Building Web Services with C# and DB2

Skill Level: Intermediate

Wayne Freeze (WFreeze@JustPC.com)
Studio B

21 Oct 2004

Developers using the Microsoft .NET Framework can easily leverage their existing knowledge to access information stored in a DB2 database using the new DB2 Managed Provider. This tutorial shows how to create a stored procedure in DB2 and then incorporate it into a Web service built using Microsoft's Visual Studio .NET and C# (pronounced C Sharp). Finally, a simple ASP.NET application demonstrates how to access the web service to display the data over the Internet.

Section 1. Tutorial introduction

What is this tutorial about?

This tutorial demonstrates the techniques needed to build a Web service in C# that accesses an IBM DB2 Universal Database database using the DB2 managed data provider. The DB2 managed data provider offers capabilities similar to the SQL Server managed data provider as well as providing a high performance, secure way to access a DB2 database from any .NET programming language. The DB2 managed data provider was written in C# by the DB2 Development organization to combine the best features and function of DB2, while exploiting the database neutral facilities included in the .NET Framework.

This sample application in this tutorial involves the JustPC.com Music Company, which maintains a database with information about various CDs it has for sale. Through a Web service it provides, users can search the database to retrieve various kinds of information, including a list of artists, a list of CDs recorded by a specific artist and the list of tracks on the CD.

In this tutorial, you'll learn the following:

- What the application accomplishes
• How the database is designed
• How to create the stored procedures that provide the data to the Web service
• How to define the Web service methods that make the data available to the outside world
• How a simple application can use the Web service to display data to the user

All of the application code was created using Visual Studio .NET and the C# programming language, while most of the database tasks were performed using the DB2 Development Add-in to Visual Studio .NET.

Should I take this tutorial?
You should take this tutorial if:

• You are familiar with DB2 and wish to learn about how to create a C# Web service using the new DB2 managed provider.
• You are familiar with C# and wish to see how to use the new IBM DB2 managed provider to access a DB2 database.
• You wish to learn more about the capabilities of the new IBM DB2 Development Add-in to Visual Studio .NET.
• You are curious to see how well a non-Microsoft database can be integrated into a .NET application.

Tools
This tutorial relies on software from both IBM and Microsoft.

• IBM DB2, Version 8.1 provides the database facilities used by this tutorial. You can download an evaluation copy of DB2 from: http://www6.software.ibm.com/dl/db2udbdl/db2udbdl-p.
• You can download a beta copy of the IBM DB2 .NET Enablement package at: http://www7b.software.ibm.com/dmdd/downloads/dotnetbeta/.
• Microsoft Visual Studio .NET (http://msdn.microsoft.com/vstudio) provides the development environment used to build the web service and including the C# compiler and the ASP.NET development libraries.
• Microsoft Windows 2000 Server (http://www.microsoft.com/windows2000/server/) provides the operating system to host DB2, plus the tools to run the web service and the C# ASP.NET application. Note that you can also use Windows 2000

- A separate C compiler to compile the stored procedures must also be available on the same machine as the database server. You should refer to the DB2 installation documentation for more details.

In order to follow along with this tutorial, you will need to create an empty database called MUSIC using the DB2 Control Center. Simply use the Create Database Wizard and specify MUSIC as the name of the database. No other information is required to create the database.

**Note:** While this tutorial was created and tested on a single Windows 2000 Server system, you may choose to run Visual Studio .NET on a Windows 2000 or XP Professional system and copy the appropriate files to the Windows Server system for execution.

You can also download all of the files in ZIP format (65 KB) for this example.

### Notices and Trademarks

Copyright, 2003 International Business Machines Corporation. All rights reserved.

IBM, DB2, and DB2 Universal Database are trademarks or registered trademarks of IBM Corporation in the United States, other countries, or both.

Windows and Windows are registered trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product, and service names may be trademarks or service marks of others.

---

### Section 2. Understanding the application

### Overview of the sample application

Before diving in, let's first get familiar with the application used in this tutorial.

The JustPC.com Music Company maintains a database with information about various CDs it has for sale. Through a Web service it provides, you can search the database to retrieve various kinds of information, including a list of artists, a list of CDs recorded by a specific artist and the list of tracks on the CD.

The customer's computer uses a Web browser to connect to a Web server which
runs the C# ASP.NET program. The C# program in turn communicates to a computer that runs the Web service. Finally, the Web service executes stored procedures on the DB2 database server to retrieve the data returned to the client.

While the above diagram shows four independent computers, there is no reason why the functions performed by each computer can't be combined. In fact this tutorial was written on a single Windows 2000 Server computer running DB2 and Visual Studio .NET. While this approach is a bit extreme, you may want to try this tutorial on a single test computer before migrating it to use multiple computers in a more production-like environment.

