Manual de Usuario para utilización de la programación del DLL CamposAdicionales

Campos Adicionales

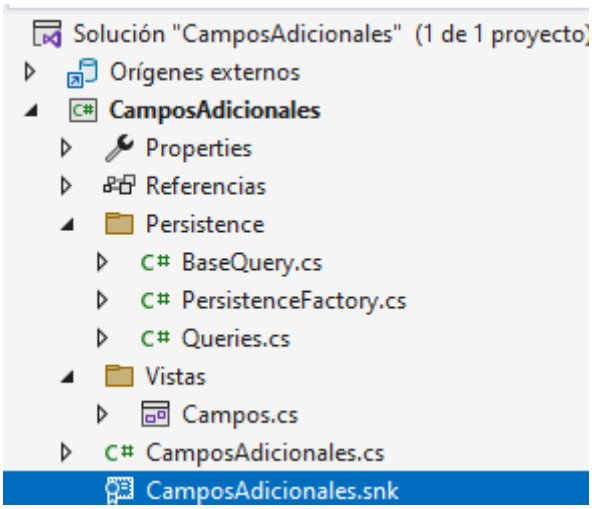
Manual de Usuario

Dpto. Software Nea F3 Master S.L.

NEAMASTER
TECNOLOGÍA DE CONFIANZA

1. Introducción

La programación se compone de un DLL principal llamado *CamposAdicional.cs* que a su vez contiene un Windows Form: *Campos.cs*.



Este programa se encarga de permitir que los usuarios puedan guardar valores adicionales en 20 campos nuevos que se han añadido en un diccionario en las tablas de **Clientes** (__CLIENTES), **Proveedores** (__PROVEED) y **Artículos** (ARTICULO):

Tabla	Descripción	Estado	Tipo	Obsoleta	Orden auxili	Externc
__CLIENTES	Tabla de clientes.	General	Maestro			✓
__PROVEED	Tabla de proveedores.	General	Maestro			✓
ARTICULO	Tabla de artículos.	General	Maestro			✓

Dichos campos son los siguientes:

- **ADI1, ADI2, ADI3, ADI4, ADI5, ADI6, ADI7, ADI8, ADI9** y **ADI10** para valores de tipo **VARCHAR**.
- **ADI11, ADI12, ADI13** y **ADI14** para valores del tipo **DECIMAL**.
- **ADI15, ADI16, ADI17** y **ADI18** para valores del tipo **DATETIME**.
- **ADI19** para valores del tipo **TEXT**.
- **ADI20** para valores del tipo **IMAGE**.

Columna	Descripción	Tipo	ongitud	Adm. Nulo	Defecto	Cuadradc	Identidad	Descr. tabla	Calculadc
ADI1	Adicional 1	varchar	100	✓					No
ADI10	Adicional 10	varchar	100	✓					No
ADI11	Adicional 11	decimal	9	✓					No
ADI12	Adicional 12	decimal	9	✓					No
ADI13	Adicional 13	decimal	9	✓					No
ADI14	Adicional 14	decimal	9	✓					No
ADI15	Adicional 15	datetime	8	✓					No
ADI16	Adicional 16	datetime	8	✓					No
ADI17	Adicional 17	datetime	8	✓					No
ADI18	Adicional 18	datetime	8	✓					No
ADI19	Adicional 19	text	16	✓					No
ADI2	Adicional 2	varchar	100	✓					No
ADI20	Adicional 20	image	16	✓					No
ADI3	Adicional 3	varchar	100	✓					No
ADI4	Adicional 4	varchar	100	✓					No
ADI5	Adicional 5	varchar	100	✓					No
ADI6	Adicional 6	varchar	100	✓					No
ADI7	Adicional 7	varchar	100	✓					No

2. CamposAdicionales

Esta clase se encarga de gestionar la interacción con la base de datos y proporcionar funcionalidades adicionales a a3ERP. Sus principales funcionalidades son:

- **Conexión a la Base de Datos**: Mediante la string *conexion*.

```
public class CamposAdicionales
{
    private static Enlace a3enlace = null;

    public static string conexion = "Data Source=AUXILIAR-MSI\\A3ERP;Initial Catalog=EjemploDDBB;User
Id=Sa;Password=demo";

    //public static string conexion = "Data Source=PABLO-ERP\\A3ERP;Initial Catalog=DEMO;User
Id=Sa;Password=demo";

    public static bool ventanaActiva = false;
    public static bool ActivarVentana = true;
    public List<string[]> refe;
}
```

- **Integración con a3ERP:** Mediante el objeto Enlace establece la comunicación con el ERP
- **Gestión de los Windows Forms.**
- **Eventos de documentos:** Implementa métodos para manejar eventos específicos relacionados con documentos, como:
    - *AntesDeGuardarDocumentoV2*: Prepara datos antes de guardar documentos, estableciendo valores en los formularios adicionales.
    - *DespuesDeGuardarDocumentoV2*: Configura el documento después de ser guardado.
    - *DespuesDeCargarDocumentoV2*: Muestra formularios y botones adicionales basados en el tipo de documento cargado.

