Plugin de personalización de Sage 50c

Guía rápida

Sage
29 06 2017

Índice

1. Descripción de las propiedades y métodos para la implementación de Listados y/o Gráficas en .net en Sage 50c (Plugin de desarrollo básico)..... 4

1.1. Definición de propiedades y métodos de la clase listados 4

1.1.1. Propiedades..... 4

1.1.2. Métodos 7

1.2. Funciones tratamiento de filtros y opciones..... 11

1.3. Extensiones sobre datatables 12

2. Configuración de un proyecto para poder realizar la implementación de un listado y depurarlo..... 14

2.1. Pasos para la configuración..... 15

2.1.1. Diseño visual de los filtros..... 15

2.1.2. Establecer las referencias en el proyecto principal..... 16

2.1.3. Establecer el código base en program.c..... 17

2.1.4. Establecer la configuración básica para la creación del listado 18

2.1.5. Ejemplo de creación de un listado 19

3. Gestión de Addons implementados en .NET para Sage 50c (Plugin de desarrollo avanzado). 19

3.1 Preparación de Visual Studio para crear proyectos (addons)..... 20

El Plugin de desarrollo avanzado requiere de la instalación de una extensión de Sage 50c en Visual Studio con el objetivo de poder crear proyectos (addons). Dicho componente se encuentra en el directorio de la carpeta de instalación del producto dentro de la carpeta “Plugin”. Los pasos para realizar dicha instalación son los siguientes:..... 20

Para que Visual Studio muestre el asistente del módulo tras crear un nuevo proyecto del tipo 20

módulo, se debe registrar la dll del proyecto al Sistema Global de Cache global de VS. Para ello..... 20

utilizaremos la herramienta Gacutil.exe. 20

3.1 Arranque del Plugin desde Visual Studio 21

3.1.1 Creación de un projecto de tipo Addon 21

3.1.2 Ejecutar Sage 50c desde Visual Studio..... 21

3.1.3 Base de datos del addon 23
3.1.4 Métodos y propiedades para la ejecución de un addon 24
3.2 Creación de mantenimientos nuevos 25
3.3 Extensión de un mantenimiento ya existente 26
3.3.1 Personalización de un mantenimiento existente 26
3.3.1 Asociación de una tabla relacionada a un mantenimiento 27
3.5 Menú de los addon 30
3.6 Pantallas y Formularios libres 31
3.7 Proceso de instalación de un addon de personalización a medida. 32
3.7.1 Generación del proceso de instalación 32
3.7.2 Instalación del addon de personalización a medida 33

Es posible consultar la ayuda técnica avanzada del plugin de personalización de Sage 50c desde la siguiente url: http://developers.sage50comunidad.es

1. Descripción de las propiedades y métodos para la implementación de Listados y/o Gráficas en .net en Sage 50c (Plugin de desarrollo básico)

1.1. Definición de propiedades y métodos de la clase listados

1.1.1. Propiedades

```
string _ReportFile { get; }
```

Nombre del report (sage reports) que se utiliza para la impresión de un listado. Cuando se está creando el listado si no existe este fichero lo crea automáticamente.

```
string _Error_Message { get; set; }
```

En caso que en la clase LISTADOS genere algún tipo de error actualizará esta propiedad _Error_Message. En caso que esta propiedad tenga un valor se presentará en el momento en que el usuario pulse el botón ACEPTAR

```
int _Decimales { get; }
```

Número de decimales en que trabaja la moneda de la empresa de Eurowin. El programador podrá utilizar esta propiedad en caso que tenga que redondear el número de decimales del DataTable.

```
string _Mascara_Precios { get; }
```

Mascara en formato .NET que utilizarán todas las columnas que presenten información de tipo PRECIOS. Esta propiedad será utilizada por el método _SetColumnsMasks()

```
string _Mascara_Importe { get; }
```

Igual que _Mascara_Precios pero para las columnas de tipo importe

```
string _Mascara_TPC { get; }
```

Igual que _Mascara_Precios pero para las columnas que presentan valores de tanto por ciento

```
string _Mascara_Unidades { get; }
```

Igual que _Mascara_Precios pero para las columnas que presentan valores de unidades

```
string _Mascara_Peso { get; }
```

Igual que _Mascara_Precios pero para las columnas que presentan valores de tipo peso

```
string _Mascara_Cajas { get; }
```

Igual que _Mascara_Precios pero para las columnas que presentan valores de tipo cajas

```
string _Pantalla { get; set; }
```

Esta propiedad contiene el valor del campo “PANTALLA” de la tabla LISTADOS de la base de datos de COMUNES.

```
string _Nombre { get; set; }
```

Esta propiedad contiene el valor del campo “NOMBRE” de la tabla LISTADOS de la base de datos de COMUNES.

```
string _Eje_X_Graficos { get; set; }
```

En esta propiedad se asignará el campo que se utilizará como eje de las X en caso que en el listado haya un botón para presentar una gráfica.

```
bool _Imprimir_CambiarFecha { get; set; }
```
Propiedad que nos permite definir si en el apartado de impresión del listado aparece la fecha de impresión. El usuario podrá cambiar esta fecha.

```
bool _Imprimir_CambiarNumPagina { get; set; }
```
Propiedad que nos permite definir si en el apartado de impresión del listado el usuario puede cambiar el número de página.

```
Dictionary<string, object> _Filtros { get; set; }
```
Diccionario que contiene todas las variables de filtros que se utilizarán para generar el DataTable. Este diccionario equivale a los filtros no visuales de un listado.

```
Dictionary<string, object> _FiltrosReport { get; }
```
Diccionario que contiene todos los filtros que se utiliza en la impresión de un listado. Estos filtros se cargan de forma automática en el método _Valid.

```
Dictionary<string, object> _Opciones { get; set; }
```
Diccionario que contiene todas las posibles opciones de agrupación de un listado. Este diccionario equivale a las opciones de un listado.

```
Dictionary<string, object> _VisualOpciones { get; set; }
```
Diccionario que contiene todas las posibles opciones que se presentan en el listado. Esta propiedad se utiliza para modificar los valores de las opciones de listado una vez éstas ya han sido pintadas en el formulario.

```
Dictionary<string, string> _NavigateButtons { get; set; }
```
Diccionario que contiene todas las descripciones de los botones que se utilizarán para navegar desde este listado hacia otras pantallas

Ejemplo:
Suponemos que la propiedad _NavigateButtons devuelve el siguiente diccionario

```
return new Dictionary<string, string>() { { "Albaven", "Ver albarán de venta" },
{ "VerFact", "Ver factura" } };
```

- En este ejemplo el listado tendría 2 botones de navegación, el primer botón se irá al documento de venta y el segundo botón irá al documento de ver la factura.
- Cuando el usuario haga click sobre el botón que tiene por título “Ver albarán de venta” entonces ejecutará el método _Navigate(“Albaven”,DataRow del grid). En caso que el usuario haga click sobre el botón de “Ver factura” entonces ejecutará el método _Navigate(“VerFact”,DataRow del grid)

```
Dictionary<string, string> _NavigateButtonsGráficas { get; set; }
```
Diccionario que contiene las posibles gráficas que puede llegar a presentar un listado. Esto implica que en el resultado del listado habrá un botón con el título de “Gráfica”. En el caso que haya más de una gráfica presentará un submenú.

```
Dictionary<string, string> _NavigateButtonsCheck { get; set; }
```
Diccionario con las posibles opciones de navegación cuando se selecciona una lista de la casilla del listado. Si hay alguna entrada en este diccionario se mostrará una casilla de check en las columnas del listado.

Dictionary<string, string> _NavigateButtonsGoogleMaps { get; set; }
Diccionario que contiene todas las posibles opciones que se utilizarán para presentar información en un mapa de GoogleMaps.

Un ejemplo sería:

```
public override Dictionary<string, string> _NavigateButtonsGoogleMaps
{
        get
        {
                bool llVentasYPromociones = this._Opcion_Logico("lIncluirventasyprom");

                if (!llVentasYPromociones)
                        return new Dictionary<string, string>() { { "unidades", "Distribución de
                ventas por unidades" }, { "ventas", "Distribución de ventas por importe" } };
                else
                        return base._NavigateButtonsGoogleMaps;
        }
}
```

string[] _Ejercicios { get; set; }
Array que contiene los diferentes ejercicios que el usuario ha seleccionado en el listado. Esta propiedad solo tiene sentido si en el listado aparece un filtro del tipo txtEjercicioDesdeHasta

List<string> _NavigableColumns { get; set; }
Lista de campos que nos permite navegar hacia otra pantalla.

List<string> _ReportColumns { get; }
Lista de campos que presentará el gridview. En caso que esta propiedad este vacía entonces el gridview presentará todos los campos del DataTable.

Un ejemplo sería:

```
Return  new List<string>()
{
        "albaran", "fecha", "almacen", "nomalm", "*pvp”
};
```

Por defecto, en el listado, siempre la primera letra está en mayúsculas. En caso que haya un “*” delante implicará que el título de la columna estará todas las letras en mayúsculas.

Dictionary<String, Grafica> _Graficas { get; set; }
Diccionario que contiene las diferentes posibles gráficas que se pueden generar a partir de un listado. El diccionario contiene un objeto de tipo gráfica.

DataTable _Resultados { get; set; }
Datatable que contendrá los resultados de la ejecución del método _DataTable()

DataTable _ResultadoOriginal { get; set; }
Datatable que contendrá el resultado original del datatable sin eliminación de columnas. En el método Execute siempre se actualiza este DataTable y se utiliza en el momento de la navegación. A diferencia del método _Resultados, nos podemos encontrar que por algún motivo el DEVELOPER necesite quitar o añadir columnas.

1.1.2. Métodos

`Virtual void _SetDefaultValues(Dictionary<string, object> toControlsFiltros = null, Dictionary<string, object> toControlsOpciones = null);`

Método que se utiliza para establecer valores por defecto en la capa visual. Normalmente no se debería establecer valores por defecto ya que los filtros ya tienen establecido por defecto estos valores. En caso que se tuviera que modificar un valor por defecto de la capa visual utilizaremos este método.

- `Dictionary<string, object> toControlsFiltros:` es un diccionario que contiene una referencia a todos los filtros establecidos **en la capa visual**.
- `Dictionary<string, object> toControlsOpciones:` es un diccionario que contiene una referencia a todos los objetos visuales del grupo de opciones de la capa visual.

En este mismo método podemos establecer si es necesario que un Filtro no aparezca el Desde/Hasta estableciendo la propiedad _Intervalo=False del objeto txtFiltroMinMax.

