**SQL Server CE introduction**

**(Part 5)**

**Topic 1: Access data through DataSets**

**Topic 2: Access data through SqlCeResultSets**



[**http://msdn.microsoft.com/rampup**](http://msdn.microsoft.com/rampup)

**Introduction**

Developing applications for Windows Mobile Devices is similar to developing applications for the desktop, especially if you make use of either Visual Basic.NET or Visual C#. You can use the same development tools to develop your applications, although there are also differences between developing applications for Windows Mobile Devices and developing desktop applications. The same is true for working with data on the device. No real application exists without data. To store data on the device, a developer is of course free to choose his or her own strategy. In this article you will read more about using SQL Server 2005 Compact Edition and about different options to access data that is stored locally on the device. In this article we will focus on performance of the different access methods regarding execution time and the amount of code you have to provide yourself. In the code samples, the Orders and Order\_Details tables of the Northwind sample database are used. This article will focus on Visual Studio 2008 in combination with SQL Server 2005 Compact Edition 3.5.

**SQL Server 2005 Compact Edition**

There might be some confusion about different names and different versions of SQL Server that are capable of running on Windows Mobile Devices. At the moment of writing this article, there is more consistency in naming the product. Here is a short overview of the most important versions of SQL Server 2005 for Windows Mobile device developers:

* **SQL Server Mobile 3.0** - released in 2005 with Microsoft Visual Studio 2005 and with SQL Server 2005. SQL Server Mobile 3.0 is supported on devices and on Tablet PCs. As long as development tools are installed on a desktop system, SQL Server Mobile 3.0 also runs on that particular desktop machine.
* **SQL Server 2005 Compact Edition 3.1** – released in 2006 and based largely upon SQL Server Mobile 3.0, this version of SQL Server CE runs both on Windows Mobile Devices and on desktop / laptop systems without any restriction. SQL Server CE is highly compatible with SQL Server editions and provides full relational database functionality in a small footprint. This version of SQL Server CE is installed in ROM on Windows Mobile 6 Devices.
* **SQL Server 2005 Compact Edition 3.5 SP1** – released in 2007 as a separate download and also integrated in Visual Studio 2008. Building upon and extending the functionality of SQL Server CE 3.1, this version of SQL Server CE also allows synchronization with backend servers through the Microsoft Synchronization Services for ADO.NET.

**SQL Server 2005 Compact Edition Background**

SQL Server CE is a lightweight database and unlike other versions of SQL Server, SQL Server CE runs in process with an application using it. This means that SQL Server CE is not implemented as a ‘real’ server that runs separately. A SQL Server CE database is stored in a single file with the extension **sdf**, allowing easy synchronization between a desktop and a device, for instance through ActiveSync (Windows XP) or the Windows Mobile Device Center (Windows Vista). More sophisticated synchronization options are possible through RDA, Merge Replication and Sync Services for ADO.NET to allow synchronization between a back-end server and multiple devices, even with support for conflict resolution in case multiple users update the same data in a database. SQL Server CE databases have different levels of security. Amongst others, databases can be password protected and can be encrypted. This makes a lot of sense for use on Windows Mobile devices, since devices can be relatively easy lost, and especially if the database is stored on a storage card, that storage card might be stolen.

For the remainder of this article, we will focus on using SQL Server 2005 Compact Edition 3.5 in combination with a managed application. You will learn about different ways to retrieve, modify and insert data from inside a managed application in a number of different ways, each with their own specific pros and cons.

To compare each solution realistically, we are going to make use of the Northwind sample database, displaying Orders and Order Details. In order to make use of the Northwind sample database that is installed in the folder **C:\Program Files\Microsoft SQL Server Compact Edition\v3.5\Samples** (default installation path with installing Visual Studio 2008), you need to run Visual Studio 2008 with administrator elevations under Windows Vista. Not doing so means you cannot create a connection to the database.

