

Curso de Patrones y Buenas Prácticas .Net

Arquitectura de Aplicaciones .Net (Capas)

Plataforma: Visual Studio .Net
Framework: Microsoft .Net Framework

Documento de Referencia y Capacitación
(Este documento está sujeto a cambios)

Fecha de Creación: 03-Jun-2010
Versión: 2.0.0.0
Autor: Julio Cesar Robles Uribe

Tabla de Contenido

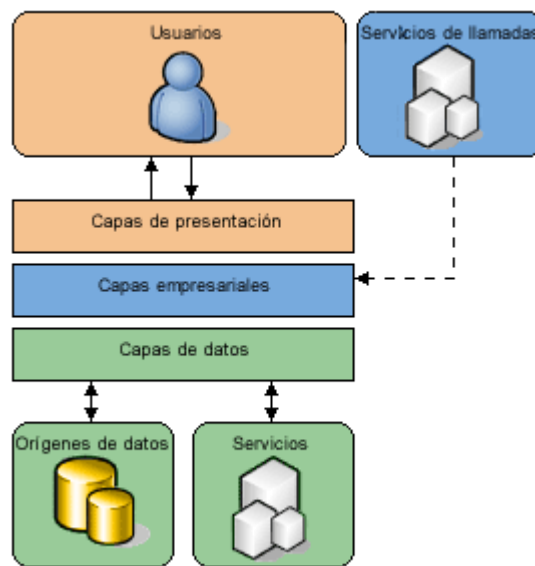
Introducción	4
Arquitectura de Aplicaciones para .Net.....	5
Tipos de Componentes.....	6
1. Componentes de interfaz de usuario (IU).....	7
2. Componentes de proceso de usuario.....	8
3. Flujos de trabajo empresariales.....	9
4. Componentes empresariales.....	10
5. Agentes de servicios.....	11
6. Interfaces de servicios.....	12
7. Componentes lógicos de acceso a datos.....	13
8. Componentes de entidad empresarial.....	14
9. Componentes de seguridad, administración operativa y comunicación.....	15
9.1 Componentes de Seguridad.....	16
9.2 Componentes de Administración Operativa.....	17
9.3 Componentes de Comunicación.....	19
Retroalimentación.....	20
Caso de Estudio.....	22
Crear Proyecto.....	25
Entidades (Entites).....	34
Lógica de Acceso a Datos (DAL).....	39
Lógica de Negocios (BL).....	51
Interfaz de Usuario (UI).....	55
Práctica.....	76
Observaciones y Recomendaciones.....	78

Introducción

Este documento proporciona una breve introducción a la arquitectura y el diseño de aplicaciones .NET. Se mostrará la partición de las funcionalidades de la aplicación en capas y se describirán sus principales características de diseño, de igual modo se explicará cómo se realiza la comunicación de cada a capa.

Arquitectura de Aplicaciones para .Net

El diseño de aplicaciones distribuidas no es una tarea sencilla. Es necesario tomar un gran número de decisiones a nivel de arquitectura, diseño e implementación. Estas decisiones tendrán un impacto en las "capacidades" de la aplicación (seguridad, escalabilidad, disponibilidad y mantenimiento, entre otras), así como en la arquitectura, el diseño y la implementación de la infraestructura de destino. Esta guía le ayudará a comprender las distintas opciones que se presentan a la hora de diseñar las capas de una aplicación distribuida; estas opciones se presentan como un conjunto de componentes que se podrán utilizar para modelar la aplicación. En la figura siguiente se muestran las capas de los componentes lógicos que este documento utiliza para estructurar su distribución.

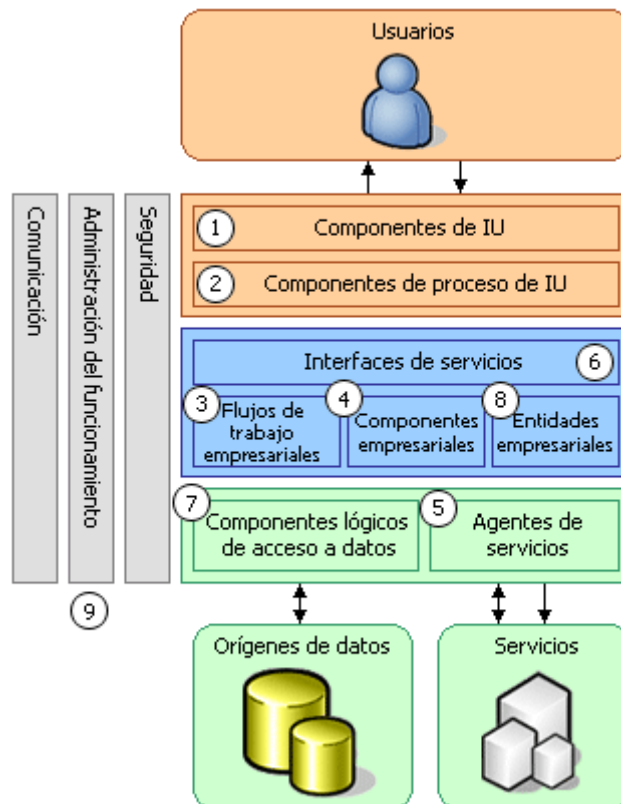


En este esquema principal se muestran diferentes fuentes de datos y sus interacciones con los diversos niveles de distribución de la aplicación. Las capas principales que se utilizan son:

- **Capa de Presentación:** que interactúa principalmente con el usuario final. También conocida como UI o de Interfaz de Usuario.
- **Capas Empresariales o de Negocios:** que interactúa con sistemas externos y con la capa de datos, controlando la lógica principal de la aplicación. También conocida como BL o capa de Lógica de Negocios.
- **Capa de Datos:** Esta capa es la que provee los recursos de información para las demás capas. También conocida como DAL o capa lógica de acceso a datos.

Tipos de Componentes

El análisis de la mayoría de las soluciones empresariales basadas en modelos de componentes por capas muestra que existen varios tipos de componentes habituales. En la figura siguiente se muestra una ilustración completa en la que se indican estos tipos de componentes.



Nota

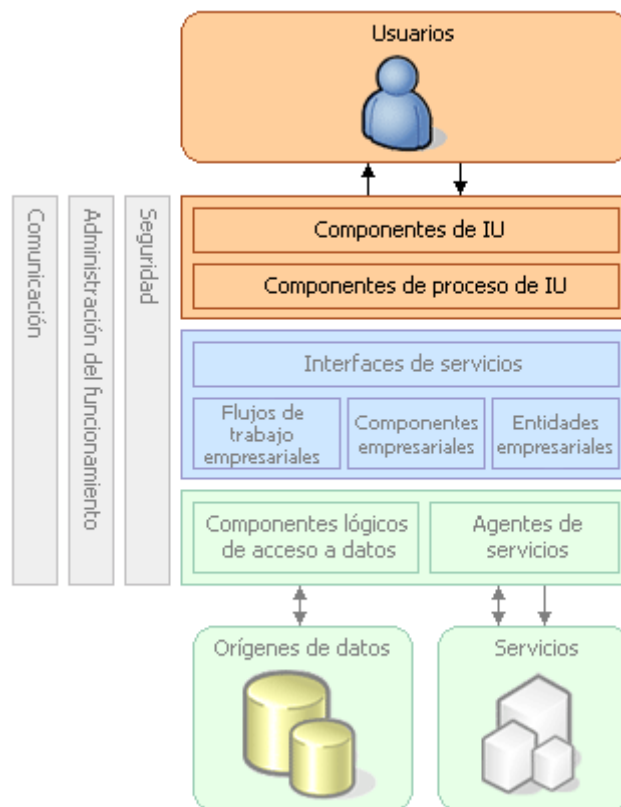
El término componente hace referencia a una de las partes de la solución total, como los componentes de software compilado (por ejemplo, los ensamblados de Microsoft .NET) y otros elementos de software, como las páginas Web.

Los tipos de componentes identificados en el escenario de diseño de ejemplo son:

1. Componentes de interfaz de usuario (IU).
2. Componentes de proceso de usuario.
3. Flujos de trabajo empresariales.
4. Componentes empresariales.
5. Agentes de servicios.
6. Interfaces de servicios.
7. Componentes lógicos de acceso a datos.
8. Componentes de entidad empresarial.
9. Componentes de seguridad, administración operativa y comunicación.

1. Componentes de interfaz de usuario (IU).

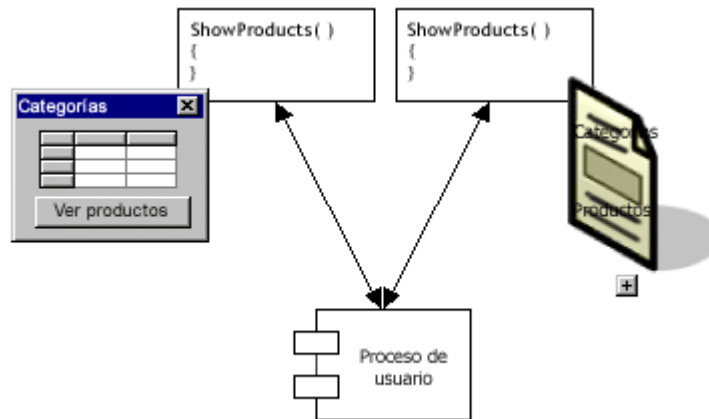
La mayor parte de las soluciones necesitan ofrecer al usuario un modo de interactuar con la aplicación. En el ejemplo de aplicación comercial, un sitio Web permite al cliente ver productos y realizar pedidos, y una aplicación basada en el entorno operativo Microsoft Windows® permite a los representantes de ventas escribir los datos de los pedidos de los clientes que han telefoneado a la empresa. Las interfaces de usuario se implementan utilizando formularios de Windows Forms, páginas Microsoft ASP.NET, controles u otro tipo de tecnología que permita procesar y dar formato a los datos de los usuarios, así como adquirir y validar los datos entrantes procedentes de éstos.



En resumen podríamos decir que los componentes de interfaz de usuario solo permiten mostrar la información e interactúan con el usuario final.

2. Componentes de proceso de usuario.

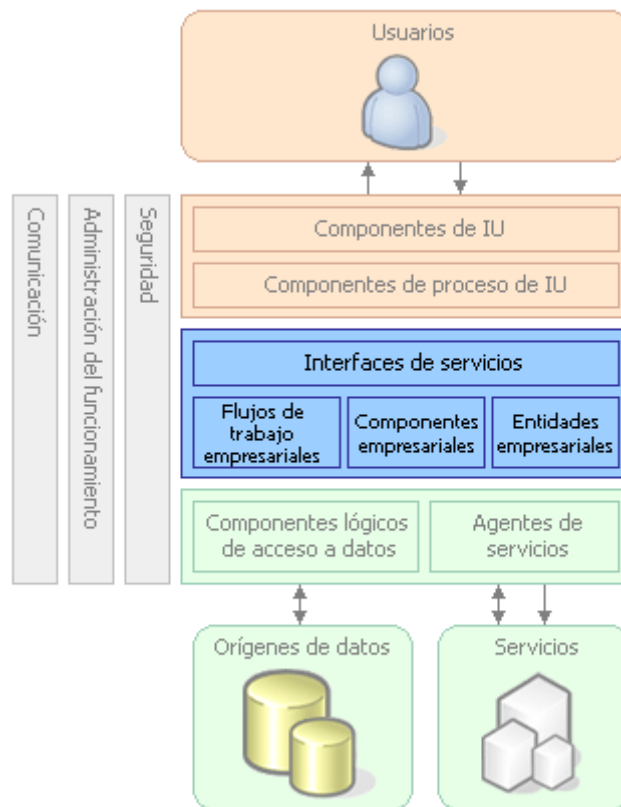
En un gran número de casos, la interacción del usuario con el sistema se realiza de acuerdo a un proceso predecible. Por ejemplo, en la aplicación comercial, podríamos implementar un procedimiento que permita ver los datos del producto. De este modo, el usuario puede seleccionar una categoría de una lista de categorías de productos disponibles y, a continuación, elegir uno de los productos de la categoría seleccionada para ver los detalles correspondientes. Del mismo modo, cuando el usuario realiza una compra, la interacción sigue un proceso predecible de recolección de datos por parte del usuario, por el cual éste en primer lugar proporciona los detalles de los productos que desea adquirir, a continuación los detalles de pago y, por último, la información para el envío. Para facilitar la sincronización y organización de las interacciones con el usuario, resulta útil utilizar componentes de proceso de usuario individuales. De este modo, el flujo del proceso y la lógica de administración de estado no se incluyen en el código de los elementos de la interfaz de usuario, por lo que varias interfaces podrán utilizar el mismo "motor" de interacción básica.



En otras palabras controlan el contenido y la información que puede ser desplegada al usuario.

3. Flujos de trabajo empresariales.

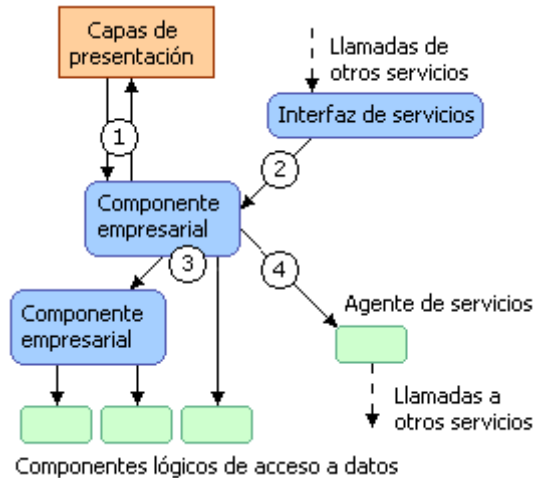
Una vez que el proceso de usuario ha recopilado los datos necesarios, éstos se pueden utilizar para realizar un proceso empresarial. Por ejemplo, tras enviar los detalles del producto, el pago y el envío a la aplicación comercial, puede comenzar el proceso de cobro del pago y preparación del envío. Gran parte de los procesos empresariales conllevan la realización de varios pasos, los cuales se deben organizar y llevar a cabo en un orden determinado. Por ejemplo, el sistema empresarial necesita calcular el valor total del pedido, validar la información de la tarjeta de crédito, procesar el pago de la misma y preparar el envío del producto. El tiempo que este proceso puede tardar en completarse es indeterminado, por lo que sería preciso administrar las tareas necesarias, así como los datos requeridos para llevarlas a cabo. Los flujos de trabajo empresariales definen y coordinan los procesos empresariales de varios pasos de ejecución larga y se pueden implementar utilizando herramientas de administración de procesos empresariales.



En otras palabras controlan el flujo de información requerida para completar una operación o transacción del lado del negocio.

4. Componentes empresariales.

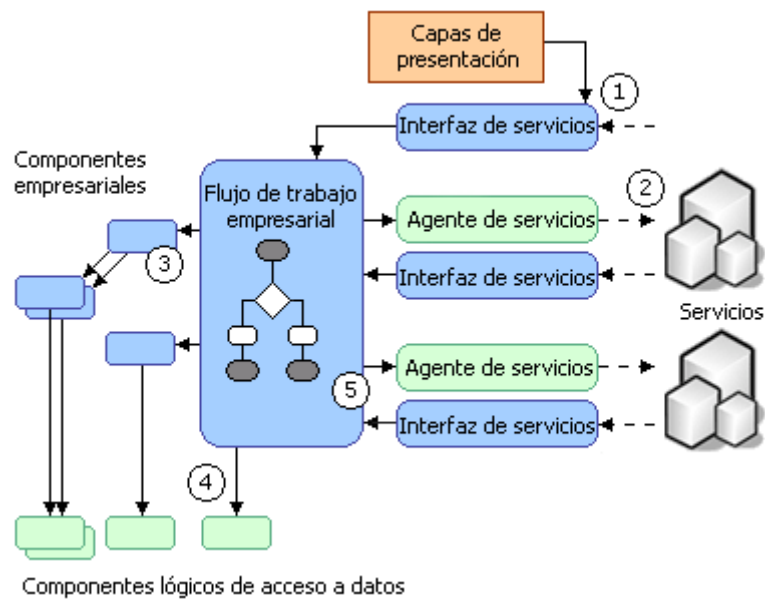
Independientemente de si el proceso empresarial consta de un único paso o de un flujo de trabajo organizado, la aplicación requerirá probablemente el uso de componentes que implementen reglas empresariales y realicen tareas empresariales. Por ejemplo, en la aplicación comercial, deberá implementar una funcionalidad que calcule el precio total del pedido y agregue el costo adicional correspondiente por el envío del mismo. Los componentes empresariales implementan la lógica empresarial de la aplicación.



En otras palabras controlan la lógica de la información realizando validaciones y efectuando operaciones sobre la información.

5. Agentes de servicios.

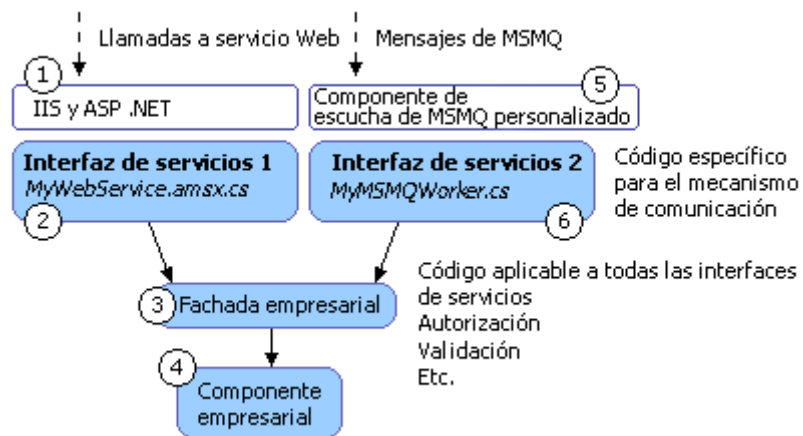
Cuando un componente empresarial requiere el uso de la funcionalidad proporcionada por un servicio externo, tal vez sea necesario hacer uso de código para administrar la semántica de la comunicación con dicho servicio. Por ejemplo, los componentes empresariales de la aplicación comercial descrita anteriormente podrían utilizar un agente de servicios para administrar la comunicación con el servicio de autorización de tarjetas de crédito y utilizar un segundo agente de servicios para controlar las conversaciones con el servicio de mensajería. Los agentes de servicios permiten aislar las idiosincrasias de las llamadas a varios servicios desde la aplicación y pueden proporcionar servicios adicionales, como la asignación básica del formato de los datos que expone el servicio al formato que requiere la aplicación.



En otras palabras permite obtener información o ejecutar funcionalidades de entes externos al sistema.

6. Interfaces de servicios.

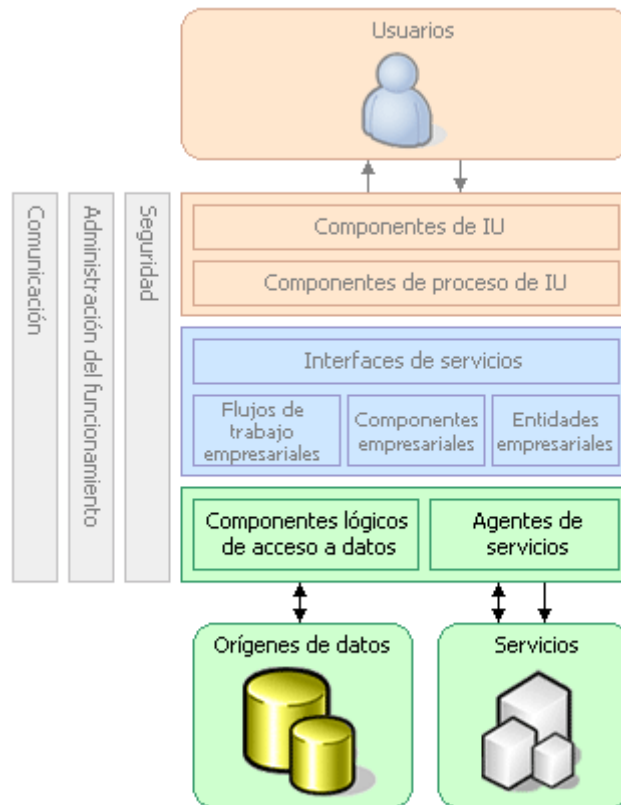
Para exponer lógica empresarial como un servicio, es necesario crear interfaces de servicios que admitan los contratos de comunicación (comunicación basada en mensajes, formatos, protocolos, seguridad y excepciones, entre otros) que requieren los clientes. Por ejemplo, el servicio de autorización de tarjetas de crédito debe exponer una interfaz de servicios que describa la funcionalidad que ofrece el servicio, así como la semántica de comunicación requerida para llamar al mismo. Las interfaces de servicios también se denominan fachadas empresariales.



En otras palabras es la que permite interactuar al sistema con servicios o sistemas externos.

7. Componentes lógicos de acceso a datos.

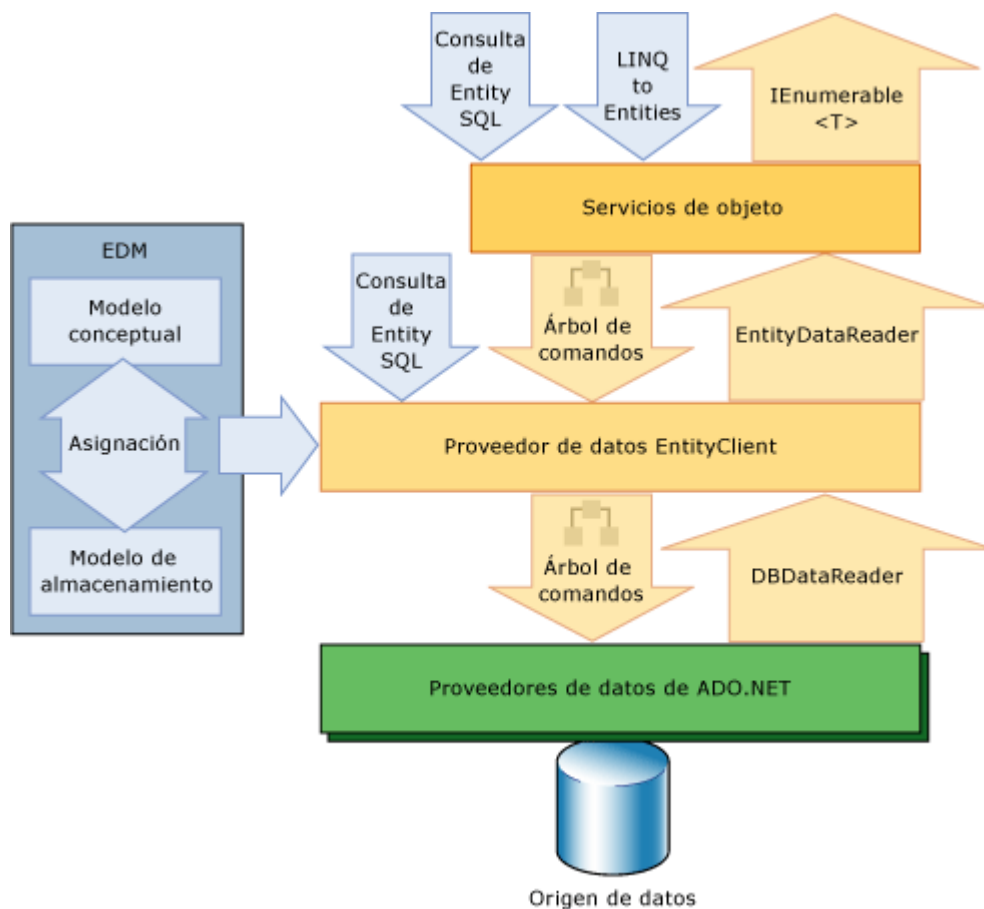
La mayoría de las aplicaciones y servicios necesitan obtener acceso a un almacén de datos en un momento determinado del proceso empresarial. Por ejemplo, la aplicación empresarial necesita recuperar los datos de los productos de una base de datos para mostrar al usuario los detalles de los mismos, así como insertar dicha información en la base de datos cuando un usuario realiza un pedido. Por tanto, es razonable abstraer la lógica necesaria para obtener acceso a los datos en una capa independiente de componentes lógicos de acceso a datos, ya que de este modo se centraliza la funcionalidad de acceso a datos y se facilita la configuración y el mantenimiento de la misma.



En otras palabras son los componentes que nos permiten acceder a los datos o a la lógica de los datos.