Database design

The database consists of two tables: one table containing a list of CDs, and a second table containing a list of the tracks for each CD in the first table.

The CDs table contains six columns:

- **CDId** -- a unique identifier for each CD in the database
- **Title** -- name of the CD
- **Artist** -- name of the individual or group that recorded the CD
- **Type** -- genre of the music on the CD
- **Year** -- year the music on the CD was originally recorded
- **Price** -- current list price for the CD

<table>
<thead>
<tr>
<th>CDId</th>
<th>Title</th>
<th>Artist</th>
<th>Type</th>
<th>Year</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1139</td>
<td>Pet Sounds</td>
<td>Rock</td>
<td></td>
<td>15.9900</td>
</tr>
</tbody>
</table>

The Tracks table contains three columns:

- **CDId** -- uniquely identifies the CD containing the track
- **TrackId** -- location of the track on the CD
- **Title** -- title of the track
Stored procedures

The stored procedures access the database tables containing information about the CDs in the music store.

The GetArtists stored procedure returns a list of artists available in the database. There are no parameters in this stored procedure, although the stored procedure needs to insure that each artist is unique.

The GetCDs stored procedure returns a list of CDs that were recorded by a particular artist. The information returned includes CDId, Title, Artist, Type, Year and Price. Note that you must pass a valid value for Artist to this stored procedure. Ideally, this name should be derived from the list of artist names returned by the GetArtists stored procedure.

Finally, the GetTracks stored procedure returns a list of tracks for a particular CD. This stored procedure will return the TrackId and Title columns. The CDId value must be passed to the stored procedure and the best place to get the CDId value is from results returned by the GetCDs stored procedure.

Web service

<table>
<thead>
<tr>
<th>CDId</th>
<th>TrackId</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1139 1</td>
<td>Wouldn't It Be Nice</td>
</tr>
<tr>
<td>2</td>
<td>1139 2</td>
<td>You Still Believe in Me</td>
</tr>
<tr>
<td>3</td>
<td>1139 3</td>
<td>That's Not Me</td>
</tr>
<tr>
<td>4</td>
<td>1139 4</td>
<td>Don't Talk (Put Your Head on My Sho...</td>
</tr>
<tr>
<td>5</td>
<td>1139 5</td>
<td>I'm Waiting for the Day</td>
</tr>
<tr>
<td>6</td>
<td>1139 6</td>
<td>Let's Go Away for Awhile</td>
</tr>
<tr>
<td>7</td>
<td>1139 7</td>
<td>Sloop John B</td>
</tr>
<tr>
<td>8</td>
<td>1139 8</td>
<td>God Only Knows</td>
</tr>
<tr>
<td>9</td>
<td>1139 9</td>
<td>I Know There's an Answer</td>
</tr>
<tr>
<td>10</td>
<td>1139 10</td>
<td>Here Today</td>
</tr>
<tr>
<td>11</td>
<td>1139 11</td>
<td>I Just Wasn't Made for These Times</td>
</tr>
<tr>
<td>12</td>
<td>1139 12</td>
<td>Pet Sounds</td>
</tr>
<tr>
<td>13</td>
<td>1139 13</td>
<td>Caroline No</td>
</tr>
<tr>
<td>14</td>
<td>1139 14</td>
<td>Unreleased Backgrounds</td>
</tr>
<tr>
<td>15</td>
<td>1139 15</td>
<td>Hang On to Your Ego</td>
</tr>
<tr>
<td>16</td>
<td>1139 16</td>
<td>Trombone Dixie</td>
</tr>
</tbody>
</table>
The Web service uses the stored procedures to provide three methods to access the data in the database.

- The GetArtists method returns the set of Artists found in the database. This method has no parameters and returns a .NET DataSet object containing the list of artists.

- The GetCDs method returns a collection of information about the CDs that are recorded by a particular artist. The method has a single parameter -- a string containing the name of the artist.

- The GetTracks method returns a list of tracks for a particular CD. You must supply a valid integer containing the CDId to the web method in order to retrieve the appropriate tracks.

C# ASP.NET program

In order to demonstrate the capabilities of the Web service, a simple ASP.NET program was written in C#. This program calls all three of the Web service’s methods discussed in the previous panels and allows the visitor to the Web site to inquire about any particular artist and CD available for sale.
Section 3. Building the customer database

Starting Visual Studio .NET

Building the customer database begins by creating a DB2 Database Project in Visual Studio .NET. To create the DB2 Database Project, start Visual Studio .NET (start => Programs => Microsoft Visual Studio .NET => Microsoft Visual Studio .NET), and then click the New Project button on the Start Page or choose File => New Project from the main menu. This will display the dialog box shown below.
Select IBM Projects as the Project Type and choose DB2 Database Project as the template. Enter **MusicStore** as the name of the project and choose the appropriate location on your computer to save the project. Press **OK** to begin creating the new DB2 Database Project.

