Wolters Kluwer

a3ERP

Guía de los eventos de DLL en a3ERP

Sumario

Eventos de DLL en a3ERP

Introducción

Los eventos de ERP no son 'manejadores' de eventos convencionales. Pero su funcionalidad es parecida, por lo que se han asimilado a este concepto. Al igual que un evento, se disparan cuando ocurre un suceso determinado. Y provee información de contexto del 'objeto' o del proceso que los genera.

Suelen estar relacionados con documentos o ficheros maestros, sobre todo con el momento previo o posterior a guardar la información. Pero no necesariamente. Hay eventos que detectan la carga de un formulario, la impresión de un documento o el cálculo de comisiones.
Un evento convencional se activa escribiendo un procedimiento que lo implemente y notificando al objeto que está disponible para el evento concreto. Esto último suelo hacerlo nuestro IDE por nosotros, de una manera visual y más cómoda.

En a3ERP, la notificación del evento funciona de una forma diferente. A ERP se le informan las DLL que van a implementar los eventos, no necesariamente todos. Cuando en a3ERP llega el momento de llamar al evento, explora en cada una de ellas si lo han implementado (es decir, hay un método con el mismo nombre), y lo llama pasando como parámetros la información de contexto.

¿Qué utilidad tienen los eventos de DLL en a3ERP?

Sirven para 'extender' las funcionalidades de a3ERP en ciertas partes del programa.

Conceptos básicos

Tipos de datos

En los distintos eventos de extensión se usan tanto tipos de datos simples (escalares) como tipos más complejos como las matrices (o también llamadas vectores o array), que pueden ser, a su vez, de valores escalares como matriz. Sobre estos definimos unos tipos compuestos específicos de la aplicación que describiremos más adelante.

Nombre	Descripción / Finalidad
Cadena	Almacenar un valor alfanumérico (Nota: Se recomiendan cadenas ANSI, no unicode).
Real	Representa un valor numérico con coma flotante.
Entero	Representa un valor numérico entero
Booleano	Contiene una afirmación con valor Verdadero o Falso.
Variante	Representa un valor que puede estar almacenado con cualquiera de los tipos anteriores. Para conocer el tipo correcto es conveniente revisar la estructura usando la herramienta Diccionario.exe proporcionada con el ERP.
Matriz	Serie de elementos consecutivos e indexados de un tipo dado, cuyo índice, generalmente y salvo que se indique lo contrario, empieza en cero.

Equivalencia entre entornos y lenguajes

A continuación indicamos la **correspondencia de los tipos usados en los eventos con los lenguajes más usados para el desarrollo de eventos de** ERP.

Nombre	Delphi	.NET (C#)	VB6
Cadena	Dependiendo del soporte Unicode: **Anterior a Delphi 2009**: String **Desde Delphi 2009**: AnsiString	System.String (String)	String
Real	Double	System.Double (float)	Double
Entero	Integer	System.Int32 (int)	Integer
Booleano	Boolean	System.Boolean (bool)	Boolean
Variante	Variant	System.Object (Object) Contiene un valor de un tipo debidamente boxed como objeto.	Variant
Matriz	Variant Contiene un VarArray	System.Object (Object) En este caso, se trata de un object que contiene realmente un Object[], recuperable mediante conversión (cast).	Variant

Parámetros complejos

Los tipos compuestos específicos, construidos a partir de los tipos de datos indicados anteriormente, se articulan a partir de matrices cuyos elementos son de alguno de los tipos básicos anteriores.

Los definidos para los eventos son los siguientes:

<table>
<tr><th>Nombre</th><th>Descripción</th></tr>
<tr><td>Identificador</td><td>Matriz de "n" elementos (mínimo 1) que representan un valor único de llave primaria, siguiendo el orden que tengan indicados en la herramienta Diccionario.exe.
<table>
<tr><th>Índice</th><th>Tipo</th><th>Contenido</th></tr>
<tr><td>0</td><td>Variante</td><td>Valor del primer campo de la llave del elemento al cual refiere el identificador.</td></tr>
<tr><td>…</td><td></td><td></td></tr>
<tr><td>n</td><td>Variante</td><td>Valor del n-ésimo campo de la llave.</td></tr>
</table></td></tr>
<tr><td>Campo</td><td>Matriz de dos elementos que refieren al nombre de un campo de una entidad de ERP (maestro, documento…) y su valor:
<table>
<tr><th>Índice</th><th>Tipo</th><th>Contenido</th></tr>
<tr><td>0</td><td>Cadena</td><td>Nombre del campo</td></tr>
<tr><td>1</td><td>Variante</td><td>Valor del campo</td></tr>
</table></td></tr>
<tr><td>Registro</td><td>Matriz multidimensional en el que el primer elemento indica el número de campos contenido y los siguientes elementos son parejas de nombre campo y su valor (Campo).
<table>
<tr><th>Índice</th><th>Tipo</th><th>Contenido</th></tr>
<tr><td>0</td><td>Entero</td><td>Número de Campos</td></tr>
<tr><td>1</td><td>Campo</td><td>Primera Pareja campo/valor</td></tr>
<tr><td>…</td><td></td><td></td></tr>
<tr><td>n</td><td>Campo</td><td>N-ésima pareja campo/valor</td></tr>
</table></td></tr>
</table>

Conjunto de Datos	Matriz multidimensional en la que el primer elemento indica el número de Registros (Registro) que lo componen.

Índice	Tipo	Contenido
0	Entero	Número de registros
1	Registro	Primer registro
…		
n	Registro	Registro nésimo

Ejemplos

DELPHI

Para acceder a una información concreta almacenada en uno de los parámetros complejos basta con acceder siguiendo la nomenclatura de acceso a elementos de matrices del pascal según la dimensión en la que se encuentre.

Por ejemplo, si queremos acceder al valor del segundo campo del tercer registro de un conjunto de datos, escribiremos lo siguiente:

```
Valor := ConjuntoDeDatos[3][2][1];
```

Si lo que queremos es el nombre del campo:

```
Campo := ConjuntoDeDatos[3][2][0];
```

C#

En este caso, la matriz que conforma el tipo conjunto de datos se encuentre envuelta (boxed) como un objeto, por lo que previamente deberemos convertirla a una matriz de objetos:

```
ConjuntoDatosObj := ConjuntoDeDatos as Object[];
```

Como, a su vez, cada uno de sus elementos es un Object, para cada elemento contenido habrá de hacerse la misma operación.

```
Registro := (ConjuntoDeDatos as Object[])[3] as Object[];
Campo := Registro[2] as Object[];
Valor := Campo[1];
```

Estructura básica DLL

DELPHI

En este entorno podemos elegir **crear una biblioteca normal (dll)** o una **COM**. Si optamos por la primera, bastará con crear una DLL (Dynamic-link library) de 32 bits (target platform 32bits) que registraremos por nombre (fichero dll) en ERP. Cada uno de los eventos se recibirá en una función que deberemos exportar con su nombre en mayúsculas. El paso de parámetros se realizará siguiendo STDCALL.

