

Best Kept Secrets in .NET

DEBORAH KURATA

Apress®

Best Kept Secrets in .NET
Copyright © 2004 by Deborah Kurata

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher.

ISBN (pbk): 1-59059-426-6

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Trademarked names may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, we use the names only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

Lead Editor: Dominic Shakeshaft

Technical Reviewer: David McCarter

Editorial Board: Steve Anglin, Dan Appleman, Ewan Buckingham, Gary Cornell, Tony Davis, Jason Gilmore, Chris Mills, Steve Rycroft, Dominic Shakeshaft, Jim Sumser, Gavin Wray

Project Manager: Laura Cheu

Copy Edit Manager: Nicole LeClerc

Copy Editor: Marilyn Smith

Production Manager: Kari Brooks

Production Editor: Katie Stence

Compositor and Artist: Point 'n Click Publishing LLC

Proofreader: Greg Teague

Indexer: Ann Rogers

Cover Designer: Kurt Krames

Manufacturing Manager: Tom Debolski

Distributed to the book trade in the United States by Springer-Verlag New York, LLC, 233 Spring Street, 6th Floor, New York, NY 10013 and outside the United States by Springer-Verlag GmbH & Co. KG, Tiergartenstr. 17, 69112 Heidelberg, Germany.

In the United States: phone 1-800-SPRINGER, e-mail orders@springer-ny.com, or visit <http://www.springer-ny.com>. Outside the United States: fax +49 6221 345229, e-mail orders@springer.de, or visit <http://www.springer.de>.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, e-mail info@apress.com, or visit <http://www.apress.com>.

The information in this book is distributed on an "as is" basis, without warranty. Although every precaution has been taken in the preparation of this work, neither the author(s) nor Apress shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in this work.

The source code for this book is available to readers at <http://www.apress.com> in the Downloads section.

Chapter 1

Hidden Treasures in Visual Studio

When you have a great tool and learn the basics of using it, you get quite comfortable with it. Take your cell phone, for example. Once you learn the basics of how to make phone calls and store and retrieve phone numbers, your cell phone quickly becomes a part of your daily routine. But if you take a moment to read through the manual, you may find that your phone has many hidden treasures—many features that you didn't even know you had available.

The same is true with your Visual Studio interactive development environment (IDE). You know how to use Visual Studio to create projects, edit code, and build an executable. Since your focus is on getting the job done, you might not have time to explore the numerous features hiding in the many menus and dialog boxes. This chapter exposes these hidden treasures.

What Will This Chapter Cover?

This chapter uncovers the following Visual Studio secrets:

- * Laying out windows
- * Organizing code snippets
- * Managing your to-do list
- * Finding code
- * Using shortcut keys
- * Executing Visual Studio commands
- * Accessing external tools

By the end of this chapter, you will have discovered many of the hidden treasures in the Visual Studio IDE. You will be able to lay out your code windows for optimal access to the routines you are working on. You will see how to manage your code snippets so the code you need is always close at hand. You will learn how to use the Task List window to manage your development tasks. You will discover a dozen different ways to get to the code you need to find. You will see how easy it is to access the many features of Visual Studio using shortcut keys and Visual Studio commands, so there is no need for your hands to ever leave the keyboard. Finally, you will learn how to hook your favorite external tools directly into Visual Studio.

The following is a sample section from this chapter.

Organizing Code Snippets

You may often find that you write a little piece of code that you want to reuse again and again. These pieces of code are called *code snippets*. Wouldn't it be great to have a place

to organize these snippets and easily find and use them when you need them? The Visual Studio Toolbox provides such a place.

Storing Code Snippets

Before you can use code snippets, you need to store them. Any bit of code that you want to reuse can be stored as a code snippet.

Store your snippets on the General tab of the Visual Studio Toolbox. (If your Toolbox is not displayed in the IDE, choose the View ➤ Toolbox option to display it.) Or, if you prefer to categorize your snippets, create additional tabs on the Visual Studio Toolbox by right-clicking a tab and selecting Add Tab from the context menu. Then store your snippets on your new tabs.