**Connecting to the Music database**

After pressing **OK**, you will be prompted to select a DB2 database connection. Since this is the first time you have used this database, you'll need to create a new database connection. Press the **New Connection** button to display the Database Connection Properties dialog box shown below.
Enter the name of the database in the Database Alias field, then enter a valid user name and password in those fields. You can verify that this information is correct by pressing the **Test Connection** button. Note that as you specify the Database Alias and User Name fields, a value for Connection Name will automatically be constructed.

Once you create this connection, you will be returned to the DB2 Data Connection dialog, where you can select the newly created connection.
Working with the DB2 database project

Once a connection to the database is established, you will see the Visual Studio .NET, Interactive Development Environment (IDE) as shown below. In the Solution Explorer frame (upper right corner of the IDE), you'll notice that under the MusicStore solution is the MusicStore project containing three nodes, Procedures, Functions and Scripts.
The Procedures icon contains the collection of stored procedures associated with this project, while all of the database functions are listed under the Functions icon. These represent elements that can be called from an application program. The elements listed under the Scripts icon, however, contain collections of SQL statements that can be executed directly from the Visual Studio .NET IDE.

Adding a Create Table script

To add a script to DB2 Database Project, right click on the Scripts icon in the Solution Explorer pane and choose Add => Add New Item from the popup menu. This will display the Add New Item dialog box shown below.

![Add New Item dialog box](image)

Choose the Create Table template and enter `CreateCds` as the name of the new script. Then pressing Open will create a new script with a script that can be modified to create a new table.

Building the CDs table

The new element contains a sample SQL script that will create a table. You can edit this script in the Visual Studio .NET IDE as shown below.
The script is grouped into four main sections. The first section drops the existing table, while the other sections will recreate the table. However including a Delete Table statement in the script is not a good idea, since any data that you may load into the table would be lost each time you build your application.

The second section in the script creates the new table by executing an SQL Create Table statement. You need to replace that statement with the one shown below:

```sql
CREATE TABLE CDs

<table>
<thead>
<tr>
<th>CDId</th>
<th>Title</th>
<th>Artist</th>
<th>Type</th>
<th>Year</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>Varchar(255)</td>
<td>Varchar(255)</td>
<td>Varchar(255)</td>
<td>Varchar(4)</td>
<td>Decimal(9,2)</td>
</tr>
</tbody>
</table>
```

The third section adds a unique index on the table. Replace the statement in the template with the one listed below:

```sql
CREATE UNIQUE INDEX CDsIndex On CDs(CDId)
```

The last section contains a series of Insert statements which add some rows to the sample table. Rather than code an Insert statement for each row in the sample data
(there are over 500 rows in the sample data file), you can simply use the DB2 Control Center to import the sample data into the table after you create the table.

Running the script

Since the script will not create the table if the table already exists, you need to verify that the table doesn't exist and manually delete it if it does. The easiest way to do this is to use the IBM Explorer (View => IBM Explorer). The IBM Explorer window contains information about the database with which you’re working.

Normally the IBM Explorer is a floating window over the Visual Studio .NET IDE. However, I like to add the IBM Explorer window as a new tabbed window in the same area as the Solution Explorer windows as shown below.

To delete a table, simply right click on the table's name and choose Delete from the popup menu. A message box will then be displayed verifying that you really want to delete the table.

Once the table has been deleted, simply choose Build => Build Solution from the main menu to create your table. You can also right click on the script in Solution Explorer and choose Compile from the popup menu. Any error messages will be displayed in the Output pane located at the bottom of the Visual Studio .NET IDE.
Building the Tracks table

The same steps used to create the CDs table should be used to create the Tracks table. You can use the following Create Table statement:

```
CREATE TABLE Tracks
(
  CDId Integer NOT NULL,
  TrackId Integer NOT NULL,
  Title Varchar(255)
)
```

This Create Index statement is used to insure that only one unique combination of CDId and TrackId exists in the table:

```
CREATE UNIQUE INDEX TracksIndex On Tracks(CDId, TrackId)
```

Finally, the **Build => Build Solution** command is used to create the second table. Once this is done you can use the DB2 Control Center to populate the table from the sample data.