Ejemplo de código:

```
public override void _SetDefaultValues(Dictionary<string, object> toControlsFiltros = null,
Dictionary<string, object> toControlsOpciones = null)
{
        (txtCajaDesdeHasta)toControlsFiltros["txtCajaDesdeHasta"]=_Intervalo = false;
}
```

`Virtual void _SetColumnCaptions(Dictionary<string, string> tldTranslate = null);`

Método que se utiliza para establecer los títulos que se asignan a cada una de las columnas de un datatable. Por defecto la clase ya controla unos nombres estándares de columnas.

- `Dictionary<string, string> tldTranslate = null:` diccionario que se utiliza para establecer el título de cada columna del Datatable donde el primer parámetro del diccionario es el nombre de la columna y el segundo parámetro es el título de la columna. Este parámetro es opcional por si el programador quiere cambiar alguno de los títulos establecidos en la clase base o si quiere añadir alguna cadena de traducción.

El sistema de listados automáticamente ya tiene en consideración si .NET trabaja con peso, cajas, tallas y colores, obra, series de artículo y lotes

En caso que un campo del Datatable contiene un campo que contiene las palabras:

- “cajas” y wl_Cajas = false entonces borrará la columna
- “talla” y wl_Color = false entonces borrará la columna
- “color” y wl_Color=false entonces borrará la columna
- “peso” y wl_peso=false entonces borrará la columna
- “obra” y wl_obra=false entonces borrará la columna
- “serieart” y wl_series=false entonces borrará la columna
- “lote” y wl_lotes=false entonces borrará la columna

`Virtual void _SetColumnWidth()`

Método que permite establecer la anchura de las columnas que forman el listado.

`virtual void _SetGraficasConfig()`

Método que permite definir los objetos de tipo Gráfica que se establecerán para el listado. Si no se sobreescribe el método en una clase heredada y por otra parte el campo GRAFICA de la tabla LISTADOS contiene un XML de definición de gráfica, éste se configurará solo de forma automática.

Definición del XML para el campo GRAFICA de la tabla LISTADOS

```
<GRAFICAS>
	<GRAFICA>
		<CLAVE>Clave única aleatoria</CLAVE>
		<TITULO>Título de la gráfica</TITULO>
		<TIPOGRAFICA>Tipo de gráfica que debe presentar</TIPOGRAF>
		<X>
			<DESCRIPCION>Descripción del campo X</DESCRIPCION>
			<CAMPO>Nombre del campo del DataTable que formará parte del
			eje de las X</CAMPO>
		</X>
		<YS>
			<Y>
				<DESCRIPCION>Descripción del campo Y<DESCRIPCION>
				<CAMPO>Nombre del campo del DataTable que formará
				parte del eje de las Y</CAMPO>
			</Y>
			…
		<YS>
	 </GRAFICA>
	...
</GRAFICAS>
```

Un ejemplo de sobreescritura del método sería:

```
public override void _SetGraficasConfig()
{
	base._SetGraficasConfig();

	//Grafica de ventas
	Grafica loGraficaVentas = new Grafica();
	loGraficaVentas._Listado = this;
	loGraficaVentas._Titulo = "Estadística de arqueos - Ventas";
	loGraficaVentas._X = new Grafica.GraficaEje("Fecha", "Fecha");
	loGraficaVentas._Y.Add(new Grafica.GraficaEje("Importe tickets", "Tickets"));
	loGraficaVentas._Y.Add(new Grafica.GraficaEje("Importe albaranes", "Albaranes"));
	loGraficaVentas._Y.Add(new Grafica.GraficaEje("Importe facturas", "Facturas"));
	loGraficaVentas._Y.Add(new Grafica.GraficaEje("Importe total", "Total Ventas"));
	this._Graficas.Add("ventas", loGraficaVentas);
	this._NavigateButtonsGraficas.Add("ventas", "Gráfica de ventas");
}
```

Codificando este método donde se crea un objeto del tipo Grafica (esta clase está definida en la librería sage.ew.listados.dll) y se definen sus propiedades entonces de forma automática aparecerá un nuevo botón llamado “Gráfica” al lado del botón de “Salir” del listado donde si el usuario hace click presentará la pantalla de gráficas.

En caso que en el método _SetGraficasConfig() se hubiera definido N gráficas entonces al hacer click sobre el botón de “Gráficas” en el listado nos presentará un menú con las diferentes opciones que el usuario ha definido como posibles gráficas.

En caso que a partir de un DataTable se quiera crear una gráfica se podría utilizar la extensión __ToListado() que permite convertir un DataTable en un Listado y por tanto te permite crear una gráfica a partir del DataTable.

Virtual void _Navigate(String tcKeyNavigate, DataRow trRowList)
Método que se utiliza para definir "que hacer" cuando el usuario hace click sobre un botón de navegación. Los parámetros que hay son

- **String tcKeyNavigate:** clave especificada en el diccionario _NavigateButtons
- **DataRow trRowList:** DataRow que actualmente está seleccionado en el grid. En caso que no haya ningún registro activo siempre contendrá el primer registro.

Existe una relación entre el _DataTable creado y el DataRow que pasa como parámetro en el método _Navigate. En el método _DataTable se puede crear como resultado final un _DataTable com múltiples campos que después necesitemos para poder navegar.

Si suponemos que creamos un DataTable con un campo CodigoCaja que lo necesitamos para poder navegar entonces podemos establecer los campos que presentará el listado pero cuando el usuario haga click sobre el botón de "Navegar" el parámetro trRowList que se pasa al método _Navigate contendrá el campo "CodigoCaja".

Virtual void _NavigateChecks(String tcKeyNavigate, DataRow trRowList);
Método que se utiliza para codificar las posibles navegaciones de un listado cuando está marcado el check de la nueva columna de checks que se puede configurar en el listado.

Virtual void _SetGraficasConfig();
Método que permite definir los diferentes tipos de gráficas que podemos llegar a ejecutar a nivel de las gráficas asociadas a un listado.

bool _PrintPersonalize(DataTable tdtDataTableListado);
Método para poder personalizar la impresión del listado pasando otro DataTable y realizando otro proceso personalizado.

Virtual void _NavigateGrafica(String tcKeyGrafica)
Método que permite navegar hacia a una gráfica concreta a partir de la clave. Si no existe necesidad de realizar algo concreto no se debe sobrescribir el método ya que de forma automática presenta el formulario de gráficas a partir de la definición establecida en el método _SetGraficasConfig().

Un ejemplo de codificación de este método se puede encontrar en el listado de "Listado de ventas por series". El código es el siguiente:

```
/// <summary>
/// Método que define las gráficas disponibles para las estadísticas de arqueos
/// </summary>
public override void _SetGraficasConfig()
{
        bool llAgruparSerie = true;

      //Opción de Agrupar por serie
      llAgruparSerie = (_Opcion_Entero("nAgrupadoDesglosad") == 0);

        if (llAgruparSerie)
      {
                base._SetGraficasConfig();

            //Grafica de ventas
            Grafica loGraficaVentas = new Grafica();
              IListados loListado = this._Resultados.__ToListado(true);
```

```
                loGraficaVentas._Listado = loListado;

                loGraficaVentas._Listado = this;
                loGraficaVentas._Titulo = "Ventas por serie de albarán";
                loGraficaVentas._X = new Grafica.GraficaEje("Serie", "serie", "Serie");
                loGraficaVentas._Y.Add(new Grafica.GraficaEje("Importe", "importe", "Importe"));
                loGraficaVentas._Y.Add(new Grafica.GraficaEje("beneficio", "beneficio", "Beneficio"));
                this._Graficas.Add("serie", loGraficaVentas); //Añado al diccionario de gráficas
                this._NavigateButtonsGraficas.Add("serie", "Ventas por serie de albarán"); //Añado al
diccionario de navegación
            }
            else
            {
                this._Graficas.Remove("serie");
                this._NavigateButtonsGraficas.Remove("serie");
            }
        }

/// Añado la gráfica al diccionario
/// </summary>
public override Dictionary<string, string> _NavigateButtonsGraficas
{
        get { return _dicNavigateButtonsGraficas; }
        set { _dicNavigateButtonsGraficas = value; }
}
```

Virtual List<GoogleMaps.Point> _GoogleMaps(String tcClave, out String tcLeyenda)
Devuelve una lista de GoogleMaps.Point en función de la clave pasada como parámetro. Una vez devuelta la lista de puntos de Google, éstos son presentados de forma automática en un mapa de Google Maps.

Virtual void _ConvertVisualFiltersOptionsToObjects(Dictionary<string,object> toFiltrosVisuals=null, Dictionary<string, object> toOpcionesVisuals=null);

Método que traduce los objetos de la capa visual tanto filtros como opciones en objetos no visuales. Esta conversión en el 90% de los casos es automática y no hace falta realizar ninguna programación. En caso que se añadiera un filtro especial o una opción especial el programador debería establecer la conversión entre el objeto visual y no visual.

- `Dictionary<string,object> toFiltrosVisuals=null`: diccionario de objetos que contiene una referencia a los filtros visuales del listado.
- `Dictionary<string, object> toOpcionesVisuals=nul`: diccionario de objetos que contiene una referencia a las opciones visuales del listado.

Una vez ejecutado este método las propiedades _Filtros() y _Opciones() estarán actualizadas La conversión que realiza es la siguiente:

Virtual bool _Valid();

Método que se utiliza para validar si existen incompatibilidades entre las diferentes opciones de las propiedades _Filtros u _Opciones por parte del usuario en el listado. En caso que hubiera alguna incompatibilidad entonces el programador describiría la incompatibilidad en la propiedad _Error_Message y devolverá un valor FALSE.

Virtual void _SetColumnCaptions(Dictionary<string, string> tldTranslate = null);
Método que se utiliza para establecer el título de cada una de las columnas del DataTable.

Virtual void _SetColumnWidth(Dictionary<string, int> tldTranslate = null);
Método que se utiliza para establecer la longitud de cada una de las columnas del DataTable.

Virtual void _SetColumnMasks();

Especifica la máscara de un campo numérico. Guarda la mascara en la propiedades extendidas de la columna. En el momento de contruir el DataGridView con los resultados (FormListado._ShowResult()), tendra en cuenta dicha máscara.

```
Virtual DataTable _DataTable();
```

Método que se utiliza para implementar el DataTable resultante que presentará el listado. Para poder filtrar el DataTable se utilizará las propiedades de _Filtros y _Opciones.

```
Virtual bool _Process();
```

Método que se utiliza para implementar un proceso. Se considera un proceso como aquella pantalla de .NET que no genera como resultado final un DataTable sino que realiza un proces. Ejemplos de proceso son: recalculo de stock, proceso de facturación general, repetición de facturas, impresión masiva de albaranes de venta, etc…

1.2. Funciones tratamiento de filtros y opciones

Tipos de filtros:

- `public string _Desde { get; set; }`
- `public string _Hasta { get; set; }`
- `public string _Unico { get; set; }`
- `public List<string> _Lista { get; set; }`

Métodos para el tratamiento de filtros y opciones:

```
protected string _Filtro_String(Dictionary<string, object> toFiltros, string teFiltro, string tcAlias, string tcCampo, _Operador_Condicional teOperador = _Operador_Condicional.And, bool tlAddCoalesce = false)
```

Este método nos devuelve un String como resultado de buscar en el diccionario de filtros un filtro especifico.