- **Manejo de la creación del Windows Form:** Cuando se llama al método **Opcion(string IdOpcion, double idDoc)** si IdOpcion es CLIENTES o ARTICULOS o PROVEEDORES, se creará el formulario y se mostrará:

```
/**
 *  Recibe como parámetro el id de la opción y el id del documento.
 */
public void Opcion( string IdOpcion, double idDoc )
{

    // La opción de guardar campos adicionales solo tiene que aparecer en Clientes, Artículos y
Proveedores.
    if ((IdOpcion.ToUpper() == "CLIENTES") || (IdOpcion.ToUpper() == "ARTICULOS") || (IdOpcion.ToUpper()
== "PROVEEDORES"))
    {
        // Creo el formulario (se muestra al inicializarse)
        var camposForm = new Campos(conexion, idDoc, IdOpcion);
    }
}
```

3. Queries

Esta clase se encarga de actualizar campos específicos de las tablas de clientes, proveedores y artículos de la base de datos. Todo ello lo hace mediante el método **ActualizarCampo()** que recibe como parámetros el nombre del campo a modificar (ADI1, ADI2…ADI20), el valor que se quiere guardar en dicho campo, el código y el nombre de la tabla.

Primero, se obtiene el nombre del código (CODCLI, CODPRO o CODART) mediante el nombre de la tabla y se declara la *query* con los parámetros sin asignar aún:

```
string nombreCodigo = "";
if (nombreTabla == "__CLIENTES")
    nombreCodigo = "CODCLI";
else if (nombreTabla == "__PROVEED")
    nombreCodigo = "CODPRO";
else if (nombreTabla == "ARTICULO")
    nombreCodigo = "CODART";

string query = $"UPDATE {nombreTabla} SET {nombreCampo} = @valor WHERE LTRIM({nombreCodigo}) =
LTRIM(@CODCLI)";
```

A continuación, añado los parámetros a la *query* y la ejecuto en función del tipo de dato que es. Si se quiere guardar un DATETIME:

```
// Si es DATETIME la query es diferente
if (nombreCampo == "ADI15" || nombreCampo == "ADI16" || nombreCampo == "ADI17" || nombreCampo == "ADI16")
{
    string queryDatetime = $"UPDATE {nombreTabla} SET {nombreCampo} = CONVERT(DATETIME, @valor, 120)
WHERE LTRIM({nombreCodigo}) = LTRIM(@CODCLI)";

    using (var command = new SqlCommand(queryDatetime, connection))
    {
        command.Parameters.AddWithValue("@valor", valorCampo ?? (object)DBNull.Value); // Manejo de null
        command.Parameters.AddWithValue("@CODCLI", codigo.Trim());

        connection.Open();
        int rowsAffected = command.ExecuteNonQuery();

        return rowsAffected > 0; // Devuelve true si se actualizó al menos un registro
    }
}
```