8. Componentes de entidad empresarial.

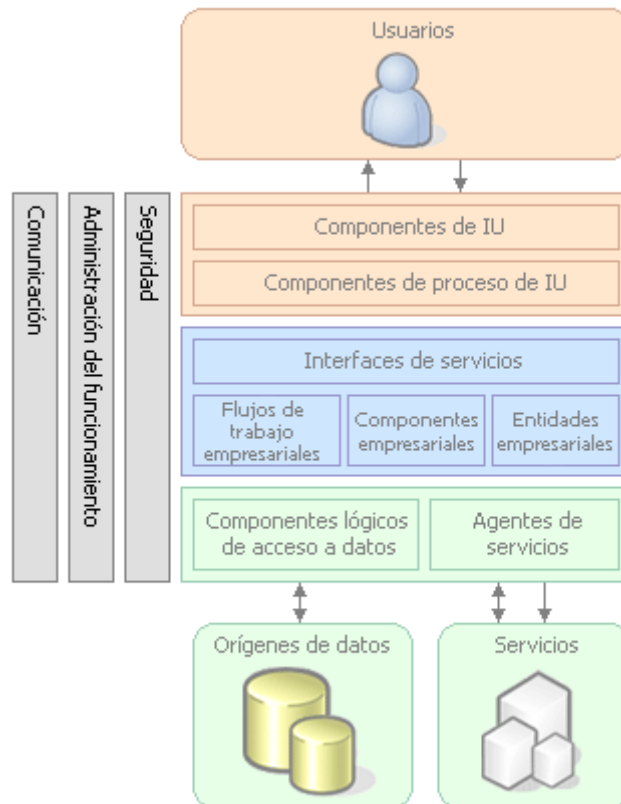
La mayoría de las aplicaciones requieren el paso de datos entre distintos componentes. Por ejemplo, en la aplicación comercial es necesario pasar una lista de productos de los componentes lógicos de acceso a datos a los componentes de la interfaz de usuario para que éste pueda visualizar dicha lista. Los datos se utilizan para representar entidades empresariales del mundo real, como productos o pedidos. Las entidades empresariales que se utilizan de forma interna en la aplicación suelen ser estructuras de datos, como conjuntos de datos, `DataReader` o secuencias de lenguaje de marcado extensible (XML), aunque también se pueden implementar utilizando clases orientadas a objetos personalizadas que representan entidades del mundo real necesarias para la aplicación, como productos o pedidos.



En otras palabras son objetos que representan los datos o la información necesaria para el sistema y que pueden viajar entre las demás capas. También se conoce como la capa de entidades. En algunos casos esta capa se dibuja transversal a todas las demás ya que puede ser utilizada por las demás en cualquier momento.

9. Componentes de seguridad, administración operativa y comunicación.

La aplicación probablemente utilice también componentes para realizar la administración de excepciones, autorizar a los usuarios a que realicen tareas determinadas y comunicarse con otros servicios y aplicaciones.



Esta conjunto de componentes se conoce también como la capa Operacional, ya que soporta todo el proceso y la operación de las demás capas.

9.1 Componentes de Seguridad.

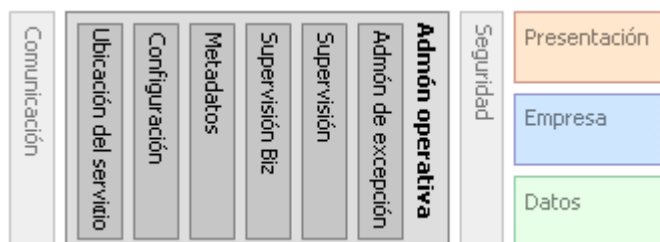
Los componentes de seguridad se ocupan de la autenticación, autorización, comunicación segura, auditoría y administración de perfiles.



- La autenticación permite manejar el ingreso de los usuarios autorizados o inscritos en el sistema.
- La autorización define que cosas puede o no hacer un usuario sobre el sistema.
- La comunicación segura define el protocolo o canal que se utilizara para realizar la comunicación así como la seguridad de los datos.
- La auditoría define el rastro o registro de las operaciones que realizan los usuarios, en caso de que se necesite validar operaciones sobre el sistema.
- La administración de perfiles, permite asignar las opciones validas que puede realizar un usuario sobre el sistema y se complementa con la autorización.

9.2 Componentes de Administración Operativa

Los componentes de administración operativa se ocupan de la ejecución constante y diaria de la aplicación y abarca aspectos como la administración de excepciones, la supervisión, la supervisión empresarial, los metadatos, la configuración y la ubicación del servicio.



- La administración de excepciones incluye la detección y generación de excepciones, el diseño de éstas, el flujo de información de las mismas y la publicación de información de las excepciones a diversos usuarios.
- La supervisión se necesita para instrumentar la aplicación de forma que proporcione al personal de operaciones información sobre el mantenimiento de la aplicación, compatibilidad con los acuerdos de nivel de servicios (SLA), así como administración de la escalabilidad y capacidad.
- La supervisión del negocio tiene como objetivo proporcionar una capacidad reactiva para aquellos que dirigen la empresa en relación al mantenimiento de la empresa, compatibilidad con SLA a nivel de empresa y administración de la capacidad organizativa. En lugar de indicarle los errores de la red, este tipo de supervisión ofrece una perspectiva con respecto a la estructura empresarial y a la eficacia de los procesos. Por ejemplo, puede determinar que procesos empresariales se detengan durante días cada vez que un determinado socio forme parte del envío y control.
- Los Metadatos es lo que hace que su aplicación sea más flexible en relación a los cambios en las condiciones de tiempo de ejecución, puede que desee proporcionarle información sobre él mismo. El diseño de la aplicación para que utilice metadatos en determinados lugares puede facilitar el mantenimiento y permitir su adaptación al cambio sin costosos procesos de implementación o modificaciones en el desarrollo. En otras palabras los metadatos son información anexa de los mismos datos, como tipo de datos, cantidad de registros, encabezados de columnas etc.
- La configuración es necesaria para que las aplicaciones puedan funcionar técnicamente. Los valores que modifican el comportamiento de las directivas (seguridad, administración operativa y comunicaciones) se consideran datos de configuración. Los datos de configuración se conservan en los archivos de configuración de .NET a nivel de usuario, equipo y aplicación. La configuración personalizada almacenada aquí se puede definir con cualquier esquema y se puede tener fácil acceso mediante el uso de la clase ConfigurationSettings en su aplicación.
- La ubicación del servicio es la información necesaria para efectuar las llamadas a servicios remotos, lo que permite determinar dónde están situados los objetos y servicios externos de .NET que pueden procesar su solicitud y cómo tener acceso a ellos. Esto resulta especialmente importante a la hora de utilizar servicios Web alojados por otras organizaciones o terceros, en otras palabras son los descubridores de servicios y las referencias a los servicios externos del sistema.

Estos componentes se encargan de verificar la salud del sistema pudiendo registrar, por ejemplo, la cantidad de usuarios conectados al sistema, la cantidad de transacciones ejecutadas, la cantidad de descargas, el numero de excepciones presentadas, etc.

9.3 Componentes de Comunicación

Los componentes de comunicación define la forma en que los componentes de la aplicación se comunicarán entre sí. Esta directiva trata cuestiones como la sincronización de la comunicación, el formato y el protocolo.



- La sincronización define la forma en cómo dos aplicaciones se envían o reciben mensajes, para ello existen diversos modelos de los cuales se pueden implementar los que se requieran según el objetivo del sistema.
- El formato determina como se envían los datos, y la forma en como se deben interpretar.
- El protocolo determina la forma en cómo se transmiten los datos, para ello se pueden utilizar diferentes formas (SOAP, HTTP, SMTP, FTP, etc.) que empaquetan los mensajes y los procesan para que sean entendidos por la parte receptora.

En otras palabras determinan como se envían los mensajes a los entes del sistema desde y hacia los servicios externos, además de implementar el formato de transmisión (binario, xml, texto, etc.) y el protocolo de envío, que es diferente del protocolo de comunicación.

Retroalimentación

1. Las capas principales en las que se puede dividir una aplicación o sistema son: UI, BL y DAL.

☐ Verdadero ☐ Falso

R/Verdadero: UI es la abreviatura para par referirse a la capa de presentación, BL para referirse a la capa de lógica de negocios y DAL para referirse a la capa de lógica de acceso a datos.

2. El termino componente se refiere a la parte principal del sistema.

☐ Verdadero ☐ Falso

R/Falso: El término componente hace referencia a una de las partes de la solución total, no determina si es la principal o no.

3. Los componentes de interfaz de usuario, solo permiten mostrar la información e interactúan con el usuario.

☐ Verdadero ☐ Falso

R/Verdadero: Estos componentes solo despliegan la información no la manipulan.

4. Los componentes de proceso de usuario muestran la información al usuario final.

☐ Verdadero ☐ Falso

R/Falso: Los componentes de proceso de usuario, controlan el contenido y el flujo de la información que se le muestra al usuario.

5. Los Flujos de trabajo empresariales controlan el flujo de información requerida para completar una operación.

☐ Verdadero ☐ Falso

R/Verdadero: el flujo determina que camino o ruta de negocio o de lógica se debe desarrollar según el contenido de la información.

6. Los componentes de empresariales implementan reglas de validación y realizan tareas específicas.

☐ Verdadero ☐ Falso

R/Verdadero: Los componentes empresariales controlan la lógica de la información realizando validaciones y efectuando operaciones sobre la información.

7. Las interfaces de servicios son las que permiten exponer funciones o servicios a otros sistemas.

☐ Verdadero ☐ Falso

R/Verdadero: Las Interfaces son las que permiten al sistema interactuar con servicios o sistemas externos.

8. Los componentes de Acceso a datos son los que nos permiten acceder a la información directamente.

☐ Verdadero ☐ Falso

R/Verdadero: Los componentes de acceso a datos recuperan o alimentan la información del sistema.

9. Los componentes de entidad empresarial son los únicos que pueden navegar entre las capas.

☐ Verdadero ☐ Falso

R/Verdadero: Los componentes de entidad empresarial o simplemente entidades (Entities) son los únicos que pueden ser llamados y manipulados en cualquiera de las capas (UI, BL, DAL).

10. Los componentes de seguridad, administración operativa y de comunicación se denominan también componentes de servicios.

☐ Verdadero ☒ Falso

R/Falso: Estos componentes se agrupan en un solo conjunto llamado componentes operacionales ya que soporta todo el proceso y la operación de las demás capas.

Caso de Estudio

En este caso queremos manejar los datos de nuestros amigos en particular los teléfonos y para ello necesitamos crear una aplicación que nos permita manejar los datos de las personas y los teléfonos asociados a ellos.

Para entender mejor el manejo de las capas vamos a realizar un caso práctico en donde trataremos de manejar de forma concreta los principales aspectos de la arquitectura de capas (Layers).

Pre-requisitos

- Conexión a la base de datos ORACLE y acceso a la instancia **DirTelDB**.
 - User= DirTelUser
 - Password= DirTelPwd
 - Instancia= DirTelDB
- las librerías de Oracle Data Access Componente with Oracle Developer Tools for Visual Studio
 - **ODTwithODAC112021.zip**
 - <http://www.oracle.com/technetwork/topics/dotnet/utilsoft-086879.html>
 - Documentación de instalación
 - <http://www.oracle.com/technetwork/topics/dotnet/downloads/install112021-200037.html>
- Archivo **instantclient-basiclite-nt-11.2.0.2.0.zip** de Conexión liviana hacia ORACLE (Instant Client)
- Archivo de soporte para Scripts de base de datos, Librería de ayuda para el acceso a datos (SQLHelper) y plantilla de código (**SupportData.Zip**)

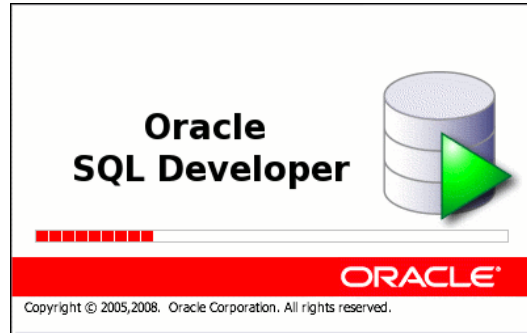
Preparar el Ambiente

1. Realice la instalación de las librerías de ODAC en el directorio **C:\ODAC**
2. Explore la estructura de directorios, después de instalar e identifique la versión de la librería de Acceso a datos, **Oracle.DataAccess.dll**, adecuada para el Framework 2.0. para ello busque en la ruta **C:\ODAC\product\11.2.0\client_1\odp.net\bin\2.x**
3. Cree el directorio de Trabajo **C:\Proyecto** y allí referencie los pasos siguientes, además descomprima el archivo **SupportData.zip**.
4. Identifique los directorios de SupportData:
 - a. **Admin:** Contiene los archivos de configuración para establecer conexión con la base de datos (TNSName.ora y SQLNet.ora)
 - b. **Librerías:** Contiene las librerías de Conexión a la base de datos
 - i. **EanT.Framework.Base.Data.dll:** SQLHelper o ayuda para el acceso a datos
 - ii. **Orracle.DataAccess.dll:** Librería de ORACLE para conectarse a la base de datos.
 - c. **Llaves:** Contiene la llave de registro para establecer donde están los registros de configuración de ORACLE.
 - d. **Plantilla:** Contiene la plantilla de referencia para documentar el código .Net
 - e. **Recursos:** Contiene imágenes e iconos que será utilizados en el proyecto.
 - f. **SQL:** Contiene los Scripts de creación de usuario, tablas y demás, necesarios para acceder a los datos.

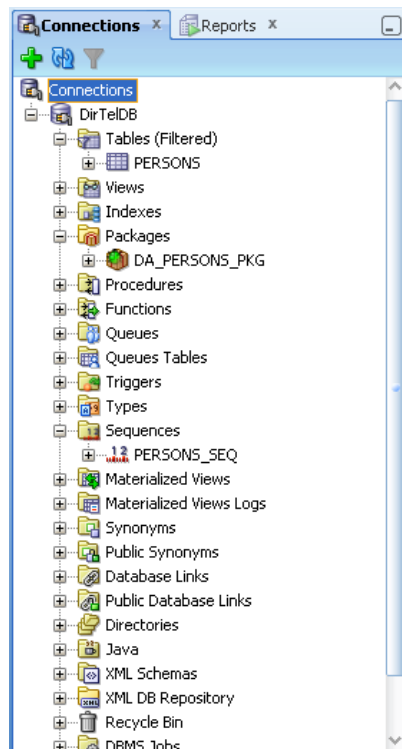
5. Descomprima el archivo conexión Liviana de Oracle (Oracle Instant Client 11.2.0.2) **instantclient-basiclite-nt-11.2.0.2.0.zip** en el directorio **C:\Proyecto\InsCli**.
6. En el directorio del Oracle Instant Client (**C:\Proyecto\InsCli**), copie los archivos contenidos en el directorio **C:\Proyecto\SupporData\Admin** y modifique los valores correspondientes a la IP, Puerto y nombre de la Instancia de base de datos, según sea su correspondiente valor.
7. Ahora en el directorio del Oracle Instant Client (**C:\Proyecto\InsCli**), copie los archivos contenidos en el directorio **C:\Proyecto\SupporData\Llaves** y ejecute el archivo **Key_INSCLI.reg** para establecer los valores de configuración para la conexión con Oracle.

Preparar la base de datos

1. Utilizando **ORACLE SQL Developer**, u otra herramienta, conectarse a la base de datos, en la instancia **DirTelDB**, usando el usuario **SYS**.



2. Ejecute el Script **DirTelUser.sql** del directorio **C:\Proyecto\SupportData\SQL\User**, para crear el usuario DirTelUser.
3. Desconectarse y volverse a conectar con este usuario (DirTelUser).
4. Ejecutar el Script **Person_Tab.Sql** y **Person_Seq.sql** del directorio **C:\Proyecto\SupportData\SQL\Persons\01.Data**.
5. Ejecutar el Script **DA_Persons_PKG.sql** y **DA_Persons_PKG_Body.sql** del directorio **C:\Proyecto\SupportData\SQL\Persons\02.SPs**.
6. Ejecutar el Script de inserción de datos **Ins_Persons.sql** del directorio **C:\Proyecto\SupportData\SQL\Scripts**.
7. Al finalizar se tendrá la siguiente estructura.

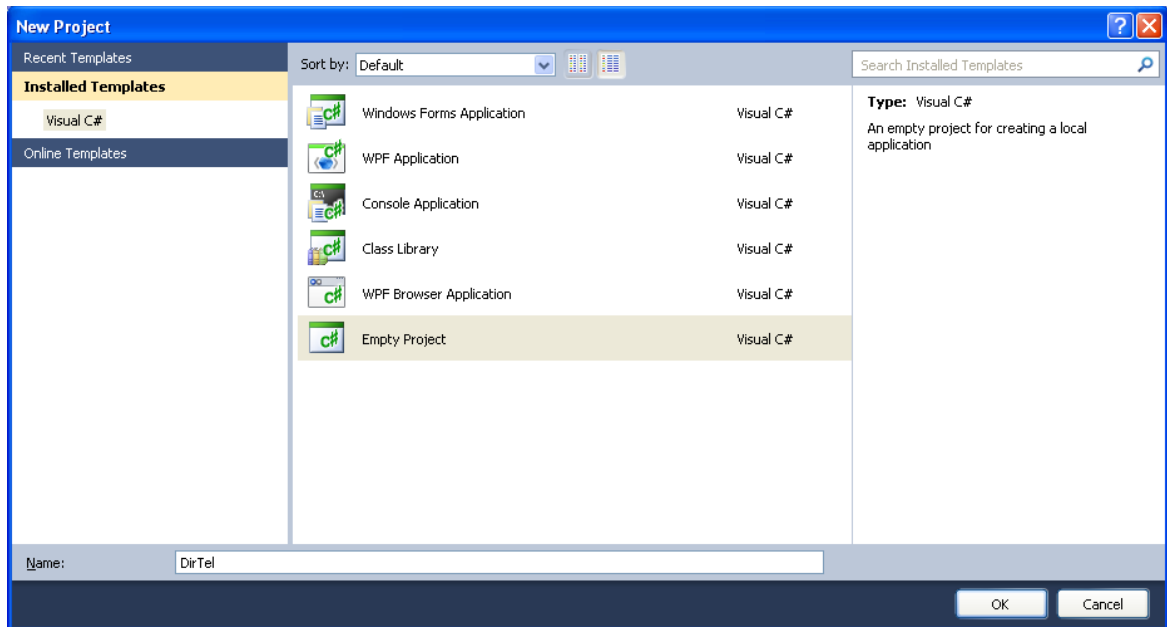


Crear Proyecto

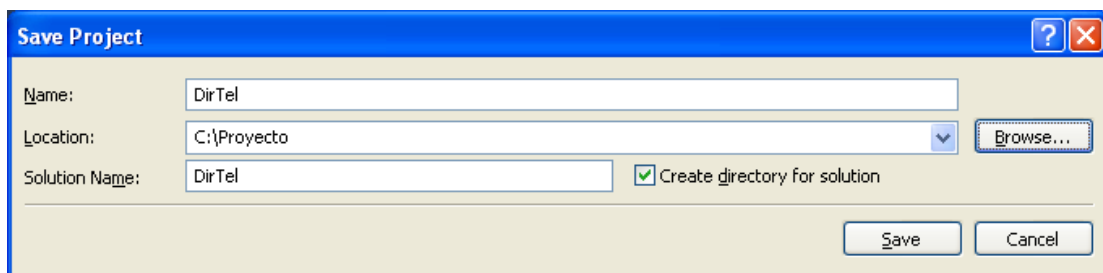
Lo primero que vamos a hacer es crear la distribución por capas de los diversos proyectos y componentes que vamos a desarrollar en este proyecto.

Para ello ejecutaremos los siguientes pasos:

1. Cree un proyecto nuevo referenciando la opción de proyecto vacío (Empty Project)

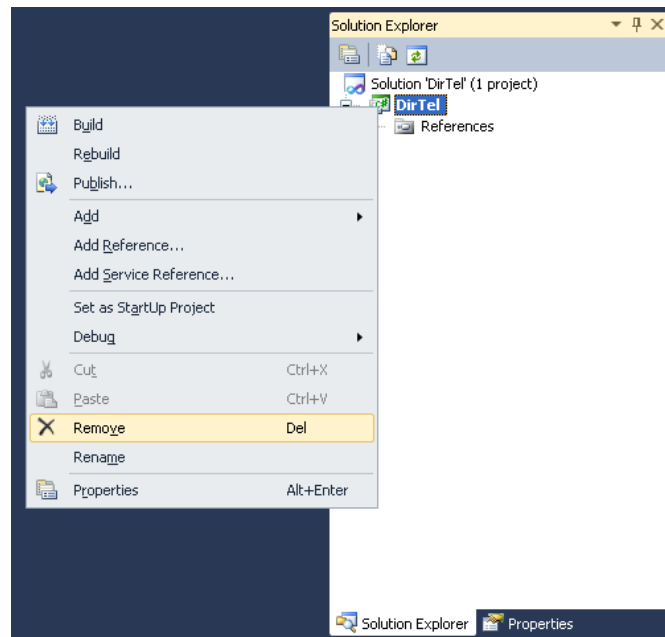


2. Asigne el nombre del proyecto como **DirTel**.
3. Grabe el proyecto incluyendo la solución, para ello utilice el botón de grabar todo, recuerde referenciar el directorio de trabajo (**C:\Proyecto**)

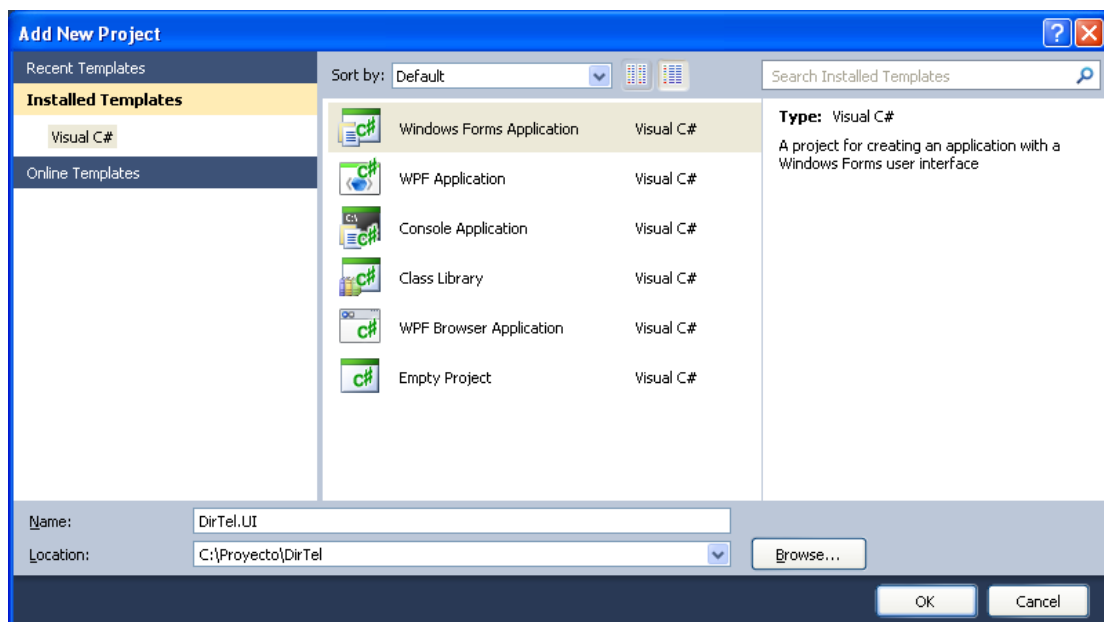


4. Con esto solamente estamos creando un directorio de referencia para nuestro proyecto.

5. Ahora borremos el proyecto inicial creado, haciendo click sobre el y presionando la tecla Suprimir (DEL) o haciendo click derecho y escogiendo la opción de remover.

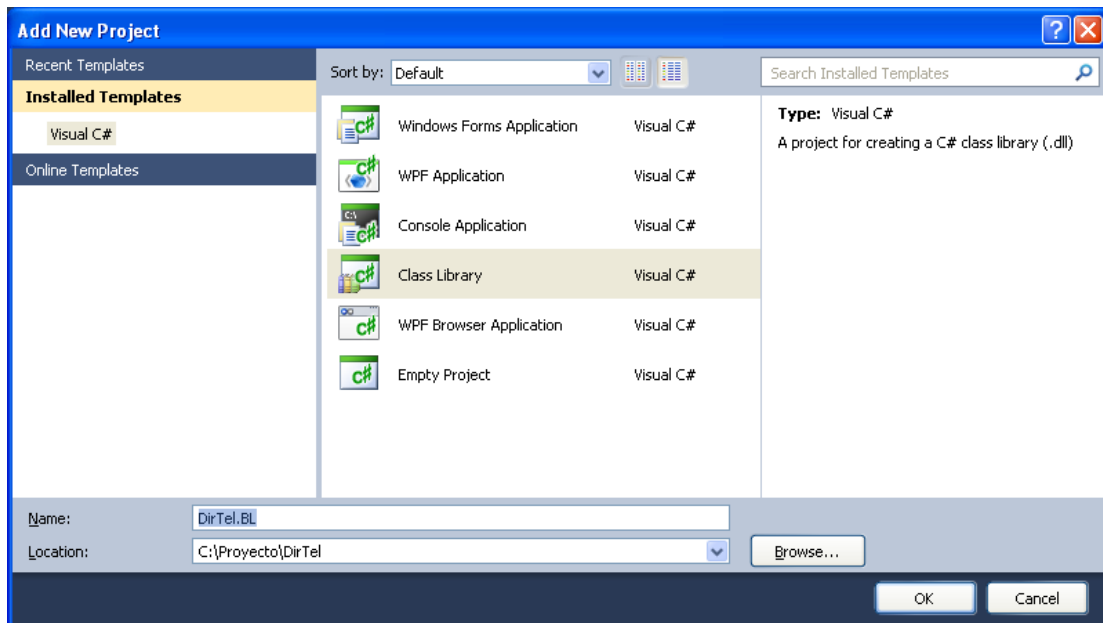


6. Ahora vamos a crear uno a uno los proyectos necesarios para realizar la distribución por capas (UI, BL, DAL, Entities). El nombre de cada proyecto será el mismo de la solución pero utilizara el sufijo de la capa a la que pertenece (DirTel.UI, DirTel.BL, DirTel.DAL y DirTel.Entities)
7. Adicionaremos un nuevo proyecto de tipo Aplicación de Windows (Windows Application) para la capa de Interfaz de Usuario (UI) y su nombre será **DirTel.UI**.