Por ejemplo, el método iniciar:

```
library MyLibrary;
uses
        System.Sysutils;
exports
        INICIAR;
begin
end.
```

En el caso de optar por una dll COM, se deberá crear una biblioteca ActiveX (ActiveX library), compilada en 32bits (target platform 32bits), dentro de la cual se deberá crear una clase ActiveX que deberá tener los métodos que se indiquen para el caso de C#, con las mismas especificaciones pero con los tipos correspondientes a Delphi.

C#

En el mundo .NET se puede optar por crear DLLS nativas (usando Visual C++ o pluggins) o accesibles desde **COM**. El primer caso sería el equivalente a lo indicado para Delphi en el apartado anterior. En cualquier caso, la biblioteca de clases que se cree deberá tener como destino de plataforma x86 (es decir 32 bits) y no AnyCPU o x64. Esto asegurará que ERP tendrá accesible la biblioteca independientemente de la arquitectura (x86, AMD64) de la máquina en la que se ejecute.

En C# lo más cómodo es optar por el segundo caso, una biblioteca de clases (DLL) accesible desde COM. Albergará una clase que ERP instanciará para realizar las llamadas correspondientes a los eventos que dispare. Para ello, deberemos marcar la solución como "**COM Visible (Propiedades de aplicación, Información del ensamblado)**" que expondrá al COM todas las clases que definamos, o indicar el atributo [System.Runtime.InteropServices.ComVisible] en las clases que se quieran hacer accesibles desde COM. Esta última opción sería la preferible al limitar las clases a las que se podrá acceder desde fuera, por el **ERP** o cualquier otro cliente. Si se escoge esta opción, como mínimo deberá estar así marcada la clase que instanciará ERP para comunicar los eventos. Esta será la que registremos en ERP para habilitarla (su nombre de clase para COM, que generalmente es el nombre completo de clase en el caso de C#, o nombre de la DLL sin extensión u nombre estricto de clase en VB.NET).

Los métodos que recibirán los distintos eventos de **ERP** se deberán marcar como públicos y, adicionalmente, se requiere un método que devuelva la lista de eventos que se quiera escuchar. Esto permite habilitar o desconectar eventos dinámicamente. Este método se llama previamente (al menos una vez) a la generación de cualquier evento para ver si está disponible respuesta desde nuestra programación.

Ejemplo:
Queremos escuchar los eventos Inicia, SePuedeFinalizar y Finalizar.

```
public object[] ListaProcedimientos()
{
    // Nota: Los valores devueltos no son sensibles a mayúsculas y minúsculas
      return new object[] { "Iniciar", "SePuedeFinalizar", "FINALIZAR" };
}
```

Obviamente, la clase deberá tener implementados los métodos con los nombres devueltos por la función.

Registro en ERP

Para que a3ERP sepa que módulos debe cargar, se deberá registrar sus binarios en la tabla DLLS. Esto se puede hacer lanzando una sentencia SQL desde la opción "**Vistas SQL**", desde el "**Management Studio**" de SQL Server o una herramienta parecida, o usando una actualización de un diccionario que forme parte del módulo.

```
if not exists(select IDDLL from DLLS where DLL = 'WK.ERP.Ejemplos.LoggerEventosERP')
begin
  declare @id int
  select @id = max(iddll) + 1 from dlls
  insert into dlls(iddll, dll, fabricante, producto, ORDENDEEJECUCION)
  values(@id, 'WK.ERP.Ejemplos.LoggerEventosERP', 'WK', 'a3ERP', 1)
end
```

En el caso de que se opte por la opción del diccionario, debemos recordar que las actualizaciones deben estar diseñadas para que se ejecuten una única vez, como ocurre en el ejemplo anterior. Si no fuese así, podríamos encontrarnos con el registro repetido en la tabla.

Los valores que habrá de indicarse en la tabla DLLS son los siguientes, según sea una dll nativa o COM (hecha en .NET o VB6, por ejemplo):

NATIVA

Campo	Uso
IDDLL	Identificador único del registro
DLL	Nombre de la DLL que el ERP deberá cargar
FABRICANTE	Descripción del fabricante del módulo
PRODUCTO	Descripción del módulo
ORDENEJECUCION	Valor no usado

A tener en cuenta…

En el caso de una **DLL nativa**, el nombre del fabricante y del producto debe corresponder con los directorios dentro de la carpeta extensiones de a3ERP en el que se ubica.

Por ejemplo, si el fabricante fuese WKE y el producto EJEMPLOSERP, el módulo se deberá encontrar en <directorioERP>\Extensiones\WKE\EJEMPLOSERP\Binarios

Recordemos que la estructura para un módulo de extensión de **ERP** es:

<directorioDelMódulo>

\Binarios: Directorio de los binarios del módulo
\Diccionarios: Directorio que albergará los diccionarios del módulo
\Menús: Ubicación del menú del módulo

Por supuesto, estos directorios son opcionales (pero al menos uno deberá existir).

COM

Campo	Uso
IDDLL	Identificador único del registro
DLL	Nombre de la clase COM que el ERP deberá instanciar
FABRICANTE	Descripción libre del fabricante del módulo
PRODUCTO	Descripción libre del módulo
ORDENEJECUCION	Valor no usado

Las liberías COM deberán estar registradas. Si es nativa con **regsvr32** y si es **.NET** con regasm usando el modificador /CODEBASE.

A tener en cuenta…

A diferencia del caso nativo, los **campos FABRICANTE** y **PRODUCTO** son libres (no tienen que coincidir con la ruta en la que se ubica, pero es recomendable seguir el mismo patrón).

En el caso del valor de DLL, clase COM a instanciar, en el caso de .NET coincide con el nombre completo de la clase (es decir, con namespaces). Generalmente el namespace principal de la biblioteca de clases coincide con el nombre de la DLL, con lo que suele quedar como "**nombredllSinExtension.nombreDeLaClase**". Pero no se debe olvidar que, si se cambia el namespace principal, cambiará el nombre de la clase a registrar.

Eventos

Los distintos eventos que produce a3ERP y que un módulo externo puede escuchar se han categorizado según sean eventos que notifiquen cambios en la empresa activa (Ciclo de Vida), de entidades (Maestros, Documentos) y de configuración.