Ejemplo:

```
String lcFiltros="";
lcFiltros+=_Filtro_String(_Filtros, _Tipo_Filtro_String.Cliente, "a", "codigo");
```

En el anterior ejemplo, en función del tipo de selección que haya especificado el usuario en el filtro de cliente el string “lcFiltros” contendrá:

1. En caso que no haya establecido ningún código de cliente en desde ni en hasta. En este caso el valor de lcFiltros será vacío.
2. En caso que se haya establecido un rango de clientes, por ejemplo del 43.1 hasta al 43.4.
    a. lcFiltros=”And a.codigo >= '43000001' AND a.codigo <= '43000004'”
3. En caso que se haya establecido un solo cliente. Por ejemplo el 43.3
    a. lcFiltros= '' And a.codigo = '43000004'"
4. En caso que el usuario haya seleccionado códigos de clientes no seguidos, entonces tendrá el siguiente valor
    a. lcFiltros=” And a.codigo IN (And a.codigo IN ('43000001','43000004','43000029')

```
protected string _Filtro_Fecha(Dictionary<string, object> toFiltros,
_Tipo_Filtro_Fecha teFiltro, string tcAlias, string tcCampo,
_Operador_Condicional teOperador = _Operador_Condicional.And)
```

Este método realiza la misma operación que el anterior método pero para campos de tipos fechas. Un ejemplo sería:

```
lcFiltros+=_Filtro_Fecha(_Filtros, _Tipo_Filtro_Fecha.Fecha, "a", "f_alta");
```

En este caso los posibles valores devueltos son:

1. No selecciono ni modifico el filtro de fechas en el listado
    a. lcFiltros= “And a.f_alta >= '1/1/2016' AND a.f_alta <= '31/12/2016'”
2. En caso que selecciono una fecha
    a. lcFiltros=” And a.f_alta = '6/1/2016' “
3. En caso que haya una selección multiple de fechas
    a. lcFiltros=” And a.f_alta IN ('4/1/2016','7/1/2016','14/1/2016')”

```
protected bool _Filtro_Vacio()
```

Nos devuelve TRUE en caso que un filtro determinado esté vacío. En caso contrario nos devuelve FALSE.

```
protected int _Opcion_Entero(string tcFiltro, int tnValor_Opcion = 0)
```

Método que nos devuelve un valor entero del diccionario de “opciones” que pasamos como parámetro de un listado. En caso que no encuentre la opción dentro del diccionario nos devolverá el valor por defecto que se haya pasado como parámetro, en este caso un valor entero.

```
protected bool _Opcion_Logico(string tcFiltro, bool tlValor_Opcion = false)
```

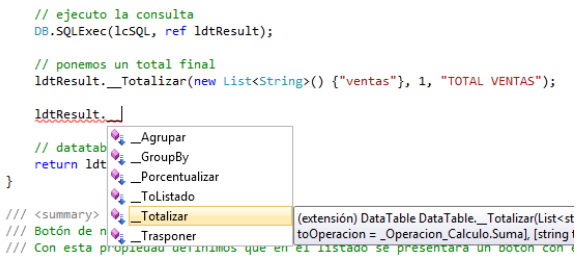
Método que nos devuelve un valor lógico del diccionario de “opciones” que pasamos como parámetro de un listado. En caso que no encuentre la opción dentro del diccionario nos devolverá el valor por defecto que se haya pasado como parámetro, en este caso un lógico.

```
protected string _Opcion_String(string tcFiltro, string tcValor_Opcion = "")
```

Método que nos devuelve un valor string del diccionario de “opciones” que pasamos como parámetro de un listado. En caso que no encuentre la opción dentro del diccionario nos devolverá el valor por defecto que se haya pasado como parámetro, en este caso un string.

1.3. Extensiones sobre datatables

Para poder implementar un listado debemos tener el siguiente using: `using sage.ew.listados;` al añadir esta librería entonces tenemos una serie de métodos que se pueden aplicar sobre un objeto de tipo DataTable:



Extensiones disponibles para el objeto DataTable

- __Agrupar()
- __GroupBy
- __Porcentualizar()
- __ToListado()
- __Totalizar()
- __Trasponer()

```
public static DataTable __Agrupar(this DataTable tdtDatos, string tcCodigoAgrupa,
string tcNombreAgrupa, string tcCampoTotaliza, bool tbTotalizar = true, string
tcNombreAgrupaDefault = "")

public static DataTable __Agrupar(this DataTable tdtDatos, string tcCodigoAgrupa,
string tcNombreAgrupa, List<string> tcCampoTotaliza, bool tbTotalizar = true,
bool lbTitulo = true, bool lbAddColumnTitulo = false, string
tcNombreAgrupaDefault = "")

public static DataTable __Agrupar(this DataTable tdtDatos, List<string>
tcCodigoAgrupa, string tcNombreAgrupa, List<string> tcCampoTotaliza, bool
tbTotalizar = true, bool tbListaCampos = false, int tnColumnaTituloTotal = 0)

public static DataTable __Agrupar(this DataTable tdtDatos, List<string>
tcCodigoAgrupa, List<string> tcNombreAgrupa, List<string> tcCampoTotaliza, bool
tbTotalizar = true)
```

Extensión que nos permite agrupar un DataTable en función de una serie de campos. Esta extensión también nos permite realizar subtotales. En el momento que el datatable entra en la función de Agrupar, éste debe estar ordenado por el código de agrupación.

Ejemplo:

```
public static DataTable __GroupBy(this DataTable tdtDatos, List<string>
tcCamposAgrupa, List<string> tcCamposSum)
```

Extensión que nos permite agrupar los datos de un datatable en función de una serie de campos y nos totaliza las columnas que son pasadas como lista.

```
public static DataTable __Porcentualizar(this DataTable tdtOrigen, List<int>
tnColumnas, bool tlIncluyeTotales = false)
```

Extensión que nos devuelve un DataTable con unas nuevas columnas que queremos calcular el %.

```
public static DataTable __Totalizar(this DataTable tdtOrigen, List<int>
tnColumnas, int tnColTexto = -1, string tcTitulo = "",
```

```
Listados._Operacion_Calculo toOperacion = Listados._Operacion_Calculo.Suma,
string tcCondicion = "")

public static DataTable __Totalizar(this DataTable tdtOrigen, List<string>
tcColumnas, int tnColTexto = -1, string tcTitulo = "",
Listados._Operacion_Calculo toOperacion = Listados._Operacion_Calculo.Suma,
string tcCondicion = "")
```

Ejemplos:

- `ldtFinal = ldtFinal.__Totalizar(loListColumnas, 0, "Total");`

```
public static DataTable __Trasponer(this DataTable tdtOrigen, string tcNombre =
"")
```

Extensión de listados para trasponer tablas. Se debe tener en cuenta que la primera columna debe contener registros únicos para poder trasponer.

```
public static IListados __ToListado(this DataTable tdtOrigen, List<string>
tlisColumnas)

public static IListados __ToListado(this DataTable tdtOrigen, bool
tlEliminar_bold = false)
```

Extensión que nos convierte un DataTable en un listado de Sage 50.

```
Grafica loGraficaVentas = new Grafica();
IListados loListado = this._Resultados.__ToListado(true);
loGraficaVentas._Listado = loListado;


if (sage._50.main_s50.Connect(lcInstancia, lcUser, lcPass, lcDBComunes,
lcEmpresa))
            {
                ListadosPersonalizables loList2 = new ListadosPersonalizables();
                //Cargar un listado en concreto
                loList2._CargarListado("ELMEULLISTAT");
                //Generar un nuevo liustado
                loList2._NuevoListado();
                //Duplicar un listado
                loList2._DuplicarListado("VEN_CA");

            }
```

2. Configuración de un proyecto para poder realizar la implementación de un listado y depurarlo

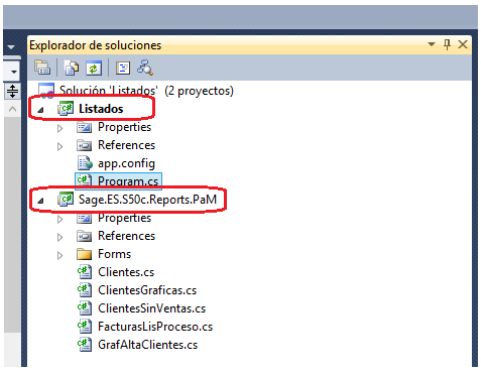
Para poder implementar un listado es necesario los siguientes requisitos:

1. Tener instalado .Net Framework 4.0
2. Tener instalado un terminal de Sage 50c
3. Haber definido el diseño visual de un listado mediante Sage 50c.
4. Tener instalada la versión Visual Studio 2010 o superior

El ejemplo se presenta mediante Visual Studio 2010.

Para poder desarrollar un proyecto de tipo listados y poder depurarlo en Visual Studio es necesario crear una solución de .Net donde albergue los siguientes proyectos:

1. Proyecto donde el objetivo de éste será realizar la comunicación y conexión con Sage 50. Este proyecto debe ser un proyecto de tipo **"Aplicación de Windows Form"**. En ningún caso se pasará el fichero .EXE.
2. Proyecto donde se definen las clases de listados que se quieren ejecutar desde Sage 50. Este proyecto debe ser un proyecto de tipo **"Biblioteca de clases"** y por tanto una vez compilado generará un fichero de tipo .DLL que será la librería que deberemos instalar en Sage 50c.

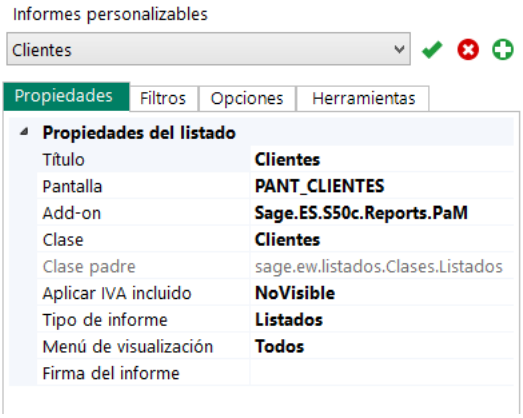


En la imagen se ve la definición de una solución en .Net que está formado por 2 proyectos: el proyecto Listados de tipo "Aplicación de Windows Form" y el proyecto Sage.ES.S50c.Reports.PaM que es un proyecto de tipo "Bibliotecas de clases".

2.1. Pasos para la configuración

2.1.1. Diseño visual de los filtros

Visualmente el usuario puede crear un listado yendo al menú Herramientas / Configuración / Informes personalizables.

