Windows® Embedded

POSReady: Using Linq to SQL in POS for .NET Applications

By Gordon H. Smith, Embedded MVP

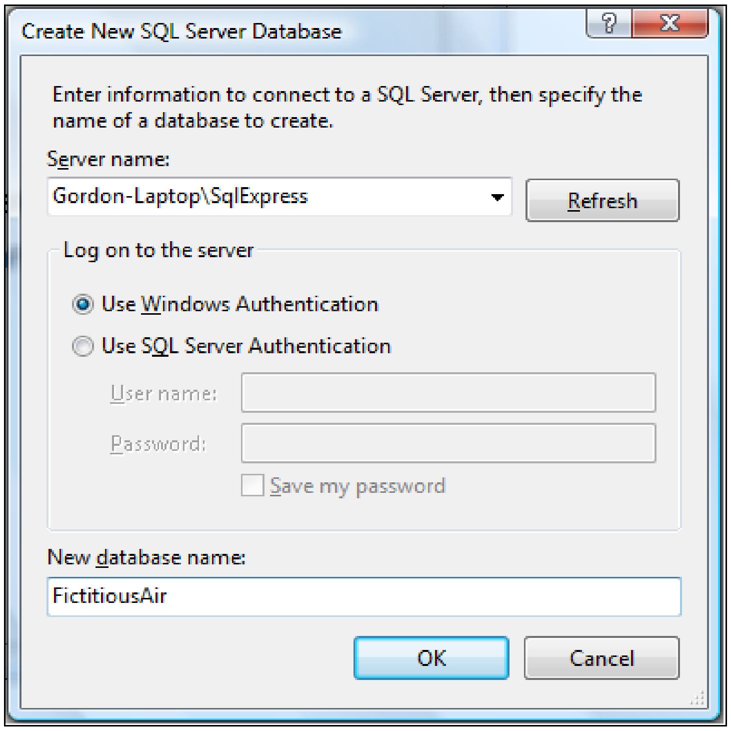
Developers of retail applications based on POS for .NET and the .NET platform realize benefits that extend beyond rich interaction with peripheral devices by connecting their applications with distributed resources such as database servers and web services. By using the .NET platform components, developers can rapidly create multi-tiered applications and connect retail devices to their enterprise. Linq to SQL one .NET feature that gives developers a natural interface from .NET applications to back-end data hosted in Microsoft® SQL Server® databases.

This article builds upon the results from the article Creating a Proof-of-Concept POS Application. At the conclusion of that article, you should have a functional prototype application with the ability to retrieve data from a Magnetic Stripe Reader (MSR), and the ability to respond to Plug-and-Play hardware insertion and deletion events. Data lookup was left as an exercise for the reader. This article covers how to look up data from a SQL Server database for the FictitiousAir prototype application.

This article assumes the presence of SQL Server Express on your system, which is available as a free download at Microsoft SQL Server 2008 Express with Tools. In a production environment, the database would reside on a remote server, but for the purposes of this prototype a local database will suffice.

You can use Visual Studio® 2008 to create a database in SQL server, populate some fictional passenger entries and enhance the kiosk application to query for that data. To create the database, open the Server Explorer pane and right-click Data Connections. Choose the Create New SQL Server Database option and fill it out as shown in Figure 1. Use your SQL Server Express instance for the server name.

Now that the database is available for use, create a table to keep track of passengers. For this prototype, you can use the following columns: ID (the primary key), First Name, Last Name, Flight Number, Destination, Seat Number and a date/time showing when the passengers printed their boarding passes.


Create New SQL Server Database

Enter information to connect to a SQL Server, then specify the name of a database to create.