---

Section 4. Creating the stored procedure in DB2

Creating a new stored procedure

Switching from the IBM Explorer window to the Solution Explorer window gives you the ability to create new stored procedures as part of a Visual Studio .NET project. Right click on the Procedures node under the DB2 Database Project node and choose **Add => New Item** from the popup menu. This will display the following dialog box.
Choose Procedures as the category and the DB2 Stored Procedure Wizard as the template, then enter GetCDs.db2sp as name of the stored procedure as the name of the stored procedure and click the Open button to create your new stored procedure.

Starting the Stored Procedure wizard

The DB2 SQL Stored Procedure wizard will then display a welcome panel that informs you that you are creating a DB2 stored procedure written in SQL. Press Next to begin creating your stored procedure.

In the first step of the wizard, you will choose a name for your stored procedure and then provide comments that describe the stored procedure. Press Next to move to step 2. Note that the name you specify here is the actual name of the stored procedure, while the name you specified when you added a new item to the project is merely the name of the file that contains the stored procedure.
Enter SQL statements

Step 2 of the DB2 Stored Procedure wizard prompts you to enter the set of SQL statements that make up the stored procedure. Each statement is assigned a name, which appears in the Statements list. Selecting one of the names in Statements makes the details available in the Statement detail section of the form.
You can change the name of the statement in the Name file, and change the statements location in the list of statements by changing the number in Order. You can also use the up and down arrows next to the Statements box to do the same thing.

In the SQL statement box, enter the following SQL statement. You should always use a unique name for a parameter and prefix the name with a colon. In this procedure, :Artist is the only parameter.

```sql
SELECT CDS.CDID, CDS.TITLE, CDS.ARTIST, CDS.TYPE, CDS.YEAR, CDS.PRICE 
FROM ADMINISTRATOR.CDS AS CDS 
WHERE CDS.ARTIST = :Artist
```

Don't include a semicolon at the end of the statement, as the wizard will automatically add one when it generates the script for the stored procedure. Also, if you want to use multiple lines in your SQL statement, press Ctrl-Enter. When you have entered all of the statements needed, press Next.

### Defining parameters

Each parameter used in the stored procedure is listed in the Parameters box as shown below. You can display detailed information about a parameter by selecting it in Parameters box. Then you can modify the attributes about the parameter in the Parameter detail section of the form.
In this case, the Artist parameter, is a Varchar(255) value, which you will have to manually define. Depending on the type you choose, you may need to enter additional information such as precision and scale for Decimal.

You can also arrange the order of the parameters in the Parameters box by pressing the up and down arrows or changing the Order value after selecting a particular parameter. You may also use the Add and Remove buttons to change the list of parameters passed to this routine. If you remove a parameter that was automatically, you may need to press the Back button and modify any statements that referenced the parameter.

Finishing the wizard

The next step in the wizard allows you to select files containing code fragments that you would like to insert into the stored procedure, along with choosing the type of error handling that should be used. Step 5 of the wizard allows you to add code to the script that will actually create the stored procedure. You may add statements before and after the actual Create Procedure statement, along with code to execute a Drop Procedure before the stored procedure is created and a Grant Execute statement afterwards.

In the final step of the wizard, you'll see a short summary of the information that will be used to create the stored procedure. You can also preview the script that will be used to generate the stored procedure by pressing the Show generated code button. Once you have verified everything, press Finish to create the stored procedure script.
Compiling the stored procedure

If you chose to view the generated code in the last step of the wizard, you'll notice that the same code is shown below. Before you can use the stored procedure you need to compile it. Right click on the stored procedure name in Solution Explorer and choose **Compile** from the popup menu.
The Output pane (bottom left pane) contains any errors that may have occurred during compilation. If you encounter any errors, you should switch from the Build pane to the IBM DB2 Output Message Pane. This pane will contain DB2 specific error messages.

Once you have successfully compiled the stored procedure, it is available for you to use. If you can switch from the Solution Explorer to the IBM Explorer and refresh the display (right click and choose Refresh from the popup menu), you will see the new stored procedure.

**Testing the stored procedure**

Switch back to IBM Explorer, right click on the Stored Procedures node and choose Refresh from the popup menu. Expand the Stored Procedures node if it isn’t expanded. You should see your new stored procedure.

To run the stored procedure, right click on its name and choose Run Stored Procedure from the popup menu. If the stored procedure has parameters, you will be prompted to enter their values using a dialog like the one shown below.
After entering a value and pressing **OK**, the results will be displayed in the Visual Studio .NET IDE. Since this stored procedure has an input set and a result set, you will see a small plus sign, which expands to list these sets. Then clicking on the result set link will display the results from the stored procedure as shown below.