Hay eventos de los que se puede encontrar distintas versiones. Esto se debe a la evolución natural del producto, recomendándose usar siempre que se pueda, la versión más moderna. Para distinguir la versión antigua de la moderna, esta última vendrá sufijada por V2 (u otro número superior en el futuro).

Recomendaciones generales

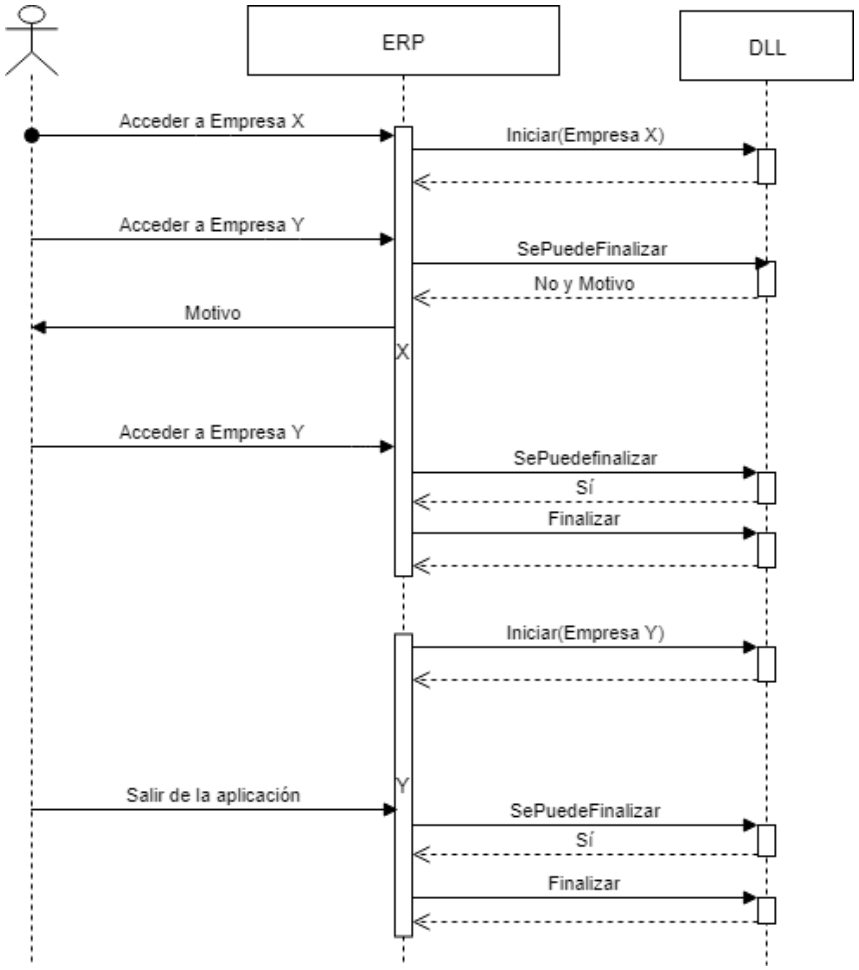
La implementación de los eventos permite, en muchos casos, acceder a los datos de **ERP**, alterarlos e incluso cambiar el comportamiento de la aplicación. Se recomienda un cuidado extremo a la hora de plantear y ejecutar tales acciones. Para dar una guía que ayude en tal tarea, recopilamos aquí algunas recomendaciones a la hora de implementar eventos de **ERP**:

- **No mostrar pantallas visuales**, salvo en los eventos en los que se indique o cuya funcionalidad, a todas luces, así lo requiera (por ejemplo: Llamadas a pantallas externas).
- **No confiar en que los eventos que implementemos se disparen en grupo para una única entidad** (ejemplo un documento concreto). Al permitir la aplicación trabajar con varios a la vez, pueden dispararse eventos para varias entidades. Ejemplo: Se podría disparar un AntesDeGuardar maestro X y a continuación otro para un maestro Y. Es por esto que hay que vigilar muy bien que información se compartirá entre eventos. Si hiciera falta, es recomendable etiquetar la información a compartir por entidad, por ejemplo.

Ciclo de Vida

Los eventos de ciclo de vida informan a la extensión de ERP del momento en éste la habilita o deshabilita para una determinada empresa. Esto ocurrirá cada vez que un usuario entre o salga de una empresa durante la ejecución del programa.

El flujo de eventos, durante la ejecución de la extensión es el siguiente:



Hay unas consideraciones a tener en cuenta, dependiendo de si la dll usa la implementación COM o no. Si se usa COM, dispondremos de dos métodos Iniciar distintos **"Iniciar"** e "**IniciarGeneral**". En otro caso dispondremos de tres: "**Iniciar**", "**IniciarConSistema**" e "**IniciarGeneral**". De implementarse todos ellos, cuando se produzca la entrada en una empresa de ERP, se llamarán de la siguiente manera:

DLL normal	COM
IniciarConSistema / Iniciar	Iniciar
IniciarGeneral	IniciarGeneral

Con una DLL normal, la existencia de IniciarConSistema anula Iniciar. Es decir, si se implementan "**Iniciar**" e "**IniciarConSistema**", sólo se llamará a IniciarConSistema.

El método de respuesta Iniciar del COM se corresponde con el IniciarConSistema de una DLL normal, ya que cuando se implementó las llamadas a COM se consideró que esta era más útil que la Iniciar antigua.

Iniciar

Este evento se ejecuta cuando se accede a una determinada empresa del ERP siempre y cuando, en el caso de una dll nativa, no se haya implementado tambien IniciarConSistema.

La signatura de este evento es distinta según se implemente en una dll nativa o mediante COM.
En el primer caso, es la siguiente:

Delphi procedure INICIAR(ConexionEmpresa: string); stdcall;

En el segundo, los parámetros corresponden a la de IniciarConSistema.

C# public void Iniciar(string conexionSistema, string conexionEmpresa)

Donde en cada uno de los parámetros se reciben la cadena de **conexión ADO** a la base de datos de sistema (conexionSistema) y la de la empresa a la que se conecta (conexionEmpresa). Entre otros, contienen el nombre del usuario y el nombre físico de la base de datos a la que se está accediendo. En el caso de una dll nativa, la cadena de conexión a la base de datos de sistema no se provee.

A tener en cuenta…
Estas cadenas no contienen la parte relativa a la contraseña.

Antiguamente se usaban estas cadenas para establecer una conexión con las bases de datos de a3ERP, usando componentes de acceso a datos compatibles con ADO. Hoy día se puede usar IniciarGeneral para obtener la conexión que usa el **ERP** y así evitar tener que autentificarse con otras credenciales y, por lo tanto, consumir una licencia adicional.