Add a code snippet to a tab of the Toolbox as follows:

1. Select the code snippet (or any text) in the code file.
2. Copy the text to the Clipboard.
3. Click the desired tab of the Toolbox.
4. Right-click the area below the tab.
5. Select Paste from the context menu.

Alternatively, simply select the text from the code file and drag it to the desired tab of the Toolbox. An example of the Toolbox with code snippets is shown in Figure 1-8.

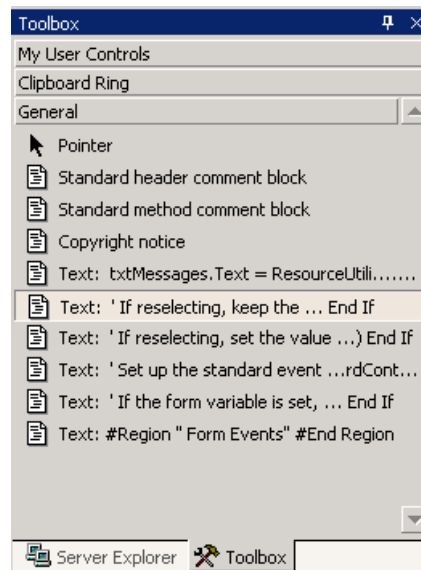


Figure 1-8. The Toolbox provides a great place for organizing your snippet library.

After you have snippets in the Toolbox, you can sort, rename, or delete them using the context menu. You can also reorganize them by dragging and dropping them onto a different tab.

The snippets in the Toolbox are retained as part of the IDE, so you can access the snippets from any solution. The contents of the Toolbox are stored in the toolbox.tbd file, located under Local Settings\Application Data\Microsoft\Visual Studio\7.1.

Store any piece of code that you may reuse as a code snippet in the Toolbox. This will greatly aid your productivity, because the next time you need the code, you can locate it quickly and insert it into your code file.

Using Code Snippets

Once you have stored a set of code snippets, you will want to use them in your code files. You can use them in any code file in any solution.

To use the code from your snippet library, follow these steps:

1. Right-click the desired snippet in the Toolbox.
2. Select Copy from the context menu.
3. Place the cursor at the desired insertion point in the code file.
4. Select Paste from the Edit menu (or the code window's context menu).

You don't need to use the Clipboard to access your snippets. Instead, place the cursor at the desired insertion point in the code file, and then double-click the snippet in the Toolbox. Alternatively, drag the snippet from the Toolbox to the desired location in the code file.

Organizing Comment Blocks

In addition to using the Toolbox to maintain a set of code snippets, you can keep comment blocks as text snippets in the Toolbox as well. For example, you can create a standard comment block for the header of every class and one for the header of every method, and store them as text snippets. This simplifies and standardizes your commenting tasks. To store and use a text snippet, follow the same techniques as defined for code snippets in the previous sections.

Organizing the Clipboard

If you have ever wished that you could have multiple items on the Clipboard, your wish has come true. The Clipboard Ring tab of the Toolbox, shown in Figure 1-9, displays the last set of items added to the Clipboard using the Cut or Copy commands within Visual Studio.

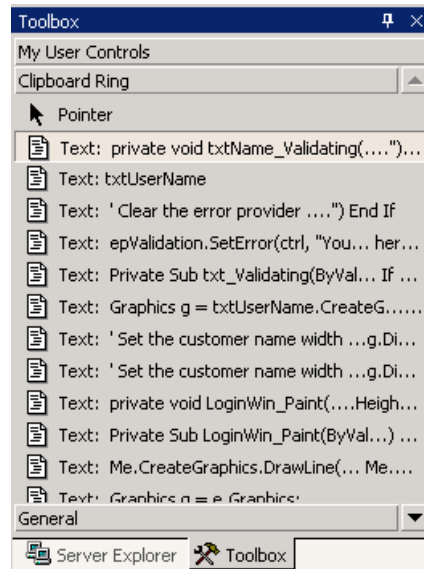


Figure 1-9. The Clipboard Ring tab gives you access to the last set of items added to the Clipboard from within Visual Studio.