Adding the other stored procedures

After you finish this stored procedure, you should repeat the same process for the remaining stored procedures that will be called by the Web service.
The GetArtists stored procedure is illustrated below. It merely returns a unique list of artists from the CDs table:

```sql
SELECT DISTINCT CDS.ARTIST
FROM ADMINISTRATOR.CDS AS CDS
```

The GetTracks stored procedure takes a single parameter, :CDId, which identifies the CD containing the tracks you want to retrieve:

```sql
SELECT TRACKS.TRACKID, TRACKS.TITLE
FROM ADMINISTRATOR.TRACKS AS TRACKS
WHERE TRACKS.CDID = GetTracks.:CDId
```

In the next section, we'll build a C# program to access the DB2 database.

Section 5. Building the C# Web service

Creating the Web service project in Visual Studio .NET

In order to access a DB2 database, you need to build a C# program that includes the appropriate classes from the DB2 managed data provider. The DB2 managed data provider is so well integrated with the tools already in Visual Studio .NET that it makes it very easy for any .NET developer to make the transition to DB2.

You can create the C# Web service by starting with the default C# Web Service template, changing the names used with the newly created Web service to reflect this particular application, and then adding Web methods that call the DB2 stored procedures to access the database.

To create the new Web service, start Visual Studio .NET (start => Programs => Microsoft Visual Studio .NET => Microsoft Visual Studio .NET), and then click the New Project button on the Start Page or choose File => New Project from the main menu. You will see the New Project dialog box as shown below:
Select Visual C# Projects in the Project Types section of the dialog box and then select ASP.NET Web Service in the templates section. Next choose the location for your Web service. This example uses http://multivac/IBM/MusicStoreWebService. Finally click the OK button to create your Web service.

Using Visual Studio .NET

After clicking OK on the New Project dialog box, Visual Studio .NET creates a template application that you can use to begin building your Web service. The Solution Explorer pane (found on the right side of the Visual Studio .NET window, just below the toolbar) contains a tree view organization of the solution.
Adding the DB2 library to the solution

Before you can start adding DB2 code to your new Web service, you need to add the DB2 Library to the solution, right click on the References icon and click **Add Reference** from the popup menu. This will display the Add Reference dialog box. Scroll through the list of Component Names until you see `IBM.Data.DB2.dll`. Select this component and then press the Select button on the right side of the dialog box. The selected component will appear in the Selected Components section of the dialog box. Finally, press the **OK** button to add the reference to your solution.
Naming the service

By default, Visual Studio .NET adds a file called `Service1.asmx` containing a prototype Web service to your project. In order to keep things straight, you should rename this file to something more meaningful, like `MusicStoreInfo`. To rename the file, right click on the file's name in the Solution Explorer and choose Rename from the popup menu. Then type `MusicStoreInfo.asmx` over the existing file name.

Next you need to open the file to make rest of the changes. Right click over the new file name and choose View Code from the popup menu. This will display the source code for the prototype in the main pane of the Visual Studio .NET development window.
Making the template code usable

As with any template code, you'll need to make a few changes to fit your current situation. The first step is to add a reference to the DB2 managed provider to the top of the listing:

```csharp
using IBM.Data.DB2;
```

Next you need to change the name of the service to complete the change from the previous step. First you need to change every reference of Service1 to MusicStoreInfo. You can either change it by using a search and replace (choose **Edit => Find and Replace => Replace** from the main menu) or by changing the three places `Service1` appears in the code, one of which is a comment. The relevant code fragment is shown below with the changes displayed in bold:

```csharp
namespace MusicStoreWebService
{
    /// <summary>
    /// Summary description for MusicStoreInfo.
    /// </summary>
    {
        public MusicStoreInfo()
        {
            // CODEGEN: This call is required by the ASP.NET Web Services Design
            InitializeComponent();
        }
        
        Component Designer generated code
    }
}
```
If you have never used C# in Visual Studio .NET, a word of caution is in order. Visual Studio .NET has embedded some code of its own in the file you are editing. This embedded code contains some methods that are used by the Visual Studio .NET design tools, along with other references that insure that the service is properly initialized and any resources it uses are disposed of properly. Do not remove this code or your Web service may not work properly.

Defining the GetArtists Web method

Now that you've customized the template to fit this particular application, you are ready to begin defining methods for the Web service. The following function implements the GetArtists method. Every Web method must be identified as a Web method by placing [WebMethod] in front of the function definition.

```csharp
[WebMethod]
public DataSet GetArtists()
{
    DataSet ds = new DataSet();
    DB2Connection conn = new DB2Connection("database=Music");
    DB2Command cmd = new DB2Command("Administrator.GetArtists", conn);
    cmd.CommandType = CommandType.StoredProcedure;
    DB2DataAdapter adpt = new DB2DataAdapter(cmd);
    adpt.Fill(ds, "Artists");
    return ds;
}
```

You can then define the function as you would normally define a function. GetArtists is a public method that returns a DataTable object containing the information extracted from the database.