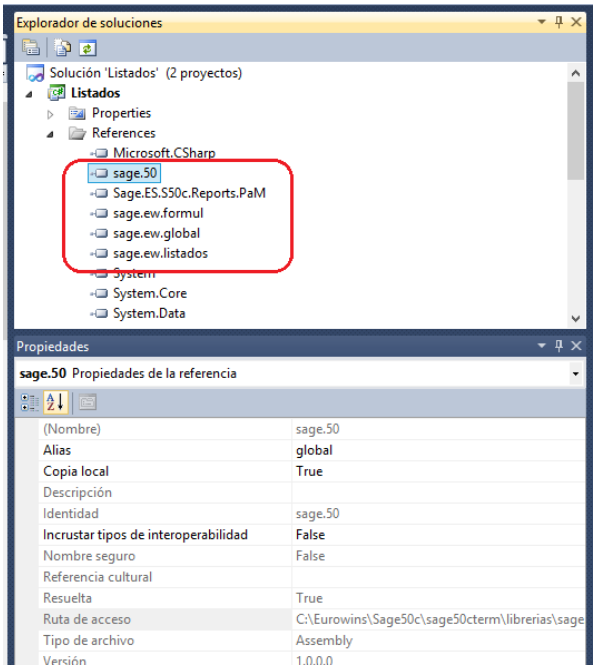


Propiedad	Definición
Título	(string) Título del listado
Pantalla	(string) Nombre único para identificar
Add-on	(string) Nombre de la librería donde hay la implementación del listado
Clase	(string) Nombre de la clase del listado
Clase padre	(string) Nombre de la clase en que se basa esta nueva clase
Aplicar IVA incluido	(boolean) Nos identifica si en el listado presentará un check para incluir los precios con IVA incluido.
Tipo de informe	(lista) identifica el tipo de informe como se va presentar: informe o gráfica.
Menú de visualización	(lista) identifica en que menú debe aparecer el nuevo listado. Debe aparecer en todos los menús (ventas, compras y contabilidad) o solo en uno.
Firma de informe	(string) firma que nos identifica que este listado es único. En caso de exportación de listado o importación nos pedirá que introduzcamos este password.

2.1.2. Establecer las referencias en el proyecto principal

Para poder ejecutar en un proyecto de Visual Studio un listado debemos añadir las siguientes referencias al proyecto de tipo “Aplicación de Windows Form”, en este ejemplo debemos añadir las referencias al proyecto “Listados”

Librería	**Ruta donde se encuentra la librería**
Sage.50	[Ruta del terminal de Sage 50]\librerias\sage.50.exe
Sage.ew.formul	[Ruta del terminal de Sage 50]\librerias\sage.ew.formul.dll
Sage.ew.global	[Ruta del terminal de Sage 50]\librerias\sage.ew.global.dll
Sage.ew.listados	[Ruta del terminal de Sage 50]\librerias\sage.ew.listados.dll
sage.ew.license.dll	[Ruta del terminal de Sage 50]\librerias\sage.ew.license.dll



También deberá tener una referencia al proyecto de tipo “Biblioteca de clases”, en este ejemplo sería al proyecto “Sage.ES.S50c.Reports.PaM”

2.1.3. Establecer el código base en program.c

Para poder arrancar la aplicación y poder conectarse con Sage 50c deberemos tener las siguientes líneas de código en program.c del proyecto de tipo “Aplicación de Windows Forms”.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;

using sage._50;
using sage.ew.formul;
using sage.ew.formul.Forms;
using sage.ew.global;

namespace Listados
{
    static class Program
    {
        /// <summary>
        /// Punto de entrada principal para la aplicación.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);

            // conectamos hacia Sage 50 que tenemos instalado
            main_s50.Connect(@"C:\Eurowins\Sage50c\sage50cterm\", "SUPERVISOR", "1");
```

Referencias que debemos que tener para poder ejecutar un listado

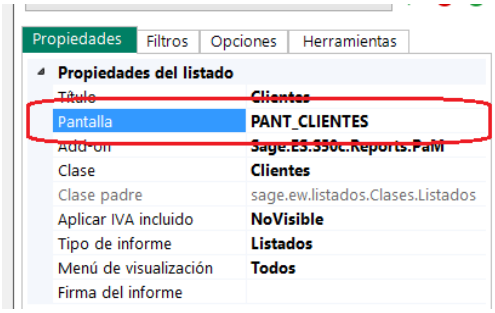
El método Connect() es necesario para que desde VS podamos acceder a los datos de Sage 50c. En este caso es necesario proporcionar la ruta absoluta donde se encuentra el terminal de Sage 50c, el usuario con el cual nos queremos conectar y el password de este usuario.

```
// 1º listado --> listado básico
            Sage.ES.S50c.Reports.PaM.Clientes ClientesSinVentas = new
Sage.ES.S50c.Reports.PaM.Clientes();
```

Para poder ejecutar el listado en modo de depuración se debe instanciar la clase. En este ejemplo, esta clase está definida en el proyecto definido como “librerías de clases”

```
            FormListado loForm = new FormListado("PANT_CLIENTES");
```

Instrucción que nos ejecuta el listado, es muy importante pasar como parámetro el texto introducido en la propiedad del listado “Pantalla” en el diseño visual del listado.



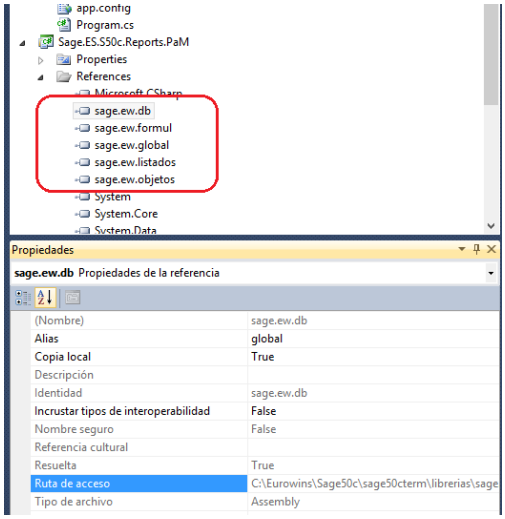

```
            loForm.ShowDialog();
        }
    }
}
```

2.1.4. Establecer la configuración básica para la creación del listado

El nombre de proyecto debe empezar por: Sage.ES.S50c.Reports.*.dll donde el "*" puede ser cualquier nombre. En caso que la librería no empiece por este nombre no se cargará en Sage 50c.

Las referencias mínimas que debe tener este proyecto son:

Librería	Ruta donde se encuentra la librería
Sage.ew.db	[Ruta del terminal de Sage 50]\librerias\sage.ew.db.dll
Sage.ew.formul	[Ruta del terminal de Sage 50]\librerias\sage.ew.formul.dll
Sage.ew.global	[Ruta del terminal de Sage 50]\librerias\sage.ew.global.dll
Sage.ew.listados	[Ruta del terminal de Sage 50]\librerias\sage.ew.listados.dll
Sage.ew.objetos	[Ruta del terminal de Sage 50]\librerias\sage.ew.objetos.dll

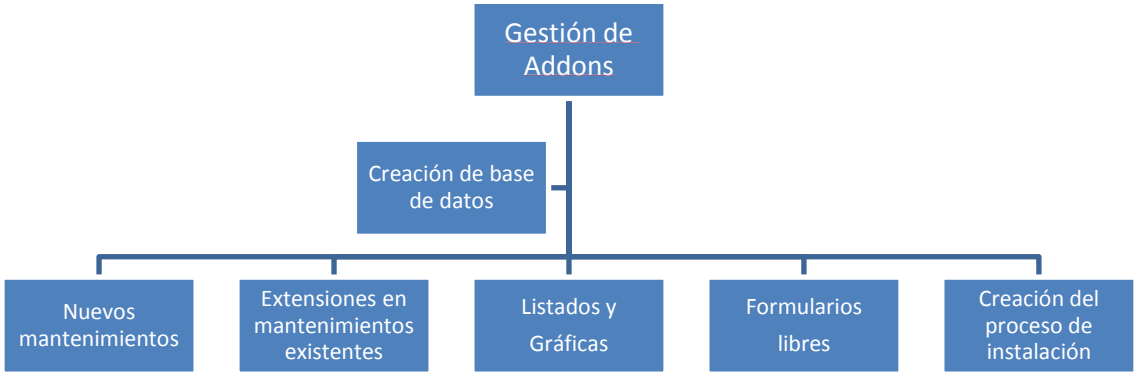


2.1.5. Ejemplo de creación de un listado

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;

using sage.ew.db;
using sage.ew.global;
using sage.ew.listados;
using sage.ew.listados.Clases;

namespace Sage.ES.S50c.Reports.PaM
{
    /// <summary>
    /// Clase de negocio para ListadoPersonalizable1
    /// </summary>
    public class Clientes : sage.ew.listados.Clases.Listados
    {}
}
```

3. Gestión de Addons implementados en .NET para Sage 50c (Plugin de desarrollo avanzado).

La gestión de Addons forma parte de las capacidades de programación avanzada de Sage 50c más allá de la creación de listados y gráficas, permitiendo los siguientes aspectos:



- Creación de nuevos mantenimientos desde cero
- Modificación de mantenimientos ya existentes.
- Creación de nuevas pantallas y formularios desde cero.
- Creación de listados y gráficas en el addon (según comentado en puntos anteriores)
- Creación de base de datos del addon.
- Creación del proceso de instalación del addon.

La imagen visual de los mantenimientos, pantallas y formularios puede seguir el estilo visual de Sage 50c con el objetivo de integrarlos correctamente en el flujo de trabajo del cliente sin que perciba diferencias respecto a la funcionalidad estándar de la solución.

3.1 Preparación de Visual Studio para crear proyectos (addons)

El Plugin de desarrollo avanzado requiere de la instalación de una extensión de Sage 50c en Visual Studio con el objetivo de poder crear proyectos (addons). Dicho componente se encuentra en el directorio de la carpeta de instalación del producto dentro de la carpeta "Plugin". Los pasos para realizar dicha instalación son los siguientes:

Paso 1. Añadir la plantilla del proyecto en Visual Studio

Se debe añadir el zip con la plantilla del proyecto (sage.addons.base.zip) a la carpeta de Templates de Visual Studio. Esta se encuentra en:

C:\Users\TU_USUARIO\Documents\Visual Studio 20XX\Templates\ProjectTemplates

Paso 2. Registrar la assembly en Visual Studio

Para que Visual Studio muestre el asistente del módulo tras crear un nuevo proyecto del tipo módulo, se debe registrar la dll del proyecto al Sistema Global de Cache global de VS. Para ello utilizaremos la herramienta Gacutil.exe.