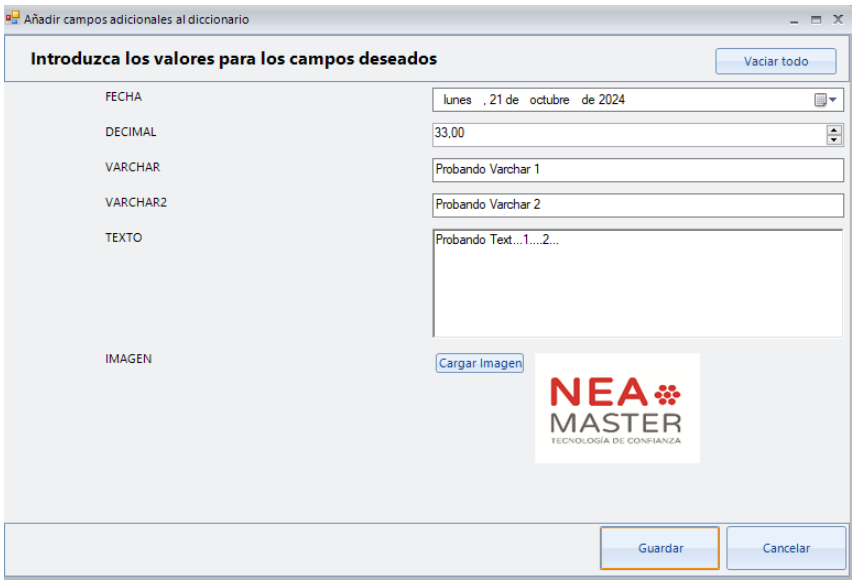
Si se quiere guardar un DECIMAL o IMAGE:

```
// Si es DECIMAL o IMAGE la query es diferente
else if (nombreCampo == "ADI11" || nombreCampo == "ADI12" || nombreCampo == "ADI13" || nombreCampo ==
"ADI14")
{
    try
    {
        using (var command = new SqlCommand(query, connection))
        {

            //command.Parameters.AddWithValue("@valor", valorCampo ?? (object)DBNull.Value); // Manejo de
null


             // Añadir el parámetro explícitamente como DECIMAL con precisión y escala
            command.Parameters.Add("@valor", SqlDbType.Decimal).Value = valorCampo ?? 0.0M;
            command.Parameters["@valor"].Precision = 18; // Definir precisión (18 es típico)
            command.Parameters["@valor"].Scale = 2; // Definir la escala (por ejemplo, 2 para decimales
como 0.00)


            command.Parameters.Add("@CODCLI", SqlDbType.VarChar).Value = codigo.Trim();

            connection.Open();
            int rowsAffected = command.ExecuteNonQuery();

            return rowsAffected > 0;
        }
    } catch (Exception e) // Para arreglar excepcion incoherente al guardar los DECIMAL
    {
        return true;
    }

} else if (nombreCampo == "ADI20")
{
    using (var command = new SqlCommand(query, connection))
    {
        // Verifica si el valor es un byte[] para manejar la imagen
        if (valorCampo is byte[] imageBytes)
        {
            // Si la imagen está vacía, asigna DBNull.Value
            if (imageBytes.Length == 0)
            {
                command.Parameters.Add("@valor", SqlDbType.VarBinary).Value = DBNull.Value; // Asigna
NULL a la base de datos
            }
            else
            {
                command.Parameters.Add("@valor", SqlDbType.VarBinary).Value = imageBytes; // Asigna la
imagen
            }
        }
        else // Para imagen null
        {
            command.Parameters.Add("@valor", SqlDbType.VarBinary).Value = DBNull.Value; // Asigna NULL si
es null
        }

        command.Parameters.Add("@CODCLI", SqlDbType.VarChar).Value = codigo.Trim();

        connection.Open();
        int rowsAffected = command.ExecuteNonQuery();

        return rowsAffected > 0; // Devuelve true si se actualizó al menos un registro
    }
} else
```

Y finalmente, si se quiere guardar un VARCHAR o TEXT:

```
// Verificación de cadena vacía para varchar o text
if (valorCampo is string && string.IsNullOrWhiteSpace((string)valorCampo))
{
    valorCampo = DBNull.Value; // Asignar NULL si la cadena está vacía
}

using (var command = new SqlCommand(query, connection))
{
  command.Parameters.AddWithValue("@valor", valorCampo??(object)DBNull.Value);
  command.Parameters.Add("@CODCLI", SqlDbType.VarChar).Value = codigo.Trim();

  connection.Open();
  int rowsAffected = command.ExecuteNonQuery();

  return rowsAffected > 0; // Devuelve true si se actualizó al menos un registro
}
```

4. Campos

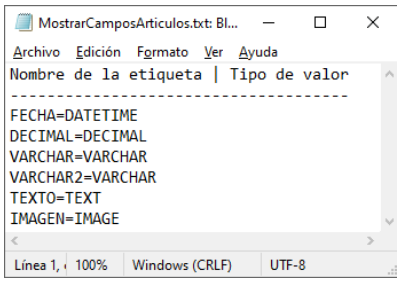


El formulario *Campos* está formado por un tableLayoutPanel donde se añaden los elementos a mostrar: **Label** para el nombre de cada campo, **NumericUpDown** para los campos de tipo DECIMAL, **TextBox** para los campos de tipo VARCHAR, **DateTimePicker** para los campos de tipo DATETIME, **RichTextBox** para los campos de tipo TEXT y **FlowLayoutPanel** para los campos de tipo IMAGE.