IniciarConSistema

Este evento se ejecuta cuando se accede a una determinada empresa del **ERP** ocultando, en el caso de una dll nativa, el evento Iniciar. Este evento sólo está disponible para dlls nativas (recordemos que en COM se llama Iniciar).

Delphi: procedure INICIARCONSISTEMA(ConexionSistema, ConexionEmpresa: string); stdcall;

Donde en cada uno de los parámetros se reciben la cadena de conexión ADO a la base de datos de sistema (conexionSistema) y la de la empresa a la que se conecta (conexionEmpresa). Entre otros, contienen el nombre del usuario y el nombre físico de la base de datos a la que se está accediendo.

A tener en cuenta…
Estas cadenas no contienen la parte relativa a la contraseña.

Antiguamente se usaban estas cadenas para establecer una conexión con las bases de datos de ERP, usando componentes de acceso a **datos compatibles con ADO**. Hoy día se puede usar IniciarGeneral para obtener la conexión que usa el **ERP** y así evitar tener que autentificarse con otras credenciales y, por lo tanto, consumir una licencia adicional.

IniciarGeneral

Al igual que Iniciar/IniciarConSistema, este evento se produce al entrar en una determinada empresa. Pero al contrario de lo que ocurre en el caso de IniciarConSistema e Iniciar, en el que implementar el primero oculta a segundo, este evento no oculta a ningún otro. Es decir, se puede implementar IniciarConSistema (por ejemplo) e iniciarGeneral, disparándose ambos.

La diferencia más significativa, con el resto de eventos de ciclo de vida, se encuentra en los parámetros. Añadiendo las cadenas de conexión ya vistas se incluye, además, las **interfaces COM** de los objetos conexión usados por ERP para acceder a las bases de datos de sistema y de empresa. Usando estas interfaces, la programación ejecutará sus sentencias dentro de las transacciones del **ERP**.

A tener en cuenta…
Una extensión JAMÁS deberá usar estas interfaces y cerrar las conexiones.

Delphi: procedure INICIARGENERAL(Parametros: Variant); stdcall;
C#: void IniciarGeneral(object parametros);

El contenido de la matriz es el siguiente (corresponde a lo que denominamos ConjuntoDeDatos):

Posición 0: Valor 1 (Indicando que le sigue un valor)
Posición 1: Matriz de 9 Valores:
- Posición 1.0: Valor 9 (Indicando que le siguen nueve valores)
- Posición 1.1: Cadena de conexión a base de datos de sistema.
- Posición 1.2: Cadena de conexión a base de datos de empresa.
- Posición 1.3: Usuario logado
- Posición 1.4: Usuario trabajo ERP (usuario a3ERP).
- Posición 1.5: Reservado.
- Posición 1.6: Reservado.
- Posición 1.7: Interfaz ADO Conection a base de datos de sistema.
- Posición 1.8: Interfaz ADO Conection a base de datos de empresa logado con usuario de trabajo ERP.
- Posición 1.9: Interfaz ADO Conection a base de datos de empresa logado con el usuario actual.

En delphi se puede acceder a cada uno de los valores indicando accediendo de manera indexada a la variable variant recibida, mientras que en C# se deberá convertir a object[] para acceder de manera indexada al segundo elemento para convertirlo nuevamente a object[] y así poder acceder a cada uno de los nueve valores contenidos en él.

A tener en cuenta…
Cada uno de los valores es, a su vez, otra matriz de dos elementos. El 0, con el nombre o ID de lo contenido y el 1 con el valor propiamente dicho.

Ejemplo de acceso al usuario logado:

Delphi: Parametros[1][3][1]
C#: ((object[]) (((object[]) ((object[])parametros)[1])[3]))[1]

SePuedeFinalizar

Este evento permite a una extensión de ERP evitar que se pueda salir de una empresa, por ejemplo porque se encuentre realizando una operación costosa sobre la misma o mostrando alguna pantalla.

Delphi: function SEPUEDEFINALIZAR(out Motivo: string): Boolean; stdcall;
C#: function bool SePuedeFinalizar(out string Motivo);

El permiso para desconectar de la empresa se otorga mediante el resultado de la función. True lo permite False lo impide. En este último caso, se debe proporcionar una cadena que explique al usuario porque no se le permite desconectar de la empresa.

Finalizar

Este evento se produce cuando **ERP** desconecta, ya sea por cierre de la aplicación o por cambio de empresa, de una determinada empresa. Es un buen punto para liberar los recursos obtenidos en cualquiera de los iniciar.

Delphi: procedure FINALIZAR; stdcall;
C#: public void Finalizar();

Entidades

Maestros

Los distintos maestros (Clientes, Proveedores, Artículos, Cuentas...) de **ERP** notifican mediante eventos cuándo se realiza modificaciones o borrados de los mismos, como notificación de dichas operaciones pudiendo impedirlas en algunos casos (AntesDeXXX). Los eventos producidos (ya sea por el usuario o por otros procesos que trabajen con ellos) son los siguientes:

AntesDeGuardarMaestro

Este evento notifica la intención de guardar los cambios de un maestro, con la posibilidad de impedirlo.

Dephi: function ANTESDEGUARDARMAESTRO(Tabla: AnsiString; var Datos: Variant): Boolean; stdcall;
C#: bool AntesDeGuardarMaestro(string tabla, ref object datos)

Nos informa del maestro que se está modificando usando el nombre de tabla de base de datos en la que se guardará, y con los datos actuales en el programa (en formato Dataset ya comentado anteriormente disponible en el parámetro Datos).

En este momento, se pueden cambiar valores de los campos que se almacenarán en base de datos, accediendo al Dataset que se recibe en la variable Datos.

Para permitir la modificación se deberá devolver el valor booleano True o False en caso contrario.

AntesDeGuardarMaestroV2

Este evento tiene la misma misión y funcionamiento (modificación incluida) que el anterior (AntesDeGuardarMaestro) con la salvedad de que además informa del tipo de operación que propicia el guardado.

Delphi: function ANTESDEGUARDARMAESTROV2(Tabla: AnsiString; var Datos: Variant; Estado: Integer): Boolean; stdcall;
C#: bool AntesDeGuardarMaestroV2(string tabla, ref object datos, int estado)

El nuevo parámetro, Estado, podrá tener los siguientes valores:

0: Alta
1: Edición
2: Borrado
3: Duplicación

A tener en cuenta…
Este evento no cancela el anterior. Es decir, se disparan los dos eventos si se encuentran implementados, aunque es preferible esta versión.