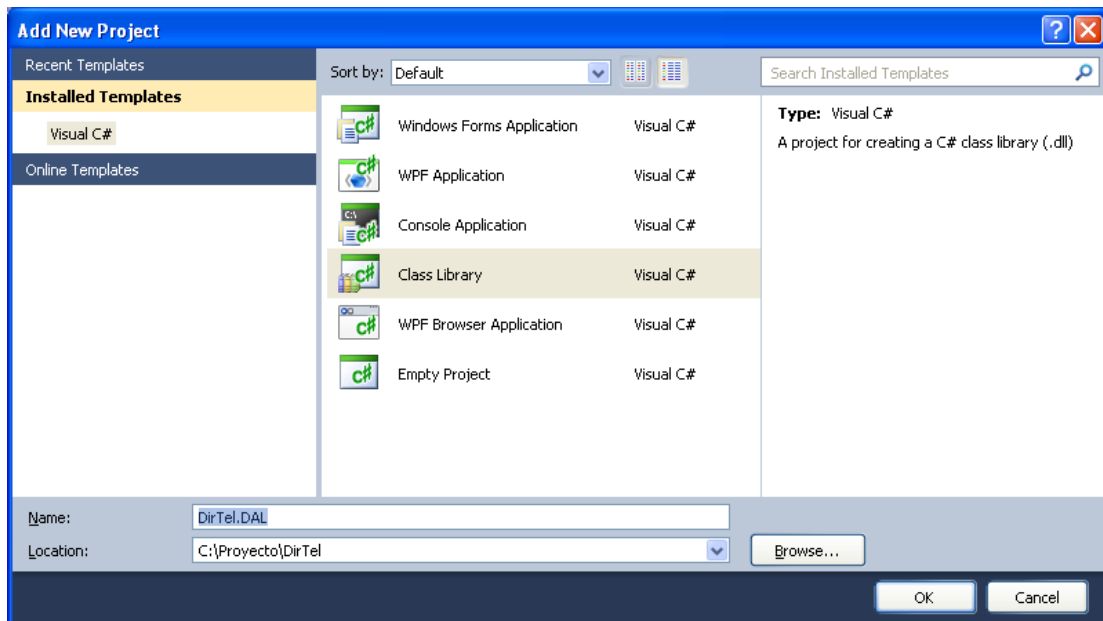


8. En este proyecto se manejara el despliegue de la información hacia el usuario.

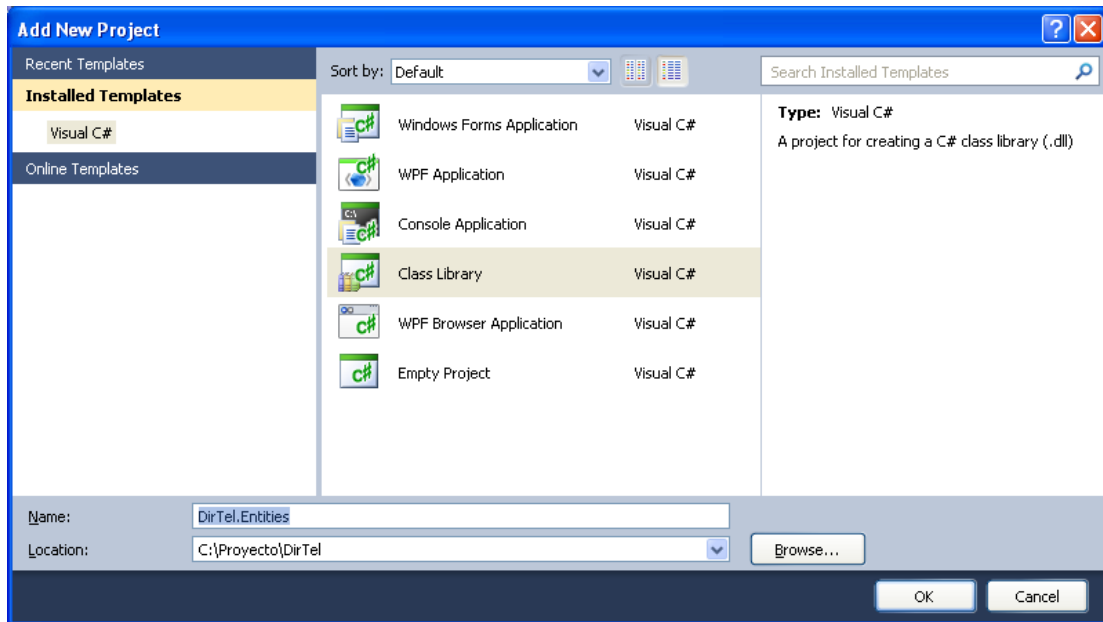
9. Ahora adicionaremos un proyecto de tipo Librería de Clases (Class Library) para manejar la lógica de negocios (**BL**) del proyecto y lo llamaremos **DirTel.BL**.



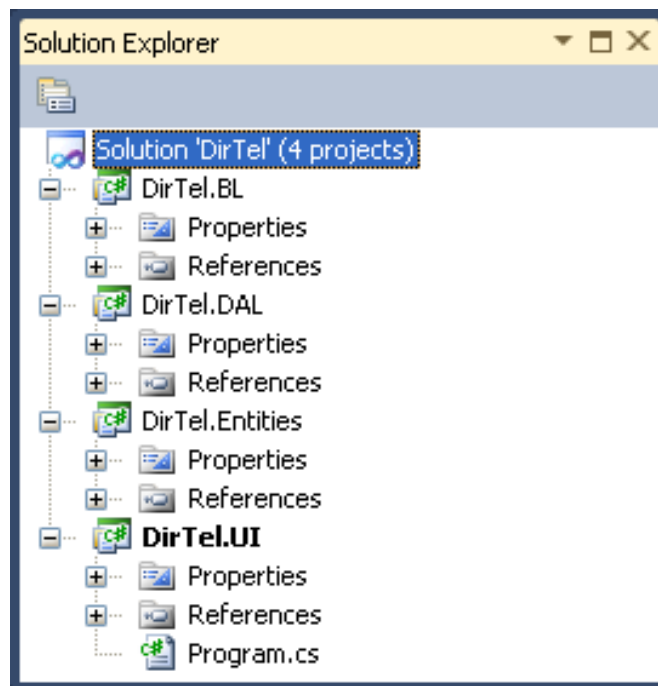
10. Luego adicionaremos un proyecto de tipo Librería de Clases (Class Library) para manejar la lógica del acceso a los datos (**DAL**) del proyecto y lo llamaremos **DirTel.DAL**.



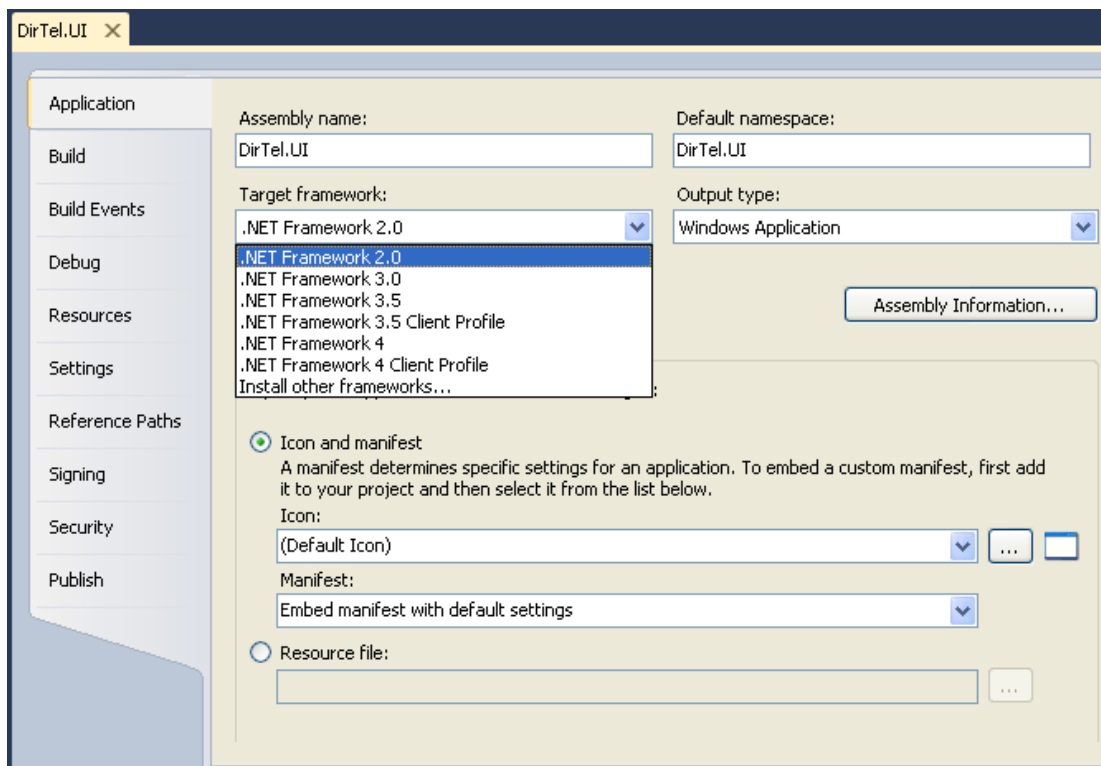
11. Por último adicionaremos un proyecto de tipo librería de clases (Class Library) para manejar los datos de las Entidades (Entities) que podrán ser utilizadas por todos los demás componentes. A este proyecto lo denominaremos **DirTel.Entities**.



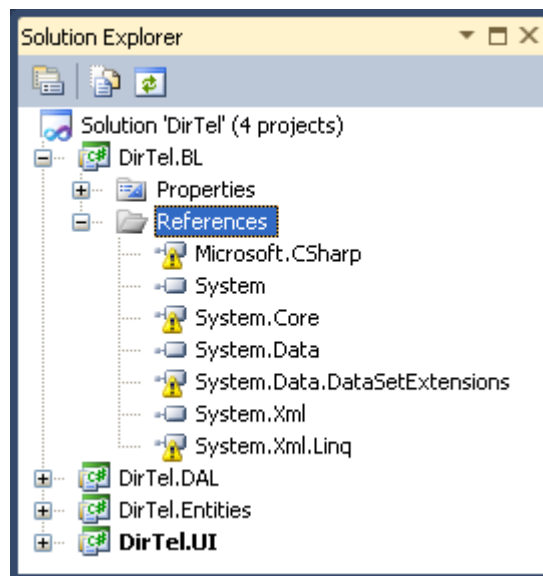
12. De todos los proyectos de tipo Librerías de Clase, elimine los Archivos Class1.cs y del proyecto de aplicación de Windows, elimine el formulario Form1.cs. Al finalizar tendremos la siguiente estructura:



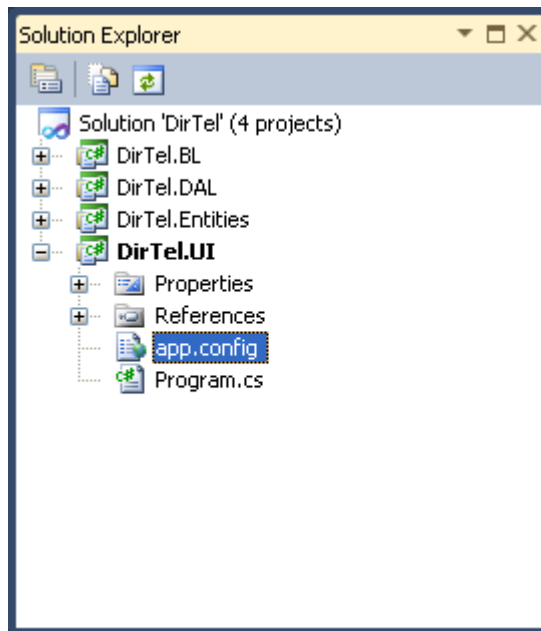
13. En cada proyecto modifique sus propiedades para que el marco de trabajo destino (Target Framework)



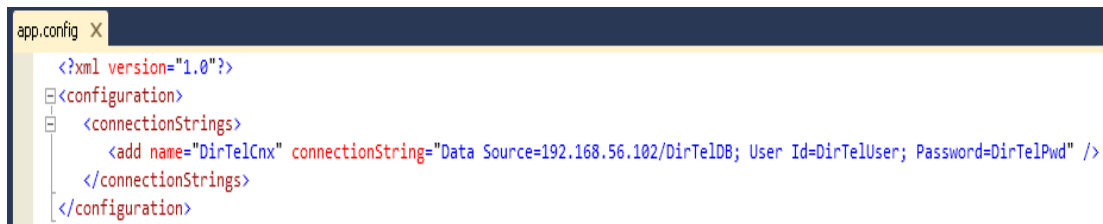
14. De cada uno de los proyectos elimine la referencia a las librerías de que aparecen con el icono de advertencia. (Microsoft.CSharp, System.Core, System.Data.DataSetExtensions, System.Xml.Linq)



15. Modifique el archivo de configuración App.config y adicione la cadena de conexión hacia la base de datos de la siguiente forma:



- Borre la línea referente al Framework inicial
`<startup><supportedRuntime version="v2.0.50727" /></startup>`
- Agregue las líneas de configuración de la cadena de conexión



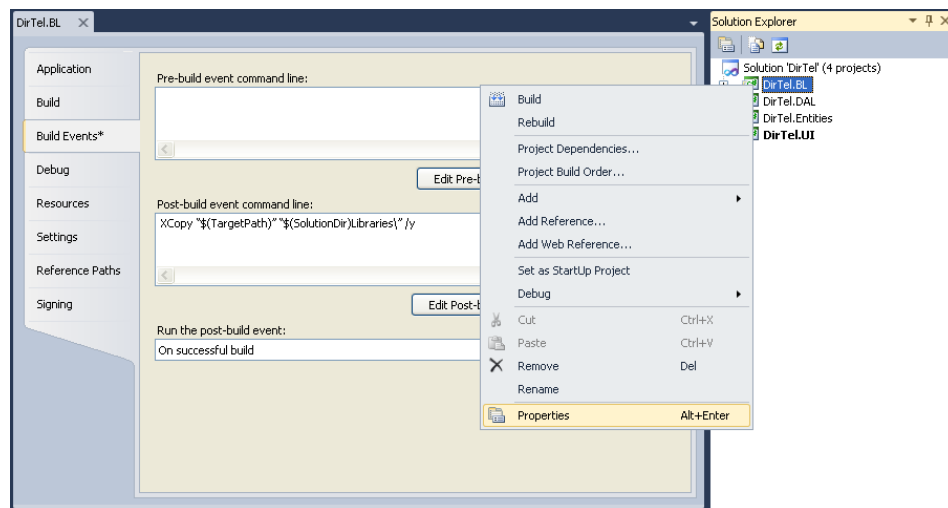
Use la siguiente cadena para completar esta acción:

```
<connectionStrings>
  <add name="DirTelCnx" connectionString="Data Source=192.168.56.102/DirTelDB; User
Id=DirTelUser; Password=DirTelPwd" />
</connectionStrings>
```

Recuerde que los valores de la **IP**, **Nombre de la Instancia**, **Usuario** y **Clave** son sugeridos en este proyecto, en caso de que usted esté utilizando otros, por favor realizar las asignaciones respectivas.

16. Guarde los cambios.

17. Ahora modifique las propiedades de cada proyecto y adicione el siguiente evento Post-Build para que se copien las librerías a un directorio de salida destino.

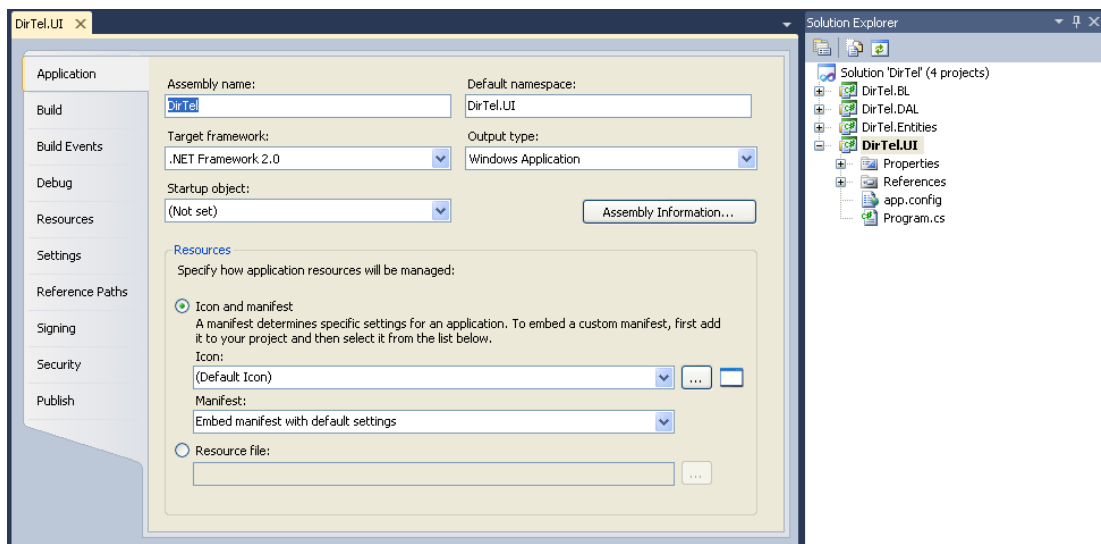


Post-build event command line:

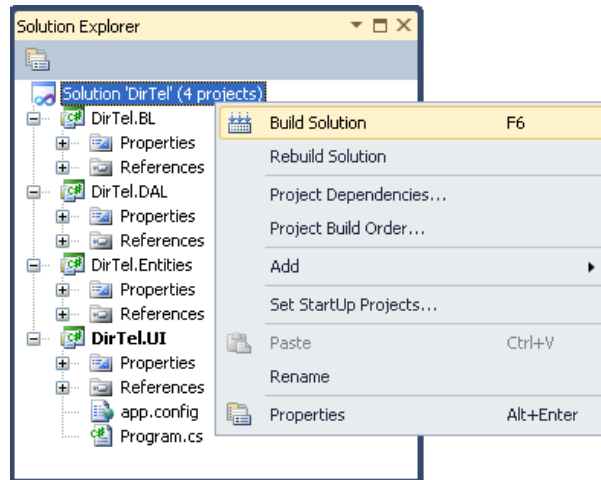
```
XCOPY "$(TargetPath)" "$(SolutionDir)\Libraries\" /y
```

Tenga presente que esto debe hacerlo en todos los proyectos (DirTel.UI, DirTel.BL, DirTel.DAL y DirTel.Entities).

18. Modifique las propiedades del Proyecto **DirTel.UI** cambiando en el TAB de Aplicación (Application) el valor para el nombre del Asembler (**Assembly name**) y cambiando su valor a Solamente **DirTel**, esto para que cuando genere el nombre de la aplicación o ejecutable este sea **DirTel.exe**.

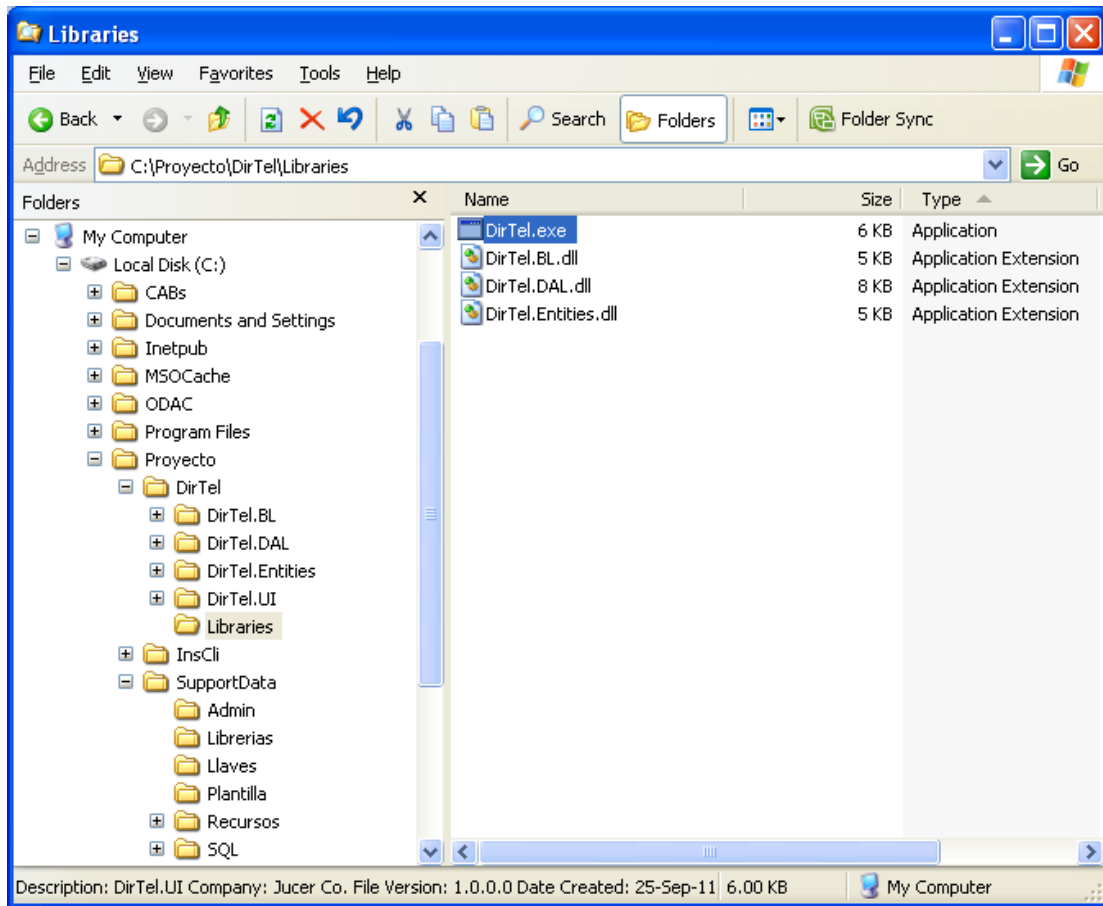


19. Compile y corrija los posibles errores que deja la referencia a otro marco de trabajo (Framework).
- Borre los **using** que posiblemente referencien a librerías **LinQ**, propias del Framework 4.0 de .Net.
 - Ponga en comentario (//) la línea del Program que presente errores.



Compile haciendo click derecho sobre la solución y seleccionando la opción Build Solution o presionando las tecla CTRL+SHIFT+B o la tecla F6.

20. Recuerde borrar el directorio interno **C:\Proyecto\DirTel\DirTel** en donde se encuentra el primer proyecto vacío que se utilizó para crear la solución. Al finalizar se obtendrá una estructura de directorios igual a la siguiente:



En resumen se crean 4 Proyectos Iniciales orientados a manejar las capas así:

- **UI:** Interfaz de Usuario
- **BL:** Lógica de Negocios
- **DAL:** Lógica de Acceso a Datos
- **Entities:** Entidades.