The GetArtists method begins by declaring a new instance of the DataSet object called ds. The DataSet object can hold a collection of DataTable objects. Then a DB2Connection object is instantiated that contains a connection string pointing to the particular instance of the database. Since the database runs on the same server, all that you need to include in the connection is database=Music.

The DB2Command object combines the DB2Connection created in the previous line of code with the fully qualified name of the stored procedure. Next, you should specify the type of command contained in the DB2Command object using the CommandType property. The last object to be created is a DB2DataAdapter, which includes a reference to the DB2Command object that you just created.

After that the data adapter's Fill method is called to populate the DataSet object ds with the results of the stored procedure. Then finally, the resulting DataSet object is returned to the calling program.

Alternately, you can store configuration strings in the application's web.config file. To modify the sample program, simply create a file called web.config in the same directory as the Web page and insert the following XML elements. If you already have a web.config file, just add the elements between the <appSettings> element beneath the existing <configuration> element:
<configuration>
  <appSettings>
    <add key="ConnectionString" value="database=Music" />
  </appSettings>
</configuration>

Then replace this statement in the C# Web service:

```csharp
DB2Connection conn = new DB2Connection("database=Music");
```

with this statement:

```csharp
DB2Connection conn = new DB2Connection(ConfigurationSettings.AppSettings("ConnectionString"));
```

Using parameters in a stored procedure

The GetArtists method is somewhat simpler than many calls to a stored procedure because no parameters were included in the stored procedure. The GetCDs method, however illustrates how to call a stored procedure with a parameter.

As you would expect, the code for this method is nearly identical to the GetArtists method. However, following the line of code where you specify the type of command in the DB2Command object, you should include the list of parameters for the stored procedure.

```csharp
[WebMethod]
public DataSet GetCDs(string Artist)
{
    DataSet ds = new DataSet();
    DB2Connection conn = new DB2Connection("database=Music");
    DB2Command cmd = new DB2Command("Administrator.GetCDs", conn);
    cmd.CommandType = CommandType.StoredProcedure;
    cmd.Parameters.Add(":Artist", DB2Type.VarChar,64).Value = Artist;
    DB2DataAdapter adpt = new DB2DataAdapter(cmd);
    adpt.Fill(ds, "CDs");
    return ds;
}
```

Within the DB2Command object is a DB2ParameterCollection object containing a DB2Parameter object for each parameter used by the stored procedure. The Add method of the DB2ParameterCollection object returns a DB2Parameter object with the specified name, data type and optional size. In this case, the name of the parameter is :Artist and its data type is Varchar(64).
Because the Add method returns a DB2Parameter object, you can use the DB2Parameter object’s Value property to store the value you wish to use in the newly created DB2Parameter object. While you could have explicitly created each parameter object and then assigned it a value, this highly compact form is far more readable if you have a stored procedure with more than a handful of parameters.

Finishing the Web service

With two of the Web methods finished, only the GetTracks method remains to be added. This Web method is very similar to the GetCDs method and is shown below:

```csharp
[WebMethod]
public DataSet GetTracks(int CDId)
{
    DataSet ds = new DataSet();
    DB2Connection conn = new DB2Connection("database=Music");
    DB2Command cmd = new DB2Command("Adminitrator.GetTracks", conn);
    cmd.CommandType = CommandType.StoredProcedure;
    cmd.Parameters.Add(":CDId", DB2Type.Integer).Value = CDId;
    DB2DataAdapter adpt = new DB2DataAdapter(cmd);
    adpt.Fill(ds, "Tracks");
    return ds;
}
```

After you enter all of the code for these Web methods, you are ready to compile your Web service. Choose Build => Build Solution from the menu bar. Assuming that you entered everything properly, you'll see a message in the Output pane (below the code pane) that indicates that the build succeeded. Otherwise, you'll find a series of error messages in the in the Output pane that will help you identify and correct the errors.
Section 6. Displaying the results in C# using ASP.NET

Creating a C# ASP.NET solution in Visual Studio .NET

Now that the Web service is ready to use, all that remains is to create the ASP.NET application. This involves creating a new Visual Studio .NET solution, adding a reference to the new Web service, designing the user interface, and adding the code behind the user interface that accesses the Web service.

While you could easily add an ASP.NET Web page to your current project, it wouldn't demonstrate the independence of the Web service from the ASP.NET application program. So if you close your existing solution in Visual Studio .NET (File => Close Solution), you can open a new solution (File => New Project) and start building the ASP.NET application.
Adding a Web reference

Once you have your new template, you need to add a Web reference to the Web service that you just created. Right click on the References icon in the Solution Explorer and choose Add Web Reference from the popup menu.