- Copiar la librería sage.addons.wizard.dll en un directorio permanente, de donde no se elimine y posteriormente:

  1. Abrir el "Simbolo de Sistema de Visual Studio" como Administrador

  2. Registrar la dll: C:\>gacutil /if <directorio con la dll>\sage.addons.wizard.dll

- Ayuda sobre gacutil:
  - Registrar: gacutil /if "ruta_hacia_la_dll/"
  - Desregistrar : gacutil /uf sage.addons.wizard.dll
  - Listar nuestra assembly: gacutil /l sage.addons.wizard.dll (la f fuerza la acción)
    Para más información: Gacutil.exe (Global Assembly Cache Tool) :
    https://msdn.microsoft.com/en-us/library/ex0ss12c.aspx

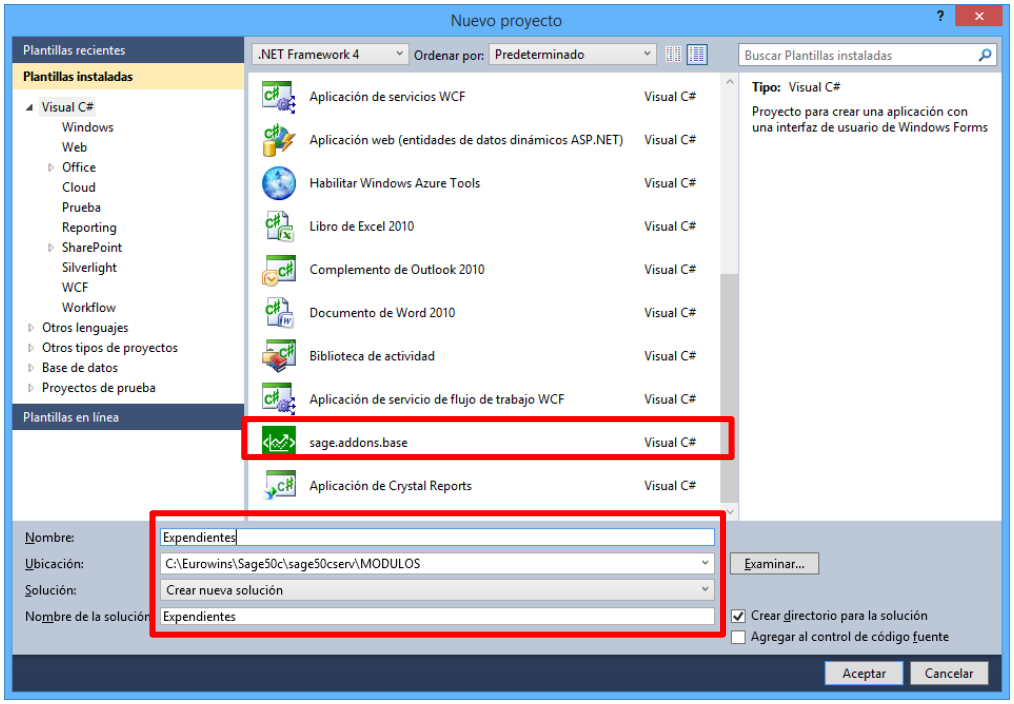
Paso 3. Instalar la extensión de Sage 50c

Finalmente deberá irse a la carpeta Plugin de la carpeta de instalación de Sage 50c comentada anteriormente, seleccionar la carpeta correspondiente a la versión de Visual Studio y ejecutar el fichero.

3.1 Arranque del Plugin desde Visual Studio

3.1.1 Creación de un projecto de tipo Addon

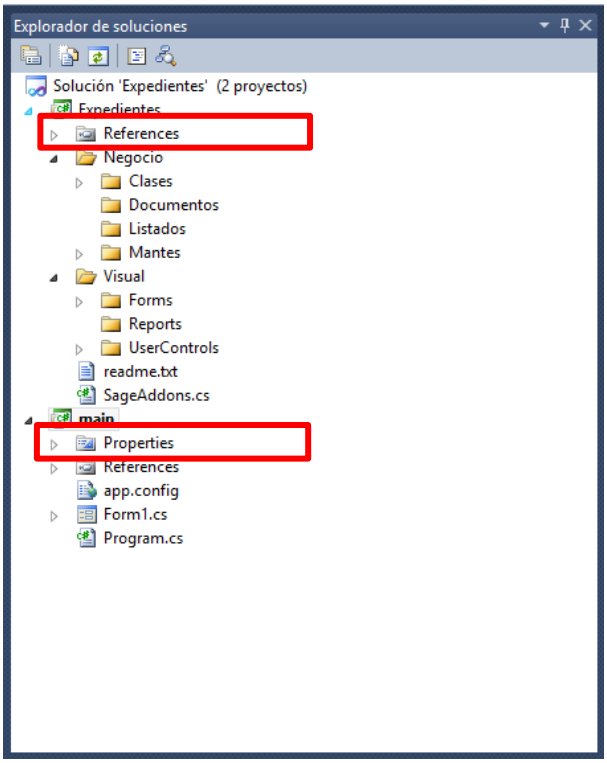
1. Seleccionar el tipo de proyecto sage.addons.base en plantillas instaladas.
2. Se debe seleccionar la ruta donde se encuentra el servidor de Sage 50, carpeta MODULOS.
3. Como nombre del proyecto estableceremos el nombre del addon.
4. Se creará automáticamente el proyecto del addon en la carpeta Modulos de la instalación de Sage 50c.



3.1.2 Ejecutar Sage 50c desde Visual Studio

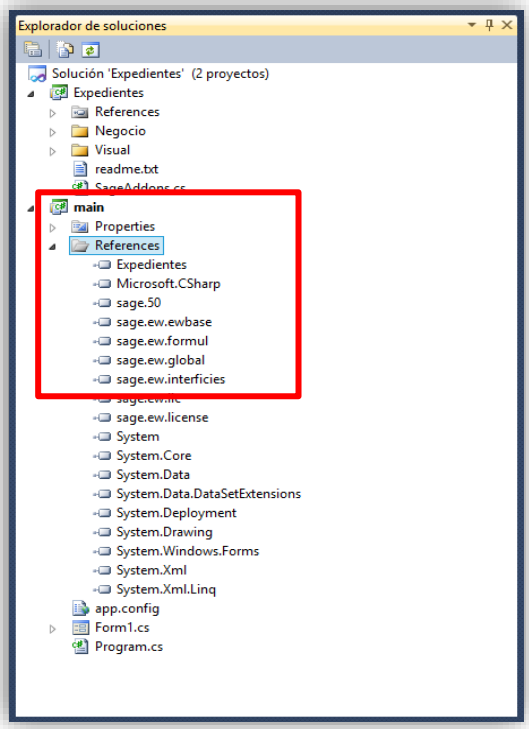
A modo de recordatorio de lo comentado en los apartados anteriores de este documento, a continuación, se detalla como ejecutar Sage 50c desde Visual Studio con el objetivo de poder probar las personalizaciones desarrolladas des del proyecto, así como depurar código apuntando a una instalación.

1. Tener 2 proyectos dentro de la solución.
2. El primer proyecto será un proyecto generado con la plantilla de sage.addons.base
3. El segundo proyecto será el proyecto que utilizamos para ejecutar Sage 50c desde Visual Studio. Este proyecto debe ser del tipo Aplicación EXE.
4. ¡Importante! Debe ser un proyecto de .Net framework 4.0; no puede ser un proyecto de .Net framework 4.0 Client Profile



5. Asociar las siguientes referencias del proyecto de tipo aplicación:

- sage.50
- sage.ew.base
- Sage.ew.formul
- Sage.ew.global
- Sage.ew.interficies
- Sage.ew.lic
- Sage.ew.licence



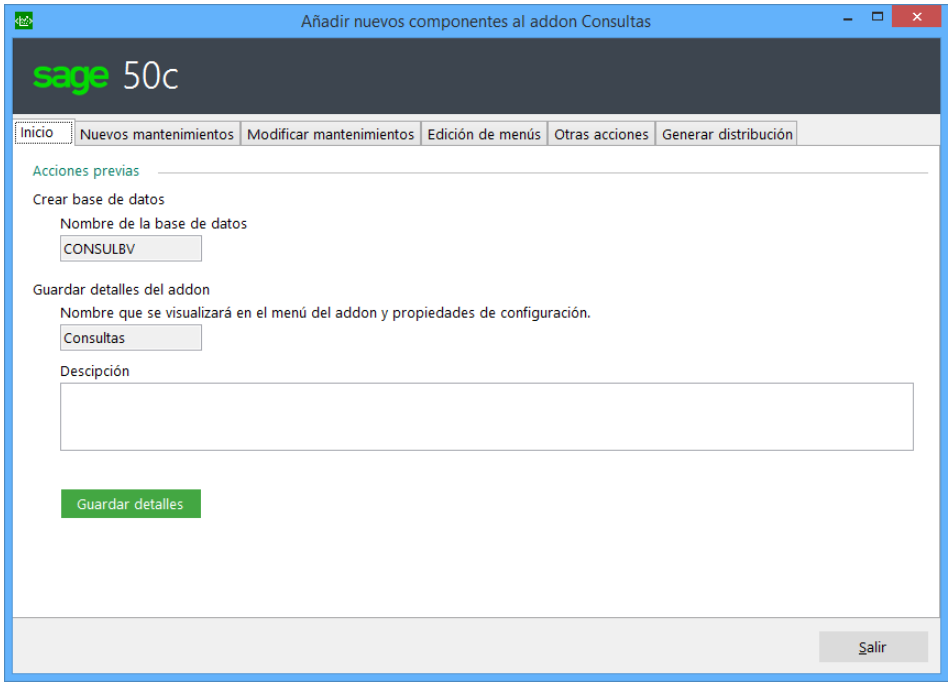
A continuación, se muestra un código de ejemplo de arranque de Sage 50c (debe tenerse en cuenta que en un entorno de Visual Studio, tan solo puede ejecutarse la parte de Sage 50c desarrollada en .NET):

```
using System.Windows.Forms;

using sage._50;
using sage.ew.formul;
using sage.ew.formul.Forms;
using sage.ew.global;

using sage.addons.Expedientes;

namespace main
{
    static class Program
    {
        /// <summary>
        /// Punto de entrada principal para la aplicación.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);

            // conectamos hacia Sage 50 que tenemos instalado
            main_s50.Connect(@"C:\Eurowins\Sage50c\sage50cterm\", "SUPERVISOR", "1");

            // presentamos el escritorio de Sage 50c, solo funcionan aquellas opciones implementadas en .NET
            main_s50._Show();
        }
    }
}
```

3.1.3 Base de datos del addon

Todo Addon tiene una base de datos asociada.