**Using Strongly Typed DataSets**

Making use of strongly typed **DataSets** gives you a simple way to access a SQL CE database. A lot of code is automatically generated for you, but a DataSet is an in-memory cache of data, retrieved from a data source like SQL Server CE. In other words, the DataSet contains a physical copy of the data you are working on. This means that it takes some time to load data from the database into the DataSet and modifications made on data inside the DataSet are not automatically updated in the database. Once the DataSet is initialized, working with data is very fast, since you are working with an in-memory copy of the data. For large databases, you might run into low memory situations, having the same data in memory twice.

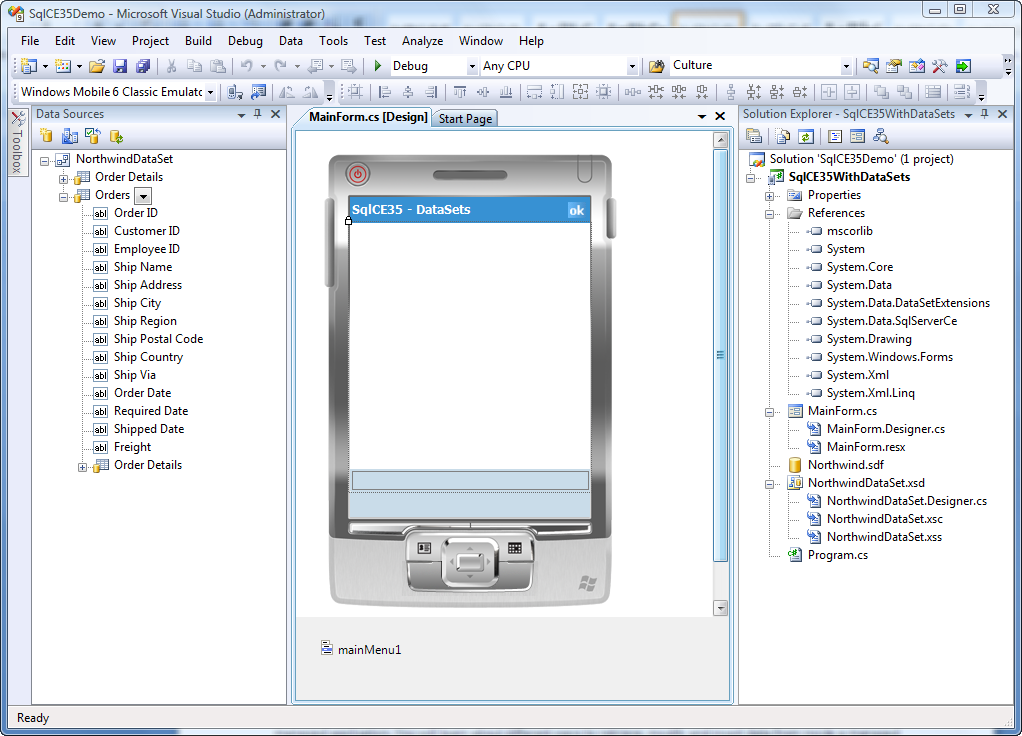


Figure - Strongly Typed DataSets

In figure 1 you see Visual Studio 2008 with a Smart Device Project for which a new Data Source was added to the Northwind sample database. Inside the Data Source wizard, two tables were selected to be added to the strongly typed DataSet, the Oder Details table and the Order tables. As you can see in the Data Sources window, the relation between the Orders and the Order Details table is automatically created for you. If you want to create a master – detail view as part of your user interface, you can simply drag and drop controls from the Data Sources window to the form you are creating. To maintain the relation automatically, it is important to select Order Detail information from within the Orders table.



Figure - Auto-generated Forms through the Visual Studio Designer

In Figure 2 you can see the Forms that were automatically generated for you. Without any coding you are ready to display information from the Northwind database. The only catch is that data will be copied to the DataSet, so you need to write some additional code to make sure that any changes will be saved in the underlying database as well. The code that Visual Studio generated for you to be able to display data on the MainForm is shown in listing 1. Similar code was added by Visual Studio to display summary information and to populate the edit dialog. Prior to filling User Interface controls you have the possibility to modify the connection string that is used to connect to the SQL Server CE database. This makes a lot of sense, especially for password protected databases, because you are running into security issues if you let Visual Studio generate connection strings containing password information.