Items on the Clipboard Ring tab can be dragged and dropped onto the active editing or design surface. If you prefer to use the keyboard, cycle through the contents of the Clipboard Ring tab while you work in an editor or designer by pressing Ctrl+Shift+V. This pastes an item from the Clipboard Ring tab at the current insertion point. Press Ctrl+Shift+V again to replace the pasted item with the next item from the Clipboard Ring tab. Repeat this process until you paste the desired item.

Chapter 2

Doing Windows Forms

If you are doing Windows Forms applications with .NET, you know the basics of building forms. You know how to use the basic controls, such as text boxes and labels. You know how to put the controls on the forms and write code to access the controls.

The purpose of this chapter is to present some of the lesser-known Windows Forms features. You may already be familiar with some of these treasures because they are reasonably discoverable. Others, such as how to draw a simple line when there is no line control, are buried treasures that are much harder to discover.

What Will This Chapter Cover?

This chapter uncovers the following Windows Forms secrets:

- * Docking for better layout
- * Anchoring for better resizing
- * Aligning for a clean look
- * Editing controls with the keyboard
- * Centering a form without using code
- * Iterating through the controls on a form
- * Implementing Enter and Leave events
- * Leveraging the DialogResult property
- * Drawing simple lines
- * Resizing controls for contents
- * Displaying validation errors with the ErrorProvider control

By the end of this chapter, you will know the secrets of how to do more with Windows Forms.

Using the Windows Forms Designer, you will discover how to apply docking and anchoring to enable end-user resizing of your forms without writing a single line of code. You will be able to improve the visual aesthetics of your forms using the alignment features. You will see how to edit controls on the Windows Forms Designer without moving your hands from the keyboard, and you will learn how to center a form without writing code.

But there are some Windows Forms features for which you do need to write code. You will see how to use recursion to iterate through all of the controls on your form. You will learn how to implement Enter and Leave events on controls to highlight the active control. You will discover how the DialogResult property can help you work with dialog boxes. You will observe the code you need to write to draw simple lines and resize

controls to the size of their contents. Finally, you will see how to display validation errors using the ErrorProvider control.

The following is a sample section from this chapter.

Iterating Through the Controls on a Form

There are many reasons why you may want to iterate through all of the controls on a form. You may want to move all of the controls due to a user action, resize each control to its contents at runtime, or set a particular property for the controls at runtime, for example.

The seemingly obvious (but potentially incorrect) solution for iterating through the controls on the form is as follows:

In VB:

```
For Each ctrl As Control In me.Controls
```

```
' Do whatever here
```

```
Next
```

In C#:

```
foreach (Control ctrl in this.Controls)
```

```
{
```

```
// Do whatever here
```

```
}
```

This code iterates through all of the controls *on* the form. But this won't always iterate through all of the controls that appear on the form, because all of those controls may not technically be *on* the form.

For example, Figure 2-4 displays a form that contains two Panel controls. The first Panel control contains a Toolbar control, two Label controls, and a ComboBox control. The second Panel control contains four Label controls and three TextBox controls. The form itself contains only the two Panel controls.

