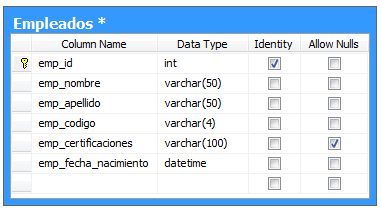
Librería Util

# Clase Util.DataObjectMapper

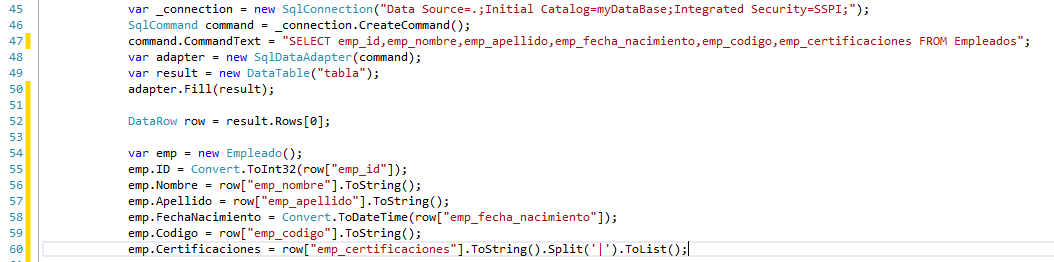
## Objetivo:

El objetivo de esta clase es sacar informaciones de un System.Data.DataRow y colocarlos en las propiedades de otro objeto especificado.

## Uso:

Suponga que se tiene el siguiente diseño:

El enfoque tradicional para transformar los datos de un System.Data.DataRow a una entidad de negocios es el siguiente:

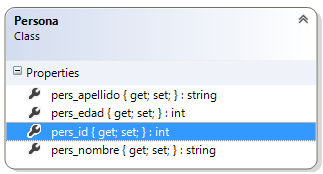


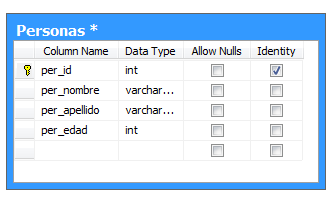
Las asignaciones realizadas desde la línea 54 a la 60 forman parte de un proceso repetitivo que realizamos cada vez sea necesario convertir del System.Data.DataRow a un tipo en específico. A continuación se muestra varios casos en los que se pueden obtener los mismos resultados utilizando la clase DataObjectMapper

### Caso 1, Sin configuración:

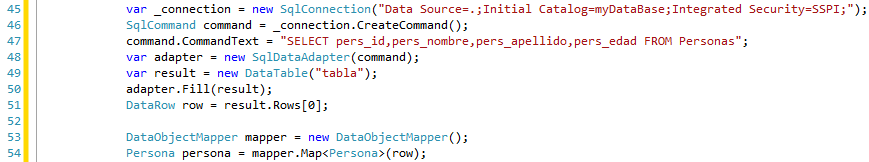
Se da cuando las propiedades de la Clase especificada tienen los mismos nombres que los campos de la consulta

***Escenario a:***

Suponga que se tiene la siguiente tabla y la siguiente clase:



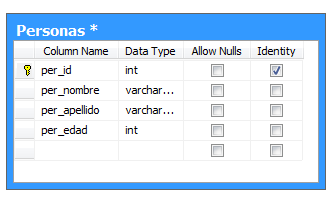
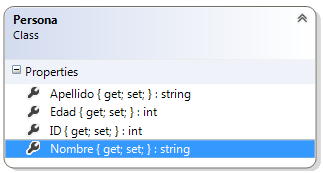
Se quiere llenar las propiedades de la clase Persona con los datos de una consulta obtenidas de la tabla persona, para esto se utiliza el método Map<T>.



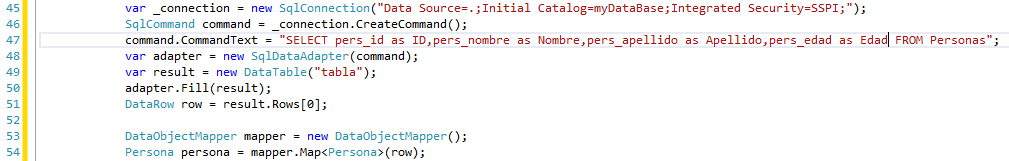
Arriba se puede ver el código para realizar esta operación. Observar en la línea 54 como se llama al método Map<Persona>. Debido a que los nombres de la clase persona concuerdan con los campos de la consulta el algoritmo interno los relaciona automáticamente sin necesidad de realizar más configuraciones y devuelve un objeto tipo Persona.

