ORACLE DATA PROVIDER TUTORIAL

GETTING THE BEST FROM ORACLE WITH C# IN VS2005

CONTENTS

1. Introduction

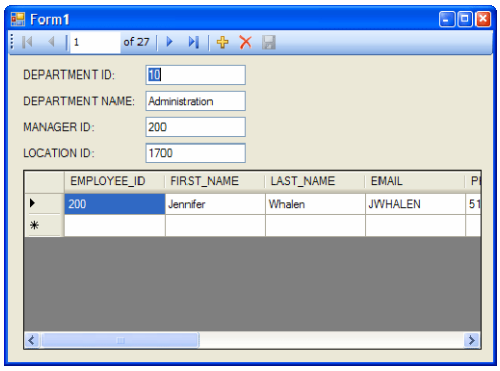
Visual Studio 2005 provides tight integration through ADO.Net, to SQL Server databases. Facilities such as drag-and-drop creation of forms with connections to the database are possible, as is the creation of strongly-typed DataSets through drag-and-drop.

Those who wish to connect to Oracle, however, have slightly less convenience (though hopefully some of the current short-comings will be ironed out in an upcoming beta release of their Oracle Data Provider.

This document has a lot of demonstrative screen-shots, so don't panic that it's 14 pages!

1.1 What You Will Achieve

Here's a screen-shot of what is achieved by the end of this tutorial. The form will be read-only (you can change data on it, but not save those changes back to the database).



We will, however, have created a means of getting data from the Oracle DB that allows us to use the latest data-bound components on top of them, with strong typing.

1.2 Technologies and Prerequisites

This document concentrates on the following technologies:

- Visual Studio 2005 SP1 (.Net framework v2) using C#;
- Oracle 10g2R2 Database server (running on Windows);
- Oracle Data Access Components 10.2.0.2.20, including Oracle Data Provider;
- If you want to use .Net Stored Procedures (outside the scope of this document) you will have had to do a manual install of the database, being careful to include the Oracle Database Extensions for .Net (ODE). Then you need to install ODAC and the included ODE update over that. Oh… and don't forget needing the .Net Framework on the server too.

You will need to have all of the software mentioned previously installed to work through this tutorial. In addition you will need to have access to the human resources demo database; typically accessed through user hr.

You will also need to be able to access the database from your development machine; you may need to set up a service name through Oracle Net Manager or Net Configuration Assistant software that was installed with ODAC.

1.3 Abbreviations Used

The following abbreviations are used in this document:

ODP	Oracle Data Provider; extensions to .Net Framework for Oracle DB access
ODT	Oracle Developer Tools; the integration with Visual Studio 2003 or 2005
ODE	Oracle Database Extensions for .Net; software required on the server to run .Net Stored Procedures (not required in this tutorial). Definitely worth knowing once you start getting into Oracle
ODAC	Oracle Data Access Components; required on the client / development machine

IDE	Integrated Development Environment – specifically Visual Studio 2005 as far as this document is concerned
DAL	Data Access Layer – the class / project that talks to the database
GUI	Graphical User Interface – the project that presents information to the user

1.4 Useful Links

ODT / ODAC Software	www.oracle.com/technology/tech/dotnet/tools/index.html
Mastering .Net with Oracle Tutorials	www.oracle.com/technology/pub/articles/mastering_dotnet_oracle/index.html (especially the Ref Cursors article, which substantially contributed to my putting some pieces of the puzzle together)
ODT Forum	forums.oracle.com/forums/forum.jspa?forumID=228
ODP.Net Forum	forums.oracle.com/forums/forum.jspa?forumID=146
My own article about solving ODAC installation issues	www.kebabshopblues.co.uk/2007/04/08/the-solution-for-oracle-odac-installation-errors/
Basic Oracle Tutorial	www.oracle.com/technology/pub/articles/cook_dotnet.html?_template=/ocom/print
A book on Data Binding	Data Binding with Windows Forms 2.0 by Brian Noyes Certainly this book is not about Oracle, but it did play some part in my being able to create this tutorial

1.5 Three Tier Architecture

The application will use a layered structure, separating out the code that handles the database from the presentation layer. I'm told that some people believe this overcomplicates Oracle projects, but as far as I can see it does make some sense in the long run; even if it is a little bit more work for a small tutorial like this.

Once aspect I like about a tiered architecture is that it separates out the different development roles a little bit. When we're coding our data access layer, we certainly will need to be thinking about SQL, column names, connection strings and so on. But once we have our dataset loaded, we can think about classes, objects, inheritance and all that, and hopefully, we won't have chunks of SQL in our GUI code. Maybe we'll even have time to give a bit of thought to usability too.