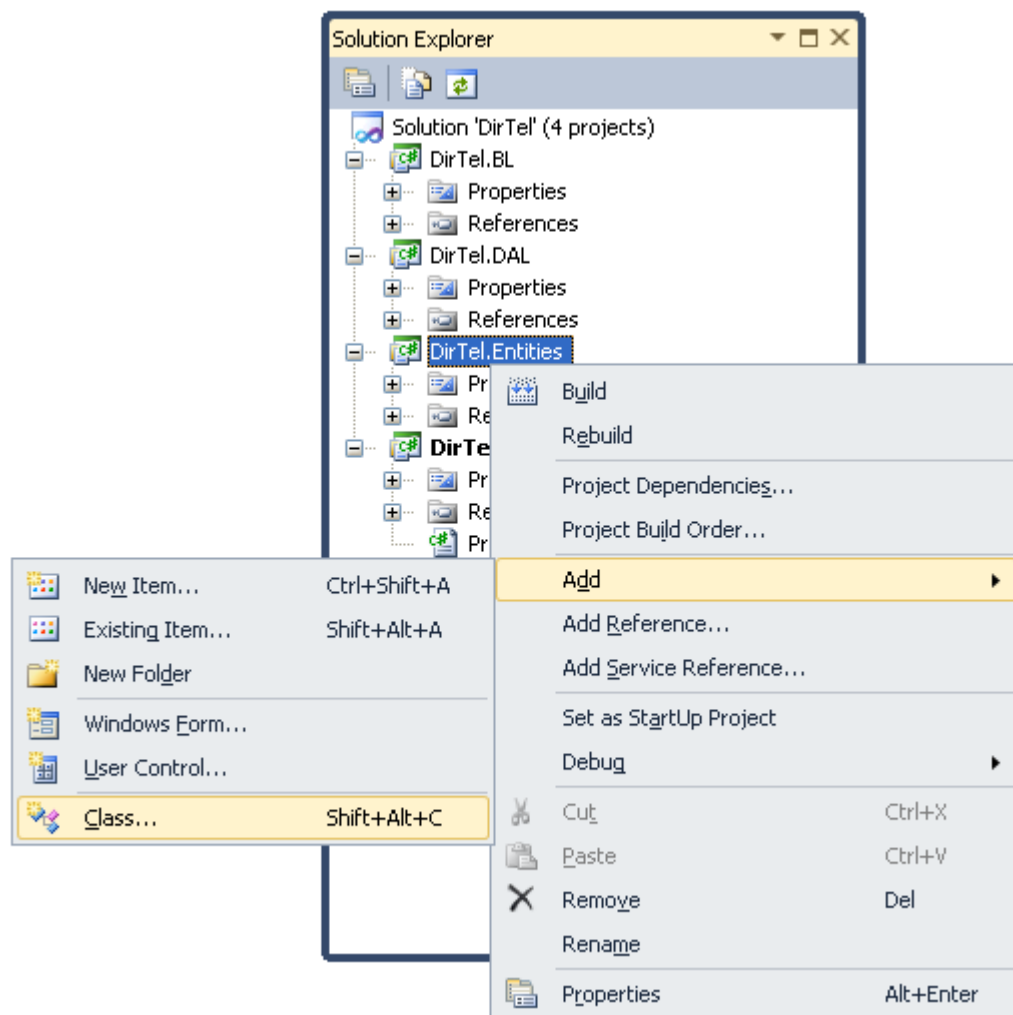
El objetivo de este ejercicio, es explicar el modelo de arquitectura de capas, tomando como ejemplo los datos de las personas.

Entidades (Entites)

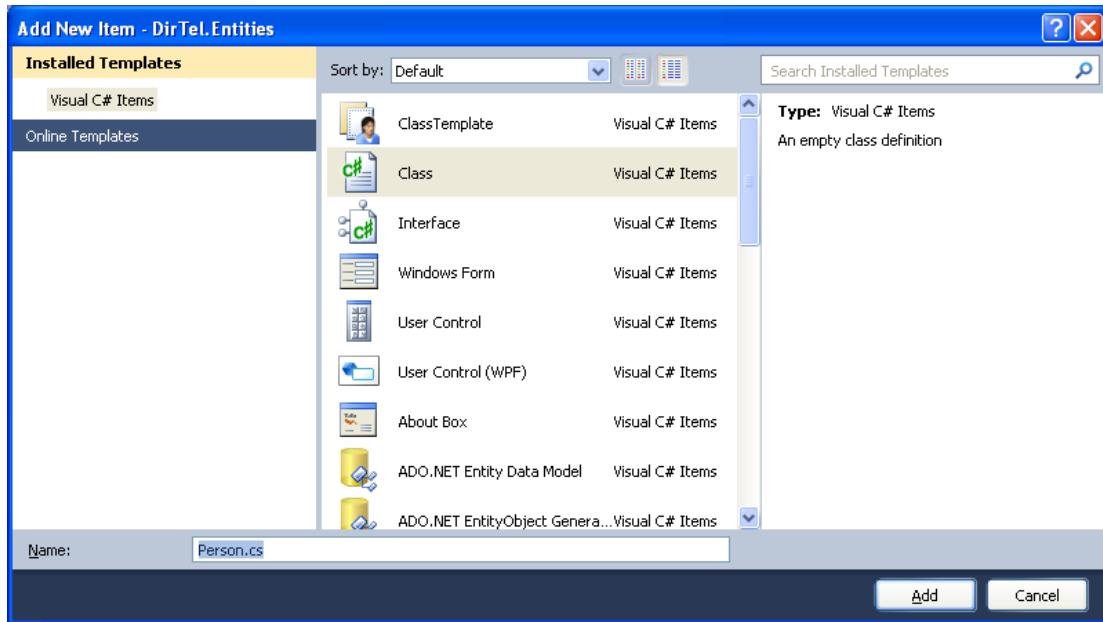
En este ejercicio vamos a desarrollar un ejemplo que muestra la implementación de las capas, partiendo desde la capa más baja (Entidades) hasta llegar a la capa de presentación (UI).

En esta primera parte vamos a construir la entidad que permite referenciar los datos de la persona.

1. Crear la entidad **Person**, que permite manejar los datos de las personas, para ello debe hacer clic derecho sobre el proyecto de **DirTel.Entities**.
2. Del menú contextual seleccionar la opción Adicionar (Add) y luego escoger la opción Clase (Class).



3. En la ventana seleccionar y verificar que se selecciona la opción **Class** y reemplazar el nombre por **Person.cs**



4. Editar el contenido de este archivo **Person.cs** y del archivo **Plantilla.cs**, copie el encabezado de la documentación y modifíquelo completando los valores con sus datos respectivos así:

```

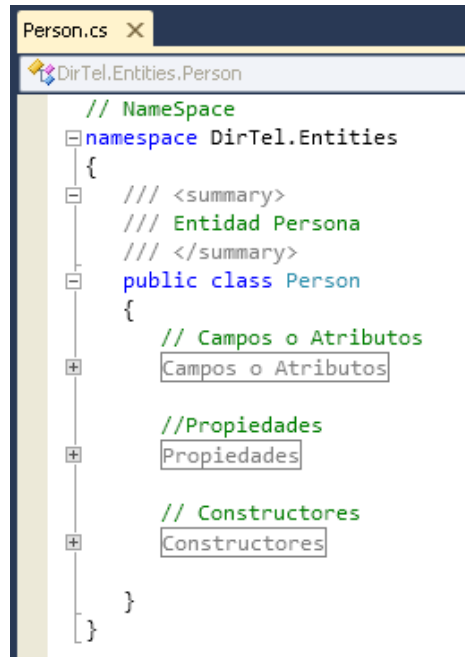
Person.cs
DirTel.Entities.Person
#region Documentación
/
*****
* Propiedad intelectual de Engineers & Tools (c).
*****
* Unidad      : Person.cs
* Descripción : Entidad para manejar los datos de las personas,
* Autor       : Julio Cesar Robles Uribe - Jucer
* Fecha       : 21-May-2010
*
* Fecha      Autor      Modificación
* =====
* 21-May-2010 Jucer      1 - Version Inicial
*****
/
#endregion Documentación

// Librerías
using System;
using System.Collections.Generic;
using System.Text;

// Namespace
namespace DirTel.Entities
{

```

5. Modificar el alcance de la clase a **public** y adicionar la documentación a la clase, tenga presente que los comentarios incluidos dentro de bloques **<summary>** se generan como documentación técnica y los que se incluyen como comentarios internos no son tomados en cuenta por el compilador.



6. Agregue los campos y atributos de la siguiente forma:

```
// Identificador de la persona
private long person_Id;
// Nombres de la Persona
private string firstName;
// Apellidos de la Persona
private string lastName;
// Fecha de Nacimiento (dd-mm-yyyy)
private DateTime birthDay;
// Sexo (M,F)
private char sex;
```

7. Cree las propiedades así:

```
/// <summary>
/// Identificador de la persona
/// </summary>
public long Person_Id
{
    get
    {
        return person_Id;
    }
    set
    {
        person_Id = value;
    }
}
```

```

/// <summary>
/// Nombres de la Persona
/// </summary>
public string FirstName
{
    get
    {
        return firstName;
    }
    set
    {
        firstName = value;
    }
}

/// <summary>
/// Apellidos de la Persona
/// </summary>
public string LastName
{
    get
    {
        return lastName;
    }
    set
    {
        lastName = value;
    }
}

/// <summary>
/// Fecha de Nacimiento (dd-mm-yyyy)
/// </summary>
public DateTime BirthDay
{
    get
    {
        return birthDay;
    }
    set
    {
        birthDay = value;
    }
}

/// <summary>
/// Sexo (M,F)
/// </summary>
public char Sex
{
    get
    {
        return sex;
    }
    set
    {
        sex = value;
    }
}

```

8. Cree el constructor por defecto y el constructor parametrizado.

```
/// <summary>
/// Constructor por defecto
/// </summary>
public Person()
{
    // Identificador de la persona
    person_Id = 0;
    // Nombres de la Persona
    firstName = string.Empty;
    // Apellidos de la Persona
    lastName = string.Empty;
    // Fecha de Nacimiento (dd-mm-yyyy)
    birthDay = DateTime.Now;
    // Sexo (M,F)
    sex = 'M';
}

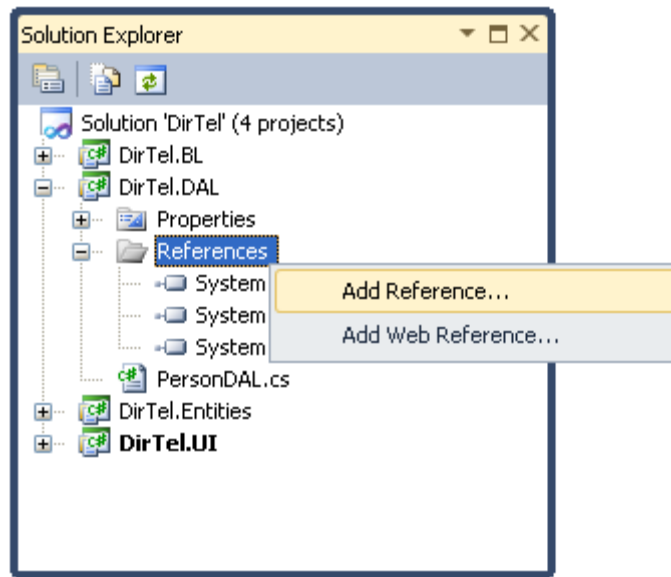
/// <summary>
/// Constructor con campos
/// </summary>
/// <param name="firstName">Nombres</param>
/// <param name="lastName">Apellidos</param>
/// <param name="birthDay">Fecha de Nacimiento</param>
/// <param name="sex">Sexo (M,F)</param>
public Person(string firstName, string lastName, DateTime birthDay, char sex)
{
    // Nombres de la Persona
    this.firstName = firstName;
    // Apellidos de la Persona
    this.lastName = lastName;
    // Fecha de Nacimiento (dd-mm-yyyy)
    this.birthDay = birthDay;
    // Sexo (M,F)
    this.sex = sex;
}
```

9. Guarde los cambios.

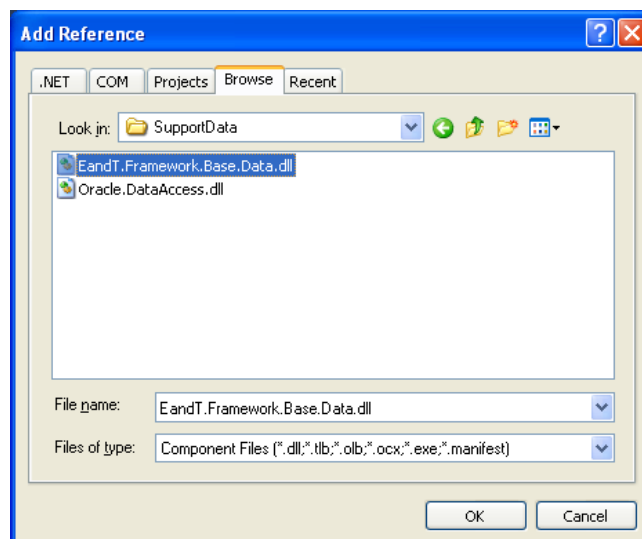
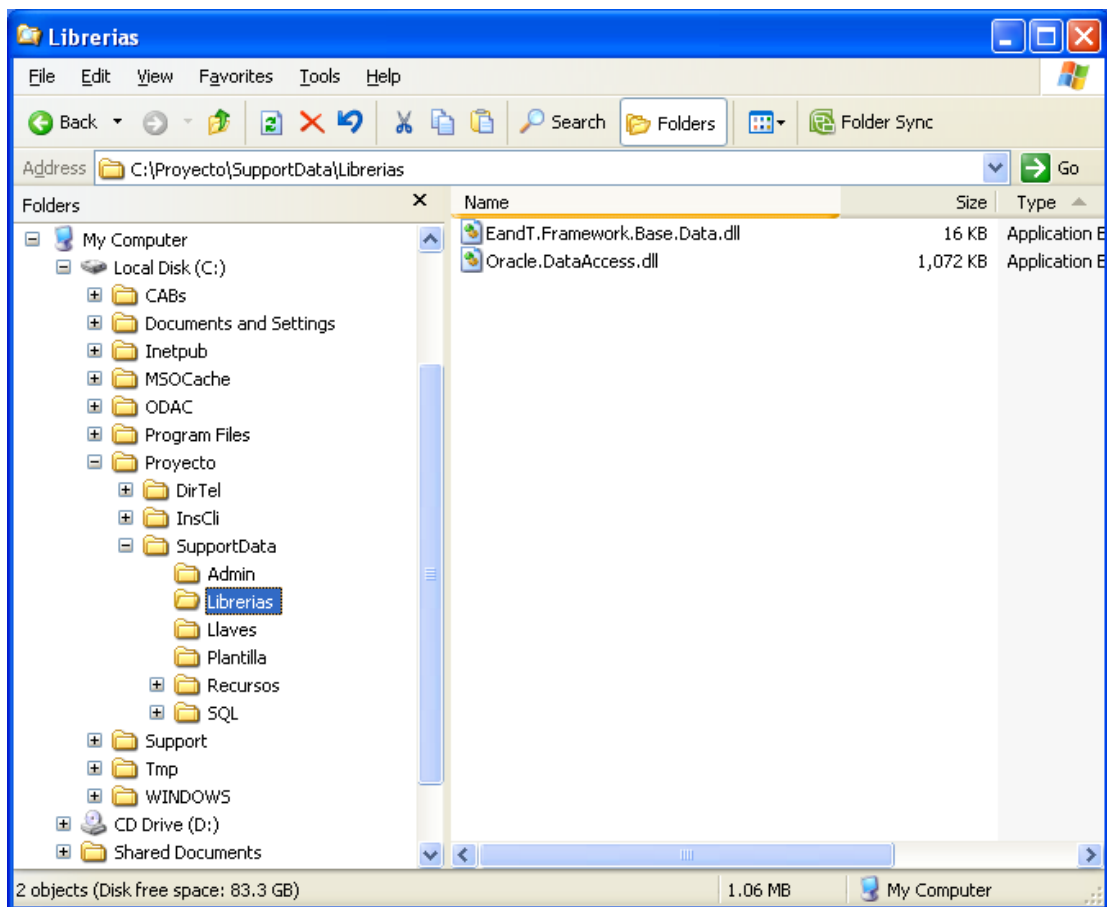
Lógica de Acceso a Datos (DAL)

En este punto vamos a hacer e implementar la lógica de acceso a datos para poder obtener la información de las personas, para ello haremos uso de la conexión a la base de datos, que para efectos de este curso apunta a una dirección IP Virtual que debe ser reemplazada por la suministrada por el Instructor. Adicionalmente utilizaremos una librería demo (**EandT.Framework.Base.Data.Dll**) para la conexión a la base de datos, además de referenciar a la clase principal de conexión por el proveedor de datos de ORACLE (ODP – ORACLE Data Provider).

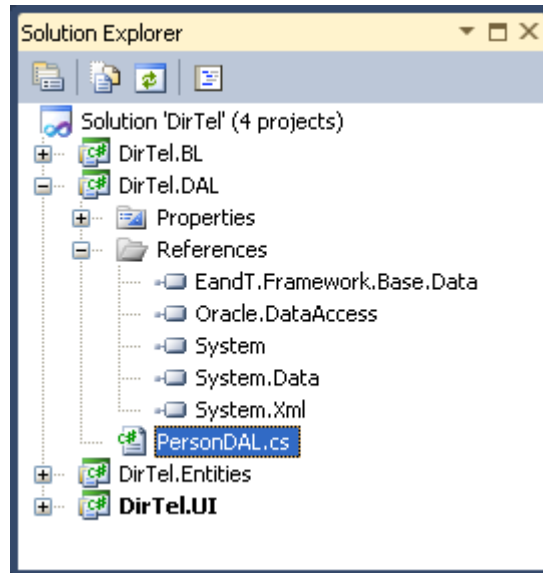
1. Adicionaremos las referencias a las librerías de ODP de ORACLE y la librería demo de conexión a base de datos EandT.Framework.Base.Data.dll. para ello seleccionaremos la sección de referencias del proyecto DAL y haremos click derecho, del menú contextual seleccionamos la opción Adicionar Referencia (Add Reference).



2. Luego de la ventana de referencia buscamos la librería EandT.Framework.Base.Data.dll del directorio **SupportData\Librerias**.



3. Luego del Directorio de **SupportData\Bibliotecas** seleccionamos la librería de **Oracle.DataAccess.Dll**.
4. Ahora adicionamos una nueva clase al proyecto **DirTel.DAL** y la llamaremos **PersonDAL.cs**



5. Modificamos la clase adicionando la cabecera de la documentación.

```

PersonDAL.cs
DirTel.DAL, PersonDAL
#region Documentación
/*
*****
* Propiedad intelectual de Engineers & Tools (c).
*****
* Unidad      : PersonDAL.cs
* Descripcion : Permite acceder a los datos de la persona
* Autor       : Julio Cesar Robles Uribe - Jucer
* Fecha       : 23-May-2010
*
* Fecha      Autor      Modificación
* =====
* 23-May-2010 Jucer      1 - Version Inicial
*****
*/
#endregion Documentación

```


6. Incluiremos los **using** necesarios para el manejo de las librerías de acceso a datos y configuración en la sección de librerías así:

```
PersonDAL.cs
DirTel.DAL.PersonDAL

*****
* Clase      : PersonDAL
* Descripcion : Permite acceder a los datos de la persona
* Autor      : Julio Cesar Robles Uribe - Jucer
* Fecha      : 23-May-2010
*
* Fecha      Autor      Modificación
* =====
* 23-May-2010 Jucer      1 - Version Inicial
***** /

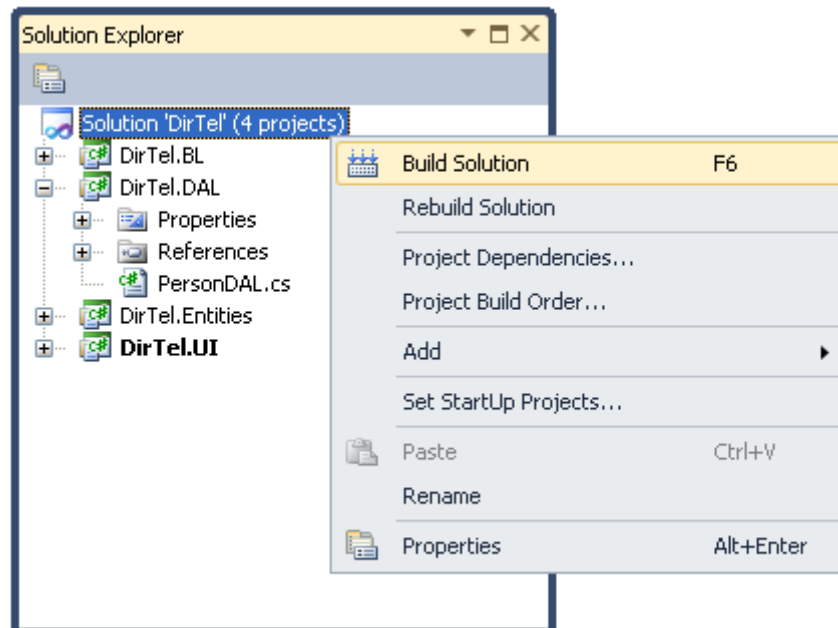
#endregion Documentación

// Librerías
using System;
using System.Collections.Generic;
using System.Text;
// Librerías de ORACLE
using Oracle.DataAccess.Client;
using Oracle.DataAccess.Types;
// Librerías de SQLHelper
using EandT.Framework.Base.Data;

// Namespace
namespace DirTel.DAL
{

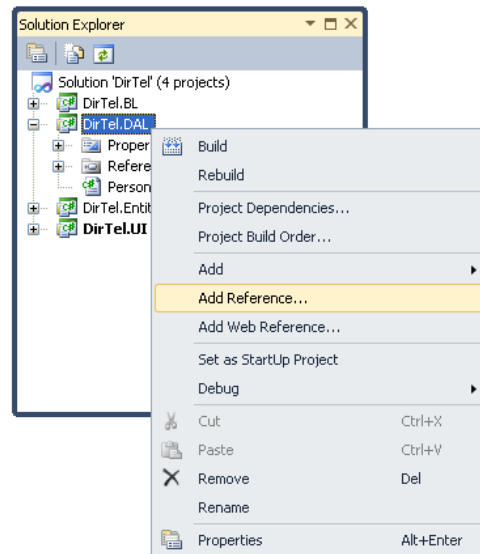
```

7. En este momento y para hacer la inclusión de las entidades compilaremos toda la solución, haciendo clic derecho sobre la solución y en el menú contextual escogemos la opción Compilar la Solución (Build Solution).

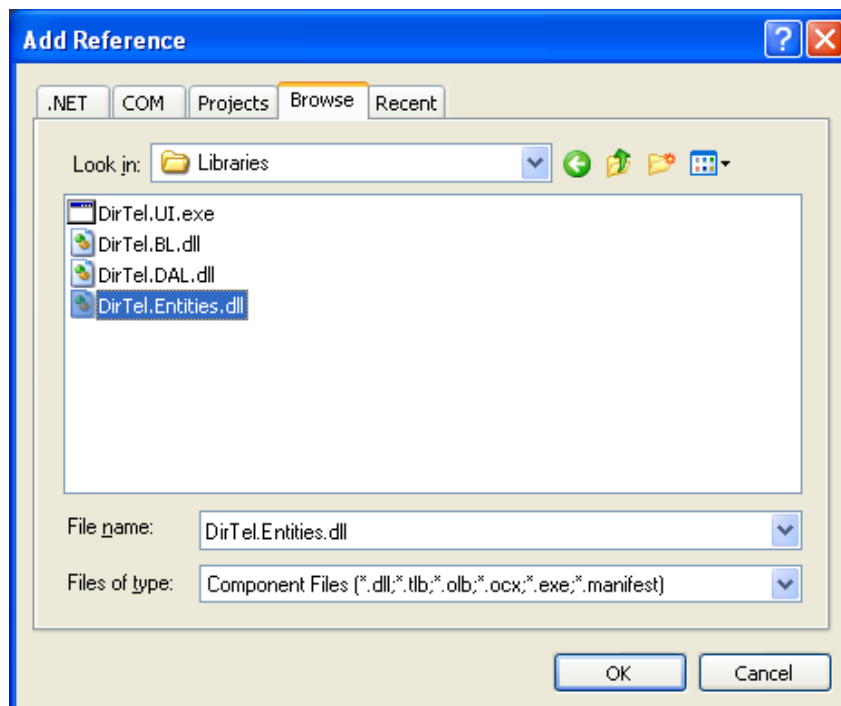


8. Esta acción genera las librerías de todos los proyectos y por la parametrización de las propiedades de los proyectos, todas quedan referencias al directorio **C:\Proyecto\DirTel\Libraries**. Este directorio (**Libraries**) será nuestra referencia de ahora en adelante para adicionar librerías creadas dentro de la solución.

9. Después de haber generado las librerías adicionaremos la referencia de la librería de las entidades DirTel.entities.Dll así:
- Click derecho sobre el proyecto DirTel.DAL y seleccionar la opción de Adicionar Referencia (Add Reference)

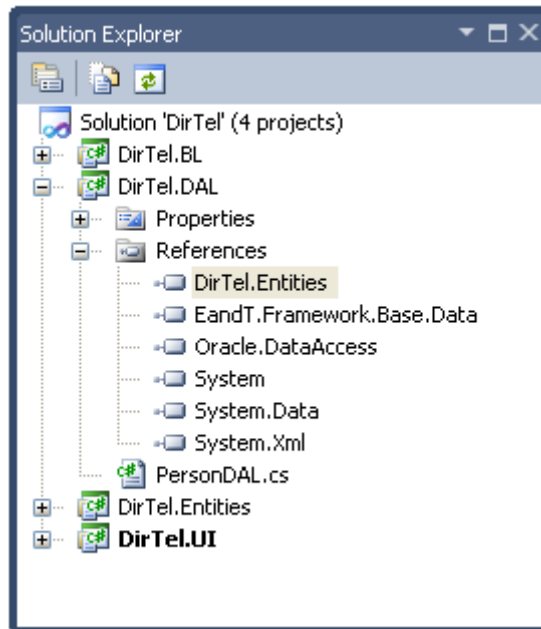


- Escoger la pestaña de examinar (Browse) y navegar hasta el directorio **C:\Proyecto\DirTel\Libraries**.