Next, enter the address the Web service you just created into the address field:

http://multivac/IBM/MusicStoreWebService/MusicStoreInfo.asmx

After you finish typing in the address, press the Enter key. You will then see the information about the Web reference displayed in the dialog box as shown in the figure below:
You may then browse the definitions and see the results through this dialog box. If you click on an individual method, you will see more detailed information about the method. You will also be given an opportunity to test the method and view the results. If everything looks fine, press the Add Reference button at the bottom of the dialog box to add the Web reference to your application.

Creating the Web form

The first step to creating the Web form is to change the default name of the page from WebForm1.aspx to JustPCMusicStore.aspx. Simply right click over the name in the Solution Explorer and choose Rename from the popup menu. Then you may enter the new name over the old name.
Visual Studio .NET supports a drag and drop design tool that helps you lay out your Web pages quickly. In this case, four elements were dragged from the Web Forms toolbox on the left side of the screen onto the Web form. A Label control was dropped and its Text property was set to The JustPC Music Store, and its Font properties were modified to display the text in bold as well as make it larger.

Next a Button control was dropped on the form and its Text property was changed to Show CDs for. Adjacent to the Button control, a DropDownList control was placed on the Web form. Below the first Button and DropDownList, you should drag and drop a second pair. This time you should change the Button’s Text property to Show Tracks for.

Finally a DataGrid control was added immediately below the pair of Button and the DropDownList controls. The DataGrid will hold the final list of tracks for the selected CD title.

If you wish, you can view the HTML associated with this Web page by switching from the Design view to HTML view at the bottom of the editing surface. Any changes you make in the HTML view will be reflected in the Design view when you switch back.

Loading the drop-down lists
Once the Web form is laid out, double click on the form's background area. This will open a new tab containing the code associated with the Web form and your cursor will be positioned in the **Page_Load** event. The **Page_Load** event is fired each time the Web page is generated, both when the page is originally requested and each time the Web page posts information back to the server.

```csharp
private void Page_Load(object sender, System.EventArgs e)
{
    if (!IsPostBack)
    {
        multivac.MusicStoreInfo ws = new multivac.MusicStoreInfo();
        DataSet ds = new DataSet();
        ds = ws.GetArtists();
        DropDownList1.DataSource = ds;
        DropDownList1.DataTextField = "Artist";
        DropDownList1.DataBind();
        ds = ws.GetCDs(DropDownList1.SelectedItem.Value);
        DropDownList2.DataSource = ds;
        DropDownList2.DataTextField = "Title";
        DropDownList2.DataBind();
    }
}
```

The best time to load the **DropDownList** is when the page is originally requested, and this code ensures that the program isn't responding to a post back request. Then it creates a new Web service using the variable `ws` from Web reference you added earlier. The specific data type for this Web service is `multivac.MusicStoreInfo`. A new `DataSet` object is also created at the start of the event. The Web service is then used to retrieve the information about the artists in the database using the **GetArtists** method.

An object reference to the data set is stored in the **DataSource** property of the drop-down list. The column name containing the information to be displayed in the drop-down list is assigned to the **DataTextField** property. Finally the drop-down list's **DataBind** method is called to bind the data from the data set to the drop-down list control.

This process is repeated for the second drop-down list, and is discussed in the next panel.

### Getting the list of CDs for a particular artist

The code to display the CDs for a particular artist exists in two places in this program. Once in the **Page_Load** event, described previously and once in the Show CDs for button's **Click** event, which is described here.

The easiest way to define the **Click** event for the button is to switch back to the Design view for the Web page and double click on the Show CDs for Button control. Visual Studio .NET will automatically define the new event for you in the code section of the Web page and place the cursor in the middle of the event.
The Web service and a data set object are defined as in the Page_Load event, but this time the GetCDs method was used to extract information about the currently selected artist in the drop down list. You can retrieve this value from DropDownList1 by using its SelectedItem property to identify the currently selected item and then the item’s Text property to get the string value of the currently selected item:

```csharp
private void Button1_Click(object sender, System.EventArgs e)
{
    multivac.MusicStoreInfo ws = new multivac.MusicStoreInfo();
    DataSet ds = new DataSet();
    ds = ws.GetCDs(DropDownList1.SelectedItem.Value);
    DropDownList2.DataSource = ds;
    DropDownList2.DataTextField = "Title";
    DropDownList2.DataValueField = "CDId";
    DropDownList2.DataBind();
}
```

Once you have the string value, it’s merely a matter of passing that value to the GetCDs method to retrieve a data set with all of the CDs for that particular artist. Once the data set has been created, you save an object reference to it in the second drop-down list’s DataSource property.