Server name: Gordon-Laptop\SqlExpress

Log on to the server

Use Windows Authentication [✓]

Use SQL Server Authentication

User name:

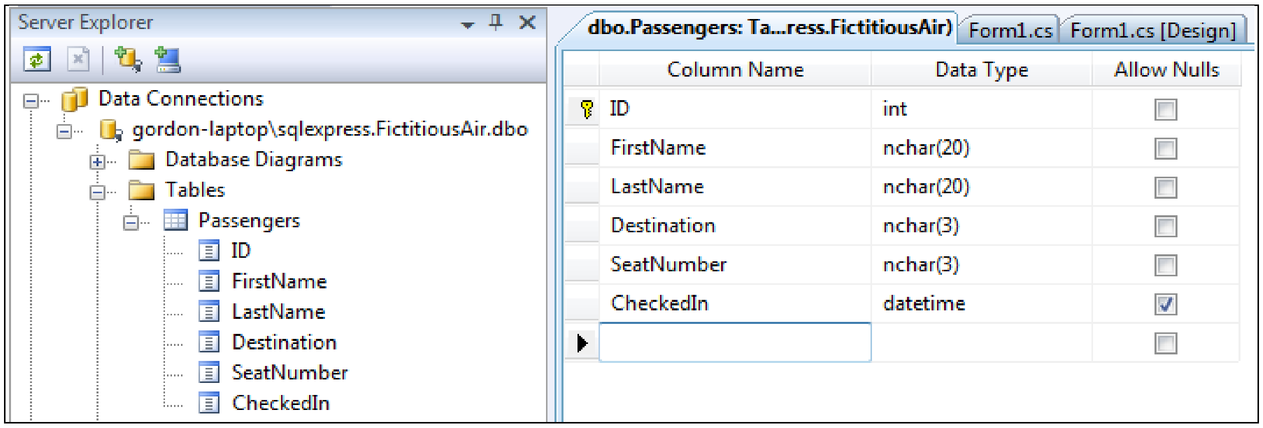
Password:

Save my password

New database name: FictitiousAir

Figure 1 Create Database dialog in Visual Studio 2008

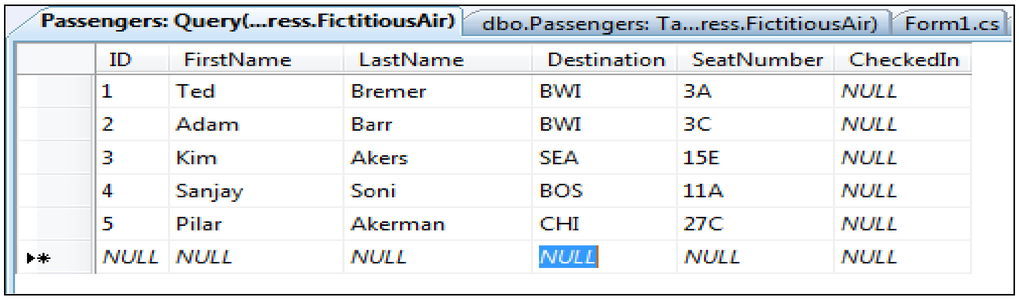
To create the Passengers table, right click the Tables node in the Server Explorer pane, choose Add New Table and enter the column metadata as shown in Figure 2. When done, close and save the table.



Column Name	Data Type	Allow Nulls
ID	int	☐
FirstName	nchar(20)	☐
LastName	nchar(20)	☐
Destination	nchar(3)	☐
SeatNumber	nchar(3)	☐
CheckedIn	datetime	✓
		☐

Figure 2 Create Table user interface in Visual Studio 2008

With the table schema saved you can enter some sample test data. Right-click the Passengers node in the Server Explorer pane and choose Show Table Data. Enter some sample data, as shown in Figure 3. For testing purposes ensure you add entries that will match names scanned from credit cards in your environment.

