The following is generated:

```
  Public Class SomeService

    Private m_Group1 As Group1Class

    Public Class Group1Class

      Private m_Group2 As Group2Class

      Public Property Group2() As Group2Class
        Get
          Return m_Group2
        End Get
        Set(ByVal Value As Group2Class)
          m_Group2 = Value
        End Set
      End Property
    End Class


    Public Class Group2Class
      Private m_Field1 As String

      Public Property Field1() As String
        Get
          Return m_Field1
        End Get
        Set(ByVal Value As String)
          m_Field1 = Value
        End Set
      End Property
    End Class

    Public Property Group1() As Group1Class
      Get
        Return m_Group1
      End Get
      Set(ByVal Value As Group1Class)
        m_Group1 = Value
      End Set
    End Property
  End Class
```

Client Proxy Test

This class contains tests to use with NUnit. A test method is generated for each method in the Client Proxy class. To run the tests, modify the TestSettings class.

Client Proxy Dataset Base

To take advantage of Visual Studio's ability to generate strongly typed datasets based on an XSD file, the Client Proxy wizard generates an XSD file. Visual Studio then converts the generated XSD file into a strongly typed dataset, which is called the BaseDataset (OrderDatasetBase in this example).

Note:
To create a strongly typed dataset, you must select the option on the **Specify Advanced Options** panel for the Client Proxy wizard. For information, see Creating a Client Proxy Class.

Example of Using the Order Client Proxy Class

To use the sample Order class:

1. Instantiate the IRemoteCaller object used to make the remote calls.
2. Logon to Remote Caller.
3. Instantiate a service by passing the Remote Caller (created in step 1) in the constructor for the business service.
4. Populate PDA properties with data to be sent.
5. Invoke the desired method.

   If you are using the dataset methods, use the overloaded method with the dataset parameter.
6. Check the method result and use the returned data as desired.

For example:

```
Imports SoftwareAG.NBS.DispatchClient
Imports SoftwareAG.NBS.BusinessServiceHelper

…
      Dim rc As IRemoteCaller
      Dim ord As Order
      Dim logResult As LogonResult
      Dim result As BusinessServiceResult
      Dim iRowsReturned As Integer

      ’ Create the remote caller.
      rc = ServiceFactory.CreateDispatcher("Some ConnectionID")

      ’ Logon
      logResult = rc.Logon("Guest", "", Nothing)
      If Not logResult.Pass Then
        ’ Handle logon error
      End If

      ’ Create the Order object
      ord = New Order(rc)
```

```
’ Populate fields to find all Orders for Customer# 2
ord.BrowseKey.OrderCustomerNumber = 2
ord.ServiceState.Inputs.RangeOption = RangeOptions.Equal

result = ord.FindByOrderCustomerNumber

If Not result.Success Then
  ’ Handle Error
End If

iRowsReturned = ord.ServiceState.InputOutputs.ActualRowsReturned

For Each row As Order.RowClass In ord.Rows.Row
  ’ Process each row here
Next
```

The following references are automatically added:

- SoftwareAG.NBS.BusinessDataTypes
- SoftwareAG.NBS.BusinessServiceHelper
- SoftwareAG.NBS.ClientConfig
- SoftwareAG.NBS.DispatchClient
- SoftwareAG.NBS.NaturalDataArea
- SoftwareAG.NBS.Shared
- SoftwareAG.NBS.XMLSerialization
- Nunit.framework (only added if the Generate Test Suite option was selected)