- Seleccionar la librería DirTel.Entities.dll

d. Al finalizar tendremos una referencia a la librería de Entidades así:



10. Ahora adicionaremos el **using** necesarios para hacer uso de esta librería así:

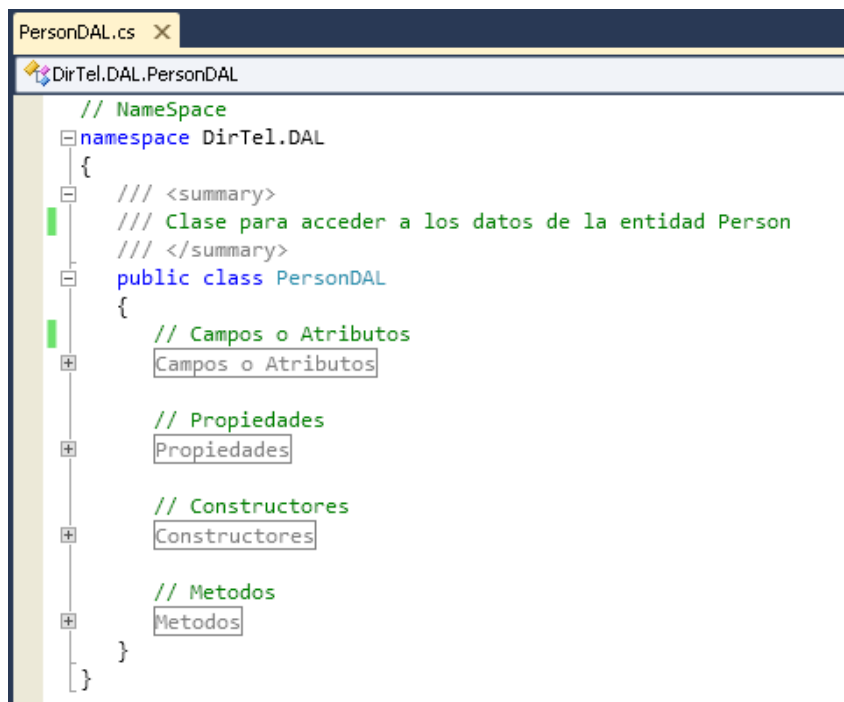
```
PersonDAL.cs
DirTel.DAL.PersonDAL

* Fecha      Autor      Modificación
* =====
* 23-May-2010 Jucer      1 - Version Inicial
*****/
#endregion Documentación

// Librerías
using System;
using System.Collections.Generic;
using System.Text;
// Librerías de ORACLE
using Oracle.DataAccess.Client;
using Oracle.DataAccess.Types;
// Librerías de SQLHelper
using EandT.Framework.Base.Data;
// Librerías de DirTel
using DirTel.Entities;

// Namespace
namespace DirTel.DAL
{
}
```

11. Ahora modificaremos la clase cambiando el alcance, poniéndole **public** y le adicionaremos la documentación propia de la misma.



```
// NameSpace
namespace DirTel.DAL
{
    /// <summary>
    /// Clase para acceder a los datos de la entidad Person
    /// </summary>
    public class PersonDAL
    {
        // Campos o Atributos
        Campos o Atributos

        // Propiedades
        Propiedades

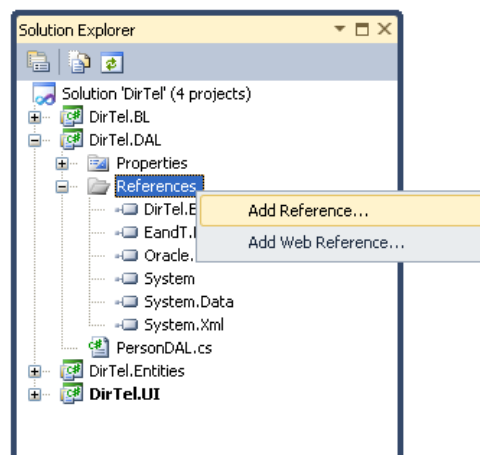
        // Constructores
        Constructores

        // Metodos
        Metodos
    }
}
```

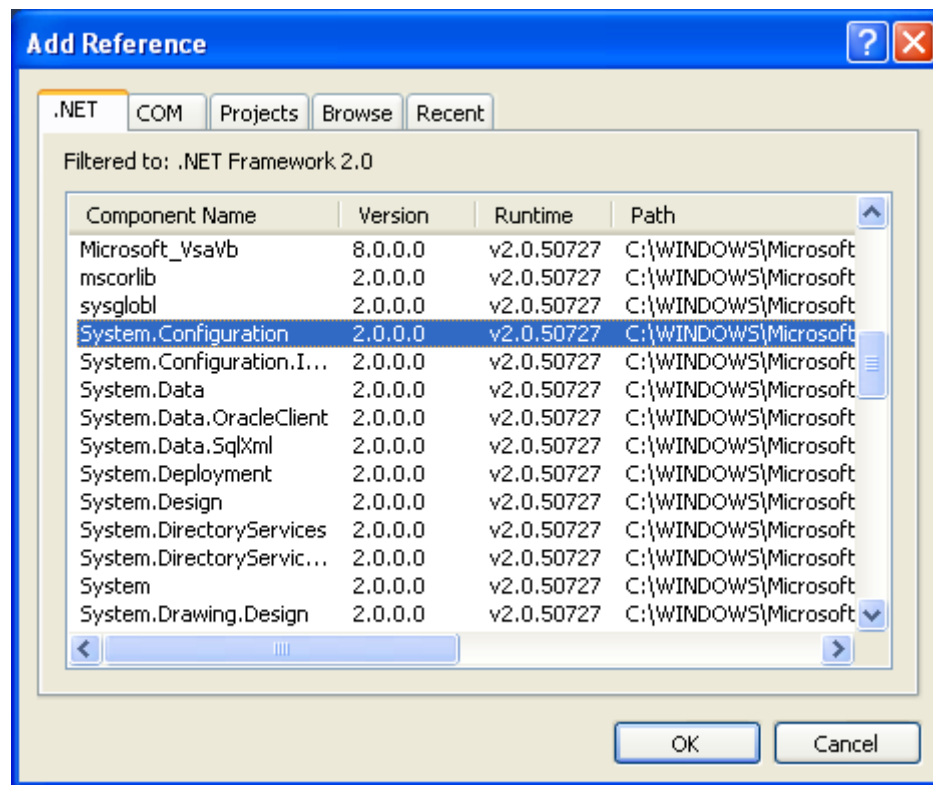
12. Creamos los campos o atributos necesarios para manejar la conexión a la base de datos.

```
// Cadena de conexion a la Base de datos
private string strCnx;
// Variable para manejar el enlace a la base de datos
private DataBase db;
// Variable para manejar la conexion a la base de datos
private IDbConnection cnx;
```

13. Referenciamos la librería de .Net para acceder a los recursos de configuración y poder utilizar los ítems descritos en el archivo App.config.



14. Seleccionamos del TAB de .Net la librería de System.Configuration.



15. Adicionaremos los **using** necesarios para acceder a la configuración y para manejar los tipos especiales de la base de datos. (System.Configuration, System.Data)

```

PersonDAL.cs
DirTel.DAL.PersonDAL
// Librerías
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Data;
using System.Text;
// Librerías de ORACLE
using Oracle.DataAccess.Client;
using Oracle.DataAccess.Types;
// Librerías de SQLHelper
using EandT.Framework.Base.Data;
// Librerías de DirTel
using DirTel.Entities;

// Namespace
namespace DirTel.DAL

```

16. Creamos el constructor por defecto y utilizamos la configuración para obtener los datos de la cadena de conexión, en este caso utilizaremos el archivo de configuración de la aplicación (**App.config**), que para efectos del ejercicio esta dentro del proyecto de **DirTel.UI** y que ya hemos modificado en la creación del proyecto. Modificando el constructor tendremos lo siguiente:

```
/// <summary>
/// Constructor por defecto
/// </summary>
public PersonDAL()
{
    // Incicializar la cadena de conexion
    strCnx = ConfigurationManager.ConnectionStrings["DirTelCnx"].ConnectionString;
}
```

17. Ahora desarrollaremos los métodos necesarios para acceder a la información de la base de datos. Para ellos tendremos los siguientes métodos:



```
PersonDAL.cs
DirTel.DAL.PersonDAL

//Campos o Atributos
Campos o Atributos

//Constructores
Constructores

//Metodos
#region Metodos

//Publicos
#region Publicos
/// <summary> ...
public long CreatePerson(Person person)...

/// <summary> ...
public IList<Person> ReadAllPersons()...

/// <summary> ...
public void DeletePerson(long person_Id)...

/// <summary> ...
public void UpdatePerson(Person person)...

#endregion Publicos

#endregion Metodos
```

18. Ahora desarrollaremos el contenido de los métodos.

```
// Metodos
#region Metodos

// Publicos
#region Publicos
/// <summary>
/// Inserta los datos de la Persona
/// </summary>
/// <param name="person">Entidad con los datos de la persona</param>
/// <returns>Identificador de la person</returns>
public long CreatePerson(Person person)
{
    // Variable para retornar el id de la persona insertada
    long person_Id=0;

    // Establecer la conexion a la base de datos
    db = new DataBase(ProviderType.Oracle, strCnx);

    // Crear la Conexion
    cnx = db.CreateAndOpenConnection();

    // Crear el Comando
    IDbCommand cmd = db.CreateCommand(cnx);
    cmd.CommandText = "DA_PERSONS_PKG.CreatePerson";
    cmd.CommandType = CommandType.StoredProcedure;
    // Asignar los parametros
    db.AddInParameter(cmd, "pm_FirstName", person.FirstName, DbType.String);
    db.AddInParameter(cmd, "pm_LastName", person.LastName, DbType.String);
    db.AddInParameter(cmd, "pm_BirthDay", person.BirthDay, DbType.Date);
    db.AddInParameter(cmd, "pm_Sex", person.Sex, DbType.String);
    // Adicionar el parametro de salida
    db.AddOutParameter(cmd, "pm_Person_Id", null, DbType.Int64);

    //Ejecutar el procedimiento
    int rowsAffected = db.ExecuteNonQuery(cnx, cmd);

    // Referencia rl parametro de salida para el Id de la persona
    OracleParameter outPerson_Id = cmd.Parameters[db.ParameterToken() +
    "pm_Person_Id"] as OracleParameter;

    // Obtener el valor del parametro
    person_Id = Convert.ToInt64(outPerson_Id.Value);

    //Cerrar la conexion
    db.CloseConnection(cnx);

    //retornar el valor
    return person_Id;
}
```

```

/// <summary>
/// Obtener todos los datos de la tabla PERSONS
/// </summary>
/// <returns>Lista con los datos de las personas</returns>
public IList<Person> ReadAllPersons()
{
    // Crear la lista para retornar los datos
    IList<Person> lstPersons = new List<Person>();

    // Establecer la conexion a la base de datos
    db = new DataBase(ProviderType.Oracle, strCnx);

    // Crear la Conexion
    cnx = db.CreateAndOpenConnection();

    // Crear el Comando especial para ORACLE
    OracleCommand cmd = (OracleCommand)db.CreateCommand(cnx,
CommandType.StoredProcedure, "DA_PERSONS_PKG.ReadAllPersons");

    // Crear el Parametro enlazado al Cursor Referenciado
    OracleParameter paramRefCursor = db.AddCommandRefCurParameter(cmd,
"cur_Persons", ParameterDirection.Output, null);

    // Ejecutar el query
    db.ExecuteNonQuery(cnx, cmd);

    // Limpiar la lista
    string strData = string.Empty;

    // Obtener el Cursor Referenciado
    OracleRefCursor refCur = (OracleRefCursor)paramRefCursor.Value;
    // Obtener el DataReader desde el cursor referenciado
    OracleDataReader dr = refCur.GetDataReader();

    // Ciclo para Recorer los datos
    while (dr.Read())
    {
        // Variable Person para obtener los datos
        Person person = new Person();

        // Leer los valores
        person.Person_Id = Convert.ToInt64(dr["Person_Id"].ToString());
        person.FirstName = dr["FirstName"].ToString();
        person.LastName = dr["LastName"].ToString();
        person.BirthDay = Convert.ToDateTime(dr["BirthDay"].ToString());
        person.Sex = dr["Sex"].ToString()[0];

        // Adicionar el Valor a la lista
        lstPersons.Add(person);
    }

    // Cerar la Conexion
    db.CloseConnection(cnx);

    // retornar la lista
    return lstPersons;
}

```



```

/// <summary>
/// Borrar el Registro de la tabla segun el Id
/// </summary>
/// <param name="person_Id">Id de la persona</param>
public void DeletePerson(long person_Id)
{
    // Establecer la conexion a la base de datos
    db = new DataBase(ProviderType.Oracle, strCnx);

    // Crear la Conexion
    cnx = db.CreateAndOpenConnection();

    // Crear el Comando
    IDbCommand cmd = db.CreateCommand(cnx);
    cmd.CommandText = "DA_PERSONS_PKG.DeletePerson";
    cmd.CommandType = CommandType.StoredProcedure;
    // Asignar los parametros
    db.AddInParameter(cmd, "pm_Person_Id", person_Id, DbType.Int64);

    // Ejecutar el procedimiento
    int rowsAffected = db.ExecuteNonQuery(cnx, cmd);

    // Cerrar la conexion
    db.CloseConnection(cnx);
}

/// <summary>
/// Actualiza los datos de la persona
/// </summary>
/// <param name="person">Datos de la Persona</param>
public void UpdatePerson(Person person)
{
    // Establecer la conexion a la base de datos
    db = new DataBase(ProviderType.Oracle, strCnx);

    // Crear la Conexion
    cnx = db.CreateAndOpenConnection();

    // Crear el Comando
    IDbCommand cmd = db.CreateCommand(cnx);
    cmd.CommandText = "DA_PERSONS_PKG.UpdatePerson";
    cmd.CommandType = CommandType.StoredProcedure;
    // Asignar los parametros
    db.AddInParameter(cmd, "pm_Person_Id", person.Person_Id, DbType.Int64);
    db.AddInParameter(cmd, "pm_FirstName", person.FirstName, DbType.String);
    db.AddInParameter(cmd, "pm_LastName", person.LastName, DbType.String);
    db.AddInParameter(cmd, "pm_BirthDay", person.BirthDay, DbType.Date);
    db.AddInParameter(cmd, "pm_Sex", person.Sex, DbType.String);

    // Ejecutar el procedimiento
    int rowsAffected = db.ExecuteNonQuery(cnx, cmd);

    // Cerrar la conexion
    db.CloseConnection(cnx);
}

#endregion Publicos

#endregion Metodos

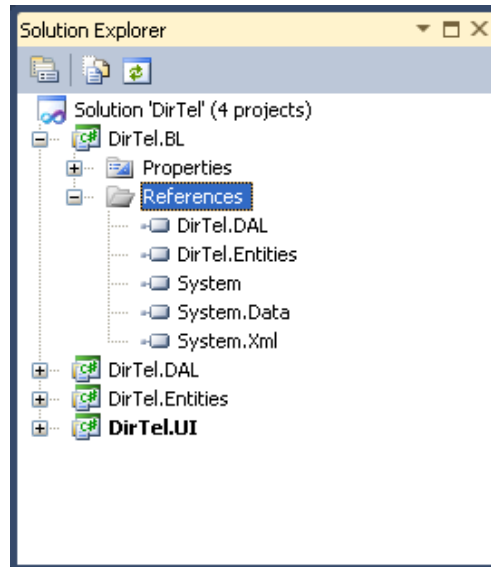
```

19. Compilamos presionando la tecla **F6** y Guardamos los cambios.

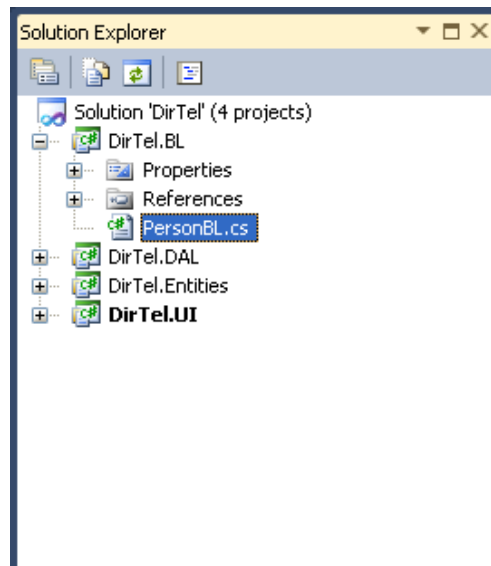
Lógica de Negocios (BL)

Ahora implementaremos la lógica de negocios de la aplicación que para este punto del proceso, es un simple llamado al acceso a datos.

1. Adicionamos las referencias a las librerías de Entidades (**DirTel.Entities.dll**) y las de acceso a datos (**DirTel.DAL.dll**) al proyecto de **DirTel.BL**. Tener en cuenta que estas referencias deben hacerse desde el directorio de **C:\Proyecto\DirTel\Libraries** y no referenciando los proyectos actuales, pues el objetivo es independizar cada proyecto de los demás.



2. Ahora adicione la clase **PersonBL**.



3. Ahora modifique el encabezado de la clase para incluir la documentación así:

```
PersonBL.cs
DirTel.BL.PersonBL

#region Documentación
/
* Propiedad intelectual de Engineers & Tools (c).
*
* Unidad      : PersonBL.cs
* Descripción : Clase para Manejar la Logica de Negocios para la entidad Person
* Autor       : Julio Cesar Robles Uribe - Jucer
* Fecha       : 24-May-2010
*
* Fecha      Autor      Modificación
* =====
* 24-May-2010 Jucer      1 - Version Inicial
/
#endregion Documentación
```

4. Modifique la sección de librerías y adicione los **using** a las librerías de entidades y acceso a datos, Note que en esta capa solo se referencia **DAL** y **Entities**.

```
PersonBL.cs
DirTel.BL.PersonBL

* Fecha      Autor      Modificación
* =====
* 24-May-2010 Jucer      1 - Version Inicial
/
#endregion Documentación

// Librerias
using System;
using System.Collections.Generic;
using System.Text;
//Librerias de DirTel
using DirTel.Entities;
using DirTel.DAL;

// Namespace
namespace DirTel.BL
```

5. Ahora modifique el alcance de la clase a **public** y adicione la documentación de la misma así:

```
PersonBL.cs
DirTel.BL.PersonBL

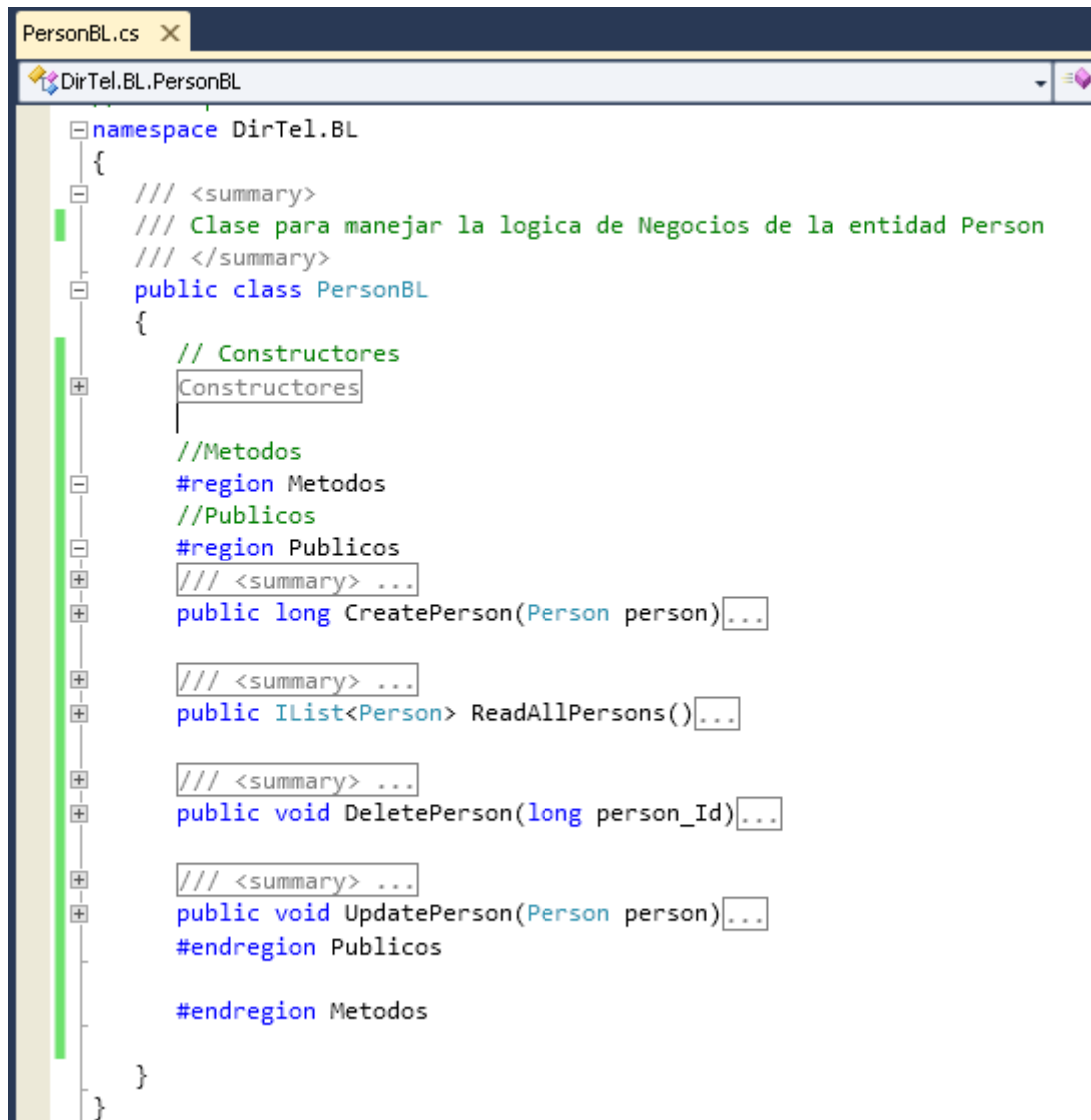
// Namespace
namespace DirTel.BL
{
    /// <summary>
    /// Clase para manejar la logica de Negocios de la entidad Person
    /// </summary>
    public class PersonBL
    {
        // Constructores
        Constructores

        // Metodos
        Metodos
    }
}
```

6. Cree un constructor vacio para tener más control sobre la case BL

```
/// <summary>
/// Constructor por defecto
/// </summary>
public PersonBL()
{
}
```

7. Adiciones los métodos para referenciar la capa de acceso a datos así:



8. Ahora desarrolle el contenido de los Métodos así:

```
//Metodos
#region Metodos
//Publicos
#region Publicos
/// <summary>
/// Inserta los datos de la Persona
/// </summary>
/// <param name="person">Entidad con los datos de la persona</param>
/// <returns>Identificador de la person</returns>
public long CreatePerson(Person person)
{
    // Crear el Objeto de Datos
    PersonDAL personDAL = new PersonDAL();

    // Ejecutar el Comando y retornar el id de la persona insertada
    long person_Id = personDAL.CreatePerson(person);

    // retornar el valor
    return person_Id;
}

/// <summary>
/// Obtener todos los datos de las personas
/// </summary>
/// <returns>Lista con las personas</returns>
public IList<Person> ReadAllPersons()
{
    // Crear el Objeto de Datos
    PersonDAL personDAL = new PersonDAL();

    // Ejecutar el Comando
    IList<Person> lstPersons = personDAL.ReadAllPersons();

    return lstPersons;
}

/// <summary>
/// Borrar el Registro de la tabla segun el Id
/// </summary>
/// <param name="person_Id">Id de la persona</param>
public void DeletePerson(long person_Id)
{
    // Crear el Objeto de Datos
    PersonDAL personDAL = new PersonDAL();

    // Ejecutar el Comando
    personDAL.DeletePerson(person_Id);
}

/// <summary>
/// Actualiza los datos de la persona
/// </summary>
/// <param name="person">Entidad Persona</param>
public void UpdatePerson(Person person)
{
    // Crear el Objeto de Datos
    PersonDAL personDAL = new PersonDAL();

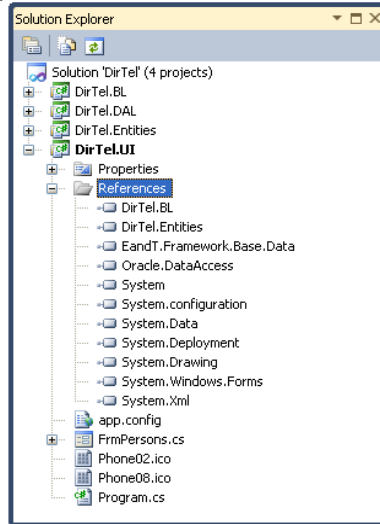
    // Ejecutar el Comando
    personDAL.UpdatePerson(person);
}
}
#endregion Publicos
#endregion Metodos
```

9. Compile el proyecto y guarde los cambios.

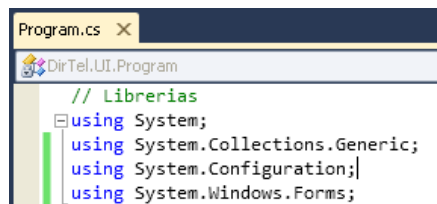
Interfaz de Usuario (UI)

En este punto vamos a utilizar lo desarrollado hasta el momento y vamos a modificar el proyecto **UI** para hacer uso de las capas de **BL** y **Entities**.