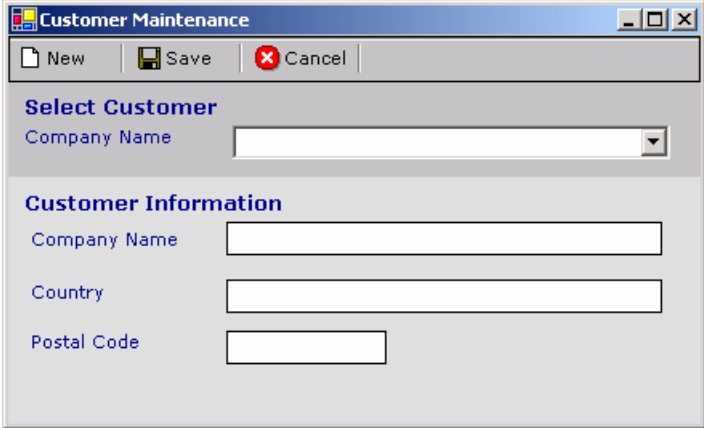


Figure 2-4. Even though all of the controls look like they are on the form, only the two panels actually reside on the form. All of the other controls reside on one of the two panels.

The prior sample code will process only the two Panel controls, because those are the only controls actually on the form, and therefore the only controls in the form's controls collection. To confirm this for yourself, add that sample code to a form with controls on panels (as in Figure 2-4), put a breakpoint in the loop, and display **me.Controls.Count** or **this.Controls.Count**.

To ensure that you process every control that appears on the form, you need to add the code that iterates through the controls to a method and call the method recursively for each control that contains other (child) controls.

In VB:

```
Private Sub MyMethod(ByVal pnl As Control)  
For Each ctrl As Control In pnl.Controls  
' Do whatever here  
  
If ctrl.HasChildren Then  
MyMethod(ctrl)  
End If  
Next  
End Sub
```

In C#:

```
private void MyMethod(Control pnl)  
{  
foreach (Control ctrl in pnl.Controls)  
{  
// Do whatever here  
if (ctrl.HasChildren)  
{  
MyMethod(ctrl);  
}  
}  
}
```

These routines use the HasChildren property of each control to determine if the control contains other controls. If so, it recursively calls the method to process the child controls. This code will then process every control that appears on the form. If more controls are added later, those controls will also be processed.

Use this technique whenever you want to iterate through all of the controls that appear on a form when you have controls that contain other controls, such as Panel, GroupBox, or Tab controls.

Chapter 3

Code Tricks

After you have been coding for a while, you'll find that you reuse coding patterns. For example, the code that processes a Toolbar control usually includes a case statement, event procedures contain the standard Try Catch block, and so on. It is easy to get into such a pattern and be so focused on getting the job done that you don't have time to try other coding techniques.

The purpose of this chapter is to present some code tricks to make your coding easier, more productive, and just better. And a few debugging techniques are thrown in, because what good is great code if it has bugs in it?

What Will This Chapter Cover?

This chapter uncovers the following coding secrets:

- * Short-circuiting Ands and Ors
- * Shortcutting the assignment operator
- * Improving string management with StringBuilder
- * Declaring on the For
- * Strictly converting your data types
- * Improving type casting
- * Aliasing data types
- * Managing regular expressions
- * Overloading procedures
- * Overloading operators
- * Exploring undiscovered regions
- * Using XML commenting
- * Obsolescing your code
- * Expanding your debugging techniques

By the end of this chapter, you'll be able to dazzle your colleagues with these code tricks. You'll find that these techniques will improve your productivity and make your code easier to develop and maintain.

The following is a sample section from this chapter.

Obsolescing Your Code

There is a growing interest in agile methodologies and agile development these days. One of the key tenets of agile development is coding for change; that is, developing code that is adaptive and easily modified as the business changes. This prevents the need to have

fixed requirements and allows your software to better meet the changing needs of the corporation and the end users. (If you are not familiar with agile methodologies, see the “Implementing a Methodology for the Design” section in Chapter 5 for more information.)

As you continue to change your code over time, you will find that you need to *refactor* a piece of logic or a routine. Refactoring is the process of rewriting routines or sets of routines to better structure your code. Refactoring is a part of agile development because adaptability requires that code be modified as needed.