Listing – Auto-generated code to initialize UI Controls with data from a DataSet

public partial class MainForm : Form

{

public MainForm()

{

InitializeComponent();

}

private void MainForm\_Load(object sender, EventArgs e)

{

if (NorthwindDataSetUtil.DesignerUtil.IsRunTime())

{

this.order\_DetailsTableAdapter.Fill(this.northwindDataSet.Order\_Details);

}

if (NorthwindDataSetUtil.DesignerUtil.IsRunTime())

{

this.ordersTableAdapter.Fill(this.northwindDataSet.Orders);

}

}

private void newMenuItemMenuItem\_Click(object sender, EventArgs e)

{

ordersBindingSource.AddNew();

SqlCE35WithDataSets.OrdersEditViewDialog ordersEditViewDialog =

SqlCE35WithDataSets.OrdersEditViewDialog.Instance(this.ordersBindingSource);

ordersEditViewDialog.ShowDialog();

}

private void ordersDataGrid\_Click(object sender, EventArgs e)

{

SqlCE35WithDataSets.OrdersSummaryViewDialog ordersSummaryViewDialog =

SqlCE35WithDataSets.OrdersSummaryViewDialog.Instance(this.ordersBindingSource);

ordersSummaryViewDialog.ShowDialog();

}

}

In the **MainForm\_Load** event handler, data from the Northwind database is copied to the DataSet, which in turn displays the data to the user through data binding. Additional code is also added to show detail information and to enter new order records. To get some information about the performance of retrieving data from a SQL CE Database through a DataSet, you can start a **Stopwatch** as the first statement in the MainForm\_Load event handler and stop the Stopwatch as the last statement in the MainForm\_Load event handler, after which you can display the elapsed time in a status bar. A similar performance measurement can be done to measure the time it takes to display detailed information when moving from one record to another by taking the time the user presses a hardware key until the time the text of one of the edit controls that is displayed on the MainForm is changed. Figure 3 shows you these performance results when using a strongly typed DataSet. As you can see, it takes a good 13 seconds on Device Emulator from the time the application is initialized to the time the data is displayed to the user. Moving from one record to another, updating especially the detailed information takes an average of 300 milliseconds. You can also see the amount of time it takes to move from the first record to the last record, which is about the same as moving to the next record, thanks to the fact that a DataSet operates entirely in memory. Measuring time to move from the first record to the last record in the database is achieved by the code as shown in Listing 2.

private void menuGotoLast\_Click(object sender, EventArgs e)

{

ordersBindingSource.MoveFirst();

sw.Reset();

sw.Start();

ordersBindingSource.MoveLast();

sw.Stop();

statusBar1.Text = "Goto last row: " + sw.ElapsedMilliseconds + " msec.";

}

Listing - Measuring time to move from the first to the last record.