1. La base de datos se crea en el SQLServer asociado a la instalación de Sage 50c
2. Las tablas asociadas a los mantenimientos se crean en esta base de datos
3. Al crear la base de datos, nos crea 6 tablas de sistema:
    - Filtros → tabla donde se guardan todos los filtros del addon
    - Listados → tabla donde se guardan todos los listados creados en el addon
    - Mantes → tabla donde se guardan la definición de los mantenimientos
    - Menu50 → tabla donde se guarda la definición del menú del addon
    - Objetos → tabla donde se guardan los diferentes objetos del addon (*)
    - Config → tabla donde se define los diferentes campos ampliables



3.1.4 Métodos y propiedades para la ejecución de un addon

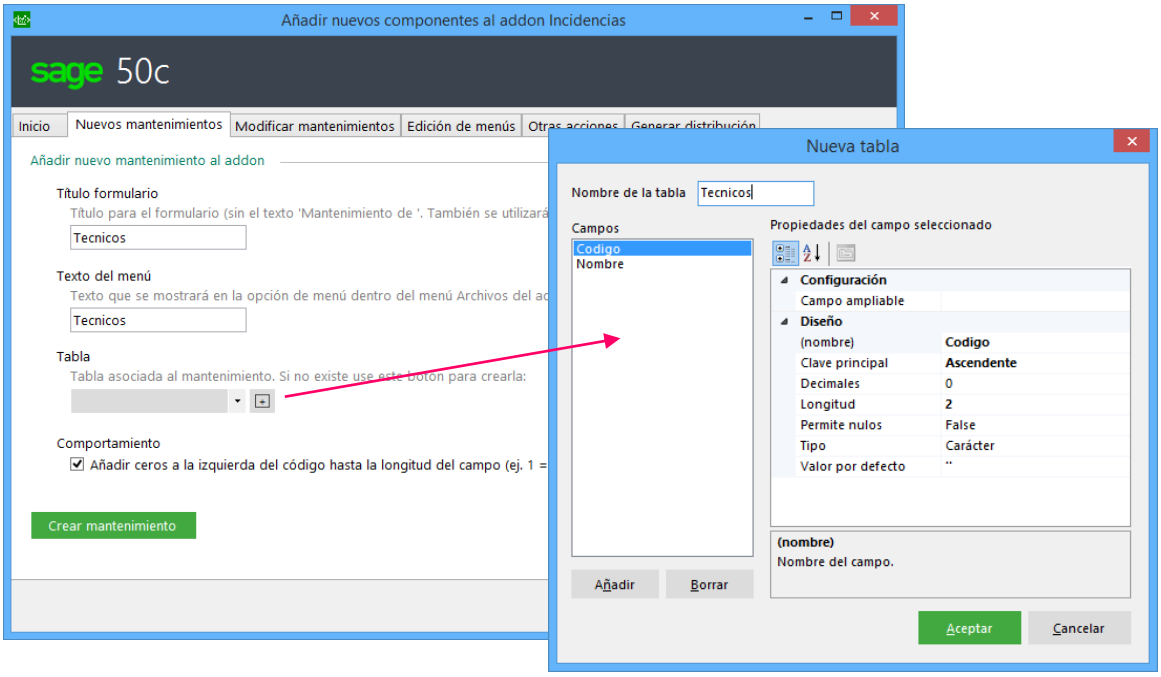
A continuación, se muestran los principales métodos y propiedades disponibles para la ejecución de un addon (SageAddons.cs - Interfaz: IModulo)

Método	Descripción
_CambiarEjercicio()	Se dispara cuando el usuario cambia de ejercicio fiscal en Sage 50c
_CambioEmpresa()	Se dispara cuando el usuario cambia de empresa en Sage 50c
_CambioUsuario()	Se dispara cuando cambiamos de usuario en Sage 50c
_Load()	Se dispara cuando cargamos el addon en Sage 50c
_Unload()	Se dispara cuando descargamos el addon de Sage 50c
_Extension()	Método que se utiliza para realizar extensiones sobre clases de tipo mantenimientos de Sage 50c
_BindForm()	Método que se utiliza para vincular la parte visual con la página de Addons de los mantenimientos de Sage 50c.

3.2 Creación de mantenimientos nuevos

Al crear un nuevo mantenimiento se realizan las siguientes acciones:

- Todo mantenimiento va relacionado con una tabla del addon (el addon lo crea en el SQLServer de forma automática)
- Los mantenimientos tienen 2 campos obligatorios: Codigo, Nombre. El usuario puede añadir más pero nunca borrar estos.



Al crear un nuevo mantenimiento genera de forma automática el siguiente código:

Tipo	Código generado	Basado en	Namesace dentro del addon
Generación de código	Clase de negocio del mantenimiento	ewMante	sage.addons.[addon].Negocio.Mantes
Generación de código	Formulario de mantenimiento	FormMante	sage.addons.[addon].Visual.Forms
Generación de código	Txtcodlabel asociado al mantenimiento	txtcodlabel	sage.addons.[addon].Visual.UserControls
Actualización datos	Nueva opción de menú dentro del addon		Tabla Menu50 en el [addon] – automáticamente crea los menús de Archivos / Listados

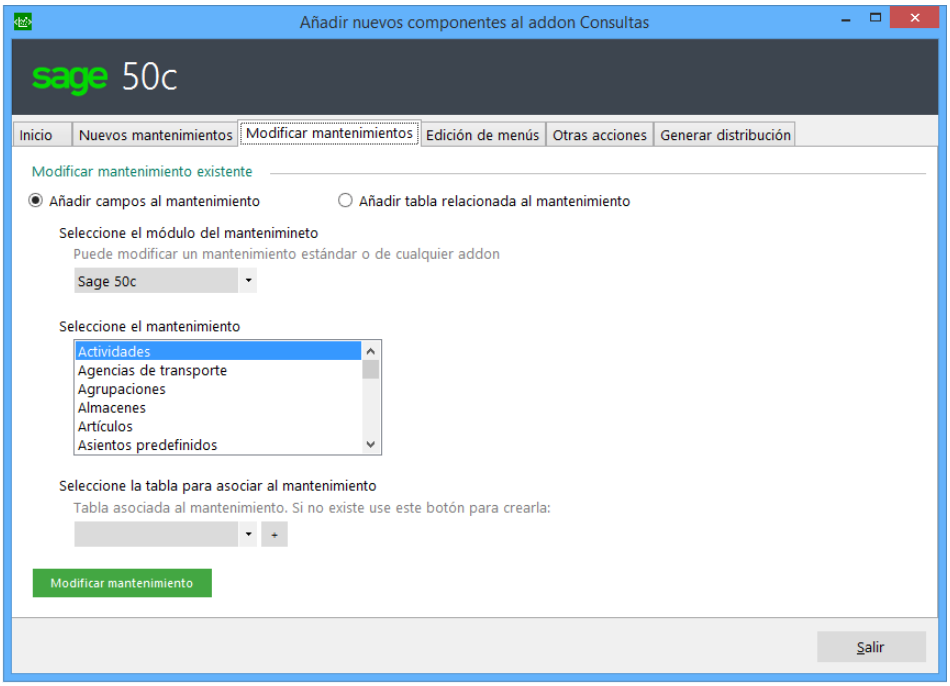
Actualización datos	Nuevo registro de tipo Mantenimiento		Tabla Mantes en el [addon]
Actualización datos	Nuevo registro de tipo Filtros		Tabla Filtros en el [addon]

3.3 Extensión de un mantenimiento ya existente

3.3.1 Personalización de un mantenimiento existente

Es posible modificar los mantenimientos ya existentes siempre según lo siguiente:

- Nos permite añadir objetos en mantenimientos de Sage 50c
- Ejemplos: Clientes, Artículos, Proveedores, Actividades, etc …
- Cualquier mantenimiento de Sage 50c que esté en .Net
- Implementar la interfaz **IExtensionMante**



A continuación, se muestran los principales métodos y propiedades disponibles:

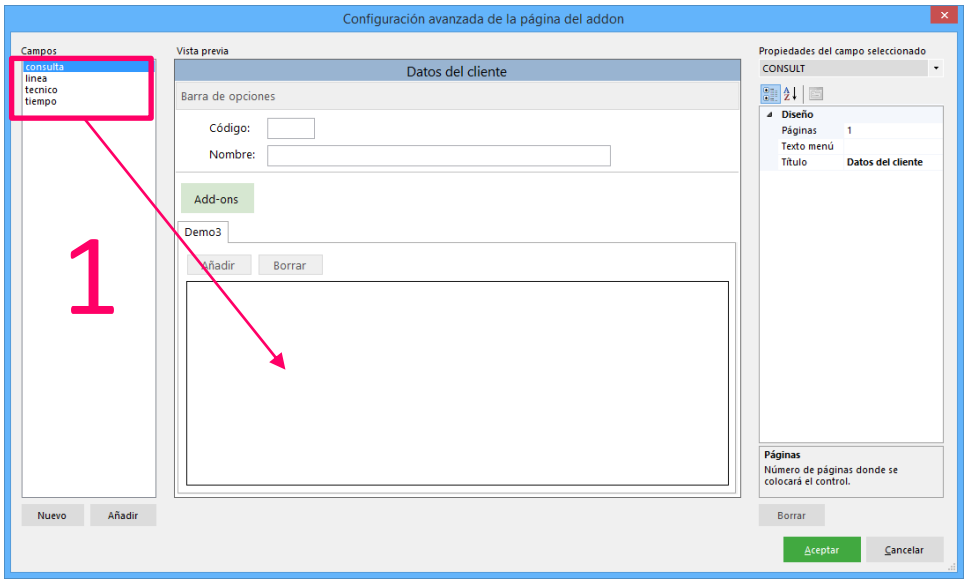
Tipo	Nombre	Descripción
Propiedad	_FormMante	Referencia al formulario del mantenimiento (visual)
Propiedad	_Mante	Referencia a la clase de negocio del mantenimiento
Método	_Init()	Se ejecuta cuando se crea el mantenimiento
Método	_New()	Se ejecuta cuando se crea un nuevo registro
Método	_Load()	Se ejecuta cuando se carga un nuevo registro
Método	_Save()	Se ejecuta cuando se guarda un registro
Método	_Delete()	Se ejecuta cuando se borra un registro