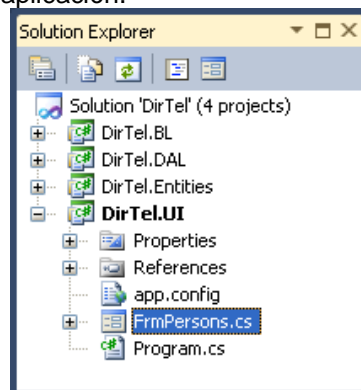
1. Adicione al proyecto de **DirTel.UI** las referencias a las librerías de **DirTel.BL.dll** y **DirTel.Entities.dll**.Tenga presente utilizar las librerías del directorio **Libraries** y no hacer referencia a los proyectos locales. Adicionalmente haga referencia a la librería de configuración **System.configuration**, la librería de **ORACLE.DataAccess.Dll** y la librería de SQLHelper **EandT.Framework.Base.Data**.



2. Agregue el using correspondiente a la librería para el manejo de la configuración, dentro del archivo de **Program.cs**.



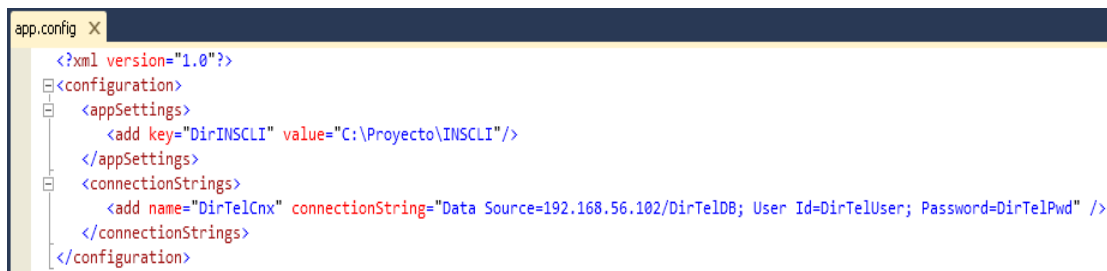
3. Agregue un nuevo formulario llamado **FrmPersons** para que sea, por ahora, el formulario inicial de la aplicación.



4. Modifique el archivo de **Program.cs** quitando el comentario y haciendo referencia al nuevo formulario ya creado.

```
// Ejecutar el llamado al formulario Principal
Application.Run(new FrmPersons());
```

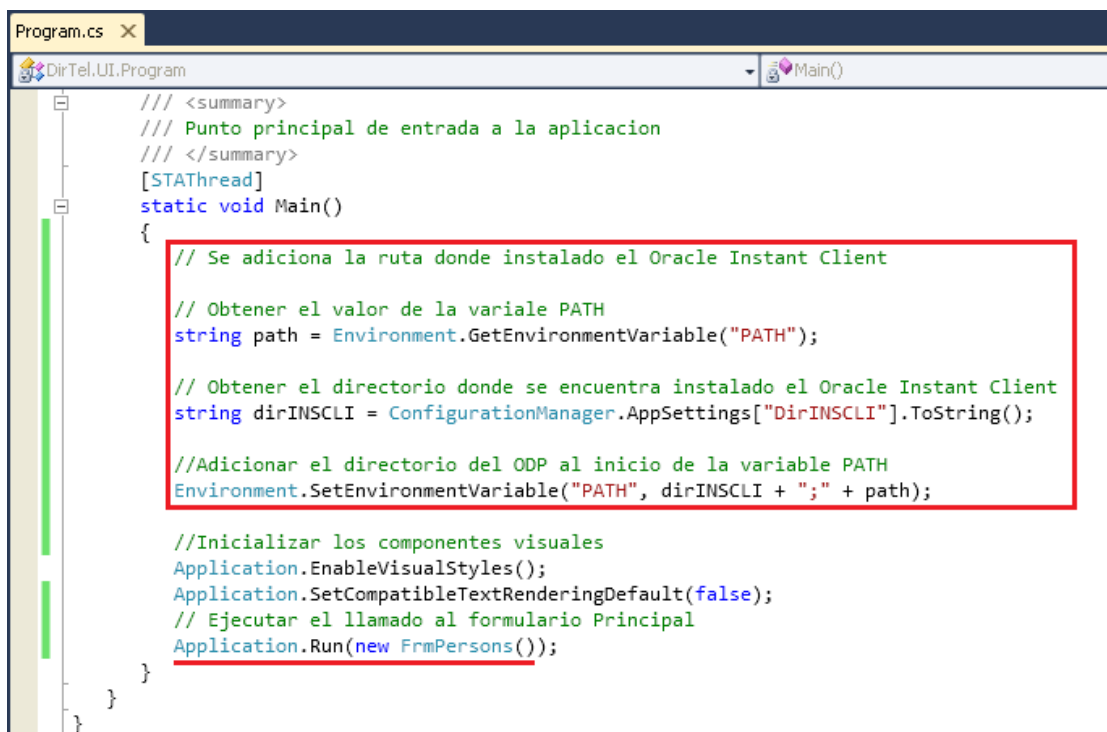
5. Modifique el archivo de configuración **App.config**, mencionado en puntos anteriores para crear la sección de **AppSettings** en donde se pueden definir llaves de configuración para la aplicación en este caso vamos a crear la llave **DirINSCLI** para referenciar en donde se encuentra el directorio de las librerías que permiten conectarse a ORACLE, para el caso concreto del proyecto las librerías del Oracle Instant client.



```
<?xml version="1.0"?>
<configuration>
  <appSettings>
    <add key="DirINSCLI" value="C:\Proyecto\INSCLI"/>
  </appSettings>
  <connectionStrings>
    <add name="DirTelCnx" connectionString="Data Source=192.168.56.102/DirTelDB; User Id=DirTelUser; Password=DirTelPwd" />
  </connectionStrings>
</configuration>
```

```
<appSettings>
  <add key="DirINSCLI" value="C:\Proyecto\INSCLI"/>
</appSettings>
```

6. Para poder hacer uso de este directorio como base para que la conexión funcione se debe modificar el archivo **Program.cs** así:



```
/// <summary>
/// Punto principal de entrada a la aplicacion
/// </summary>
[STAThread]
static void Main()
{
    // Se adiciona la ruta donde instalado el Oracle Instant Client

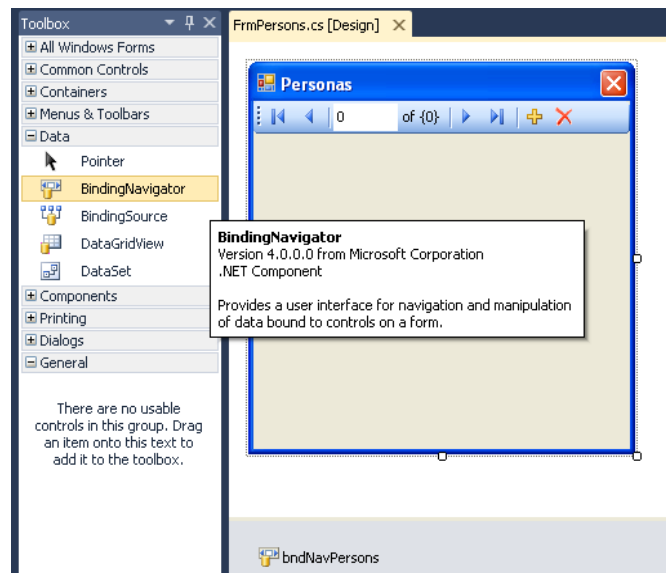
    // Obtener el valor de la variable PATH
    string path = Environment.GetEnvironmentVariable("PATH");

    // Obtener el directorio donde se encuentra instalado el Oracle Instant Client
    string dirINSCLI = ConfigurationManager.AppSettings["DirINSCLI"].ToString();

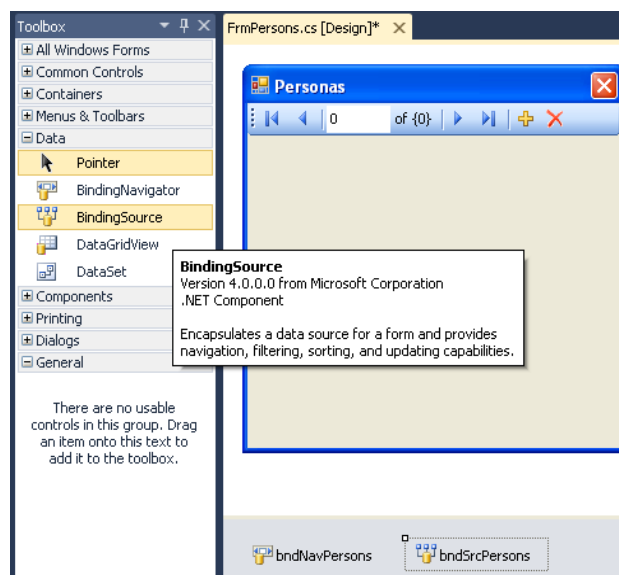
    //Adicionar el directorio del ODP al inicio de la variable PATH
    Environment.SetEnvironmentVariable("PATH", dirINSCLI + ";" + path);

    //Inicializar los componentes visuales
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);
    // Ejecutar el llamado al formulario Principal
    Application.Run(new FrmPersons());
}
```

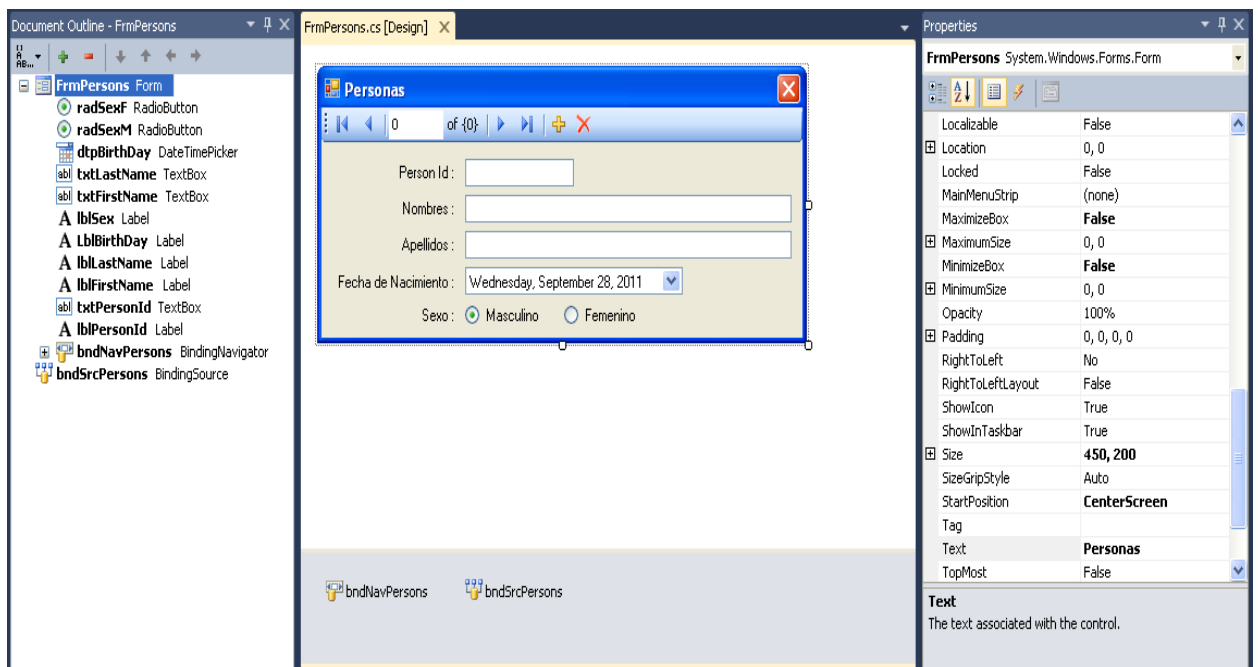
7. Modifique el formulario **FrmPersons** y cambie las siguientes propiedades:
 - a. **AutoSizeMode**=GrowAndShrink (No permitir que se redimensione)
 - b. **MaximizeBox**=False (Quitar el botón de Maximizar)
 - c. **MinimizeBox**=False (Quitar el botón de Minimizar)
 - d. **Text**=Personas (Titulo del Formulario)
 - e. **StartPosition**=CenterScreen (Poner centrado el formulario)
8. Ahora vamos a agregar los controles al formulario, para ello iniciaremos agregando un navegador enlazado de registros (BindingNavigator), que lo podemos encontrar en el conjunto de controles de tipo **Data** de la caja de herramientas y controles y cambie el nombre del control **bindingNavigator1** por **bndNavPersons**.



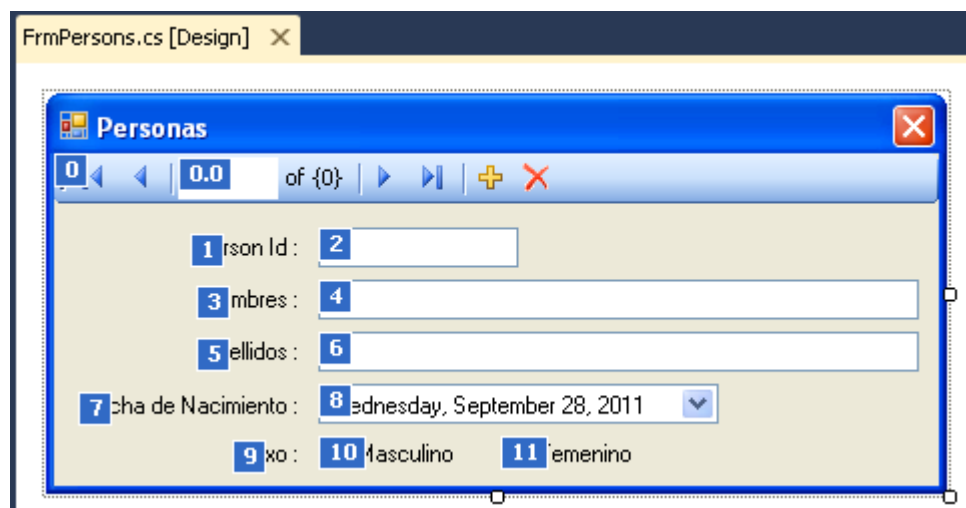
9. De este mismo grupo adicione un control de tipo fuente de datos enlazado (BindingSource) y cambie el nombre de **bindingSource1** a **bndSrcPersons**.



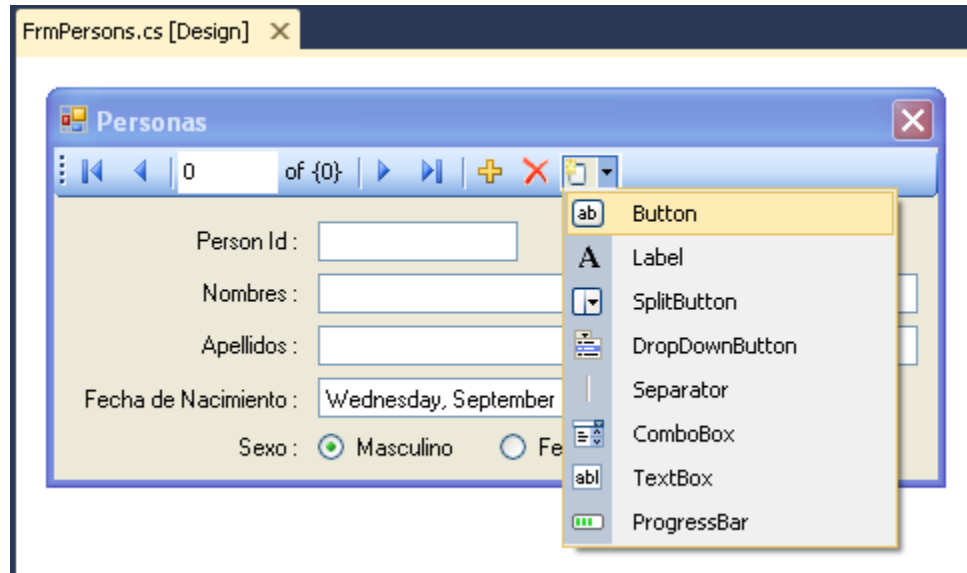
10. Modifique el formulario en tamaño y agregando los controles respectivos hasta completar la siguiente presentación:



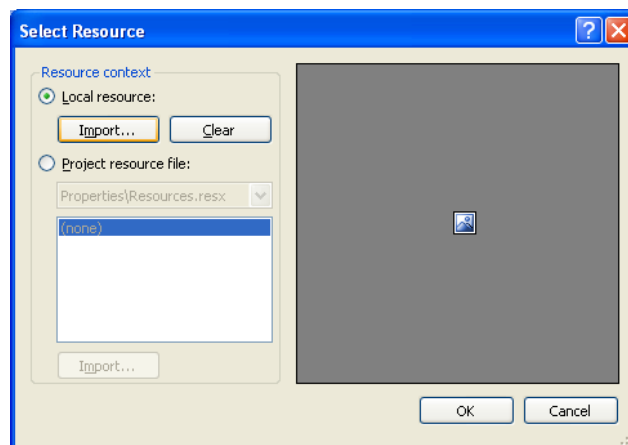
11. Asigne los nombre de los controles de entrada de datos de la siguiente forma:
- txtPersonId
 - txtFirstName
 - txtLastName
 - dtpBirthDay
 - radSexM (Activar Propiedad **Checked**=True)
 - radSexF
12. Modifique el Tab Order para que quede consecutivo el orden de navegación entre los controles.



13. Vamos a adicionar un botón nuevo al Navegador de datos y lo denominaremos SaveItems. Para ello seleccionamos la barra del Navegador y el ultimo botón abrimos el menú desplegable y escogemos la opción que hace referencia al tipo Botón así:

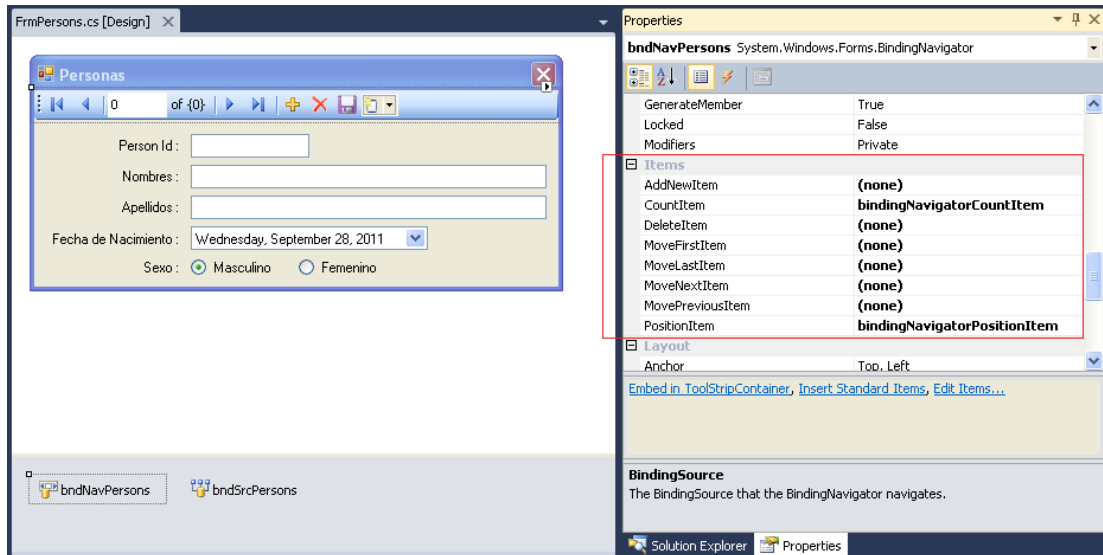


14. Cambiamos las propiedades de este nuevo botón de la siguiente forma:
- a. **Name=** bindingNavigatorSaveItems
 - b. **Text=** SaveItems
 - c. **ToolTipText=** Save Items
15. Seleccionamos el nuevo botón y desde la ventana de propiedades escogemos la propiedad **Image**, para lo cual se despliega una ventana como la siguiente:

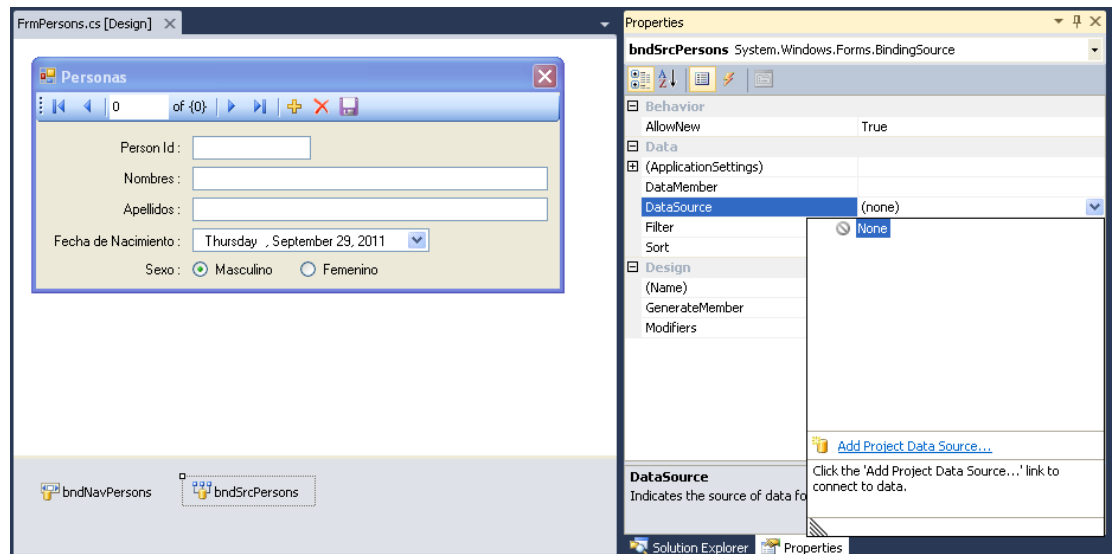


16. Escogemos la opción de importar del contexto local para que la imagen quede asociada al formulario y no al proyecto, esto nos facilita que al copiar el formulario a otra solución o proyecto este se vaya con la imagen incrustada dentro del formulario.
17. Escogemos del directorio de **C:\Proyecto\SupportData\Recursos\Png**, la imagen de **bindingNavigatorSaveItem.Image.png** y aceptamos el nuevo recurso

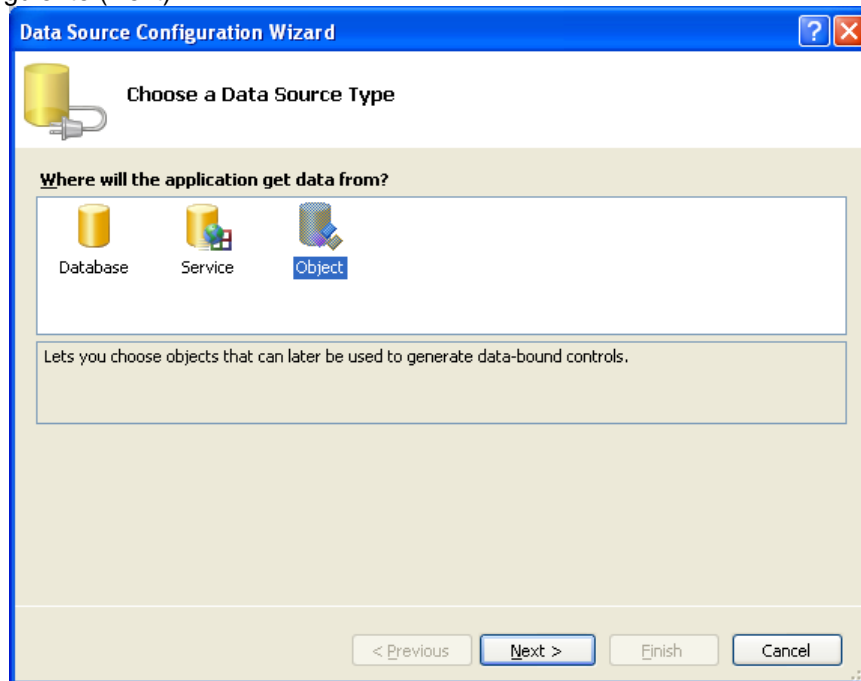
18. Ahora seleccionando nuevamente todo el Navegador (**bndNavPersons**) vamos a modificarlo eliminando las funciones de movimiento entre los ítems y las funciones de adición y eliminación ya que se van a personalizar en este ejercicio. Para facilitar esta modificación organizamos las propiedades por categorías.