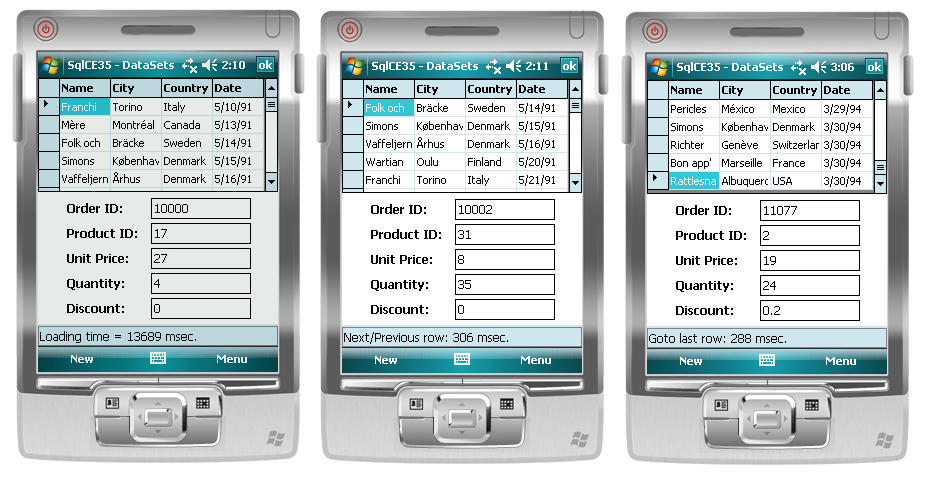


Figure - Initial time it takes to display data and moving from record to record

Because of the fact that a DataSet keeps all data locally in memory itself, at some time it is necessary to commit changes back to the database. It is of course up to you to determine when you want to commit changes. One approach could be to work locally on the DataSet as long as possible, and commit changes to the database once the user closes the application. However, especially when you need to synchronize data with a back-end server it is probably better to commit changes to the database just prior to synchronizing. Also, with all data kept locally in a data cache like a DataSet, if the application crashes it means that data might be lost. In a typical application you might want to commit changes back to the database at several moments in time, for instance when the application is closed, when data needs to be synchronized and when the application goes to the background. You can also decide to commit changes to the database immediately once they are made, but that makes the DataSet almost like a really redundant local data cache. Listing 3 shows the code you need to provide to commit changes to the database. In this sample, data is written back to the database inside the **Closing** event handler. Only if changes are found in the DataSet it is necessary to commit changes back to the database.

Listing - Committing changes back to a database on application termination.

private void MainForm\_Closing(object sender, CancelEventArgs e)

{

if (northwindDataSet.HasChanges())

{

ordersTableAdapter.Update(northwindDataSet);

order\_DetailsTableAdapter.Update(northwindDataSet);

}

}

When you are using a strongly typed DataSet, lots of code is generated for you and once the DataSet is initialized, retrieving data is very fast. Basically, the only code you have to provide is code to commit changes back to the database. Additionally, you might want to add code to navigate through the data, and provide a non-default way to fill the DataSet, maybe filtering the data you are interested in. In this introduction in using DataSets we omitted some more advanced topics like delay-loading data from the database to the DataSet.

**Using Strongly Typed SqlCeResultSets**

Making use of strongly typed **SqlCeResultSets** gives you a relatively simple way to access a SQL CE database. Some code is generated for you, although not as much as you saw when using a strongly typed DataSet. One of the big advantages of a SqlCeResultSet is that it operates directly on the database. In other words, data is not duplicated which is great on a memory scarce device like a Windows Mobile device. A **strongly typed SqlCeResultSet** combines the ease of use of a **DataSet** with the performance of a **DataReader**. To compare the performance between a SqlCeResultSet and a DataSet we are going to use a similar application, working on the same tables of the Northwind database and providing the same functionality. Performance is again measured through a **Stopwatch** object. In figure 4 you can see in the Properties Window that we specified a **custom tool** to generate a strongly typed SqlCeResultSet instead of a DataSet. The initial approach is identical. You add a DataSource to your project, being the Northwind database. When adding the DataSource, a strongly typed DataSet is automatically created for you by Visual Studio 2008. After that you change the custom tool in the Properties Window to generate SqlCeResultSets. What you also can see in figure 4 is that Visual Studio created two strongly typed SqlCeResultSets for you, but that there is no relation between the two. In order to have a master-detail view on the MainForm you must provide your own code to bind the detail view to the master view.

The code that Visual Studio generated for you to be able to display data on the MainForm is shown in listing 1. Note that only the data grid, containing Orders data, is populated automatically by Visual Studio 2008. Also, no additional forms are generated to display summary information or an edit dialog.