Primero, se cargan los elementos en el formulario con el método de **CargarCampos()**, que lee el fichero .txt correspondiente (*MostrarCamposArticulos.txt* para artículos, *MostrarCamposClientes.txt* para clientes y *MostrarCamposProveedores.txt* para proveedores) mediante el método de **leerFichero()**:

```
/*
 * Lee el fichero .txt y almacena los valores en un diccionario.
 */
public Dictionary<string, string> leerFichero()
{
    string rutaArchivo = "";

    // Dependiendo de la opción que sea se utilizará un archivo u otro
    if (this.idOpcion == "CLIENTES")
        rutaArchivo = Path.GetFullPath("..\\..\\MostrarCamposClientes.txt");
    else if(this.idOpcion == "ARTICULOS")
        rutaArchivo = Path.GetFullPath("..\\..\\MostrarCamposArticulos.txt");
    else if(this.idOpcion == "PROVEEDORES")
        rutaArchivo = Path.GetFullPath("..\\..\\MostrarCamposProveedores.txt");

    // Creo el diccionario
    Dictionary<string, string> nombreTipoDiccionario = new Dictionary<string, string>();

    // Solo leo el .txt si el archivo existe
    if (File.Exists(rutaArchivo))
    {
        var lineas = File.ReadAllLines(rutaArchivo);

        // Salta las líneas del encabezado del .txt
        for (int i = 2; i < lineas.Length; i++)
        {
            // Ignorar líneas vacías
            if (string.IsNullOrWhiteSpace(lineas[i]))
                continue;

            var linea = lineas[i].Split('=');
            if (linea.Length == 2)
            {
                string nombreCampo = linea[0].Trim();
                string tipoCampo = linea[1].Trim().ToUpper();
                // Compruebo que no haya otro elemento en el .txt con el mismo nombreCampo
                if (nombreTipoDiccionario.ContainsKey(nombreCampo))
                    MessageBox.Show($"El campo con nombre '{nombreCampo}' en la línea {i + 1} ya existe.
Se ha ignorado.");
                else // Añado al Diccionario el nombre del campo y el tipo de campo si nombreCampo no
existe en el .txt
                    nombreTipoDiccionario.Add(nombreCampo, tipoCampo);
            }
            else
            {
                MessageBox.Show($"La línea {i + 1} del archivo está mal formateada: {lineas[i]}");
            }
        }
        return nombreTipoDiccionario;
    }
    return nombreTipoDiccionario;
}
```

Por otro lado, en el archivo .txt se indican los campos a mostrar en la ventana con el nombre del campo y el tipo del campo separados por un símbolo de “=”:



Después, verifica que no se sobrepase la cantidad de elementos de cierto tipo en el .txt (por ejemplo, solo puede haber 10 elementos de tipo VARCHAR como máximo), agrega los controles de forma dinámica a la ventana y carga el valor de la base de datos para mostrarlo en la ventana (si ese campo ya tuviera un valor asignado previamente) y finalmente muestro la ventana:

```
/*
 * Método que carga los campos del txt correspondiente en el formulario.
 */
public void CargarCampos()
{
    tableLayoutPanel.Refresh();
    // Limpia el TableLayoutPanel antes de agregar nuevos controles
    tableLayoutPanel.Controls.Clear();

    // Limpiar los estilos de fila existentes
    tableLayoutPanel.RowStyles.Clear();

    // Obtener la cantidad de campos del diccionario
    Dictionary<string, string> diccionarioNombreTipo = leerFichero();
    int numberOfRows = diccionarioNombreTipo.Count; // Número de filas a agregar
    int rowHeight = 50; // Altura de las filas en píxeles

    // Verificación de si se sobrepasa la cantidad de elementos de cada tipo. (Omitido)

    // Agrega los controles correspondientes
    foreach (KeyValuePair<string, string> elemento in diccionarioNombreTipo)
    {
        // Creo la label con el texto correspondiente
        Label label = new Label();
        label.Text = elemento.Key;
        label.AutoSize = true;  // Activar AutoSize para que el tamaño se ajuste automáticamente
        label.Margin = new Padding(90, 5, 5, 5); // Margen para el label

        // Agregar la label al TableLayoutPanel en la primera columna
        tableLayoutPanel.Controls.Add(label);

        // Actualiza actualADI antes de crear el campo
        string asignacionColumna = ""; // Para almacenar la columna asignada
        if (elemento.Value == "VARCHAR" && contadorVarchar < 11)
        {
            asignacionColumna = "ADI" + (contadorVarchar); // ADI1 a ADI10
            actualADI = "ADI" + contadorVarchar;
            contadorVarchar++;
        }
        else if (elemento.Value == "DECIMAL" && contadorDecimal < 15)
        {
            asignacionColumna = "ADI" + (contadorDecimal); // ADI11 a ADI14
            actualADI = "ADI" + contadorDecimal;
            contadorDecimal++;
        }
        else if (elemento.Value == "DATETIME" && contadorDatetime < 19)
        {
            asignacionColumna = "ADI" + (contadorDatetime); // ADI15 a ADI18
            actualADI = "ADI" + contadorDatetime;
            contadorDatetime++;
        }
        else if (elemento.Value == "TEXT" && !textAsignado)
        {
            asignacionColumna = "ADI19"; // ADI19 para TEXT
            actualADI = "ADI19";
            textAsignado = true;
        }
        else if (elemento.Value == "IMAGE" && !imageAsignado)
        {
            asignacionColumna = "ADI20"; // ADI20 para IMAGE
            actualADI = "ADI20";
            imageAsignado = true;
        }

        // Creo el campo para que el usuario pueda meter el valor correspondiente
        Control inputControl = addElementoPanelSegunTipo(elemento.Value);

        // Cargo el valor de la base de datos si es que existe
        object valor = ObtenerValorDesdeBaseDeDatos(actualADI);
        AsignarValorControl(inputControl, valor, IDoc.ToString());

        // Agregar el control de entrada al TableLayoutPanel en la segunda columna
        tableLayoutPanel.Controls.Add(inputControl);
    }

    this.Show();
}
```

Al clicar el botón de *Guardar*, se crea un diccionario con los datos del fichero (método **leerFichero()** visto con anterioridad) y con el método **AsignarColumnas(Dictionary <string, string> diccionario)** asigna cada dato del diccionario con su campo ADI correspondiente dependiendo del tipo del dato y llama al método **ActualizarCampo()** de Queries para guardar los valores en la base de datos y finalmente muestra un mensaje indicando que los campos se han guardado con éxito:

```
/*
 * Guarda los datos que el usuario ha introducido en los inputs en sus ADIs correspondientes en la base
de datos.
 */
private void radButtonGuardar_Click(object sender, EventArgs e)
{
    try
    {
        Dictionary<string, string> diccionario = leerFichero();
        AsignarColumnas(diccionario);

        // Mensaje de confirmación
        MessageBox.Show("Los campos han sido guardados correctamente.");
    }
    catch (Exception excepcion)
    {
        MessageBox.Show(excepcion.Message);
    }
}
```

Al clicar el botón de *Cancelar*, simplemente se cierra el formulario:

```
/*
 * Cierra el formulario.
 */
private void radButtonCancelar_Click(object sender, EventArgs e)
{
    this.Close();
}
```

Finalmente, al clicar el botón de Vaciar todo se limpian todos los campos de input llamando al método BorrarCamposDeInput (Control control):

```
/*
 * Acción al clicar el botón de 'Vaciar todo'
 */
private void radButton1_Click(object sender, EventArgs e)
{
    BorrarCamposDeInput(this);
}

/*
 * Borra todos los campos de input del formulario.
 */
private void BorrarCamposDeInput(Control parent)
{
    foreach (Control control in parent.Controls)
    {
        if (control is TextBox textBox)
        {
            textBox.Clear(); // Borra el texto del TextBox
        }
        else if (control is RichTextBox richTextBox)
        {
            richTextBox.Clear(); // Borra el texto del RichTextBox
        }
        else if (control is ComboBox comboBox)
        {
            comboBox.SelectedIndex = -1; // Restablece la selección del ComboBox
        }
        else if (control is NumericUpDown numericUpDown)
        {
            numericUpDown.Value = 0.0M;
        }
        else if (control is DateTimePicker dateTimePicker)
        {
            dateTimePicker.Value = DateTime.Now; // O establece un valor por defecto, como la fecha
actual
        }
        else if (control is PictureBox pictureBox)
        {
            pictureBox.Image = null; // Elimina la imagen del PictureBox
        }

        // Si el control tiene otros controles dentro de él, llama recursivamente
        if (control.HasChildren)
        {
            BorrarCamposDeInput(control); // Llama a la función recursivamente
        }
    }
}
```