Visual Studio 2005 will have many features to make the refactoring process easier. You will be able to convert any piece of code into its own method, for example, using a context menu. This feature will make it easy to break a large routine into smaller routines or separate out a piece of logic so it can more easily be reused.

One feature of refactoring that is available now is the ability to obsolete your properties or methods. Say you have a method (or other member) in your application that is no longer needed or whose signature needs to be changed. You could delete or change the method, and then find and modify all of the code that uses the method. But a better approach is to mark the method as obsolete, basically declaring it to be deprecated. By marking a method as obsolete, instead of deleting or changing it, you avoid needing to modify any of the code that uses the method.

Code that uses the obsolete method will get a warning or a syntax error (depending on how you obsolete the method) the next time that code is compiled. This allows you to define a logical obsolescence path for your code. For example, when building components for a team of developers, your standards may require that you obsolete a method for six months with a warning and another six months with a syntax error, before the method is actually removed from the code.

To mark a routine as obsolete, use the `ObsoleteAttribute`.

In VB:

```
<ObsoleteAttribute("This method is obsolete. Please use Retrieve(iID) instead", _  
    False)> _  
  
Public Function Retrieve(ByVal sProduct As String) As DataSet  
    ' Code here performs the retrieve  
End Function
```

In C#:

```
[ObsoleteAttribute("This method is obsolete. Please use Retrieve(iID) instead",  
    false)]  
  
public DataSet Retrieve(string sProduct)  
{  
    // Code here performs the retrieve  
}
```

The first parameter of the `ObsoleteAttribute` is the message text that the developer will see wherever the obsolete method is used. This message text will appear in the Task List window, as shown in Figure 3-6, and in the Quick Info, as shown in Figure 3-7.

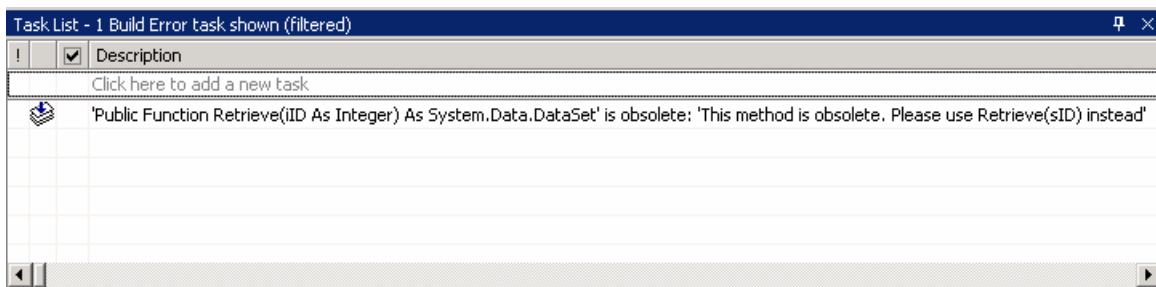


Figure 3-6. When a class member, such as a method, is marked as obsolete, any code that uses the member will appear in the Task List window.

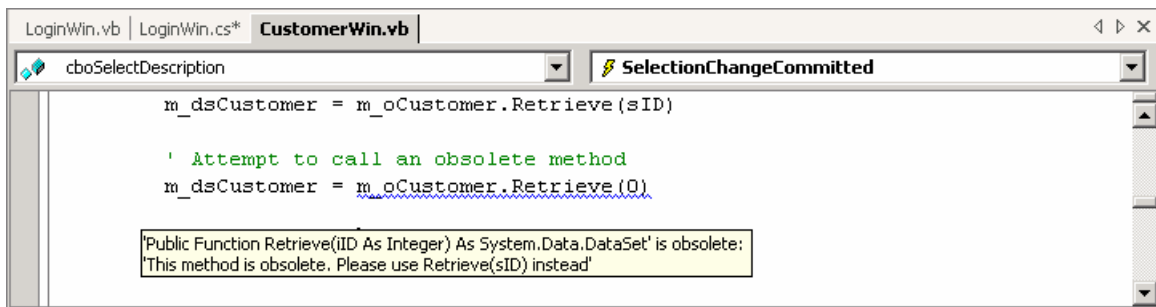


Figure 3-7. Any references to an obsolete member are marked in the code file, and the Quick Info displays the message defined in the `ObsoleteAttribute`.