Listing - Auto-generated code to initialize UI Controls with data from a SqlCeResultSet.

public partial class MainForm : Form

{

private SqlCe35WithResultSets.NorthwindResultSets.OrdersResultSet ordersResultSet;

public MainForm()

{

InitializeComponent();

}

private void MainForm\_Load(object sender, EventArgs e)

{

ordersResultSet = new SqlCe35WithResultSets.NorthwindResultSets.OrdersResultSet();

ordersResultSet.Bind(this.ordersResultSetBindingSource);

}

}

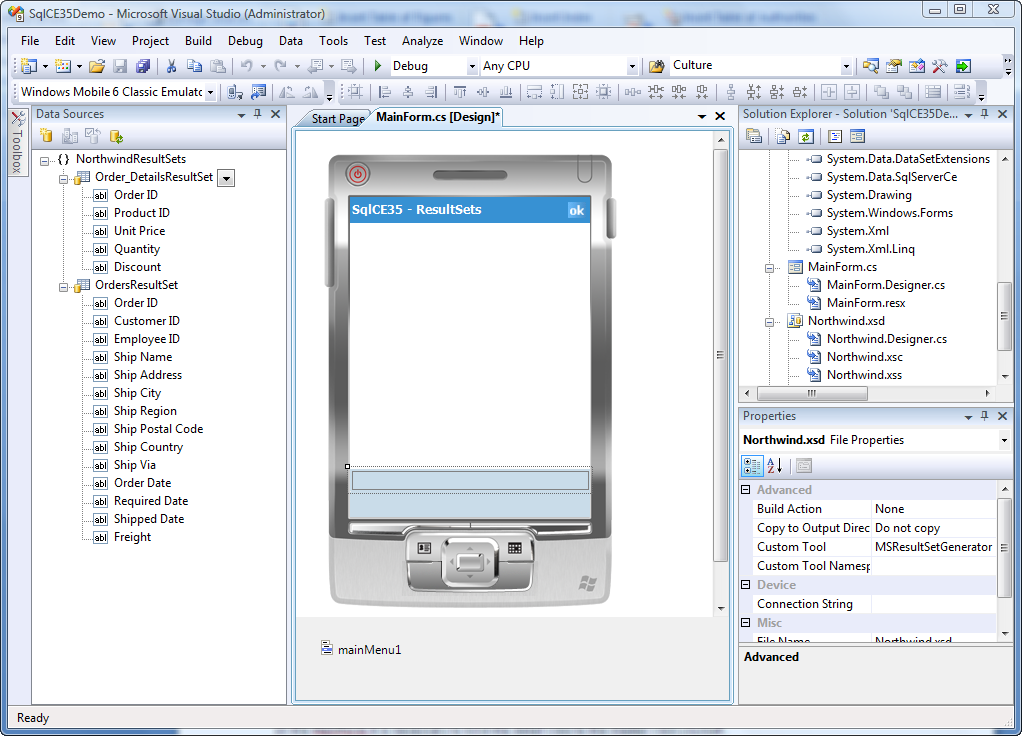


Figure - Strongly Typed SqlCeResultSets

To display information of the **Order\_Details** table in combination with the record that is selected from the Orders table, you have to provide some code yourself. A nice approach to write this code is to start extending the functionality of the **Order\_DetailsResultSet**. You can do that by right clicking on the Northwind.xsd file in Solution Explorer. Visual Studio opens a new file **Northwind.cs** for you, in which you can extend the functionality of the Order\_DetailsResultSet thanks to the fact that the designer generated code is implemented as a partial class. In listing 5 you see the additional functionality that is added to the Order\_DetailsResultSet. A new constructor is created with a Boolean flag, indicating if the Order Details table needs to be opened or not. If you call this constructor, indicating that you don’t want to open the table, you can call an additional Open method, which is also shown in listing 5, to run a query against the Order Details table. In this way, you create an Order\_DetailsResultSet that only contains records that match the query. In this whitepaper we are not optimizing the database at all by providing additional indexes to have a real comparison between the performance of a DataSet and a SqlCeResultSet. However, adding indexes might even give you better performance.

public partial class Order\_DetailsResultSet

{

public Order\_DetailsResultSet(bool openTable)