DespuesDeGuardarMaestro

Informa de la finalización del proceso de guardado de un maestro. El maestro se encontrará modificado, borrado o duplicado en la tabla correspondiente.

Delphi: procedure DESPUESDEGUARDARMAESTRO(Tabla: AnsiString; Datos: Variant); stdcall;
C#: void DespuesDeGuardarMaestro(string tabla, object datos)

Los parámetros son idénticos a los de AntesDeGuardarMaestro.

DespuesDeGuardarMaestroV2

Este evento es análogo al anterior (DespuesDeGuardarMaestro) pero informa del tipo de operación que propició el guardado mediante los valores de la propiedad Estado (nueva propiedad respecto al método antiguo), que están descritos en AntesDeGuardarMaestroV2, **con la salvedad de que los datos son un Row y no un Dataset**.

Delphi: procedure DESPUESDEGUARDARMAESTROV2(Tabla: AnsiString; Datos: Variant; Estado: Integer); stdcall;
C#: void DespuesDeGuardarMaestroV2(string tabla, object datos, int estado)

A tener en cuenta…
Este evento no cancela el anterior. Es decir, se disparan los dos eventos si se encuentran implementados, aunque es preferible esta versión.

AntesDeBorrarMaestro

Notifica la intención de borrar un determinado maestro. Este borrado puede impedirse.

Delphi: function ANTESDEBORRARMAESTRO(Tabla: AnsiString; IdMaestro: Variant): boolean; stdcall;
C#: bool AntesDeBorrarMaestro(string tabla, object idMaestro)

El maestro concreto vendrá indicado por el campo **IdMaestro** y la clase del mismo por el campo Tabla (que hace referencia a la tabla de b.d. en la que se almacena). El campo IdMaestro contiene un array de valores correspondientes (y con el mismo orden) a los campos de la llave primaria de la tabla en el que se almacena el maestro. Es decir, si la llave primaria se compusiera de tres campos, habrá tres elementos en dicho array.

Por ejemplo, si en el maestro en cuestión es ARTICULO (cuya la llave primaría se compone únicamente de CODART), recuperaríamos el identificador del artículo siendo borrado así:

Delphi: CodArt := IdMaestro[0];
C#: var codArt = ((object[])idMaestro)[0];

La función deberá devolver True si se permite el borrado, False en caso contrario.

DespuesDeBorrarMaestro

Notifica la desaparición de un determinado maestro. Deja de existir en B.D.

Delphi: procedure DESPUESDEBORRARMAESTRO(Tabla: AnsiString; IdMaestro: Variant); stdcall;
C#: void DespuesDeBorrarMaestro(string tabla, object idMaestro)

Los parámetros son análogos a los de AntesDeBorrarMaestro.

Documentos

Las entidades en las que se plasma el trabajo habitual en la aplicación, tales como ofertas, pedidos, albaranes, depósitos, facturas, cuotas, expedientes, etc. son capaces de notificar a un módulo de terceros de las operaciones CRUD (alta, modificación, baja) que se realizan sobre ellos. Para ello pone a nuestra disposición eventos que se dispararán antes y después de la carga de un documento, cambio de un campo, guardado de línea o guardado del documento mismo.

Ciclo de vida

Para avisar del ciclo de vida de un documento solo disponemos, por el momento, de los eventos que nos notifican de la carga de uno dado. Pero pueden combinarse con eventos de modificaciones para que estos últimos tengan más información (como que cambios se han producido, por ejemplo, si guardamos todos los valores, o parte de ellos, que tenía el documento).

DespuesDeCargarDocumento

Este evento se dispara tras haberse cargado en memoria un documento. Nos indica el tipo de dicho documento, sus datos (cabecera y líneas) así como si es nuevo o no.

Delphi: procedure DespuesDeCargarDocumento(const TipoDocumento: AnsiString; const IdDoc: **Double**; var Cabecera, Lineas: Variant; const Estado: Integer); stdcall;
C#: void DespuesDeCargarDocumento(string tipoDoc, float IdDoc, object cabecera, object lineas, int estado)

El tipo de documento valdrá uno de los siguientes valores según corresponda:

OC: Oferta de compra
OV: Oferta de venta
PC: Pedido de compra
PV: Pedido de venta
AC: Albarán de compra
AV: Albarán de venta
CUOTA: Cuota
TQ: Tique de TPV
DC: Depósito de compra
DV: Depósito de venta
FC: Factura de compra
FV: Factura de venta
RE: Regularización
TR: Traspaso
EXPEDIENTE: Expediente

En el parámetro cabecera se enviará el conjunto de datos guardado en la tabla cabecera del documento (ejemplo con factura de ventas: CABEFACV). En el parámetro línea, el conjunto de datos con las líneas del documento (siguiendo el mismo ejemplo, el contenido para el documento de la tabla LINEFACT).

Estado nos indicará si es nuevo con el valor 0, o uno existente con el valor 1.

DespuesDeCargarDocumentoV2

Esta versión del evento **DespuesDeCargarDocumento** permite, además, cambiar valores del documento situados en la cabecera tras la carga.

> **Delphi:** function DespuesDeCargarDocumentoV2(const TipoDocumento: AnsiString; const IdDoc: Double; var Cabecera, Lineas: Variant; const Estado: Integer): Variant; stdcall;
> **C#:** object DespuesDeCargarDocumentoV2(string tipoDoc, float IdDoc, object cabecera, object lineas, int estado)

Para ello, se deberá devolver un registro (una matriz que siga su definición) con el campo o campos que se quiere modificar, junto con sus valores.

Ejemplo Delphi:
return VarArrayOf(2, VarArrayOf([“CODCLI”, ” 1”]), VarArrayOf([“REFERENCIA”, ” Test”]));

Ejemplo C#:
return new [] {2, new [] {“CODCLI”,” 1”}, new []{[“REFERENCIA”, ” Test” }};
O, en caso de no requerir cambio, se puede devolver una matriz con valor 0 en la primera posición, o Null en C#.

Ejemplo: return new [] {0} // return null;

Modificaciones

En siguiente conjunto de eventos nos permite conocer los cambios producidos en documentos y, en algunos casos, realizar modificaciones adicionales.

Campos

Se notifican los cambios a nivel de campo, ya sea en cabecera o en alguna línea, mediante los eventos descritos a continuación.

CamposAEscucharEnDocumento

El primer evento lo dispara **ERP** para conocer, cuando se procede a trabajar con un documento, de que campos queremos ser notificados en caso de que se produzca un cambio en sus valores. Implementar una respuesta para este evento es fundamental para escuchar los cambios en campos y, por lo tanto, para que se disparen el siguiente conjunto de eventos.