Of course, in a tutorial like this, done by one person it all seems a bit of a mess sometimes as we seem to get nowhere for the first little while!

2. Data Access Layer

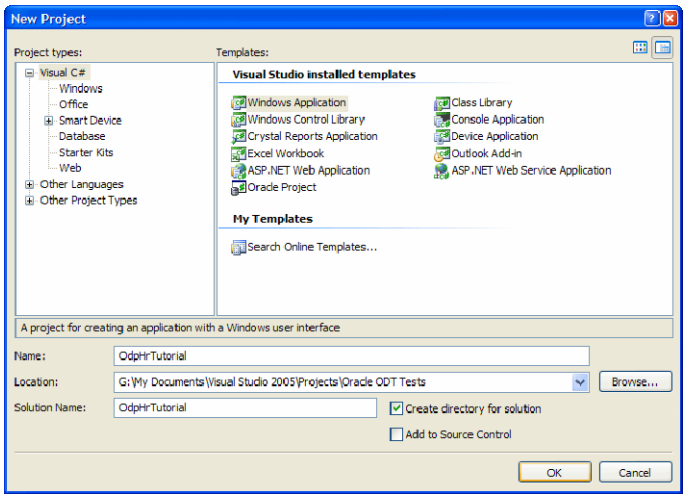
Our simple project is going to access the employees and departments tables of the human resources sample Oracle database.

We would like to use Visual Studio strongly-typed datasets so that our data is more safely handled in the code. We're going to have to do some hacks to get that, though.

2.1 Create Two Project Types

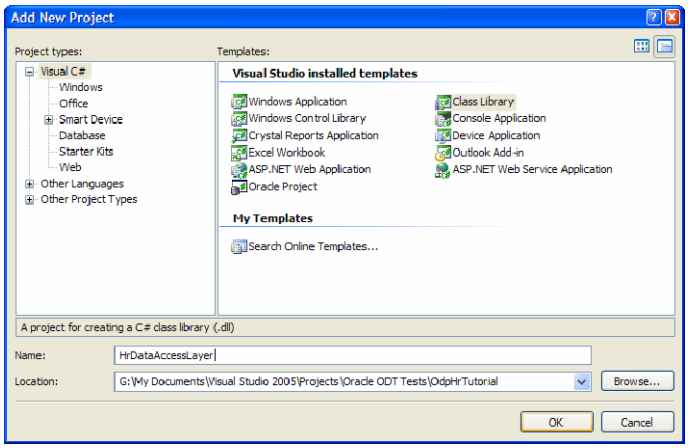
In Visual Studio 2005, create a new project by:

- Select the File | New | Project;
- Select the Windows Application template under the Visual C# node, and call it OdpHrTutorial; this will become the GUI part of our project, though in the short term we need it to do some hacks.

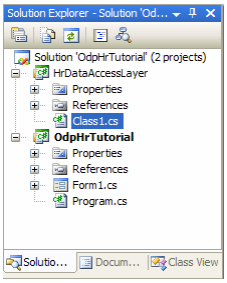


We're now going to add a new class project to this solution:

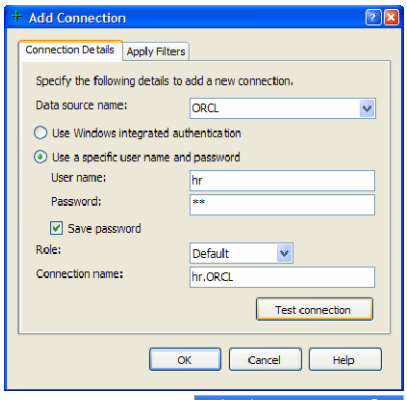
- In the Solution Explorer window (View | Solution Explorer menu if it's not open), select the main *solution*, right-click and select Add | New Project...
- This time, select the Class Library template, and call it HrDataAccessLayer;



The Solution Explorer will now look something like this:



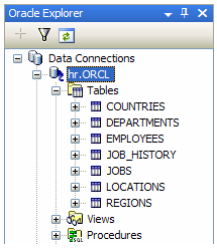
2.2 Adding an Oracle Database Connection



Open the Oracle Explorer window with the View | Oracle Explorer menu.

Create a connection to the hr schema, if you don't already have one, by:

- Right-click the Data Connections root node in the Oracle Explorer window, and selecting Add Connection...[1]
- Complete the form, following guidelines in the following text.



The data source name should be the name you set up as a Service in Oracle Net Configuration Assistant or Net Manager. In my case, I set up the service with the name ORCL, which is the same name as the database instance on my server.