{

// Create default options

//

resultSetOptions = System.Data.SqlServerCe.ResultSetOptions.Scrollable |

System.Data.SqlServerCe.ResultSetOptions.Sensitive |

System.Data.SqlServerCe.ResultSetOptions.Updatable;

if (NorthwindUtil.DesignerUtil.IsDesignTime())

{

// Designtime Connection String

resultSetConnectionString =

"Data Source=C:\\Users\\Maarten\\Documents\\Visual Studio 2008\\Projects\\SqlCE35Demo\\Sq" +

"lCE35WithDataSets\\Northwind.sdf";

}

else

{

// Runtime Connection String

resultSetConnectionString = ("Data Source =" +

(System.IO.Path.GetDirectoryName(

System.Reflection.Assembly.GetExecutingAssembly().GetName().CodeBase) +

"\\Northwind.sdf;"));

}

if (openTable)

this.Open();

}

public void Open(string query)

{

System.Data.SqlServerCe.SqlCeCommand sqlCeSelectCommand = null;

try

{

// Open a connection to the database

//

sqlCeConnection = new

System.Data.SqlServerCe.SqlCeConnection(this.resultSetConnectionString);

sqlCeConnection.Open();

// Create the command

//

sqlCeSelectCommand = sqlCeConnection.CreateCommand();

sqlCeSelectCommand.CommandText = query;

sqlCeSelectCommand.CommandType = System.Data.CommandType.Text;

// Generate the ResultSet

//

sqlCeSelectCommand.ExecuteResultSet(

System.Data.SqlServerCe.ResultSetOptions.Scrollable, this);

}

finally

{

if ((sqlCeSelectCommand != null))

{

sqlCeSelectCommand.Dispose();

}

}

}

}

Listing - Extending the Order\_DetailsResultSet

After extending the Order\_DetailsResultSet you have to do some additional work. The first thing to do is make sure that you will bind the User Interface controls that you are using for the detailed view to a new **BindingSource**. When dragging and dropping Order\_DetailsResultSet controls, they try to bind to the binding source that was generated when dragging and dropping an OrderResultSet data grid to the form as is illustrated in Figure 5. This works fine for the Order ID field, since this is a key value, used in both tables. However, binding other members of the Order\_DetailsResultSet to the **ordersResultSetBindingSource** leads to exceptions, since those members are not defined for this data source.

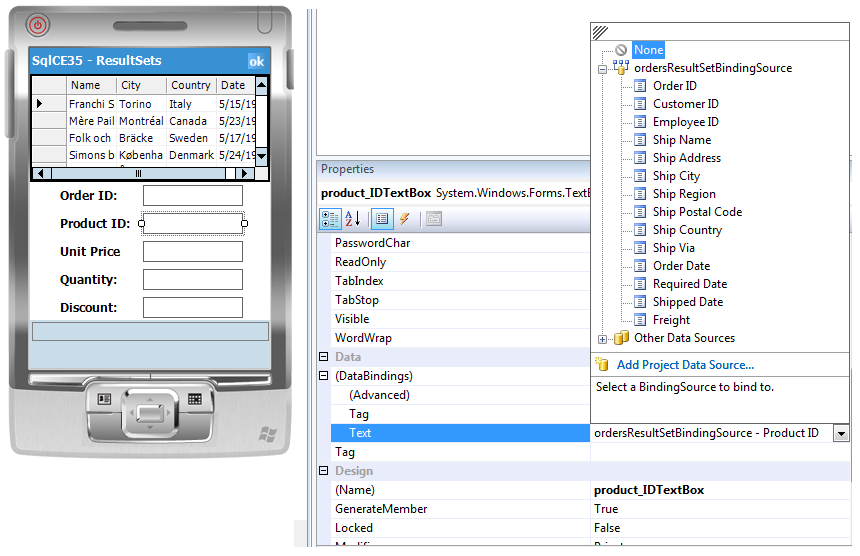


Figure - Binding the Product ID text box to the wrong data source.