DropDownLists have a useful feature that allows you to associate two values with each item in the drop down list. The DataTextField represents the visible content in the list. The DataValueField represents a "hidden" value that's distinct from the text value. Thus you can display the title of the CD, while keeping track of the CD's unique identifier.

Finding the tracks for a CD

Clicking on the Show Tracks for button will fire the Button2_Click event shown below. This time the GetTracks method is called to return the tracks for a particular CD. The CDId value is extracted from the drop-down list's SelectedItem.Value property in the same way the SelectedItem.Text property was used to extract the text name for the Artist in the first drop-down list:

```csharp
private void Button2_Click(object sender, System.EventArgs e)
{
    multivac.MusicStoreInfo ws = new multivac.MusicStoreInfo();
    DataSet ds = new DataSet();
    ds = ws.GetTracks(Convert.ToInt32(DropDownList2.SelectedItem.Value,10));
    DataGrid1.DataSource = ds;
    DataGrid1.DataBind();
}
```

The extracted value is converted to an integer value by using the Convert.ToInt32 method and specifying that the string returned by the SelectedItem.Value property contains a base 10 value.
Next an object reference to the returned data set object is stored in the data grid's DataSource property. Calling the DataBind method of the data grid will populate the data grid using all of the rows and columns extracted from the Tracks table. The DataGrid control will automatically create all of the necessary columns and rows based on the information stored in the data set.

Displaying the results on the Web form

With the coding complete, you can choose Debug => Start from the main menu to see everything work. When the program starts, the Web page will be generated and the contents of the drop down form will be loaded. Then after selecting one of the artists in the drop down list and pressing the Show CDs for button, you'll see a list of titles associated with that artist. After choosing one of the titles and pressing the Show Tracks for button, you'll see a Web page similar to the one shown below:
Section 7. Summary

Building Web Services with C# and DB2 summary

In this tutorial you learned how to create a C# ASP.NET Web page, along with a C# Web service that would access a DB2 database though a set of stored procedures using the DB2 Managed Provider. The DB2 managed provider works with all of the standard Microsoft classes like `DataSet`, which makes it very easy to incorporate DB2 into a .NET application.

The work was about evenly split between the database activities (defining the tables, loading the data and creating the stored procedures) and programming activities (building the Web service and the application program). Visual Studio .NET coupled with the DB2 Development Add-in makes short work of most tasks.

The DB2 Development Add-in was used to define the database tables and to create and test the stored procedures that provide the data to the Web service. Visual Studio .NET was used to create both the Web service and the sample application that used the Web service to access the database. The DB2 managed data provider is well integrated in Visual Studio .NET and makes it extremely easy to access a DB2 database.
Resources

Learn

- Learn how to create stored procedures using the DB2 development center in the IBM developerWorks tutorial Creating stored procedures with DB2.
- Learn how to create SQL queries using the SQL Assist for DB2 tool in the article Creating SQL queries the easy way with SQL Assist for DB2 UDB Version 8.1 from IBM DB2 Developer Domain.

Get products and technologies

- Download a copy of the DB2 .NET Enablement beta, which includes the DB2 managed data provider, from: http://www7b.software.ibm.com/dmdd/downloads/dotnetbeta/.
- Download a copy of the .NET Framework, including a command-line version of the C# compiler from: http://www.asp.net/download.aspx?tabindex=0&tabid=1.
- Download a copy of ASP.NET Web Matrix, a free ASP.NET development tool, from: http://www.asp.net/webmatrix/default.aspx?tabindex=4&tabid=46. This tool requires the .NET Framework listed above.

Discuss

- Participate in the discussion forum for this content.

About the author

Wayne Freeze
Wayne S. Freeze, a Studio B author, is a full-time author and computer consultant with more than a dozen books and over fifty articles to his credit. He has nearly 30 years of experience using all types of computers, from small, embedded microprocessor control systems to large-scale IBM mainframes. His most recent book include Windows Game Programming with Visual Basic and DirectX [Que, 2001], and the Visual Basic 6 Database Programming Bible [Hungry Minds, 2001], plus his articles have appeared in such magazines as aspnetPRO and Microsoft Office Solutions. While Wayne specializes in web and database programming using Visual Basic, he can't resist his first love, game programming. Wayne has a master's degree in management information systems as well as degrees in computer science and engineering. You can visit his Web site at http://www.JustPC.com and send him e-mail at WFreeze@JustPC.com. He loves reading e-mail from his readers, whose ideas, questions, and insights often provide inspiration for future books and articles.