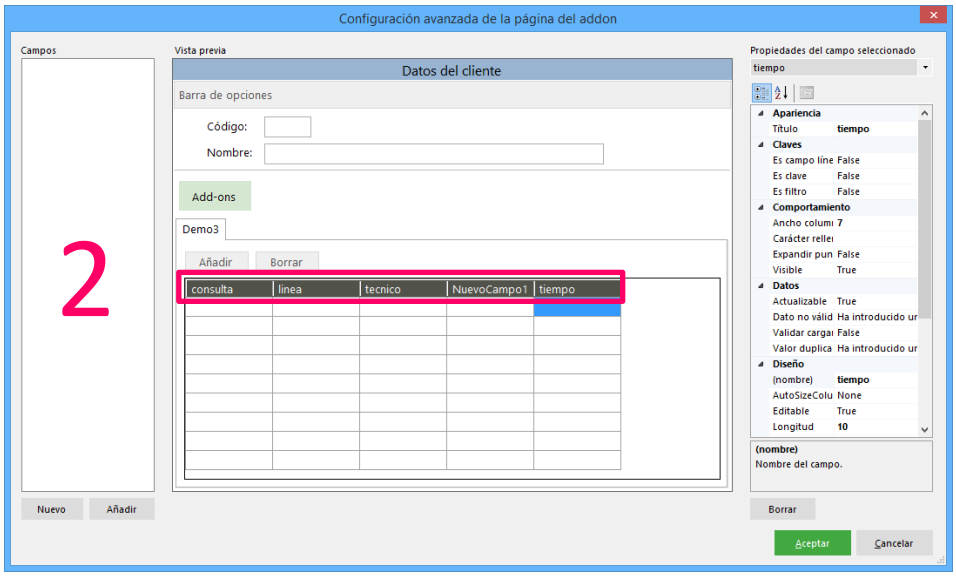
3.3.1 Asociación de una tabla relacionada a un mantenimiento

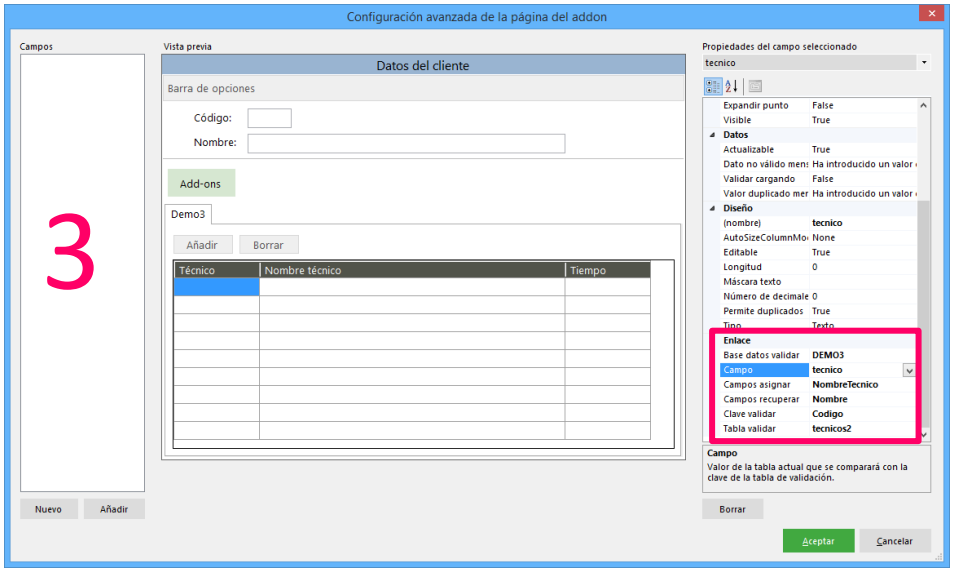
Para la asociación de una tabla relacionada a un mantenimiento hay que tener en cuenta lo siguiente:

- Creación de una clase ManteTRel<Linea, Clave>
    - Definición de la clase Linea
    - Definición de la clase Clave
- Clase **Clave**: coincide con la clave principal de la tabla, si el código lo genera de forma automática el plugin, tendrá el campo LINEA (entero)
- Clase **Linea**: definimos que campos aparecerán en esta lista asociada a un mantenimiento.

A modo de ejemplo, a continuación, se muestra un ejemplo de creación de un ManteRel en el cual se asocia una tabla de “Técnicos” relacionada:


Configuración avanzada de la página del addon
Campos
consulta
linea
tecnico
tiempo
1
Vista previa
Datos del cliente
Barra de opciones
Código:
Nombre:
Add-ons
Demo3
Añadir
Borrar
Propiedades del campo seleccionado
CONSULT
Diseño
Páginas 1
Texto menú
Título Datos del cliente
Páginas
Número de páginas donde se colocará el control.
Nuevo
Añadir
Borrar
Aceptar
Cancelar


Configuración avanzada de la página del addon
Campos
2
Vista previa
Datos del cliente
Barra de opciones
Código:
Nombre:
Add-ons
Demo3
Añadir
Borrar
consulta
linea
tecnico
NuevoCampo1
tiempo
Propiedades del campo seleccionado
tiempo
Apariencia
Título tiempo
Claves
Es campo líne False
Es clave False
Es filtro False
Comportamiento
Ancho columi 7
Carácter relle
Expandir pun False
Visible True
Datos
Actualizable True
Dato no válid Ha introducido ur
Validar carga False
Valor duplica Ha introducido ur
Diseño
(nombre) tiempo
AutoSizeColu None
Editable True
Longitud 10
(nombre)
Nombre del campo.
Nuevo
Añadir
Borrar
Aceptar
Cancelar


Configuración avanzada de la página del addon
Campos
3
Vista previa
Datos del cliente
Barra de opciones
Código:
Nombre:
Add-ons
Demo3
Añadir
Borrar
Técnico
Nombre técnico
Tiempo
Propiedades del campo seleccionado
tecnico
Expandir punto False
Visible True
Datos
Actualizable True
Dato no válido men Ha introducido un valor
Validar cargando False
Valor duplicado mer Ha introducido un valor
Diseño
(nombre) tecnico
AutoSizeColumnMo None
Editable True
Longitud 0
Máscara texto
Número de decimale 0
Permite duplicados True
Tipo Texto
Enlace
Base datos validar DEMO3
Campo tecnico
Campos asignar NombreTecnico
Campos recuperar Nombre
Clave validar Codigo
Tabla validar tecnicos2
Campo
Valor de la tabla actual que se comparará con la clave de la tabla de validación.
Nuevo
Añadir
Borrar
Aceptar
Cancelar

- Campo “consulta” y “línea”: deben tener la propiedad Visible=false
- Campo “NuevoCampo1”: propiedad (nombre) → NombreTecnico
- Campo “técnico”: se deben configurar las siguientes propiedades
    - Base datos validar: módulo en el cual estamos validando la información, en este caso DEMO
    - Campo: nombre del campo donde guardará el código del técnico, en este caso TECNICO
    - Campo asignar: nombre de la columna donde actualizará el nombre del técnico, en este caso será “NombreTecnico”
    - Campo a recuperar: nombre del campo que se recupera de la tabla “Tabla validar”, en este ejemplo es “Nombre”
    - Clave validar: nombre del campo que se utiliza para validar en la tabla “Tabla validar”, en este ejemplo es “Código”
    - Tabla validar: nombre de la tabla donde tenemos que validar el código del técnico, en este caso es “tecnico2”

De igual forma, a continuación, se muestra un ejemplo de código para instanciar ManteTRel en negocio. Para ello en la clase de negocio creamos una propiedad de tipo **consutec** que está derivada de ewManteTec

```
namespace sage.addons.Consultas.Negocio.Mantes
{
    public partial class consulta : ewMante
    {
        private consutec _tecnicosParticipados = null;
        public consutec _TecnicosAplicados
        {
            get
            {
                if (_tecnicosParticipados == null)
                {
                    _tecnicosParticipados = new consutec(_Codigo);
                    _tecnicosParticipados._ewMantePrincipal = this;
                    _tecnicosParticipados._Load();
                    this._AddManteTRel(_tecnicosParticipados);
                }
                return _tecnicosParticipados;
            }
        }

        /// <summary>
        ///  Fecha
```

El hecho de añadir en el diccionario de ManteTRel nos gestiona de forma automática el _Load(), _Save(), _New() a nivel de mantenimientos.

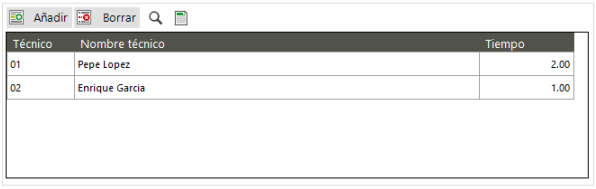
Con el objetivo de presentarlo integrado por ejemplo en un mantenimiento es posible realizar un link entre el ManteTREl y un mantegrid:

- Podemos añadir un objeto de tipo mantegrid (sage.ew.objetos.dll)
- Este objeto está formado por 3 objetos (botón añadir, botón borrar, grid)
- Si asignamos al ewManteTREL el mantegrid, automáticamente configura todo el grid

```
protected override void OnLoad(EventArgs e)
{
    base.OnLoad(e);
    if (_ewMante != null)
    {
        consulta loConsulta = (consulta)_ewMante;
        loConsulta._TecnicosAplicados._Grid = manteTecnicos;
    }
}
```

+

=

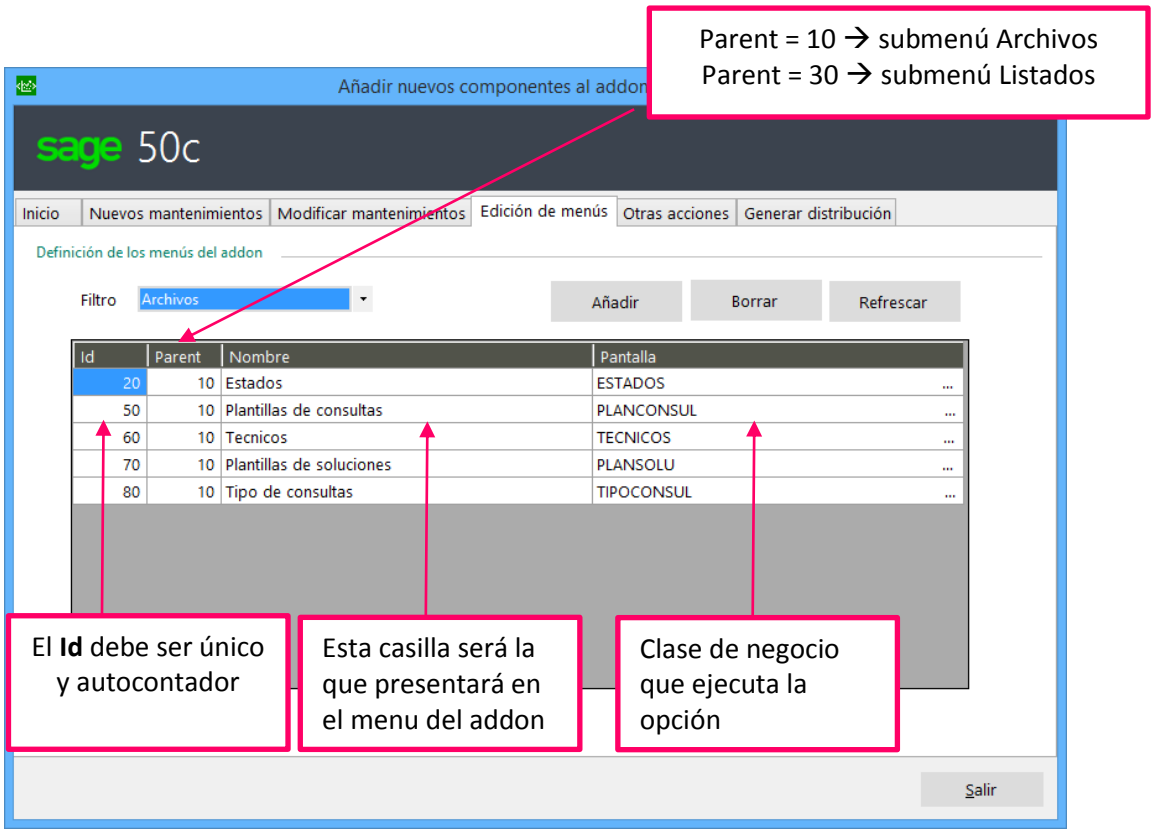


- Crea las columnas automáticamente
- Gestiona automáticamente F4 (browser)
- Gestiona automáticamente F5 (mante)