You need the connection to access the hr schema, and the hr user is probably best for that (most usually that has a password of hr… however you will need to be sure the account has been unlocked and the sample schemas installed along with the sample database).

You should substitute the values appropriate for your environment.

After creating and testing the connection you should notice a new node under the Data Connections root of the explorer. I've opened the Tables node to see that we have now connected to the schema, and ODT is showing us the information in a nice format comparable with the other tools in VS.

In addition, your DB Connection will be remembered by Visual Studio for any future projects that you do against the same schema.

2.3 Hacking a Typed Dataset

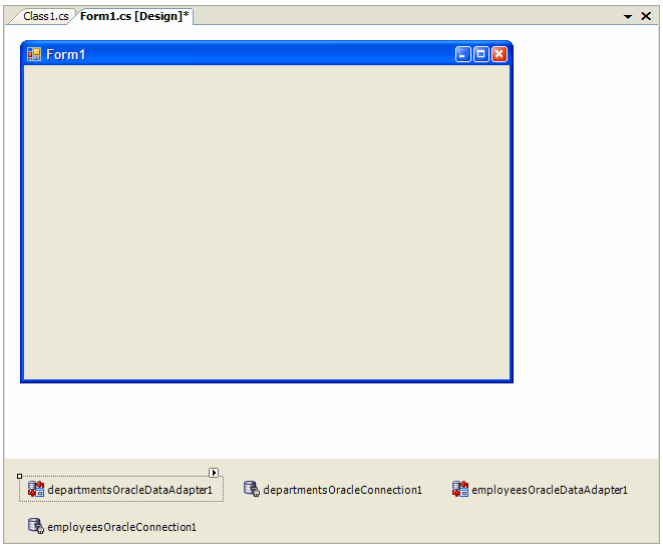
2.3.1 Getting Oracle to Generate Individual Data Tables

Now we're going to use the tools that are available to help build our own typed dataset:

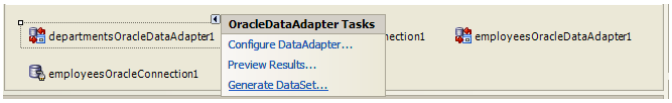
- Double-click the Form1 form from your OdpHrTutorial GUI project, so that you are looking at the Forms designer;
- In the Oracle Explorer window, click and drag the Departments node onto the form. You may need to respond to a dialog about storing the password in plain-text – click No to avoid storing it.
  (Nothing will have changed on the main form canvas, but a departmentsOracleDataAdapter1, and departmentsOracleConnection1 will have been created underneath the forms designer);
- Repeat the process by dragging the Employees table node onto the form.

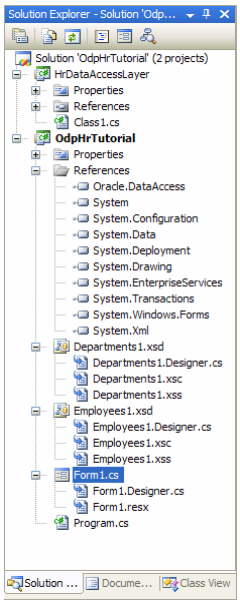
At this point, the forms designer should look a bit like the following image:

[1] Alternatively, select the Tools | Connect to Oracle Database... menu to get to the Add Connection form.



- Select the departmentsOracleDataAdapter1 component, and then click the little arrow at the top-right of the component.
- Click Generate Dataset from the drop-down menu;
- *Repeat* the process for the employeesOracleDataAdapter1.





At this stage, Solution Explorer will show two new dataset files as part of the OdpHrTutorial project (Departments1.xsd and Employees1.xsd).

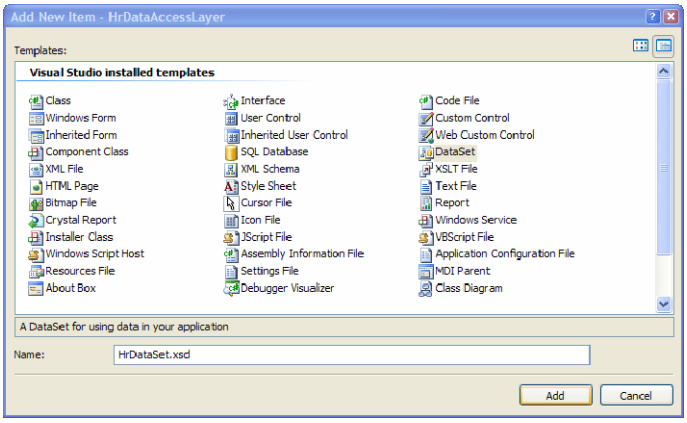
We don't actually want these xsd files here in the long run… we're still just working-round the limitation of not having drag-and-drop dataset design in ODT as yet.