The easiest way to solve this issue is to add a new **binding source** to the solution, and setting its **DataSource** property to the Order\_DetailsResultSet as you can see in Figure 6. After having done this, dragging and dropping User Interface controls from the Order\_DetailsResultSet to the MainForm results in Visual Studio asking you which binding source you want to use. You can now specify that you want to use the just added binding source, which fixes the data binding problem we just described. The next thing you need to do is provide code to show detail information for the currently selected record inside the data grid. This can be done by adding some additional code to the MainForm, making use of the extensions to the Order\_DetailsResultSet that you saw in listing 5.

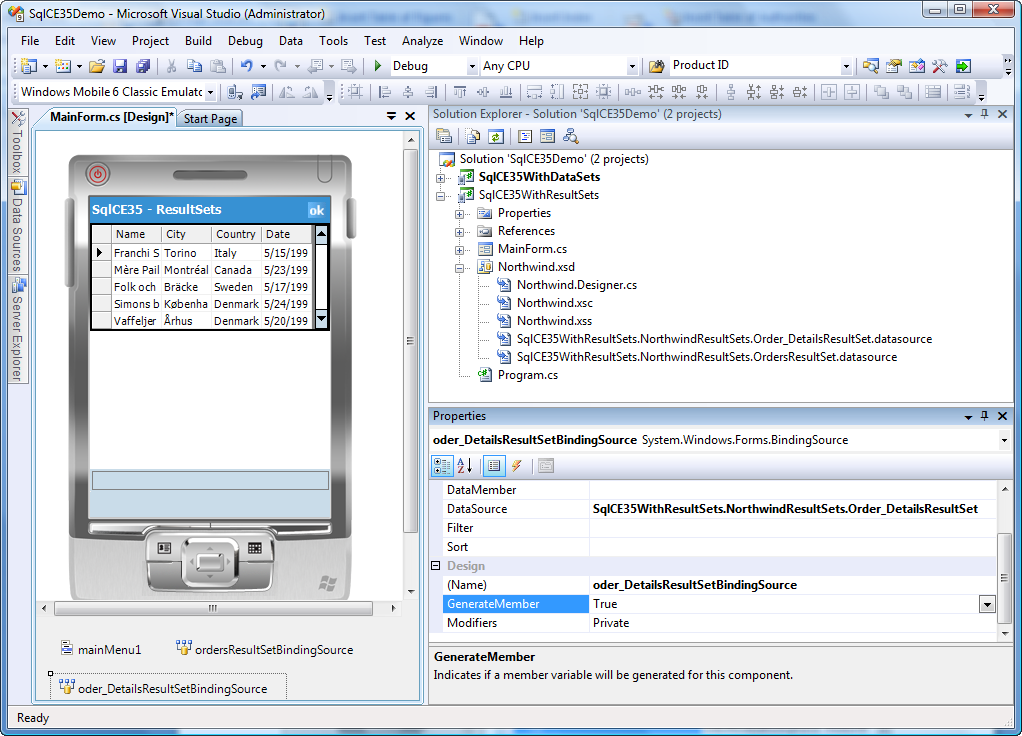


Figure - Manually adding a binding source and setting it to the Order\_DetailsResultSet.

Each time the user selects a new record in the DataGrid that is shown in Figure 6, you need to find out what the Order ID is of the currently selected record. Using that Order ID you can then build a query to retrieve order details from the Order Details table. Once a new Order\_DetailsResultSet is created, containing only the record that match the query, the next thing to do is binding the BindingSource to the newly created result set, after which you must dispose the original result set. Listing 6 shows you a method that connects the Orders table to the Orders Detail table through code. This code can be called each time the current record in the data grid changes.

private void ordersResultSetBindingSource\_PositionChanged(object sender, EventArgs e)

{

SqlCE35WithResultSets.NorthwindResultSets.Order\_DetailsResultSet orgDetailsRS =

order\_DetailsResultSet;