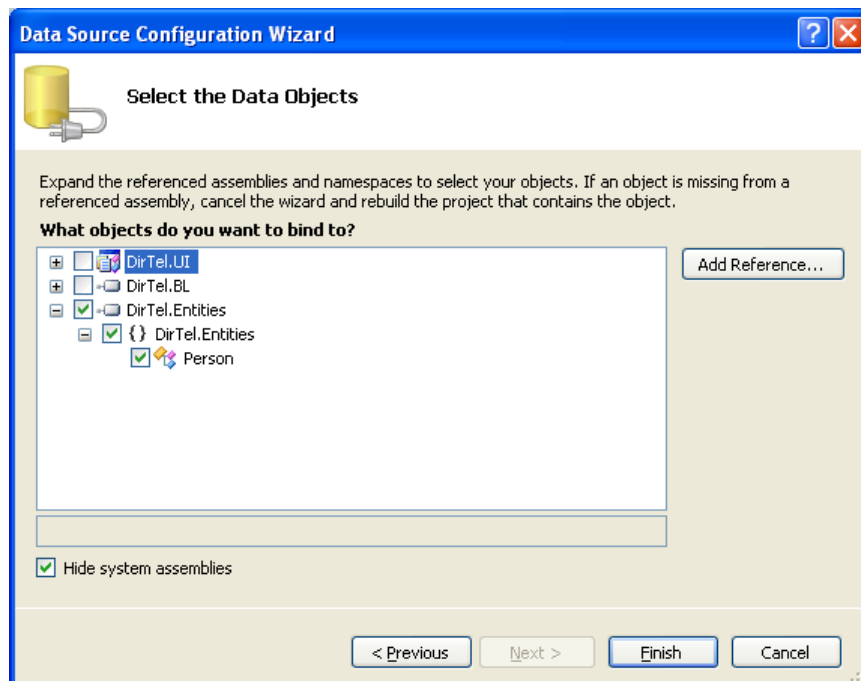
19. Cambiamos su valor a ninguno (none) para todas a excepción de **CountItem** y **PositionItem**.
20. Ahora vamos a enlazar la capa de presentación (UI) con la capa de lógica de negocio (BL) y referenciaremos las entidades. Para ello primero cambiaremos la fuente de datos del control **bndSrcPersons**, modificando la propiedad **DataSource**, adicionando una nueva fuente de datos.



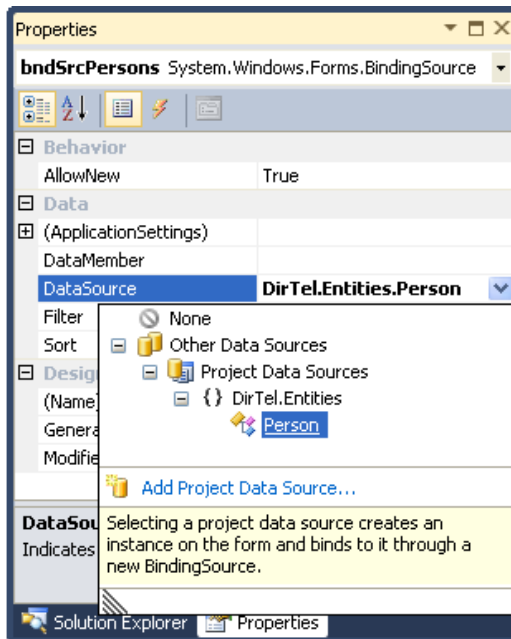
21. Creamos la fuente de datos del tipo objeto (**ObjectDataSource**) y presionamos siguiente (Next)



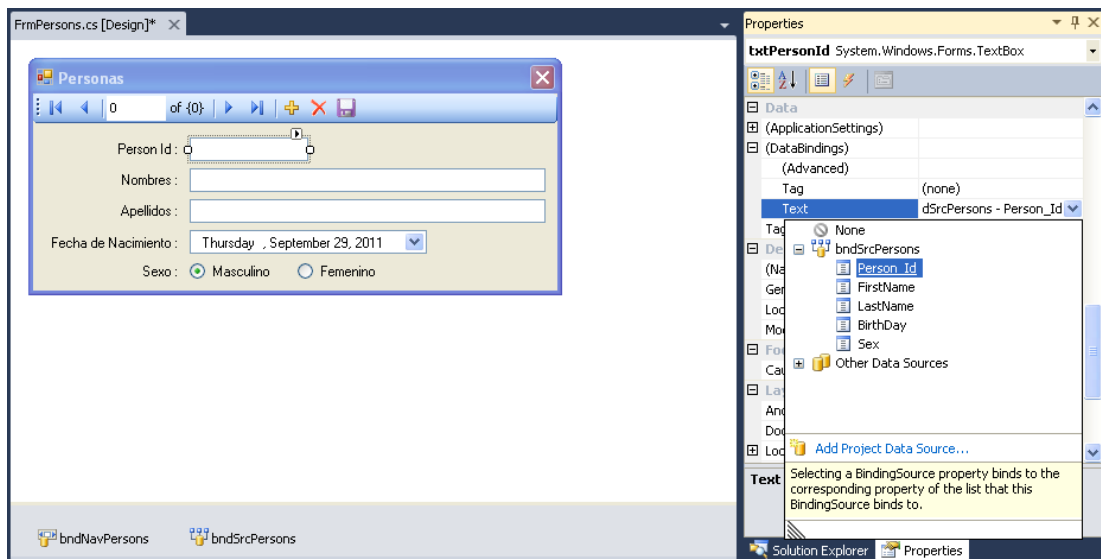
22. Seleccionamos la entidad relacionada y presionamos siguiente (Next) y por ultimo finalizamos (Finish)



23. Al finalizar queda registrado el enlace a la librería de **DirTel.Entites**.

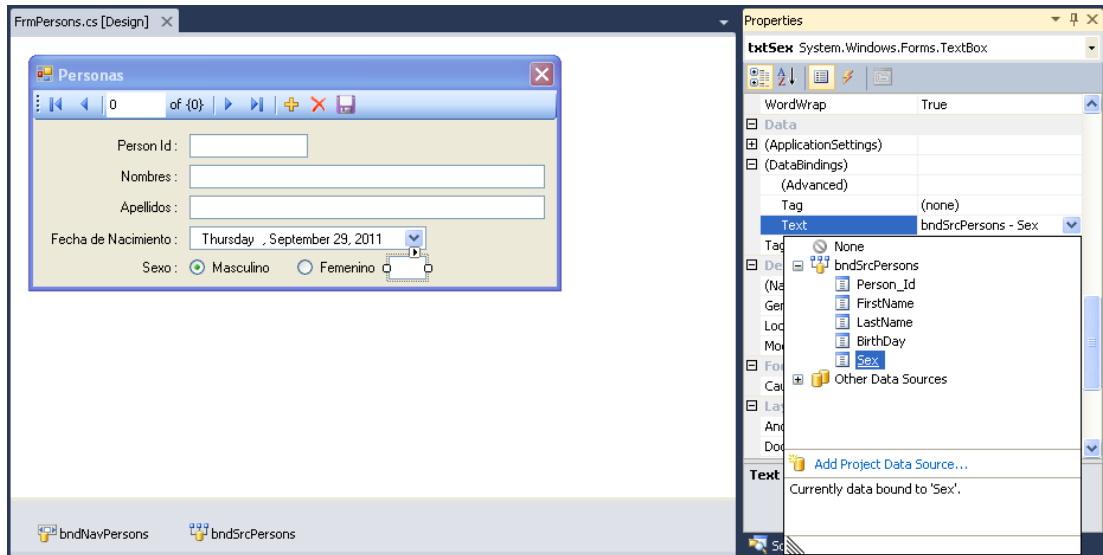


24. Modificamos las Propiedades de los campos del formulario, enlazando los campos de las entidades con los controles respectivos. Para ello utilizaremos la propiedad compuesta de **DataBindings** de cada campo y para cada uno de ellos escogeremos el campo **Text**. Así:

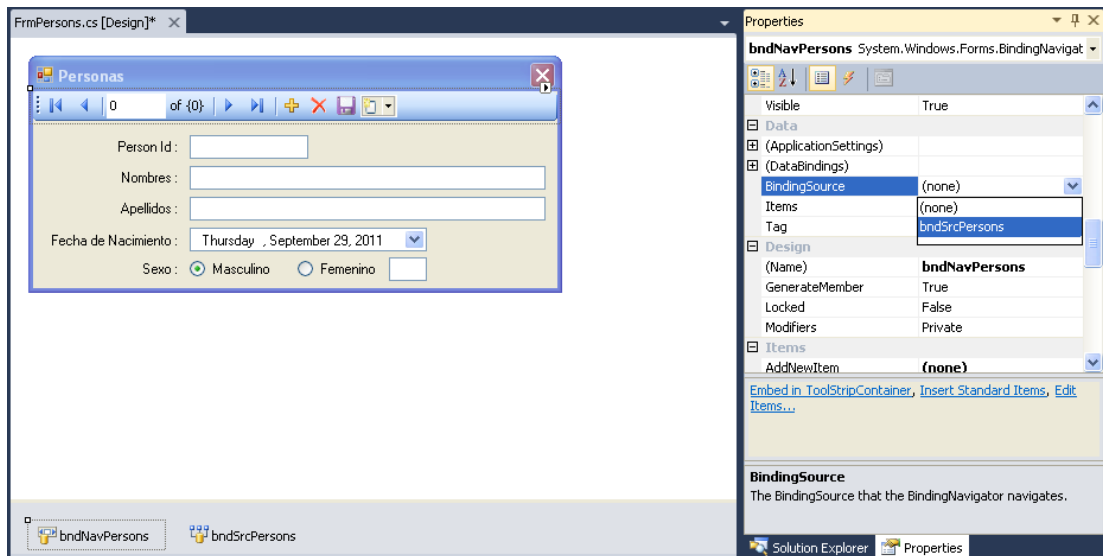


Se realiza la misma operación para los campos de Nombres y Apellidos. Para la Fecha de Nacimiento se utiliza la propiedad **Value** en lugar de la **Text**. Adicionalmente se inactiva el control `txtPersonId` poniendo en **True** la propiedad de **ReadOnly**.

25. Para enlazar el Sexo, nos toca crear una caja de texto llamada **txtSex**, definirla como no visible (**Visible=False**) y enlazar este campo a la propiedad **Text** de la sección de **DataBindings**, esto se hace por que el tipo de dato de la propiedad Sex dentro de la entidad **Person** es carácter y no boolean.



26. Ahora enlazamos el Navegador (**bndNavPersons**) a la fuente de datos (**bndSrcPersons**).



27. Ahora modifiquemos el código del formulario, para ello seleccionamos el formulario y presionamos la tecla **F7** y entramos a la sección de edición de código. Y Lo primero que vamos a hacer es adicionar el encabezado de comentarios.

```
FrmPersons.cs  X  FrmPersons.cs [Design]
DirTel.UI.FrmPersons  FrmPersons()

#region Documentación
/
* Propiedad intelectual de Engineers & Tools (c).
*
* Unidad      : FrmPerson.cs
* Descripción : Formulario para manejar los datos de la entidad Person
* Autor       : Julio Cesar Robles Uribe - Jucer
* Fecha       : 21-May-2010
*
* Fecha      Autor      Modificación
* =====
* 21-May-2010 Jucer      1 - Version Inicial
*
/
#endregion Documentación
```

28. Incluimos los **using** necesarios para referencias las librerías de entidades (**Entities**) y de lógica de negocios (**BL**).

```
FrmPersons.cs  X  FrmPersons.cs [Design]
DirTel.UI.FrmPersons  FrmPersons()

#endregion Documentación

// Librerias
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
// Librerias DirTel
using DirTel.BL;
using DirTel.Entities;

// Namespace
namespace DirTel.UI
```

29. Modificamos la clase para estructurar su métodos y su documentación así:

```
FrmPersons.cs  X  FrmPersons.cs [Design]
DirTel.UI.FrmPersons

// Namespace
namespace DirTel.UI
{
    /// <summary>
    /// Formulario para el manejo de los datos de las Personas
    /// </summary>
    public partial class FrmPersons : Form
    {
        // Enumeraciones
        Enumeraciones

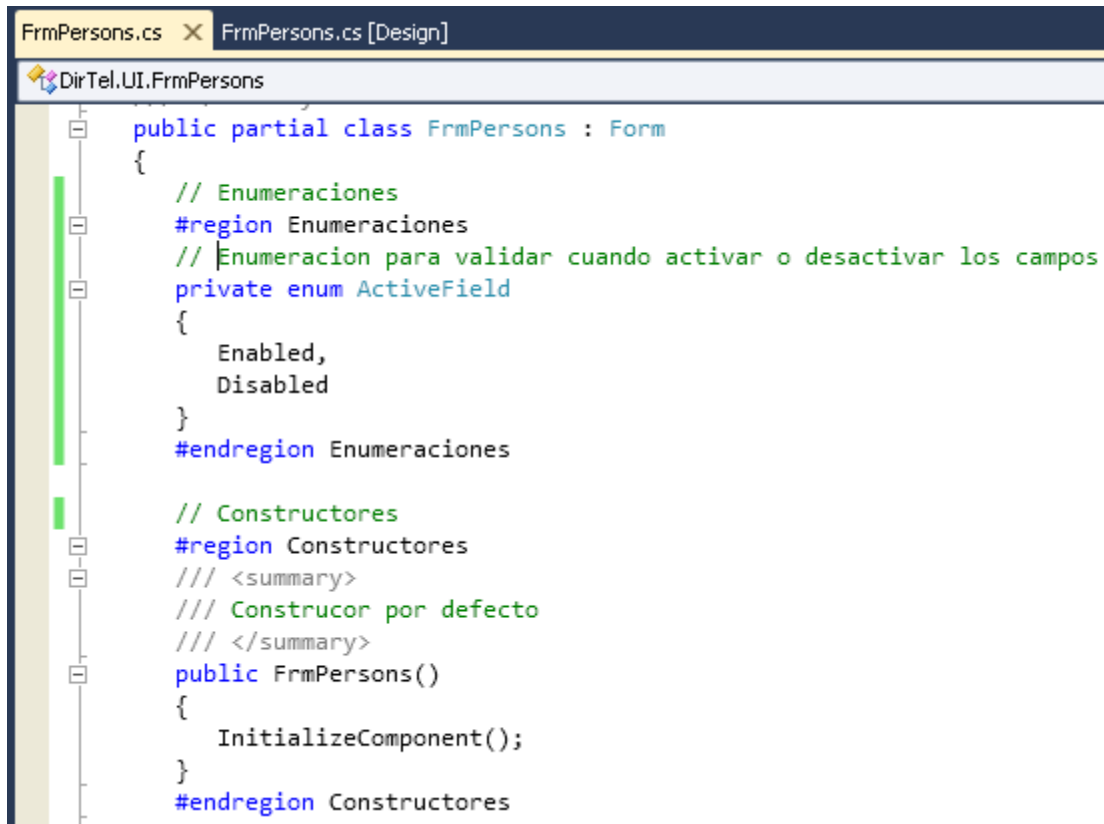
        // Constructores
        Constructores

        // Metodos
        Metodos
    }
}
```

30. Se crea una enumeración a nivel del formulario para poder activar y desactivar campos del formulario.

```
// Enumeracion para validar cuando activar o desactivar los campos
private enum ActiveField
{
    Enabled,
    Disabled
}
```

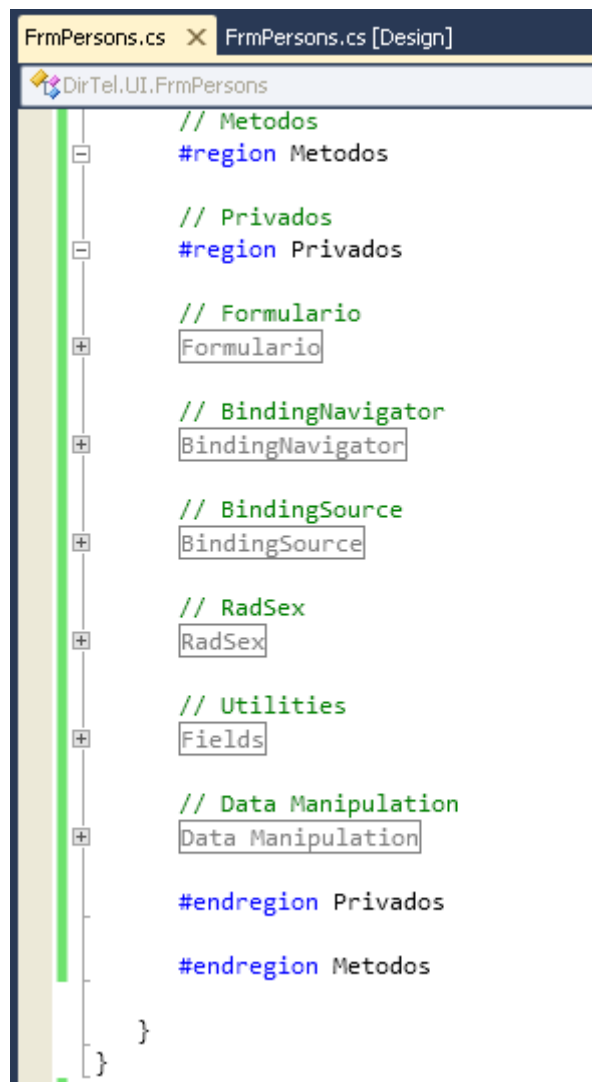
31. Se modifica, se regionaliza y se documenta el constructor.



```
public partial class FrmPersons : Form
{
    // Enumeraciones
    #region Enumeraciones
    // Enumeracion para validar cuando activar o desactivar los campos
    private enum ActiveField
    {
        Enabled,
        Disabled
    }
    #endregion Enumeraciones

    // Constructores
    #region Constructores
    /// <summary>
    /// Construcor por defecto
    /// </summary>
    public FrmPersons()
    {
        InitializeComponent();
    }
    #endregion Constructores
}
```


32. Se categorizan los métodos necesarios para manipular la información y los datos del formulario.



- **Formulario:** Agrupa las funcionalidades para manipular el formulario.
- **BindingNavigator:** Agrupa las funcionalidades para manejar la navegación entre registros (Primero, Siguiente, Anterior, Ultimo) y acciones de grabar, borrar y actualizar.
- **BindingSource:** Agrupa las funcionalidades de cambio de registro y los campos propios de la entidad y los controles del formulario.
- **RadSex:** Maneja la funcionalidad de Activación del tipo de sexo.
- **Fields:** Maneja la activación y desactivación de los controles del formulario.
- **Data Manipulation:** Permite manejar los datos de la entidad y ejecutar los procesos de lógica de negocio.

33. Desarrollamos los métodos así:

```
// Metodos
#region Metodos

// Privados
#region Privados

// Formulario
#region Formulario
/// <summary>
/// Cargar el formulario obteniendo los datos y desactivando los controles
/// </summary>
/// <param name="sender">Form</param>
/// <param name="e">Load Event</param>
private void FrmPersons_Load(object sender, EventArgs e)
{
    // Obtener los datos de todas las personas
    this.ReadAllPersons();

    // Validar si no existen Registros en el BindingSource Desactivar los
    Controles
    if (bndSrcPersons.List.Count == 0)
    {
        // Desactivar los controles
        this.ActiveFields(ActiveField.Disabled);
    }
}

#endregion Formulario

// BindingNavigator
#region BindingNavigator
/// <summary>
/// Ir al Primer Registro
/// </summary>
/// <param name="sender">Binding Source</param>
/// <param name="e">MoveFirstItem Click</param>
private void bindingNavigatorMoveFirstItem_Click(object sender, EventArgs e)
{
    bndSrcPersons.MoveFirst();
}

/// <summary>
/// Ir al Registro Anterior
/// </summary>
/// <param name="sender">Binding Source</param>
/// <param name="e">MovePreviousItem Click</param>
private void bindingNavigatorMovePreviousItem_Click(object sender, EventArgs e)
{
    bndSrcPersons.MovePrevious();
}

/// <summary>
/// Ir al Registro Siguiente
/// </summary>
/// <param name="sender">Binding Source</param>
/// <param name="e">MoveNextItem Click</param>
private void bindingNavigatorMoveNextItem_Click(object sender, EventArgs e)
{
    bndSrcPersons.MoveNext();
}
```

```

/// <summary>
/// Ir al Ultimo Registro
/// </summary>
/// <param name="sender">Binding Source</param>
/// <param name="e">MoveLastItem Click</param>
private void bindingNavigatorMoveLastItem_Click(object sender, EventArgs e)
{
    bndSrcPersons.MoveLast();
}

/// <summary>
/// Adicionar un registro
/// </summary>
/// <param name="sender">AddNewItem</param>
/// <param name="e">Click Event</param>
private void bindingNavigatorAddNewItem_Click(object sender, EventArgs e)
{
    // Poner el Foco en el campo del Id
    txtPersonId.Focus();

    // Validar si no existen Registros en el BindingSource Activar los Controles
    if (bndSrcPersons.List.Count == 0)
    {
        // Activar los campos
        this.ActiveFields(ActiveField.Enabled);
    }

    // Crear un objeto person en blanco
    Person person = new Person();
    // Adicionar un Nuevo Registro
    bndSrcPersons.Add(person);
    bndSrcPersons.MoveLast();
}

/// <summary>
/// Eliminar el Registro Activo
/// </summary>
/// <param name="sender">DeleteItem</param>
/// <param name="e">Click Event</param>
private void bindingNavigatorDeleteItem_Click(object sender, EventArgs e)
{
    // Poner el Foco en el campo del Id
    txtPersonId.Focus();

    // Validar si el Id de la persona no es vacio
    if (!txtPersonId.Text.Trim().Equals(string.Empty))
    {
        // Preguntar si en realidad lo desea eliminar
        DialogResult dlgResult = MessageBox.Show("Esta seguro de eliminar este
registro?", "Pregunta", MessageBoxButtons.YesNo, MessageBoxIcon.Question,
MessageBoxDefaultButton.Button2);

        // Validar si la respuesta fue positiva
        if (dlgResult == DialogResult.Yes)
        {
            // Borrar el Registro
            DeletePerson();

            // Validar si no existen Registros en el BindingSource DesActivar los
Controles
            if (bndSrcPersons.List.Count == 0)
            {
                // Activar los campos
                this.ActiveFields(ActiveField.Disabled);
            }
        }
    }
}

```

```

/// <summary>
/// Actualizar los datos de la tabla con los items del BindingSource
/// </summary>
/// <param name="sender">BindingSource</param>
/// <param name="e">Click Event</param>
private void bindingNavigatorSaveItems_Click(object sender, EventArgs e)
{
    // Poner el Foco en el campo del Id
    txtPersonId.Focus();

    // Obtener La cantidad de datos
    int countPersons = bndSrcPersons.List.Count;

    // Variable para Obtner los datos de la persona
    Person person = null;

    // Ciclo Para Recorrer los datos
    for (int i = 0; i < countPersons; i++)
    {
        // Obtener los datos de la lista
        person = bndSrcPersons.List[i] as Person;

        // Validar la accion del registro
        if (person.Person_Id == 0)
        {
            // Crear la Persona
            this.CreatePerson(person);
            // Actualizar el identificador en el formulario
            txtPersonId.Text = person.Person_Id.ToString();
        }
        else
        {
            // Actualizar los datos Modificados
            this.UpdatePerson(person);
        }
    }
}

#endregion BindingNavigator

```

```

// BindingSource
#region BindingSource
/// <summary>
/// Controlar el Cambio de registro en el formulario
/// </summary>
/// <param name="sender">BindingSource</param>
/// <param name="e">CurrentChanged Event</param>
private void bndSrcPersons_CurrentChanged(object sender, EventArgs e)
{
    // Obtener los datos de la persona actual
    Person person = bndSrcPersons.Current as Person;

    // Validar si hay personas
    if (person != null)
    {
        // Validar si el Sexo
        if (person.Sex == 'M')
        {
            // Activar Masculino
            this.radSexM.Checked = true;
        }
        else
        {
            // Activar Femenino
            this.radSexF.Checked = true;
        }
    }
}
#endregion BindingSource

// RadSex
#region RadSex
/// <summary>
/// Asignar el sexo Masculino (M) o Femenino (F) al campo oculto SEX
/// </summary>
/// <param name="sender">RadioButton</param>
/// <param name="e">Click Event</param>
private void radSex_Click(object sender, EventArgs e)
{
    // Obtener el control
    RadioButton radSex = sender as RadioButton;

    // Asignar el valor por defecto
    char sex = 'M';

    // Validar el control
    if (radSex.Text.Equals("Femenino"))
    {
        sex = 'F';
    }

    // Obtener una referencia al registro actual
    Person person = bndSrcPersons.Current as Person;
    // Cambiar el Sexo
    person.Sex = sex;

    // Asignar el valor al campo de texto
    txtSex.Text = person.Sex.ToString();
}
#endregion RadSex