You may also note that the References node under the OdpHrTutorial project has opened-up, and there are several references there that ODT added when we dragged the tables onto the form. Additionally, two new dataset items have been added to the components under the forms designer.

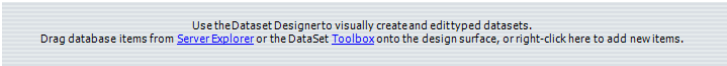
What we have done up to this point is utilise the one place where Oracle will do some code generation of a typed dataset for us. These typed datasets are very much separate… but don't panic, we'll get that sorted out next.

2.3.2 Building our Own Typed DataSet

- In Solution Explorer, right-click the HrDataAccessLayer project, and select Add | New Item;
- The Add New Item dialog opens. Select the DataSet template, and call the file HrDataSet.xsd. Close the dialog by clicking Add.



You'll now have a blank dataset designer in view, with a message about dragging tables into it. You can't do this with Oracle – yet (that's why we've been through this whole work around)!



What we are going to do now is to copy the DataTable elements that Oracle created for us, and paste them into our new dataset:

- Double-click the departments1.xsd file in the GUI project;
- Select the table with a mouse-click, and Ctrl+C to copy it;
- Open your new HrDataSet designer, and paste the table into place;
- *Repeat* for the Employees table in the employees1 dataset.

After carrying out these actions, your new dataset will look something like the first image in 'Figure 1 - The DataSet before and after adding a Relation' below.

But there's something missing - a relationship between the two tables (or perhaps more accurately, there is a missing relation between the two classes derived from `System.Data.DataTable`).

From now on in this tutorial, I will try to refer to these objects as 'data tables' to help remind you we are not talking about the actual tables in the database.

Sidebar

Just a quick reminder, we're working on a DataSet here, and we are manipulating classes that will store data from database tables for us later on.

We are *not* making any changes to the database tables that they represent.

These classes will be local stores of data that we can manipulate; but when we do so we are only changing the local copies of the data held in these stores.

The data table classes track additions, deletions and changes made to the rows of data they contain, so that sensible updates can be passed back to the DB when we are ready to do that.

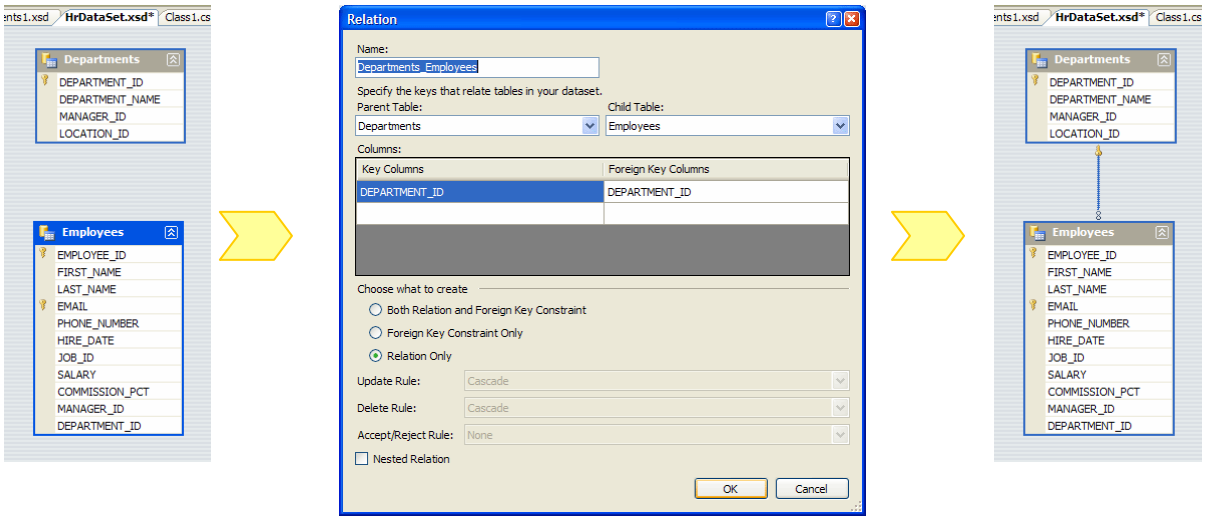


Figure 1 - The DataSet before and after adding a Relation

2.3.3 Adding a Relationship

Add a relationship by:

- Right-click the designer surface (not one of the tables) and select Add | Relation… from the menu;
- The IDE will display a dialog that should correctly populate all the fields for the relation we want. That is, the parent table is Departments and the parent key is DEPARTMENT_ID, and the child table is Employees with a key of DEPARTMENT_ID too;
- Click OK to accept the settings for the new relationship (see 'Figure 1 - The DataSet before and after adding a Relation').