3.5 Menú de los addon

Es posible gestionar los menús de los addons según lo siguiente:

- Permite modificar y/o añadir nuevas opciones del menú del addon.
- Permite ver todo el menú o filtrar por los grupos de Archivos o Listados.
- Esta opción permite configurar una opción de menú para ejecutar un formulario independiente.
- Para añadir una opción de formulario libre debemos escribir en la casilla de Pantalla: FORM(“sage.addons.Incidencias.Visual.Forms.[Nombre]”)
- Toda esta información actualiza la tabla Menu50 del addon.
- Tan solo debe modificarse el menú del addon a partir de eta opción, de lo contrario puede no funcionar.



3.6 Pantallas y Formularios libres

Es posible crear pantallas y formularios libres desde 0 e incorporarlos dentro de un addon, tomando como base el estilo visual basado en uno de los siguientes formularios de Sage 50c:

- FormBase
- FormDialog
- FormSalir
- FormSalirClienteArticulo
- FormSalirProveedorArticulo

Estos formularios están definidos en:

- Librería: sage.ew.formul.dll
- Namespace: sage.ew.formul.Forms
- Llamada desdel Menu: FORM("sage.addons.Incidencias.Visual.Forms.frmDialog")

A continuación, se muestra un ejemplo de cómo iniciar la creación de un formulario desde 0 basado en el estilo visual de un formulario base ya existente de Sage 50c:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

using sage.ew.formul.Forms;

namespace sage.addons.Incidencias.Visual.Forms
{
    public partial class frmDialog : FormSalirClienteArticulo
    {
        public frmDialog()
        {
            InitializeComponent();
        }
    }
}
```

A partir de este punto la programación será libre a nivel de construcción visual y de negocio del formulario, pudiendo hacer referencia siempre al modelo de negocio de Sage 50c con el objetivo de interactuar con la aplicación pudiendo obtener datos, así como grabar información a partir de dicho modelo de objetos (por ejemplo, crear documentos, asientos, clientes, proveedores, etc…).

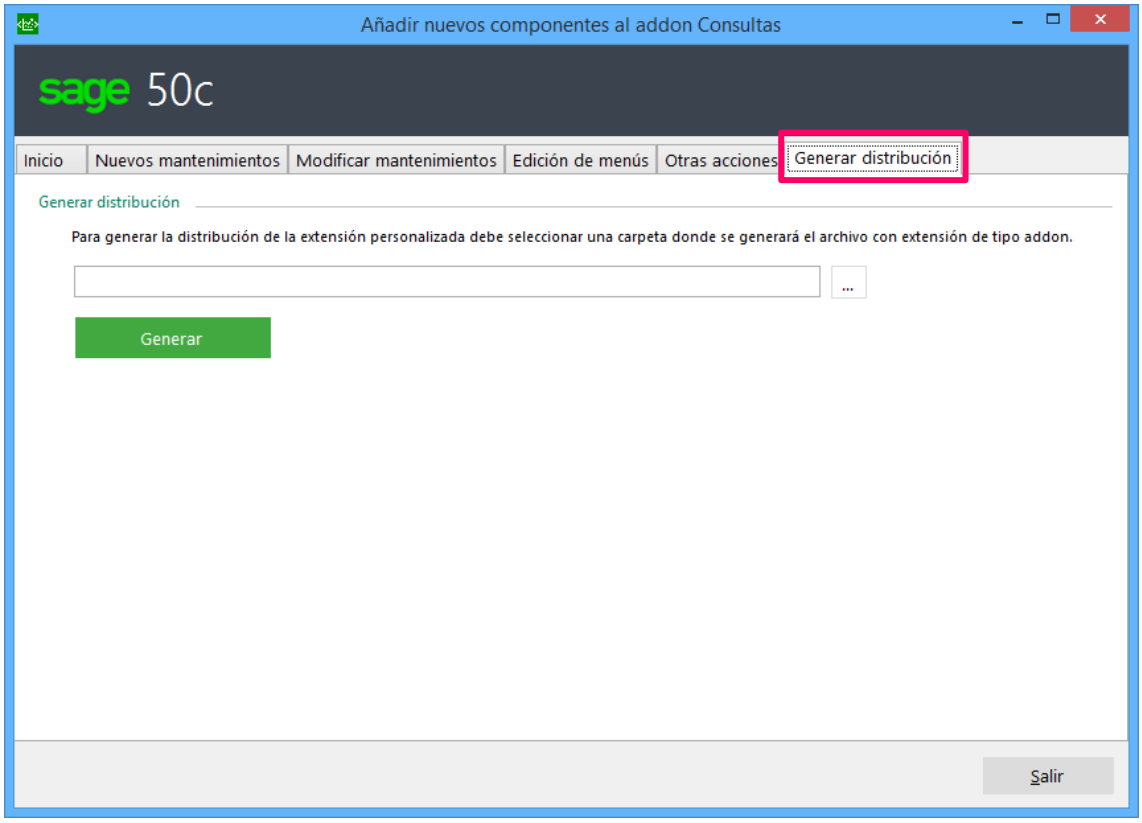
En este punto se recuerda que es posible consultar toda la ayuda técnica de Sage 50c en la siguiente web: http://developers.sage50comunidad.es/

3.7 Proceso de instalación de un addon de personalización a medida.

3.7.1 Generación del proceso de instalación

El proceso de instalación se genera des del propio plugin de personalización a medida en Visual Studio.

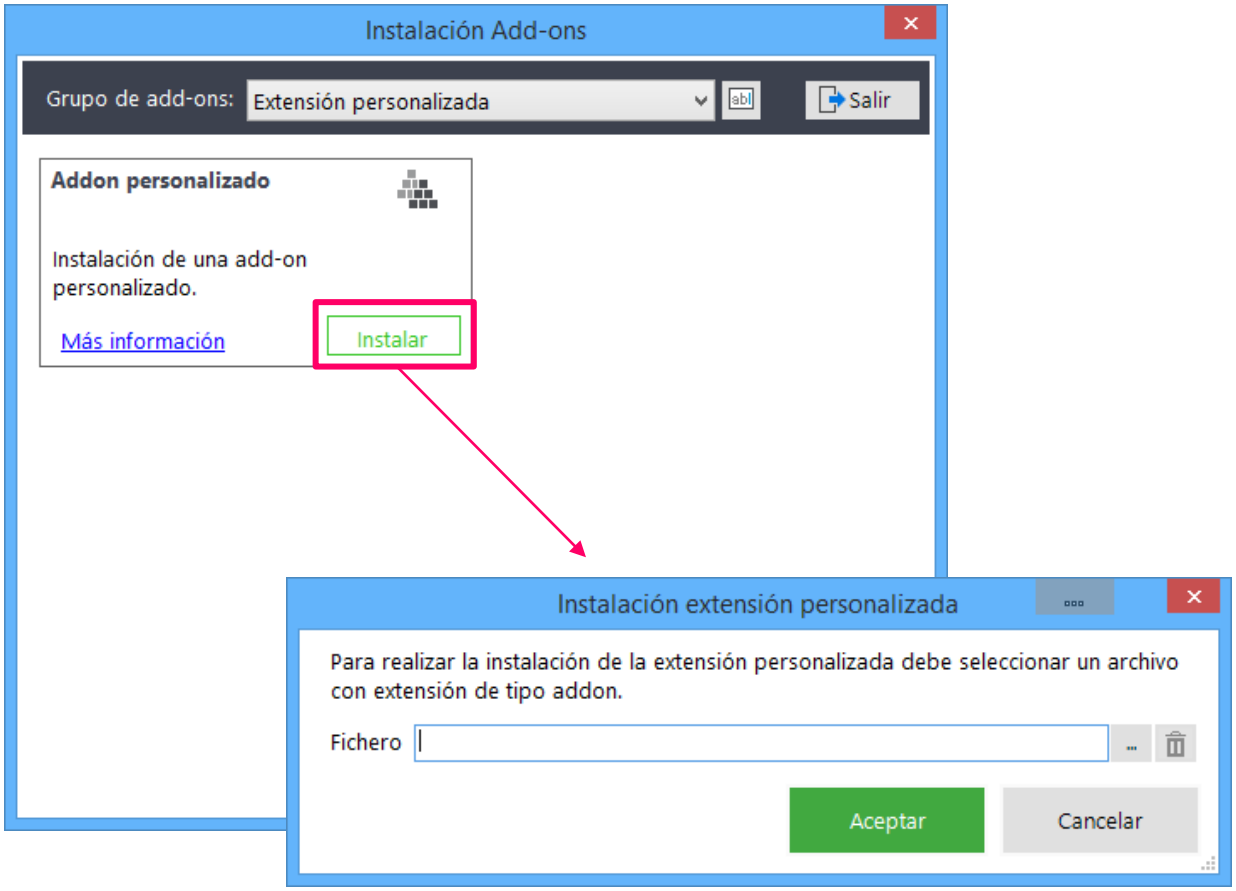
- Generamos el proceso de instalación desde dentro del plugin.
- Crea un fichero con extensión addon
- Este fichero contiene la definición del menú, las dlls y la base de datos del addon, en formato ZIP, con extensión . addon



3.7.2 Instalación del addon de personalización a medida

El addon generado se instala como cualquier otro addon de Sage 50c:

- Menú Usuario / Instalación de addons
- Grupo: extensión personalizada
- Al pulsar Instalar nos pide la ruta donde se encuentra nuestro fichero con extensión .addon
- Técnicamente: la librería se guarda junto a todas las librerías en la carpeta Sage50serv\Librerias.


Instalación Add-ons
Grupo de add-ons:
Extensión personalizada
Salir
Addon personalizado
Instalación de una add-on personalizado.
Más información
Instalar
Instalación extensión personalizada
Para realizar la instalación de la extensión personalizada debe seleccionar un archivo con extensión de tipo addon.
Fichero
Aceptar
Cancelar