The second parameter of the `ObsoleteAttribute` is whether the obsolete member usage is considered to be an error. Set the parameter to `False` to specify that the developer using the method will get a warning message in the Task List window, or to `True` to generate a syntax error if the method is used.

TIP Plan for change. Define an obsolescence plan in your coding standards, and use the `ObsoleteAttribute` to help execute the plan.

Chapter 4

Much ADO

Most applications work with some type of data: enterprise data, customer data, experimental data, configuration data, and so on. ADO.NET, which is an intrinsic part of the .NET Framework, supplies the tools for working with data in your application, and Visual Studio provides valuable tools for building your database.

The purpose of this chapter is to show you some of the lesser-known but very useful features of ADO.NET. It also presents a tutorial for working with databases in Visual Studio.

What Will This Chapter Cover?

This chapter uncovers the following ADO.NET and Visual Studio database secrets:

- * Working with your database using the Server Explorer
- * Managing stored procedures with a database project
- * Using the Microsoft Data Access Application Block
- * Configuring your connection
- * Viewing datasets as XML
- * Filtering datasets using data views
- * Building smarter datasets using extended properties

By the end of this chapter, you will understand why there is much ado about ADO.NET and the Visual Studio database features.

You will discover how to use Visual Studio's Server Explorer to access your database and view the database structure or contents. You will learn how to create your database, tables, and stored procedures, and how to manage those stored procedures using a database project within Visual Studio.

You will find out the secret to minimizing your data access code using the Microsoft Data Access Application Block. You will see how to use a configuration file for your connection strings. You will learn valuable techniques for working with your dataset, including how to view the dataset as XML to improve your debugging experience and how to filter datasets. Finally, you will discover how to use extended properties to expand the capabilities of your dataset, such as storing validation rules with a database column.

The following is a sample section from this chapter.

Viewing Datasets As XML

Have you ever tried to use the Watch or Locals window to view the contents of your dataset? You end up with something like Figure 4-22. It is very difficult to find your data in there!

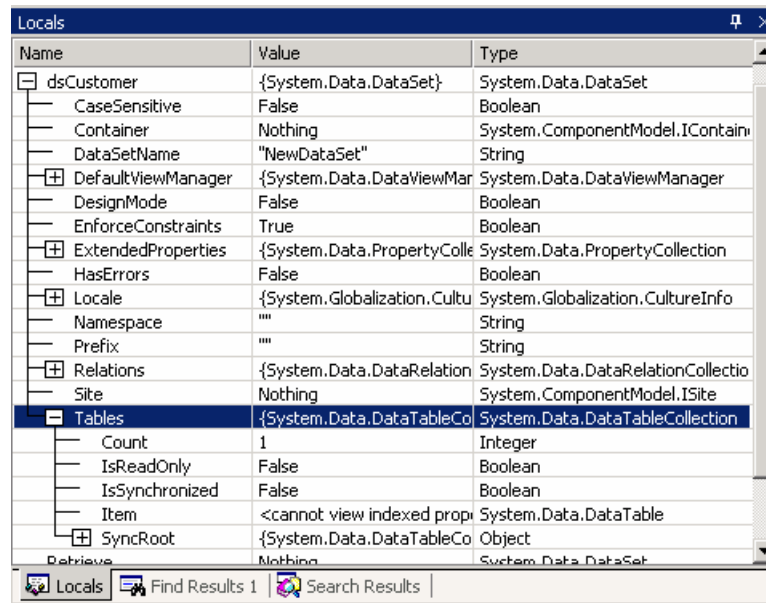


Figure 4-22. The Locals window is very useful for debugging, but not when it comes to viewing the contents of your dataset.