One thing you may have noticed was 'missing' from our HrDataSet was the TableAdapters that a similar project using MS data providers would have created for us from click-and-drag operations. For SQL Server, table adapters's control the flow of data between dataset and database table (at least to some degree), and as I understand it they are strongly typed. We're going to have to manually create *data* adapters, but that's yet to come.

2.3.4 Tidying Up the Working Files

Now for a bit of tidying-up:

- Delete Departments1.xsd from your GUI project;
- Delete Employees1.xsd from the GUI project too;
- Open Form1 in design view and delete the components that were added when we dragged the tables to the form; we've finished with them too.

2.4 Using Stored Procedures to Retrieve Data

For the purposes of this tutorial, I decided to use Oracle PL/SQL Ref Cursors to get the data from the two tables. This was pretty much because I wanted to try them out instead of a direct SQL select command or a stored procedure with a basic select.

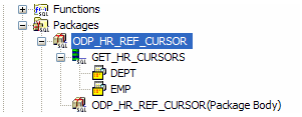
2.4.1 Creating the Package and Stored Procedure

I'm not going to cover creating the stored procedure here, though I did actually use the ODT to help me do it through Visual Studio… though some manual coding was necessary.

You can either attempt to create the package and procedure through the IDE, or just run the SQL that will be supplied with the sample project download as user hr, using SQL*Plus or similar. The SQL filename is OdpHrTutorial_Package.sql.

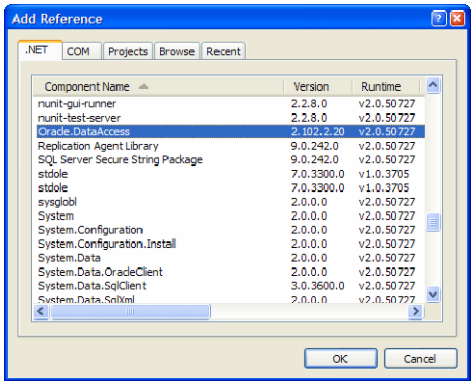
```
PACKAGE "HR"."ODP_HR_REF_CURSOR" IS

  -- Declare strongly typed Ref-Cursors, based on tables
  TYPE DEPT_CUR_TYPE IS REF CURSOR RETURN departments%ROWTYPE;
  TYPE EMP_CUR_TYPE IS REF CURSOR RETURN employees%ROWTYPE;

  PROCEDURE "GET_HR_CURSORS" (
    "DEPT" OUT DEPT_CUR_TYPE,
    "EMP" OUT EMP_CUR_TYPE);

END "ODP_HR_REF_CURSOR";

PACKAGE BODY "HR"."ODP_HR_REF_CURSOR" IS

  -- Package body for odpHrTutorial by Nigel Rheam
  PROCEDURE "GET_HR_CURSORS" (
    "DEPT" OUT DEPT_CUR_TYPE,
    "EMP" OUT EMP_CUR_TYPE) IS

    BEGIN
      -- Simply open the two ref cursor parameters as select statements
      OPEN dept FOR
        SELECT *
        FROM departments
        ORDER BY department_id;

      OPEN emp FOR
        SELECT *
        FROM employees
        ORDER BY last_name, first_name;

    END "GET_HR_CURSORS";

END "ODP_HR_REF_CURSOR";
```

After successfully creating the package, you should be able to expand the Packages node in Oracle Explorer, and see something like the image, right.



2.4.2 Using the Stored Procedure in the Data Access Layer

Before we can do any constructive coding in our DAL, we need to add a reference to the Oracle.DataAccess library:



- In the Server Explorer window, right-click the References node under the HrDataAccessLayer project, and select Add Reference…
- Select the Oracle.DataAccess library as shown in the image, and click OK;
- The Oracle.DataAccess entry will appear under the References node.

Let's rename the Class.cs file in the HrDataAccessLayer project:

- Right click Class1.cs in Server Explorer, and select Rename;
- Change the file name to HrDAL.cs, press Return, and accept the dialog suggestion to rename references to Class1;
- Open HrDAL.cs to view the code.

Let's now add a few `using` commands to the file to make our life easier with the coding.