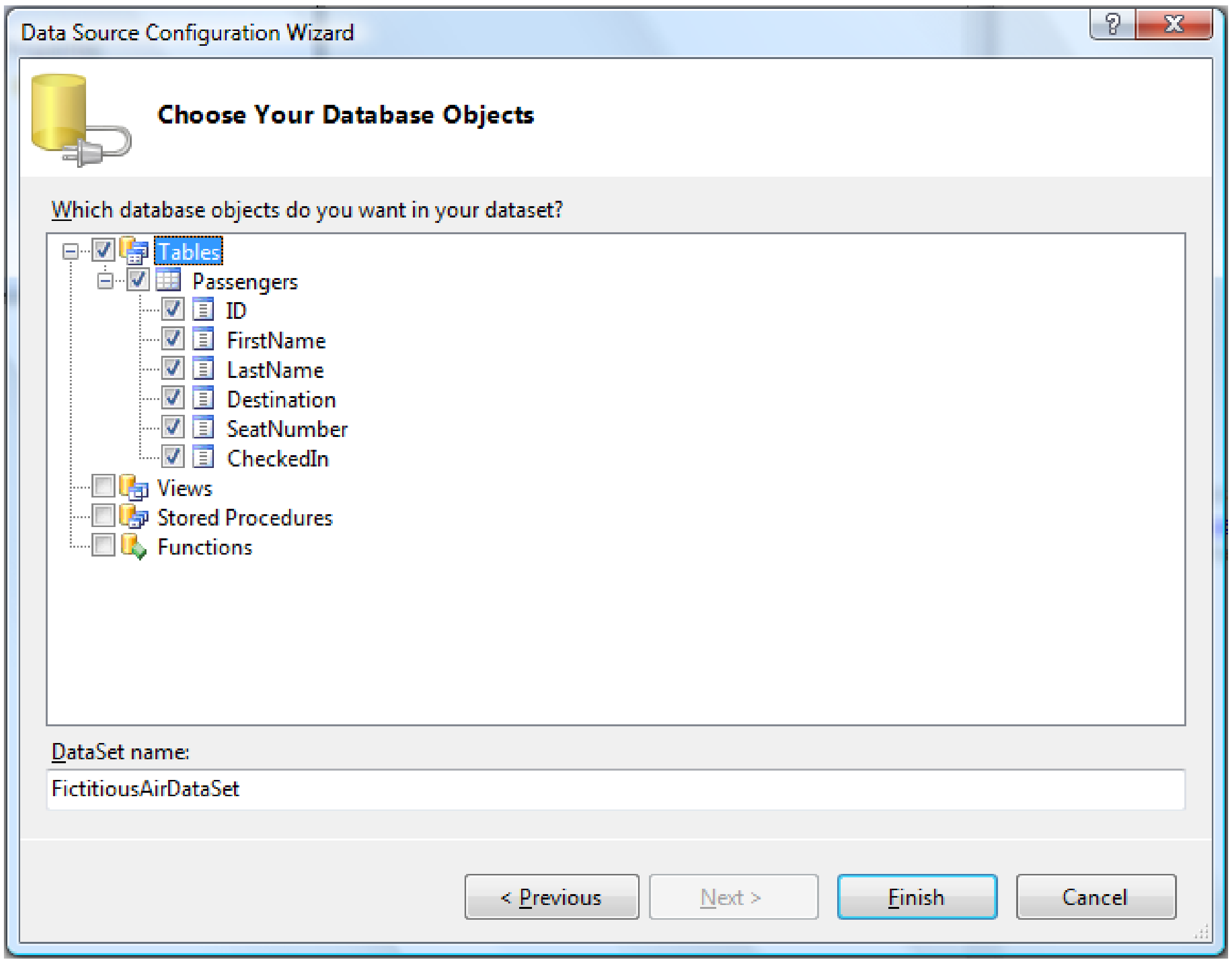

Passengers: Query(...ress.FictitiousAir) | dbo.Passengers: Ta...ress.FictitiousAir) | Form1.cs

	ID	FirstName	LastName	Destination	SeatNumber	CheckedIn
	1	Ted	Bremer	BWI	3A	NULL
	2	Adam	Barr	BWI	3C	NULL
	3	Kim	Akers	SEA	15E	NULL
	4	Sanjay	Soni	BOS	11A	NULL
	5	Pilar	Akerman	CHI	27C	NULL
▸*	NULL	NULL	NULL	NULL	NULL	NULL

Figure 3 Sample data entry

This data model is too simplistic for a possible final solution, but it demonstrates the ease of integrating SQL Server access into the prototype application. Now that the Passengers table has been created and populated, you must enhance the application needs to query and update that data. In the Solution Explorer pane, right-click the FictitiousAir project and choose Add and New Item. In the Add New Item dialog, select the LINQ to SQL Classes template and accept the default filename. Next, drag the Passengers table from the Server Explorer pane into the DataClasses1.dbml design pane and build the solution. Classes describing the Passenger table now exist in the FictitiousAir project; however, the project needs to know where the table resides.