Delphi: function CAMPOSAESCUCHARENDOCUMENTO(ClaseDocumento: AnsiString; IdDoc: Double): OleVariant; stdcall;
C#: object CamposAEscucharEnDocumento(string claseDocumento, float idDoc)

En **ClaseDocumento** recibiremos el tipo de documento:

OC: Oferta de compra
OV: Oferta de venta
PC: Pedido de compra
PV: Pedido de venta
AC: Albarán de compra
AV: Albarán de venta
CUOTA: Cuota
TQ: Tique de TPV
DC: Depósito de compra
DV: Depósito de venta
FC: Factura de compra
FV: Factura de venta
RE: Regularización
TR: Traspaso
IN: Inventario

Y en IdDoc, el valor de su clave primaria (Ejemplo, el valor de IDFACV en el caso de una factura de venta [CabeFacv]). Con esto podemos determinar si para ese documento en concreto (o cualquiera de esa clase) queremos que se nos notifique el cambio en algún campo.

Para indicar a **ERP** que nos notifique de los cambios de algún campo debemos devolver un Registro con los nombres de los campos siguiendo el siguiente patrón:

Nombre de campo -> CABECERA.NombreCampoFisicoCabecera o LINEA.NombreCampoFisicoLinea

Así, si queremos escuchar dos campos (por ejemplo: Referencia y Artículo, el primero de la cabecera y el segundo de las líneas) devolveríamos un array con los valores {2, CABECERA.REFERENCIA, LINEA.CODART}.

Ejemplo Delphi: result := VarArrayOf([2, 'CABECERA.REFERENCIA', 'LINEA.CODART'])
Ejemplo C#: return new object[] {2, 'CABECERA.REFERENCIA', 'LINEA.CODART'};

También se puede devolver un registro vacío o nulo, para indicar que no se quiere escuchar ningún campo.

Ejemplo:
Result := unnasigned; // o VarArrayOf([0]);
return new object[] { 0 };

AntesDeCambioDeCampoEnDocumento

Este evento nos notifica del cambio de valor de un campo antes de que se aplique a los datos en memoria de **ERP**, permitiéndonos alterarlo. Nota: Este evento es anulado por la implementación del siguiente. Es decir, si se implementa este y la nueva versión solo se disparará este último.

Delphi: procedure ANTESDECAMBIODECAMPOENDOCUMENTO(ClaseDocumento: AnsiString; IdDoc: Double; Campo: string; ValorAnterior: OleVariant; var NuevoValor: OleVariant); stdcall;
C#: void AntesDeCambioDeCampoEnDocumentoProc(string claseDocumento, float idDoc, string campo, object valorAnterior, ref object nuevoValor)

En **claseDocumento** e **idDoc** recibiremos el tipo e ID concreto del documento, como en el evento anterior. El parámetro Campo indicará el campo en el que se produce el cambio (precedido de CABECERA. o LINEA.) mientras que en los parámetros ValorAnterior y ValorNuevo recibiremos el valor que tenía el campo antes de la modificación y el valor modificado del campo, respectivamente. Este último podemos modificarlo con cualquier valor que permita el campo, como asignarle de nuevo el ValorAnterior, por ejemplo.

AntesDeCambioDeCampoEnDocumentoV2

Se trata de una versión extendida de la anterior que permite cambiar adicionalmente otros campos de la tabla (cabecera o línea) en la que se ha producido el cambio. Nota: Este evento anula el anterior. Si se implementan los dos, sólo se disparará este.

Delphi: function ANTESDECAMBIODECAMPOENDOCUMENTOV2(ClaseDocumento: AnsiString; IdDoc: Double; Campo: string; const ValorAnterior, NuevoValor: OleVariant): OleVariant; stdcall;
C#: object AntesDeCambioDeCampoEnDocumentoV2(string claseDocumento, float idDoc, string campo, object valorAnterior, object nuevoValor)

Los parámetros tendrán el mismo valor y significado que en el evento anterior con la diferencia de que en este caso hay valor de retorno con el que podemos devolver un registro con los campos de la misma tabla (no hay que prefijarlos con CABECERA o LINEA) a modificar (es decir, si el campo modificado es de la cabecera, los campos a devolver serán de la cabecera) y valores que queremos que tengan.

Ejemplo Delphi: result := VarArrayOf([2, VarArrayOf(REFERENCIA', 'NuevaRef'), VarArrayOf(['CODCLI', ' 1']));

Ejemplo C#: return new [] {2, new [] {"REFERENCIA", "NuevaRef"}, new [] {"CODCLI", " 1"}};

DespuesDeCambioDeCampoEnDocumento

Finalmente, este evento nos notifica de que el cambio producido se ha aplicado a los datos en memoria del **ERP**. Nos puede servir para trazar el cambio o realizar operaciones con datos propios.

Delphi: procedure DESPUESDECAMBIODECAMPOENDOCUMENTO(ClaseDocumento: AnsiString; IdDoc: Double; Campo: string; ValorAnterior, NuevoValor: OleVariant); stdcall;
C#: void DespuesDeCambioDeCampoEnDocumento(string claseDocumento, float idDoc, string campo, object valorAnterior, object nuevoValor);

Líneas

Este conjunto de eventos se disparará durante la edición de un documento al guardar cambios en memoria de alguna de las líneas de un documento y, por lo tanto, antes de los eventos de guardado del documento en base de datos.

AntesDeGuardarLinea

Evento que notifica el guardado en memoria de las modificaciones de una línea de un documento dado. Permite modificar valores de campos de dicha línea y cancelar el guardado.
La implementación de este evento anula AntesDeGuardarLineaConDetalle.

Delphi: function ANTESDEGUARDARLINEA(const TipoDocumento: AnsiString; Cabecera: Variant; Linea: Variant): variant;
C#: object AntesDeGuardarLinea(string tipoDocumento, object cabecera, object linea)

TipoDocumento contiene una cadena que describe el tipo de documento al que pertenece la línea:

OC: Oferta de compra
OV: Oferta de venta
PC: Pedido de compra
PV: Pedido de venta
AC: Albarán de compra
AV: Albarán de venta
CUOTA: Cuota
TQ: Tique de TPV
DC: Depósito de compra
DV: Depósito de venta
FC: Factura de compra
FV: Factura de venta
RE: Regularización
TR: Traspaso
EXPEDIENTE: Expediente

En Cabecera el registro de cabecera correspondiente al documento de la línea que se procede a guardar.
En Linea el registro de la línea que se está guardando.