- Paste the following three lines under the existing `using` commands in the file:

```
using System.Data;                // Needed for various DB related types
using Oracle.DataAccess.Client;   // Needed for most / all Oracle related projects
using Oracle.DataAccess.Types;    // Needed for types to run Stored Procedures
```

2.4.3 Coding the DAL

If you are working through the exercise, paste the following method into your HrDAL.cs file; it should go between the braces that come after the `class HrDAL` line.

```
/// <summary>
/// Fill tables in an HrDataSet
/// </summary>
/// <param name="dataset">The dataset instance to populate</param>
public static void FillWithRefCursors(HrDataSet dataset)
{
    // TODO: Change the connection string to suit your installation
    OracleConnection connection =
        new OracleConnection("User Id=hr;Password=hr;Data Source=ORCL;");
    connection.Open();

    // Create the command object and set the command to be package.proc_name
    OracleCommand cmd = new OracleCommand("odp_hr_ref_cursor.get_hr_cursors",
                                          connection);
    cmd.CommandType = CommandType.StoredProcedure;

    // Init. parameters for the procedure, which are our RefCursors
    // First parameter is the name of the parameter in the proc (I think)
    OracleParameter deptRefCursor =
        new OracleParameter("DEPT",
                            OracleDbType.RefCursor,
                            ParameterDirection.Output);
    OracleParameter empRefCursor =
        new OracleParameter("EMP",
                            OracleDbType.RefCursor,
                            ParameterDirection.Output);

    // Add the parameter to the Parameters collection
    cmd.Parameters.Add(deptRefCursor);
    cmd.Parameters.Add(empRefCursor);

    // Execute the command (which does not retrieve rows,
    //  hence ExecuteNonQuery)
    cmd.ExecuteNonQuery();

    // At this point after the query, our parameter variables
    //  should be pointing to REF CURSOR types on the DB Server

    // Create a data adapter to use with the data set
    OracleDataAdapter dataAdapter = new OracleDataAdapter();

    // Fill the tables in the dataset using the ref cursors.
    //  Value property returns an object, so needs to be cast to
    //  OracleRefCursor class.
    // Note strong typing on the dataset means we can refer
    //  to tables as datasetInstance.TableName
    dataAdapter.Fill(dataset.Departments,
                     (OracleRefCursor)deptRefCursor.Value);
    dataAdapter.Fill(dataset.Employees,
                     (OracleRefCursor)empRefCursor.Value);

    dataAdapter.Dispose();
    deptRefCursor.Dispose();
    empRefCursor.Dispose();
    cmd.Dispose();
    connection.Dispose();

}
```

Here's a quick run-down of the code (though refer to the Ref Cursors article linked to at the start of this document for more information):

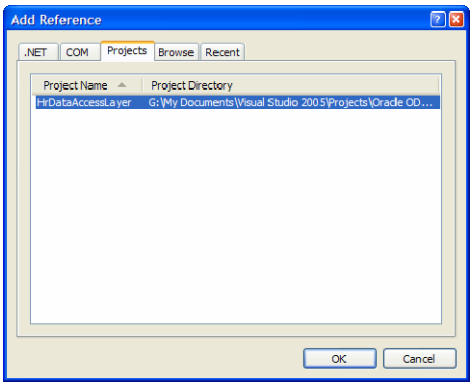
- The method takes a dataset of type `HrDataSet` as a parameter (this is the typed dataset we created in section 'Building our Own Typed DataSet' on page 6). As you'll see later on, the form will already have a dataset initialised the way we're doing this, so using a parameter will come in handy;
- We open an `OracleConnection` using a hard-coded connection string. You will need to change this to suit your environment. Plus, of course I should put the obligatory warning about putting passwords in code, but the approach is suitable for this tutorial;
- We create an `OracleCommand` where the strong command is just the package name (that we created on page 8) followed by a dot and then the procedure name. We also have to set a property on the parameter to let the command know it's a stored procedure;
- We now set up information about the two parameters. I picked the easy constructor which allowed me to set the `OracleDbType.RefCursor` parameter type (this is very important!) and the parameter direction – both of them are `OUT` parameters if you remember the PL/SQL code (page 8);
- The parameter objects are then added to the command, and the command is executed with the `ExecuteNonQuery()` method;
- We then instantiate an `OracleDataAdapter` to control the population of the dataset, and fill each table individually by type, with the appropriate ref cursor parameter. Behind the scenes, the `Fill()` method obviously takes the cursor and scans through it to fill the data table;
- OK, so we don't have any error handling in this code so far, but we probably should still dispose of the relevant objects ☺

We've finished the DAL code for now.

Save your work, have a cup of coffee – we're about to sort out the GUI, but it's relatively easy-going on from now on, because we've done almost all the coding we need to do for our simple application, and most steps from now are click-and-drag in the IDE.