It is much easier to view your dataset using the XML features provided in ADO.NET. Use the GetXML method to display the XML associated with your dataset.

In VB:

```
Debug.WriteLine(dsCustomer.GetXml())
```

In C#:

```
Debug.WriteLine(dsCustomer.GetXml());
```

In this example, dsCustomer is the dataset containing fields for a particular customer in the Northwind database's Customers table. The GetXML method displays the contents of the dsCustomer dataset in XML format, as follows:

```
<NewDataSet>
  <Customer>
    <CustomerID>AROUT</CustomerID>
    <CompanyName>Around the Horn</CompanyName>
    <Country>UK</Country>
    <PostalCode>WAI IDP</PostalCode>
  </Customer>
```

</NewDataSet>

You can also use the GetXML method from the Command window to view the dataset while you are debugging, as shown in Figure 4-23.

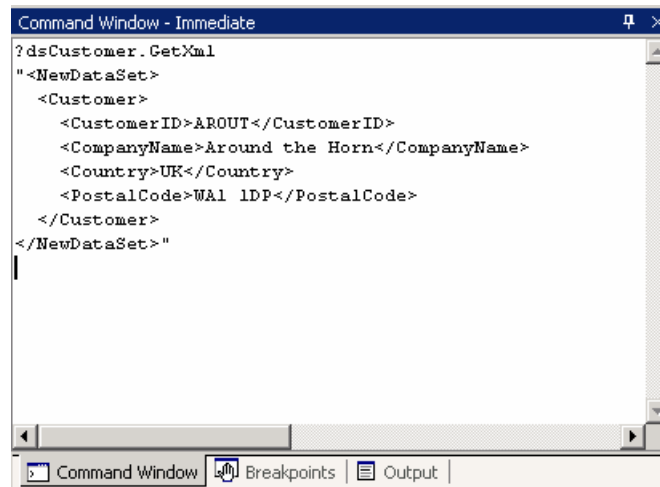


Figure 4-23. Using the GetXML method in the Command window provides a nicer view of the dataset than using the Locals or Watch window.

If the dataset is large or if you want to output the XML to a file that you can search, use the WriteXML method instead to write the dataset to a defined file.

In VB:

```
dsCustomer.WriteXml("c:\temp\customerDataset.xml")
```

In C#:

```
dsCustomer.WriteXml("c:\temp\customerDataset.xml");
```

And if you want to see the structure of the database, view the schema in XML format using the GetXMLSchema method.

In VB:

```
dsCustomer.GetXmlSchema()
```

In C#:

```
dsCustomer.GetXmlSchema();
```

As an example, the schema definition for the dsCustomer dataset is as follows:

```
<?xml version="1.0" encoding="utf-16"?>  
<xs:schema id="NewDataSet" xmlns=""  
  xmlns:xs="http://www.w3.org/2001/XMLSchema"  
  xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">  
  <xs:element name="NewDataSet" msdata:IsDataSet="true">  
    <xs:complexType>  
      <xs:choice maxOccurs="unbounded">  
        <xs:element name="Customer">
```



```
<xs:complexType>
  <xs:sequence>
    <xs:element name="CustomerID" type="xs:string" minOccurs="0" />
    <xs:element name="CompanyName" type="xs:string" minOccurs="0" />
    <xs:element name="Country" type="xs:string" minOccurs="0" />
    <xs:element name="PostalCode" type="xs:string" minOccurs="0" />
  </xs:sequence>
</xs:complexType>
</xs:element>
</xs:choice>
</xs:complexType>
</xs:element>
</xs:schema>
```

The schema lists each field of the dataset as an XML element. Within each element, it lists the name and properties of each field as an XML attribute. In this example, the dataset contains the CustomerID, CompanyName, Country, and PostalCode fields.

Use these XML features of the dataset to assist you in debugging your data access code.