GetOrderDetails();

if (orgDetailsRS != null)

orgDetailsRS.Dispose();

}

private void GetOrderDetails()

{

int orderID =

(int)((RowView)this.ordersResultSetBindingSource.Current).UpdatableRecord["Order ID"];

string query = "SELECT \* FROM [Order Details] WHERE [Order ID] = '" + orderID +"'";

order\_DetailsResultSet =

new SqlCE35WithResultSets.NorthwindResultSets.Order\_DetailsResultSet(false);

order\_DetailsResultSet.Open(query);

order\_DetailsResultSet.Bind(order\_DetailsResultSetBindingSource);

}

Listing - Connecting the details view to the master view

# When the user selects a new record, the **PositionChanged** event on the ordersResultSetBindingSource is fired. This event can be used to retrieve the currently selected Order ID. The currently active order\_DetailsResultSet will be disposed and a new one will be created using the Order ID to setup a query to return only Order Details for matching Order ID’s.

# When you read about the DataSet, you also noticed that the designer can create Summary and Edit forms for you. This functionality is not available when Visual Studio creates SqlCeResultSets for you. If you want that functionality you have to create your own forms and provide your own code to populate those forms. In this article, this functionality is omitted for the SqlCeResultSet since we are only focusing on performance during data retrieval. On the other hand, if you are making changes, you don’t have to explicitly commit them back to the database, as was the case when using a DataSet. Since a SqlCeResultSet immediately operates on the database, changes will be immediately committed for you.

# To get some information about the performance of retrieving data from a SQL CE Database through a SqlCeResultSet, you can start a Stopwatch as the first statement in the MainForm\_Load event handler and stop the Stopwatch as the last statement in the MainForm\_Load event handler, after which you can display the elapsed time in a status bar. A similar performance measurement can be done to measure the time it takes to display detailed information when moving from one record to another by taking the time the user presses a hardware key until the time the text of one of the edit controls that is displayed on the MainForm is changed. This functionality is similar to the DataSet. Figure 7 shows you these performance results when using a strongly typed SqlCeResultSet. As you can see, it takes around 300 milliseconds on Device Emulator from the time the application is initialized to the time the data is displayed to the user. Moving from one record to another, updating especially the detailed information takes an average of 50 milliseconds. You can also see the amount of time it takes to move from the first record to the last record, which is a little longer, since more records have to be evaluated in order to move from the first to the last record.



Figure - Initial time it takes to display data and moving from record to record

# **Conclusion**

# Making use of strongly typed DataSets in your application is very easy to do. You hardly have to provide any code, since most code can be generated for you by Visual Studio 2008. Especially in the case of a master – detail presentation, this is very convenient. However, since all data from the database is duplicated in memory, it will take more time to initially display your form, and you will use up precious memory on your Windows Mobile device.

# The SqlCeResultSet gives you much better performance, especially when initially loading your form. The reason for that is that the SqlCeResultSet works directly on the database and only needs to retrieve the records that are visible in the datagrid. However, to implement a master – view presentation you have to provide more code yourself. The speed of the SqlCeResultSet can even be more improved by making use of index files on the database. If you need Edit / Summary screens, you have to provide them yourself for a SqlCeResultSet. However, the improved performance is definitely worth investing in a bit of code to get that functionality available inside your own application.

# **Additional resources and references**

* [Window Mobile Developer Center](http://msdn.microsoft.com/en-us/windowsmobile/default.aspx)
* [Blog Entry: Getting Started with the SqlCeResultSet](http://blogs.msdn.com/sriram/archive/2005/11/12/492070.aspx)
* [SQL Server Compact 3.5 Tutorials](http://msdn.microsoft.com/en-us/sqlserver/bb895908.aspx)
* [Deployment, Setup, Security and You](http://msdn.microsoft.com/en-us/library/bb158571.aspx)
* [Video: How do I Improve Data Performance Using Visual Studio 2005](http://msdn.microsoft.com/en-us/windowsmobile/bb496523.aspx)