3. Getting Going with the GUI

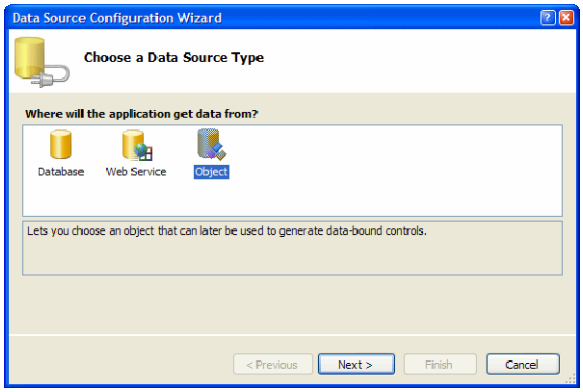
We're going to start using `Form1.cs` in the GUI project again (OdpHrTutorial project). Before we do that, however, let's add a reference to the DAL project in our GUI project (OdpHRTutorial):



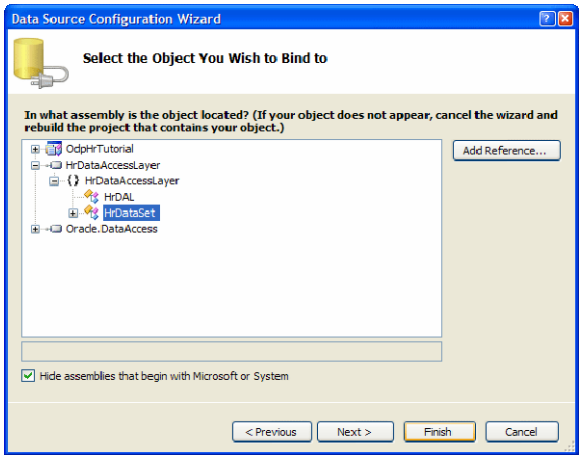
- Right-click the References node under the GUI project and select Add Reference...;
- Select the Projects tab, where you should see our HrDataAccessLayer project DLL. Select that and OK the dialog.

3.1 Creating a Data Source

Now, we want to create a new data source:

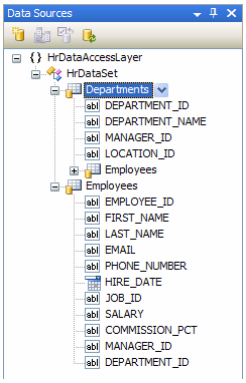


- Select the Data | Add New DataSource menu;
- In the Data Source Configuration Wizard dialog, select the Object source;
- …and click Next.



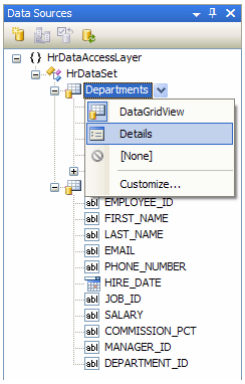
- Select the HrDataSet item within the HrDataAccessLayer assembly and namespace. (You won't see this here if you forgot to carry out the step of adding a reference to the DAL in the GUI project);
- Click Finish to complete the wizard.

The Data Sources window will open (if not, view it with the Data | Show Data Sources menu). After expanding a few nodes, it will look like the image shown right.



Bear in mind that we are now looking at a tree representation of the `DataTable` classes we created earlier (see page 6) in our new typed `DataSet`, which in turn represents the tables in the hr schema.

Note that there are *two* Employees data tables listed; one of them appears inside the Departments table. This child table is created automatically for us as a result of the relationship that we created in section 'Adding a Relationship' on page 8.

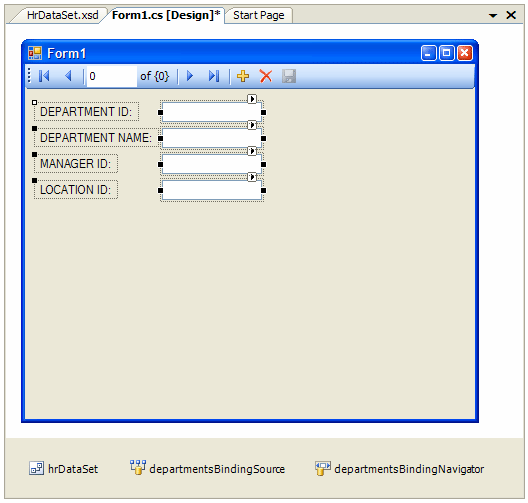


The icons next to each node on the tree are significant, and represent how Visual Studio will display each node by default. The icons next to each table represent a `DataGridView`, and most of the rest of the fields represent text edit boxes, with the exception of the `Hire_Date` column in the employees table.