***Escenario B:***

Suponga que se tiene la siguiente tabla y la siguiente clase:



Al igual que el escenario anterior se desea llenar un objeto de tipo Persona con los datos de la tabla Personas sin embargo esta vez la tabla y la clase son diferentes en los nombres:



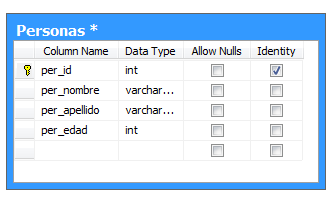
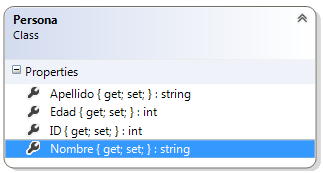
Si se observa la línea 47 se puede ver que esta vez la consulta coloca alias a los campos de forma que concuerden con los nombres de las propiedades. En este caso tampoco es necesario realizar más configuraciones para obtener el objeto tipo Persona con los datos de la consulta especificada.

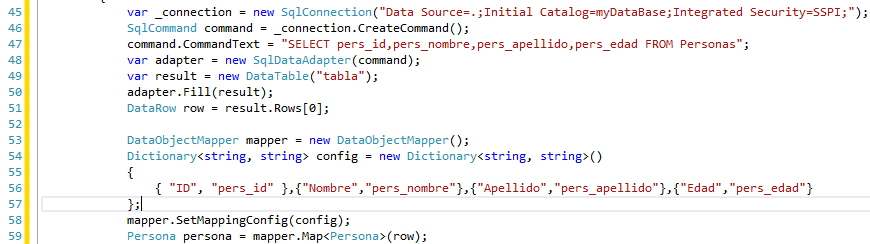
### Caso 2, Configurando:

Sucede cuando los nombres de los campos en la consulta son diferentes a los nombres de las propiedades de la Clase, se desea especificar manualmente que propiedad va relacionado con que columna o es necesario especificar una forma diferente de asignar valor a una propiedad especifica.

***Escenario A:***

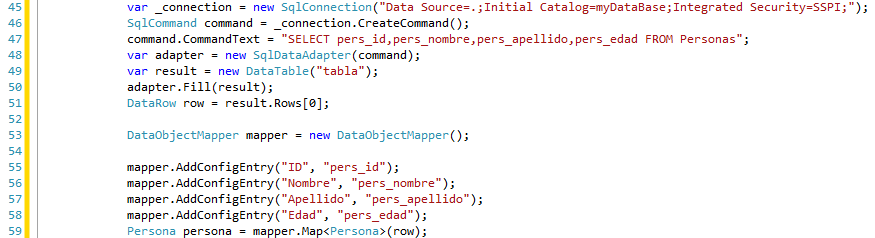
Suponga que se tiene la siguiente tabla y la siguiente clase:



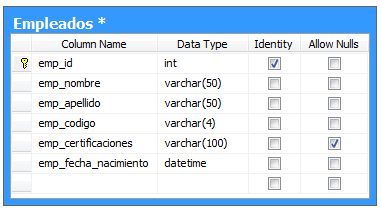


En este escenario se puede observar que los campos de la consulta no son iguales los nombres de las propiedades. En este caso se puede crear un diccionario que indique la relación Propiedad->Campo y utilizar el método SetMappingConfig para establecerlo. El algoritmo utilizara este diccionario para saber qué campo de la consulta está relacionado con que propiedad y asignar el valor adecuado.

También es posible utilizar el método AddConfigEntry para agregar las relaciones de las propiedades uno a uno en la siguiente imagen las líneas de la 55 a la 58 producen el mismo efecto que las líneas 54 a la 58 de la imagen anterior



***Escenario B:***

Suponga que se tiene el siguiente diseño:

En el diseño se decidió que las certificaciones conseguidas por un empleado se guardarían en un campo varchar cada una separada por el carácter barra (**|**), sin embargo en la aplicación para tener un manejo más versátil se decidió que se guardaría en una lista de string.