Como retorno se puede devolver un registro con cambios para campos de la línea. Si devuelve una matriz con un único valor 0, se entenderá que no hay cambios adicionales para la línea. Si el valor es una cadena "F", se entenderá que se quiere cancelar el guardado.

Ejemplos:
Delphi: result := VarArrayOf([0]); // VarArrayOf(['F']);
C#: return new object[] { 0 }; // new object[] { "F" };

AntesDeGuardarLineaV2

Este evento se disparará aunque esté implementado el anterior, añadiendo información del estado de la línea y permitiendo, explícitamente, cancelar el guardado de la línea.

La implementación de este evento anula AntesDeGuardarLineaConDetalleV2.

Solo se encuentra disponible para los siguientes tipos de documento:

OC: Oferta de compra	**DC**: Depósito de compra
OV: Oferta de venta	**DV**: Depósito de venta
PC: Pedido de compra	**FC**: Factura de compra
PV: Pedido de venta	**FV**: Factura de venta
AC: Albarán de compra	**RE**: Regularización
AV: Albarán de venta	**EXPEDIENTE**: Expedientes

Delphi: function ANTESDEGUARDARLINEAV2(const TipoDocumento: AnsiString; Cabecera: Variant; Linea: Variant; const Estado: Integer; var PermitirGuardar: Boolean): variant;
C#: object AntesDeGuardarLineaV2(string tipoDocumento, object cabecera, object linea, int estado, ref bool permitirGuardar)

Aparte de los parámetros ya conocidos de la versión AntesDeGuardarLinea, el parámetro estado indicará:

0 para nueva línea
1 para línea existente
2 para línea que se está borrando

El parámetro PermitirGuardar deja guardar o no según el valor booleano que se se le asigne (True = sí se guardará, False = se impedirá).

AntesDeGuardarLineaConDetalle

Versión de AntesDeGuardarLinea que incorpora un conjunto de datos con el detalle de la línea que se está guardando. Útil para acceder a la información desglosada de lotes, números de serie y fechas de caducidad. Este evento no es compatible con AntesDeGuardarLinea, es decir, de implementarse los dos solo se llamará a la versión original AntesDeGuardarLinea.

Solo se encuentra disponible para los siguientes tipos de documento:

OC: Oferta de compra	**DC**: Depósito de compra
OV: Oferta de venta	**DV**: Depósito de venta
PC: Pedido de compra	**FC:** Factura de compra
PV: Pedido de venta	**FV**: Factura de venta
AC: Albarán de compra	**RE**: Regularización
AV: Albarán de venta	**TR**: Traspaso

Delphi: function ANTESDEGUARDARLINEACONDETALLE(const Documento: AnsiString; Cabecera: Variant; Linea: Variant; Detalle: variant): variant;
C#: object AntesDeGuardarLineaConDetalle(string tipoDocumento, object cabecera, object linea, object detalle)

AntesDeGuardarLineaConDetalleV2

Versión de AntesDeGuardarConDetalle que, adicionalmente al conjunto de datos con el detalle de la línea que se está guardando, incorpora información sobre el estado de la línea y permite explícitamente cancelar el guardado (como ocurre con el evento AntesDeGuardarLineaV2).

Como en el caso anterior, este evento no es compatible con AntesDeGuardarConDetalle, es decir, de implementarse los dos solo se llamará a esa versión (AntesDeGuardarConDetalle)

Solo se encuentra disponible para los siguientes tipos de documento:

OC: Oferta de compra	**DC**: Depósito de compra
OV: Oferta de venta	**DV**: Depósito de venta
PC: Pedido de compra	**FC**: Factura de compra
PV: Pedido de venta	**FV**: Factura de venta
AC: Albarán de compra	**RE**: Regularización
AV: Albarán de venta	**TR**: Traspaso

Delphi: function ANTESDEGUARDARLINEACONDETALLEV2(const Documento: AnsiString; Cabecera: Variant; Linea: Variant; Detalle: variant; const Estado: Integer; var PermitirGuardar: Boolean): variant;
C#: object AntesDeGuardarLineaConDetalleV2(string tipoDocumento, object cabecera, object linea, object detalle, int estado, ref bool permitirGuardar)

DespuesDeGuardarLinea

Evento que notifica que la línea se ha almacenado en el documento en memoria de manera satisfactoria. Los parámetros y disponibilidad coinciden con los de AntesDeGuardarLinea.

Delphi: procedure DESPUESDEGUARDARLINEA(const TipoDocumento: AnsiString; Cabecera: Variant; Linea: Variant);
C#: void DespuesDeGuardarLinea(string tipoDocumento, object cabecera, object linea)

DespuesDeGuardarLineaV2

Al igual que el evento anterior, notifica que la línea se ha almacenado en el documento en memoria de manera satisfactoria añadiendo la información del estado original de la línea.

Los parámetros y disponibilidad coinciden con los de **AntesDeGuardarLineaV2**.

C#: void DespuesDeGuardarLineaV2(string tipoDocumento, object cabecera, object linea, int estado)
Delphi: procedure DESPUESDEGUARDARLINEAV2(const TipoDocumento: AnsiString; Cabecera: Variant; Linea: Variant; Estado: Integer);

Guardado / Cancelado

AntesDeGuardarDocumento

Este evento notifica la intención de guardar los cambios de un documento. Es una función booleana. Si devuelve el valor false, cancela el guardado.

Dephi: function ANTESDEGUARDARDOCUMENTO(Documento: AnsiString; const IdDoc: Double; var Cabecera: Variant; var Lineas: Variant): Boolean; stdcall;
C#: bool AntesDeGuardarDocumento(string documento, double idDoc, object cabecera, object lineas);

Se **aplica a los documentos comerciales**, de **stock** y a **órdenes de producción**.

El parámetro '**Documento**' informa el tipo de documento que ha generado el evento. Admite las siguientes codificaciones:

OC: Oferta de compra
OV: Oferta de venta
PC: Pedido de compra
PV: Pedido de venta
AC: Albarán de compra
AV: Albarán de venta
PR: Órdenes de producción
DC: Depósito de compra
DV: Depósito de venta
FC: Factura de compra
FV: Factura de venta
RE: Regularización
TR: Traspaso

- '**IdDoc**' informa el identificador del documento que ha originado el evento. En caso de alta, viene a cero.
- '**Cabecera**' contiene los datos de la cabecera del documento. Es un conjunto de datos con un solo registro. Los campos que contiene se pueden actualizar desde el evento, con lo que se pueden modificar los datos de cabecera del documento. Los valores recibidos se volverán a validar exactamente igual que si el valor lo hubiera entrado el usuario desde el interfaz. Es imprescindible que estos cambios cumplan las reglas de negocio.
- '**Líneas'** contiene los datos de las líneas del documento. Es un conjunto de datos con al menos un registro. Sus valores no son actualizables desde el evento.