For the purposes of the tutorial, we'll implement a standard Master-Detail type view on a form, and to do this, we need to change the way that the Department will be represented:

- Click on the drop-down list on the table name, and select Details from the drop-down list;
- Note how the icon changes after doing this.

3.2 Designing the Form



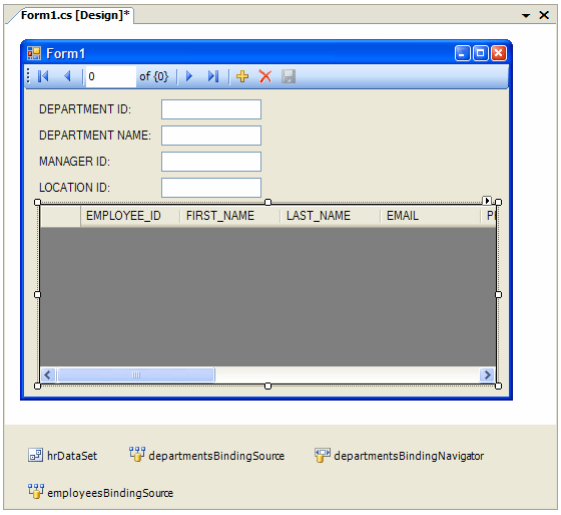
Now:

- Click and drag the Departments data table node onto your empty Form1. Release the mouse, after which several labels and text boxes will have been added to the form;
- While you have the controls selected as a group, move them to the top-left of the form.

Note the additions to the components below the form; the hrDataSet is an instance of our dataset from the DAL. The departmentsBindingSource and departmentsBindingSourceNavigator are new to us.

Very briefly, a binding source sits between form controls and the dataset. If you inspect any of the text boxes we just added, you will see that they have been bound to the binding source. The binding source, in turn, is connected to the dataset and the specific data table within the dataset.

The departmentsBindingSourceNavigator is the new data navigation component that was automatically added to the top of the form. It too is bound to the binding source, and it interfaces with methods and events there to do its job of tracking and allowing navigation through the data.



We'll now add the employees data table to the form:

- Drag the Employees data table to the form… but *make sure* that you drag the node that is a *child* of the Departments table – this ensures that the relationship between the two is kept automatically;
- Resize the form and data grid view to give it some room.

Note the new employeesBindingSource component has been added to the form automatically. Remember, the binding source

sits between the controls (in this case, a DataGridView) and the dataset where our data will be available in data tables!

3.2.1 The Last Push – Populating the DataSet

We're so nearly ready! Yet if we run the form now, you will be rather disappointed, as we have not yet told the form to populate the dataset.

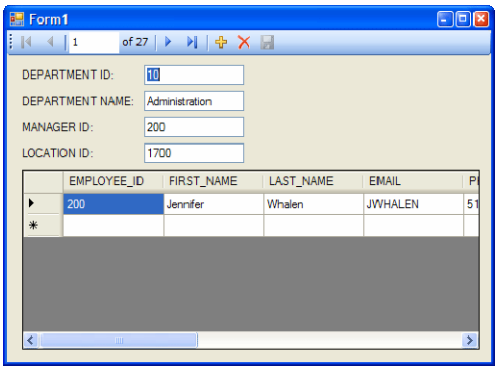
- Double-click a blank area of the form; this is a shortcut to creating the default event handler for the form, which is `Form1_Load`.
- Add the following `using` command at the top of the file to simplify our reference to the DAL code:

```
using HrDataAccessLayer;
```

- Then change the `Form1_Load` event so it looks like this:

```
private void Form1_Load(object sender, EventArgs e)
{
    HrDAL.FillWithRefCursors(hrDataSet);
}
```

- …wait for it…
  Press `F5` to compile and run the solution.



Woo Hoo! We can use the navigator element to traverse through the Department table, and as we do so, we see that the employees in that department that are listed in the table.

3.3 Summary of Achievements

During this exercise, we:

- Created two layers (GUI and DAL) of a three-tier architecture, and added a stored package on the third layer (the database itself);
- Created a typed dataset and data access layer that can be reused.

However, as much as it is fun to finally achieve an Oracle connection with the pretty table, it really does not take long to see many points that we would like to improve. Not least:

- Being able to save any changes to the data back to the database;
- Improve on the ridiculous-looking all-capital labels;
- Some (any!) resilience to errors like the database being unavailable.

I hope that this tutorial has been of use to you. Hopefully I will be able to follow-up to resolve some of the points above.