To add the data source to the project, choose Show Data Sources from the Data menu and click Add New Data Source in the Data Sources pane. Use the Data Source Configuration wizard that opens to select the sample database created. When prompted, be sure to select the Passengers table and click finish, as shown in Figure 4.


Data Source Configuration Wizard

Choose Your Database Objects

Which database objects do you want in your dataset?

- [✓] Tables
  - [✓] Passengers
    - [✓] ID
    - [✓] FirstName
    - [✓] LastName
    - [✓] Destination
    - [✓] SeatNumber
    - [✓] CheckedIn
- [ ] Views
- [ ] Stored Procedures
- [ ] Functions

DataSet name: FictitiousAirDataSet

Figure 4 Data Source Configuration Wizard

The next step is to make the application utilize the data in a meaningful way. The remaining goals for this article are to look up the passenger's information when the user presses the Look up itinerary button and to update the database with the check-in time when the user presses the Print Boarding Pass button. In the form designer, double-click the Look up itinerary button, navigate to its Click event, and enter the code below. Additionally, because you are likely to swipe more than one credit card during the test, re-enable the DataEvent, as shown below.

```
private void activeMsr_DataEvent(object sender, DataEventArgs e)
{
    // Read the track data.  For the sake of this prototype
    // just the name data needs to be gathered.
    //
    txtFirstName.Text = activeMsr.FirstName;
    txtMiddleInitial.Text = activeMsr.MiddleInitial;
    txtLastName.Text = activeMsr.Surname;

    // Re-enable the event
    //
    activeMsr.DataEventEnabled = true;
}
```

```
private void btnLookup_Click(object sender, EventArgs e)
{
    DataClasses1DataContext dc = new DataClasses1DataContext();

    var results = from p in dc.Passengers
                  where p.FirstName == txtFirstName.Text &&
                        p.LastName == txtLastName.Text &&
                        p.CheckedIn == null
                  select new { p.Destination, p.SeatNumber };

    if (results == null || results.Count() != 1)
    {
        MessageBox.Show("Please see agent for assistance.");
    }
    else
    {
        var x = results.First();
        txtDestination.Text = x.Destination;
        txtSeatAssignment.Text = x.SeatNumber;
    }
}
```

As you can see in the code sample above, Linq to SQL provides an elegant way of querying for data with support integrated into the language. The developer does not need to create multiple objects for managing connection state to the database, nor deal with the complexities of sending commands and retrieving results as present in previous data-retrieval API models.

Last, the application needs to update the database to indicate that the user has completed checking in. In the form designer, double-click the Print Boarding Pass button, navigate to its Click event, and enter the following code.

```
private void btnPrintBoardingPass_Click(object sender, EventArgs e)
{
    DataClasses1DataContext dc = new DataClasses1DataContext();

    var results = (from p in dc.Passengers
                  where p.FirstName == txtFirstName.Text &&
                        p.LastName == txtLastName.Text &&
                        p.Destination == txtDestination.Text &&
                        p.SeatNumber == txtSeatAssignment.Text
                  select p).SingleOrDefault();

    results.CheckedIn = DateTime.Now;

    dc.SubmitChanges();

    // Reset the form
    //
    txtFirstName.Text = "";
    txtLastName.Text = "";
    txtDestination.Text = "";
    txtSeatAssignment.Text = "";
}
```

Developers of retail applications who adopt POS for .NET as their peripheral device interface can also take advantage of the myriad of other technology advancements enabled by the .NET platform. Retail applications that use SQL Server can gain the benefits of an easier-to-use programming paradigm to access back-end data through Linq to SQL.