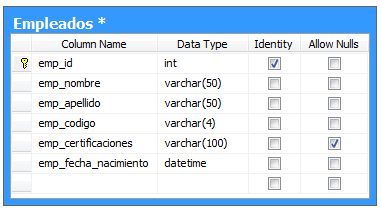
En este caso no es posible para el algoritmo interno realizar la operación de transformar la cadena de texto en una lista pero es posible especificar un procedimiento alterno para realizar el proceso.



En la imagen anterior en la línea 61 se puede observar que utilizando el método “AddGetValueAlgortihm” se puede cambiar la forma en que se obtiene el valor del objeto DataRow. El método recibe dos parámetros: el nombre de la propiedad y una función recibe por parámetro un DataRow y devuelve un valor tipo object. En el ejemplo anterior la función está representada por la expresión lambda ***dr => dr[“emp\_certificaciones”].ToString().Split(‘|’).ToList()***

### Caso 3, Creando listas de objetos:

Si se desea crear listas de objetos también es posible realizarlo usando cualquiera de las variantes del método MapToList<T>.

Suponga que se tiene el siguiente diseño:

En el siguiente código se ilustra cómo crear una lista de empleados a partir del diseño anterior:



Observar en la línea 63 como se utiliza el método MapToList<T>.

## Funcionamiento interno:

El algoritmo de la clase DataObjectMapper es el siguiente:

1. Crea un objeto de tipo T
2. Obtiene todas las propiedades del objeto creado utilizando Reflexión
3. Por cada propiedad encontrada realiza lo siguiente:
   1. Si se especificó una función para obtener el valor de esta propiedad se llama la función y se asigna el valor que devuelva a la propiedad.
   2. De lo contrario se verifica que la propiedad implemente la interfaz IConvertible antes de continuar, si no la implementa continua con la siguiente interfaz.
   3. Si se configuró la relación Propiedad->[Nombre Campo] se obtiene el nombre del campo de la configuración de lo contrario se toma el nombre de la propiedad como si fuera el nombre del campo.
   4. Si el DataRow contiene una columna con el nombre del campo y el valor de dicha columna no es igual a DBNull.Value se asigna el valor que tenga a la propiedad.
4. Se retorna el objeto creado en el paso 1.

# Clase Util.Helpers.Filler

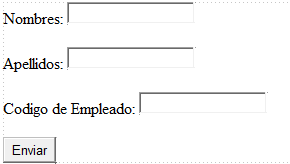
## Objetivo

Tiene como propósito tomar los datos de una interfaz web y con ellos llenar las propiedades de un objeto especificado.

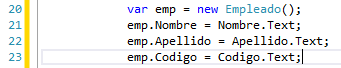
## Uso

### Caso 1, Sin Configuración:

Se utiliza cuando los nombres de los controles de la interfaz son iguales a los nombres de los controles de la clase que se desea llenar.

Suponga que se tiene el siguiente formulario y diseño de clase:

Se desea guardar los datos del formulario en una instancia de la clase empleado. A continuación se muestra el enfoque tradicional:

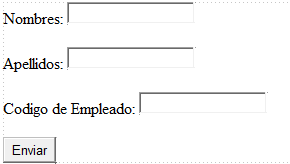


Utilizando la clase Filler el mismo ejemplo seria de la siguiente forma:

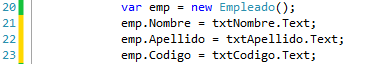


### Caso 2, Configurando:

Se da cuando los nombres de los controles son diferentes a las propiedades del objeto que se desea llenar.

Suponga que se tiene el siguiente formulario y diseño de clase:

Se desea guardar los datos del formulario en una instancia de la clase empleado. A continuación se muestra el enfoque tradicional:



Esta vez los nombres de los controles son diferentes a los nombres de las propiedades y es necesario especificarle al algoritmo la relación entre ellos de la siguiente forma:



## Restricciones:

Al momento de escribir este documento la clase Filler solo puede tomar los valores de los siguientes controles:

* Controles que extiendan de ListControl
* Controles que extiendan de HtmlInputControl
* Controles que implementen la interfaz ITextControl