```

```

// Utilities
#region Fields
/// <summary>
/// Activa o desactiva los controles del Formulario
/// </summary>
/// <param name="activeField">Enabled/Disabled</param>
private void ActivateFields(ActiveField activeField)
{
    // Variable para controlar el cambio
    bool enabled = true;
    // Validar si el cambio es desactivar los controles
    if (activeField == ActiveField.Disabled)
    {
        // Asignar desactivar los controles
        enabled = false;
    }
    // Desactivar los controles
    txtFirstName.Enabled = enabled;
    txtLastName.Enabled = enabled;
    dtpBirthDay.Enabled = enabled;
    radSexM.Enabled = enabled;
    radSexF.Enabled = enabled;
}

#endregion Utilities

// Data Manipulation
#region Data Manipulation
/// <summary>
/// Obtener los registros de la tabla y asignarlos al BindingSource
/// </summary>
private void ReadAllPersons()
{
    // Instanciar el Objeto de Negocio
    PersonBL personBL = new PersonBL();

    // Efectuar el llamado al Comando
    IList<Person> lstPersons = personBL.ReadAllPersons();

    // Recorrer la lista y asignar los valores
    foreach (Person person in lstPersons)
    {
        //asignar los valores al bindSource
        bndSrcPersons.List.Add(person);
    }
}

/// <summary>
/// Permite insertar los valores de la persona en la base de datos
/// </summary>
/// <param name="person">Entidad persona con los datos</param>
private void CreatePerson(Person person)
{
    // Instanciar el Objeto de Negocio
    PersonBL personBL = new PersonBL();

    // Ejecutar el comando
    long person_Id = personBL.CreatePerson(person);

    // Actualizar el valor de laid en la entidad person
    person.Person_Id = person_Id;
}

```

```

/// <summary>
/// Borrar el Registro Actual
/// </summary>
private void DeletePerson()
{
    // Obtener el registro activo
    Person person = bndSrcPersons.Current as Person;
    // Obtener el Id
    long person_Id = person.Person_Id;

    // Validar que el Id no sea cero para no ir a la base de datos
Innecesariamente
    if (!person_Id.Equals(0))
    {
        // Instanciar el Objeto de Negocio
        PersonBL personBL = new PersonBL();
        // Ejecutar el comando
        personBL.DeletePerson(person_Id);
    }

    // Eliminar el Registro
    bndSrcPersons.RemoveCurrent();
}

/// <summary>
/// Actualizar los datos de la persona
/// </summary>
/// <param name="person">Entidad persona con los datos cambiados</param>
private void UpdatePerson(Person person)
{
    // Instanciar el Objeto de Negocio
    PersonBL personBL = new PersonBL();

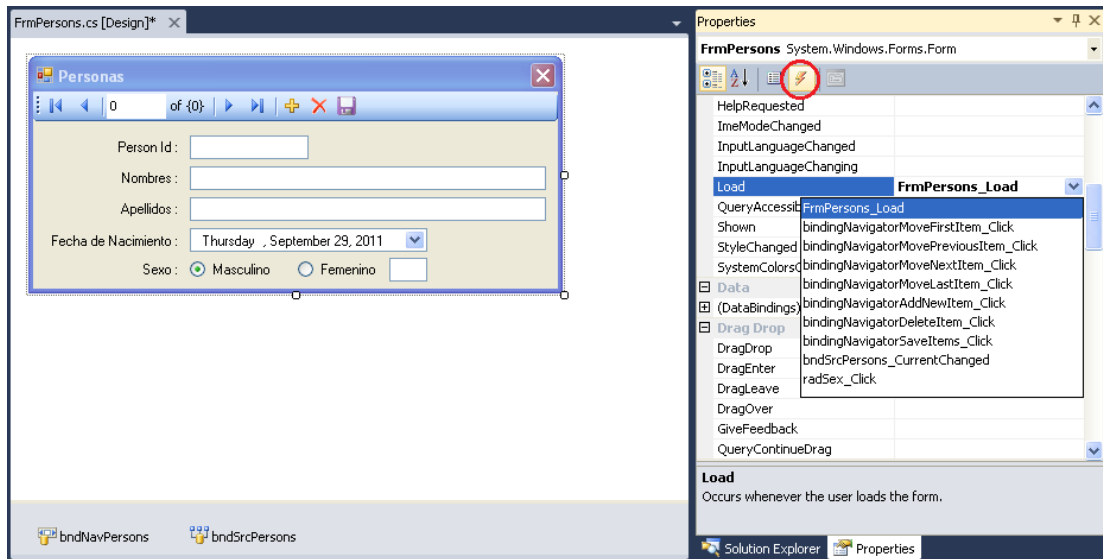
    // Ejecutar el comando
    personBL.UpdatePerson(person);
}
#endregion Data Manipulation

#endregion Privados

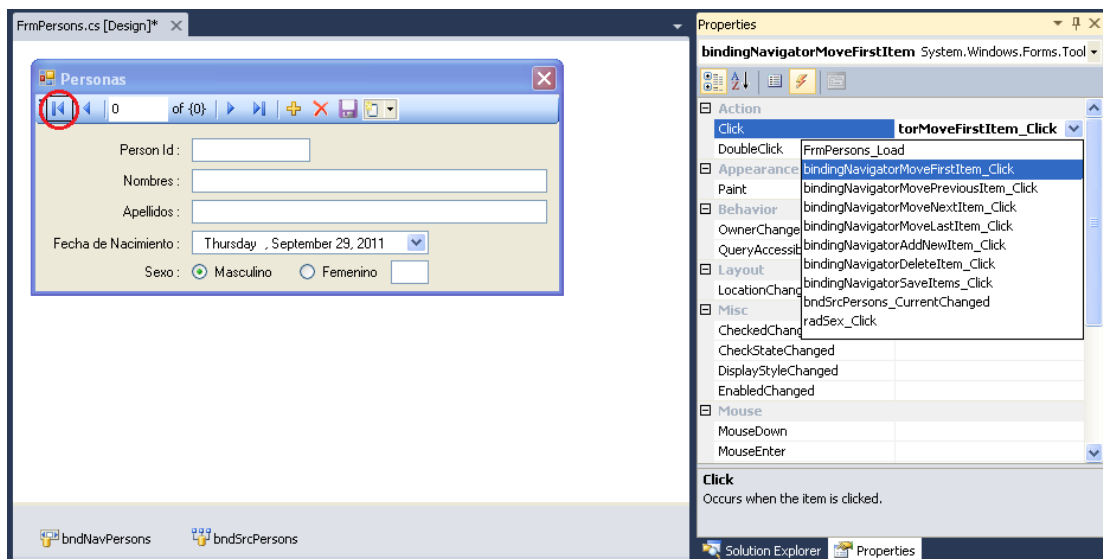
#endregion Metodos

```

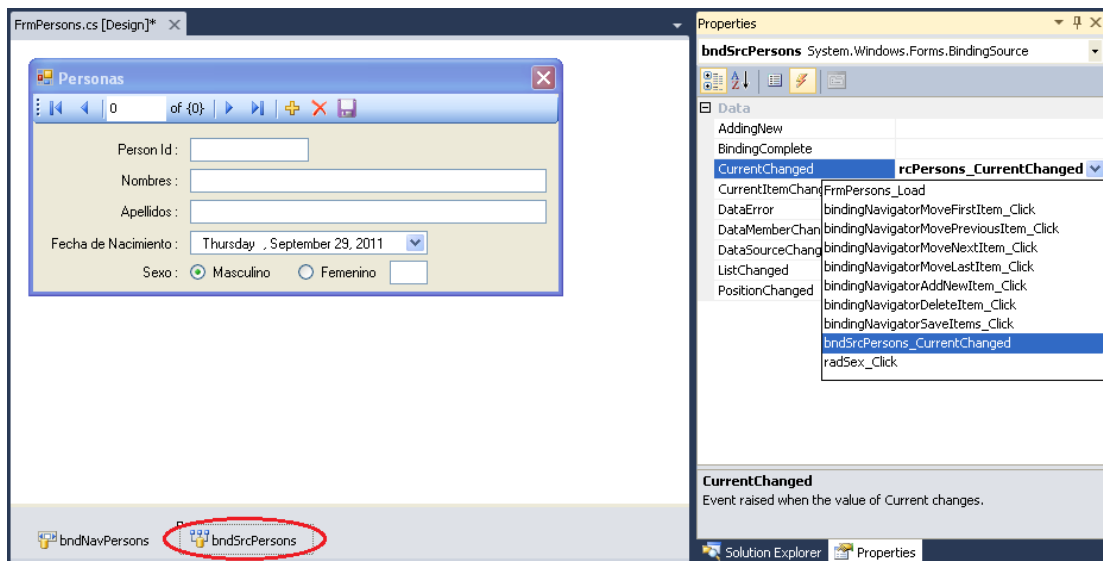
34. Ahora enlazamos estos métodos a los controles y objetos del formulario. Primero modificaremos el evento **Load** del formulario. Es importante que seleccione la hoja de eventos activando el botón identificado por el trueno.



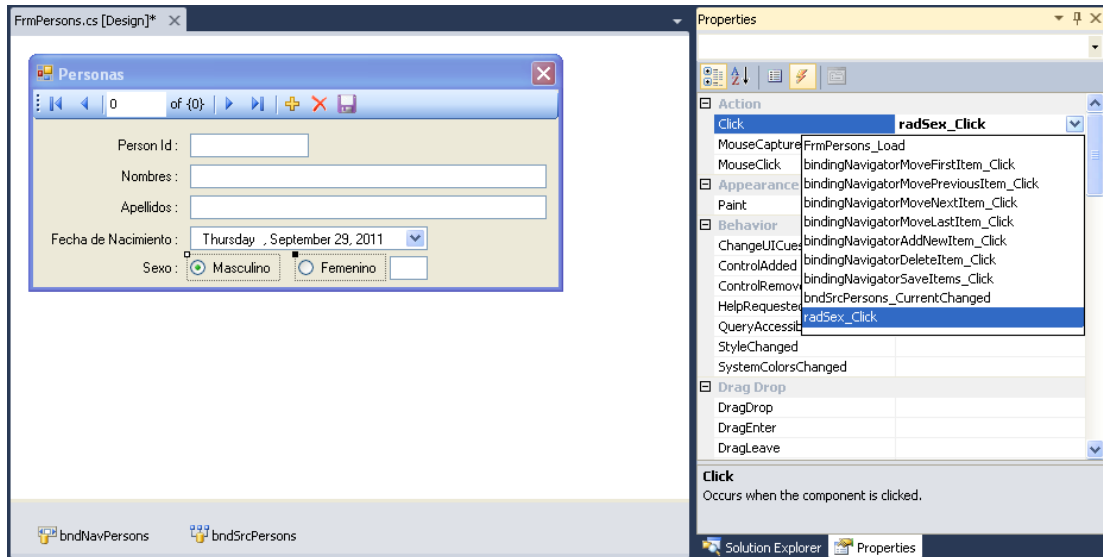
35. Para la navegación debe seleccionar los botones de la barra y modificar el evento **Click** para cada uno de los botones, respectivamente.



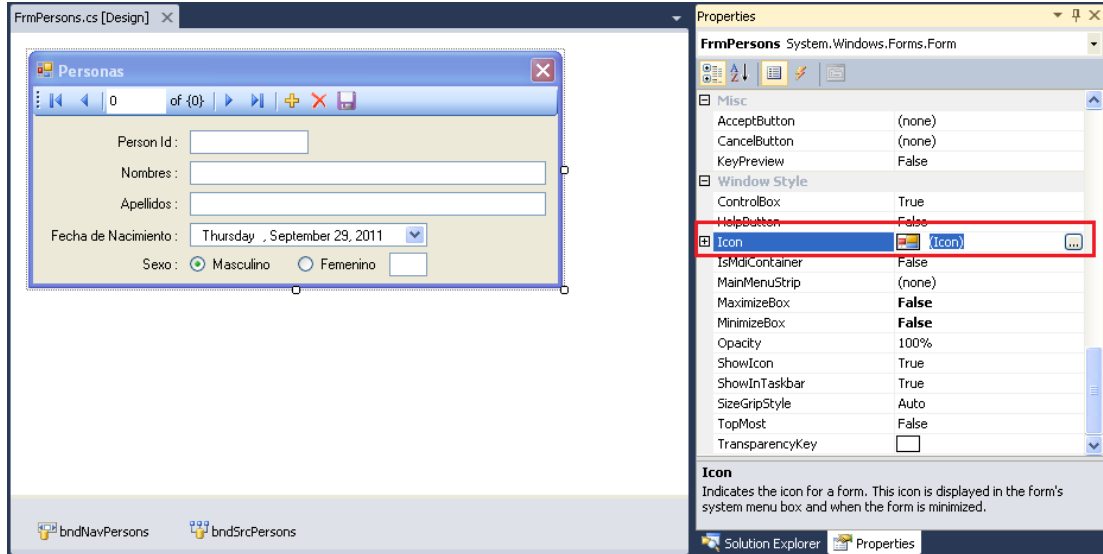
36. De igual forma se realiza la asignación del evento de cambio de registro (**CurrentChanged**), para el control de **bndSrcPersons**.



37. Para los que identifican el sexo de tipo Radio Botón debemos seleccionar ambos controles, esto lo podemos lograr haciendo un solo clic sobre el control de Masculino y luego presionando la tecla control (**CTRL**) hacemos clic en el control de Femenino y a ambos les cambiamos el evento **Click**.

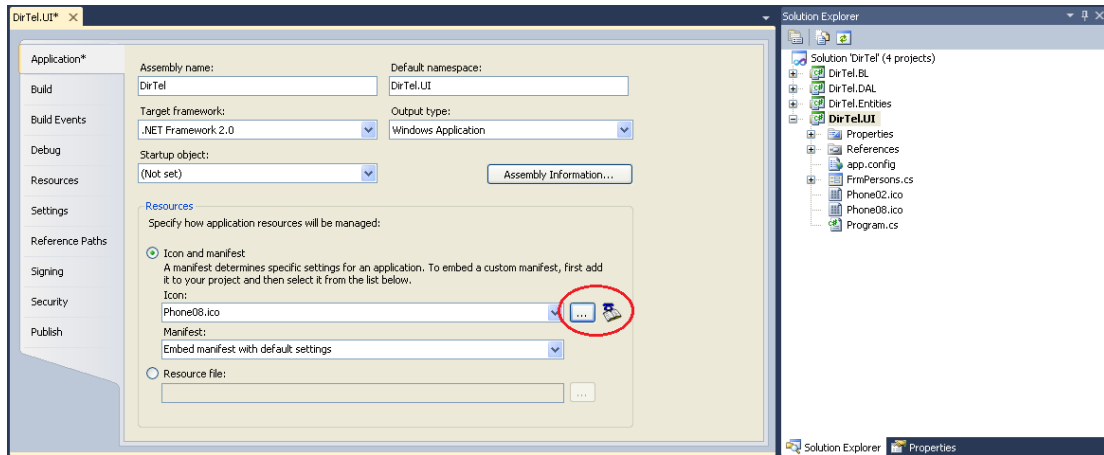


38. Cambie el icono del formulario, para ello seleccione el formulario y cambie la propiedad de Icono.



Seleccione de la carpeta **C:\Proyecto\SupportData\Recursos\lco** el icono llamado **Contacts03.ico**.

39. Modifique el icono para la aplicación, para ello seleccione, de la ventana de Exploración de la Solución el proyecto de **DirTel.UI** y abra la ventana de propiedades, haciendo click derecho sobre el proyecto. Modifique la sección de Icono y seleccione el archivo **Phone08.ico**.



40. Compile y guarde los cambios.
41. Ejecute la aplicación y pruebe las funcionalidades.

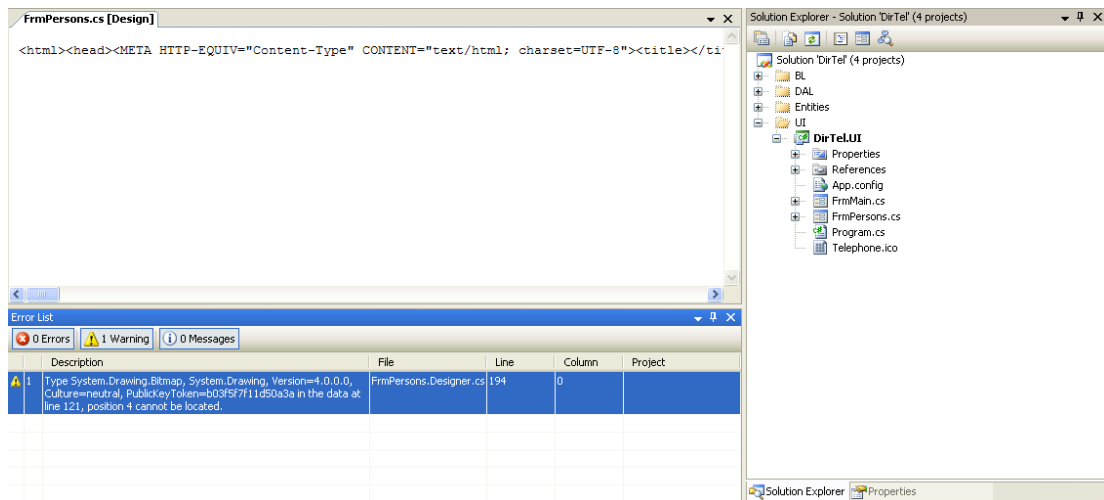
Práctica

1. Implemente la funcionalidad para el manejo de teléfonos de las personas, tenga en cuenta que una persona puede tener muchos teléfonos (Casa, Celular, FAX, etc). Para ello cree sus propios Scripts de tablas y paquetes o procedimientos de base de datos.
2. Adicione un menú para hacer el llamado de cada uno de los formularios.

Observaciones y Recomendaciones

Error al cargar el formulario

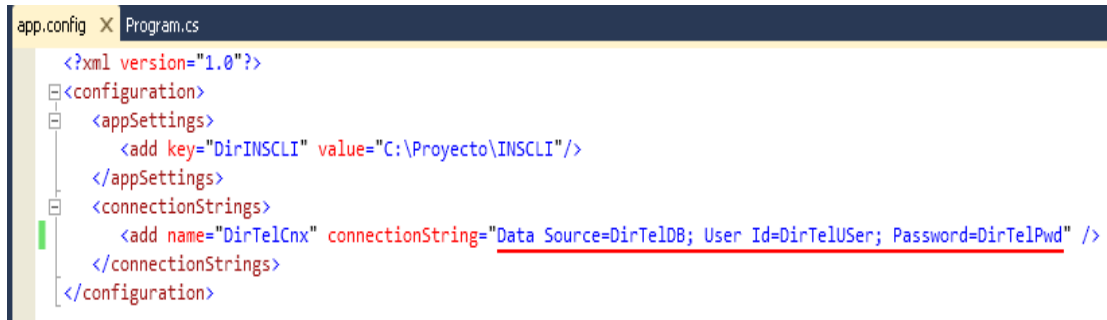
Si se llega a presentar este inconveniente:



Por favor cierre el formulario y recompile la aplicación (Rebuild Solution), para solucionar el inconveniente, esto se da porque se ha referenciado una librería que no está actualizada en ese momento.

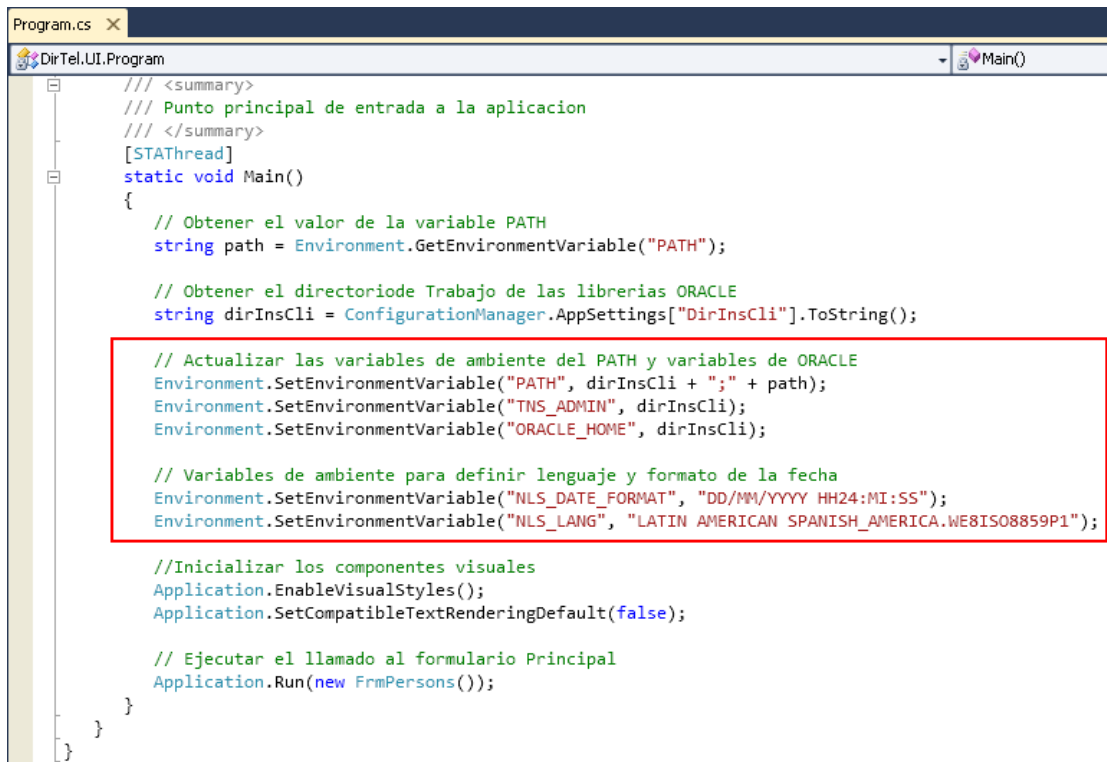
Uso del TNSNames y archivos .Ora

Si desea cambiar el estilo de conexión directa a conexión mediante el archivo de configuración **TNSNAMES.Ora** cambie su archivo de configuración **App.config**, refiriéndose únicamente al nombre de la instancia de la base de datos así:

The image shows a snippet of an XML configuration file named 'app.config'. It contains sections for 'configuration', 'appSettings', 'connectionStrings', and 'configuration'. The 'appSettings' section has an entry for 'DirINSCLI' with a value 'C:\Proyecto\INSCLI'. The 'connectionStrings' section has an entry for 'DirTelCnx' with a connection string 'Data Source=DirTelDB; User Id=DirTelUser; Password=DirTelPwd'.

```
<?xml version="1.0"?>
<configuration>
  <appSettings>
    <add key="DirINSCLI" value="C:\Proyecto\INSCLI"/>
  </appSettings>
  <connectionStrings>
    <add name="DirTelCnx" connectionString="Data Source=DirTelDB; User Id=DirTelUser; Password=DirTelPwd" />
  </connectionStrings>
</configuration>
```

Y modifique el archivo de **Program.cs** del proyecto UI adicionando las variables necesarias para realizar la conexión, asignando los valores de **TNS_ADMIN** y **ORACLE_HOME**, adicionalmente se asignan las variables de lenguaje y formato de la fecha así:

The image shows a snippet of C# code in a file named 'Program.cs'. The code is for a 'Main()' method. It first gets the 'PATH' environment variable and the 'DirInsCli' value from 'AppSettings'. Then, it updates the 'PATH' and sets 'TNS_ADMIN' and 'ORACLE_HOME' environment variables. A red box highlights the section where environment variables are updated. Finally, it sets 'NLS_DATE_FORMAT' and 'NLS_LANG' environment variables, initializes visual components, and runs the application.

```
/// <summary>
/// Punto principal de entrada a la aplicacion
/// </summary>
[STAThread]
static void Main()
{
    // Obtener el valor de la variable PATH
    string path = Environment.GetEnvironmentVariable("PATH");

    // Obtener el directorio de Trabajo de las librerias ORACLE
    string dirInsCli = ConfigurationManager.AppSettings["DirInsCli"].ToString();

    // Actualizar las variables de ambiente del PATH y variables de ORACLE
    Environment.SetEnvironmentVariable("PATH", dirInsCli + ";" + path);
    Environment.SetEnvironmentVariable("TNS_ADMIN", dirInsCli);
    Environment.SetEnvironmentVariable("ORACLE_HOME", dirInsCli);

    // Variables de ambiente para definir lenguaje y formato de la fecha
    Environment.SetEnvironmentVariable("NLS_DATE_FORMAT", "DD/MM/YYYY HH24:MI:SS");
    Environment.SetEnvironmentVariable("NLS_LANG", "LATIN AMERICAN SPANISH_AMERICA.WE8ISO8859P1");

    // Inicializar los componentes visuales
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);

    // Ejecutar el llamado al formulario Principal
    Application.Run(new FrmPersons());
}
```