Este evento permite dos posibilidades:

- Poder añadir lógica de negocio particular de nuestra aplicación, al poder realizar validaciones extra, e impedir guardar el documento en caso de no superarlas.
- Poder cambiar el valor de los campos de la cabecera, incluyendo campos de diccionario de terceros.

A tener en cuenta…
Se desaconseja usar esta versión del evento. Debería usarse *AntesDeGuardarDocumentoV2*, que se describe a continuación.

AntesDeGuardarDocumentoV2

Este evento es una evolución del anterior. Añade un parámetro '**Estado**'.

Dephi: function ANTESDEGUARDARDOCUMENTOV2(Documento: AnsiString; const IdDoc: Double; var Cabecera: Variant; var Lineas: Variant; Estado: integer): Boolean; stdcall;
C#: bool AntesDeGuardarDocumentoV2(string documento, double idDoc, object cabecera, object lineas, int estado);

El nuevo parámetro '**Estado**' es un entero. Informa qué tipo de operación se está realizando con el documento. Los valores posibles son:

Valor	Significado
0	Alta
1	Modificación
2	Borrado
3	Sirviendo líneas.
4	Anulando líneas.
5	Revertir anulación de líneas.

Se aplica a los documentos comerciales, de stock, expedientes, cuotas, punto de venta y órdenes de producción.

Los valores correspondientes al parámetro '**documento**' son:

OC: Oferta de compra
OV: Oferta de venta
PC: Pedido de compra
PV: Pedido de venta
AC: Albarán de compra
AV: Albarán de venta
PR: Órdenes de producción
CUOTA: Cuotas
DC: Depósito de compra
DV: Depósito de venta
FC: Factura de compra
FV: Factura de venta
RE: Regularización
TR: Traspaso
TQ: tique (TPV)
EXPEDIENTE: expedientes

El resto de la funcionalidad del evento es igual que en **AntesDeGuardarDocumento**.

DespuesDeGuardarDocumento

Este evento notifica que los cambios de un documento se han finalizado. Ya están 'en firme' en la base de datos.

Delphi: procedure DESPUESDEGUARDARDOCUMENTO(Documento: AnsiString; const IdDoc: Double);
C#: void DespuesDeGuardarDocumento(string documento, double idDoc);

Se aplica a los documentos comerciales, de stock y órdenes de producción. No está disponible para cuotas ni expedientes.

A tener en cuenta…
Se desaconseja usar esta versión del evento. Debería usarse *DespuesDeGuardarDocumentoV2*, que se describe a continuación.

DespuesDeGuardarDocumentoV2
Este evento es una ampliación del evento anterior.

Delphi: procedure DESPUESDEGUARDARDOCUMENTOV2(Documento: AnsiString; const IdDoc: Double; Estado: integer);
C#: void DespuesDeGuardarDocumentoV2(string documento, double idDoc, int estado);

Se aplica a los documentos comerciales, de stock, órdenes de producción, cuotas y expedientes.
El parámetro '**Documento**' informa el tipo de documento que ha generado el evento. Admite las siguientes codificaciones:

OC: Oferta de compra	**DC: Depósito de compra**
OV: Oferta de venta	**DV**: Depósito de venta
PC: Pedido de compra	**FC**: Factura de compra
PV: Pedido de venta	**FV**: Factura de venta
AC: Albarán de compra	**RE**: Regularización
AV: Albarán de venta	**TR**: Traspaso
PR: Órdenes de producción	**EXPEDIENTE**: expedientes
CUOTA: Cuotas	

- **'IdDoc'** informa el identificador del documento que ha originado el evento. Mediante este parámetro se puede recuperar la información del documento, por lo que no se proporciona más información de contexto.
- **'Estado'** es un entero. Informa qué tipo de operación se está realizando con el documento. Los valores posibles son:

Valor	Significado
0	Alta
1	Modificación
2	Borrado
3	Sirviendo líneas.
4	Anulando líneas.
5	Revertir anulación de líneas.

Todo el resto de los parámetros y de la funcionalidad son iguales que en el evento anterior.

Cuándo se aconseja utilizar este evento.

- Cuando se necesita modificar información del documento y no se puede hacer mediante los eventos ***AntesDeGuardarDocumento*** o ***AntesDeGuardarDocumentoV2***. Por ejemplo, modificaciones en campos de líneas, informar detalles (aunque hay eventos específicos para esto), añadir líneas al documento, etc. Cualquier cambio en el documento debe realizarse utilizando los objetos de **a3ERPActiveX.**

- Cuando se quiere modificar tablas de terceros, y se necesita tener el documento completo y 'en firme' en la base de datos.

DespuesDeCancelarDocumento

Se dispara este evento cuando se cancela la edición / alta de un documento.

> **Delphi**: procedure DESPUESDECANCELARDOCUMENTO(Documento: AnsiString; const IdDoc: Double; Cabecera: Variant; var Lineas: Variant; Estado: integer);
> **C#**: void DespuesDeCancelarDocumento(string documento, double idDoc, object cabecera, object lineas, int estado);

Sólo se aplica a los documentos comerciales y a las cuotas.
El parámetro '**Documento**' informa el tipo de documento que ha generado el evento. Admite las siguientes codificaciones:

OC: Oferta de compra
OV: Oferta de venta
PC: Pedido de compra
PV: Pedido de venta
AC: Albarán de compra
CUOTAS: Cuotas
DC: Depósito de compra
DV: Depósito de venta
FC: Factura de compra
FV: Factura de venta
AV: Albarán de venta

- '**IdDoc**' informa el identificador del documento que ha originado el evento. En caso de alta, viene a cero.
- **'Cabecera'** y **'Líneas'** son conjuntos de datos.
- **'Cabecera'** tiene un solo registro, que informa los datos de la cabecera del documento. 'Líneas' contiene los datos de las líneas del documento., con al menos un registro. Ninguno de los dos es actualizable, sólo es a nivel informativo. Sólo están informados en caso de que se esté cancelando una modificación.
- **'Estado'** es un entero. Sólo tiene dos valores, 0 si era un alta y 1 si era una modificación.

Comunes a documentos y maestros

RePintar

Este evento se lanza tras el guardado de cualquier documento del circuito de compra / venta (incluyendo cuotas, expedientes) y maestros.

Su finalidad es permitir recargar el documento o maestro por si se hubiese realizado algún cambio desde fuera de la lógica del programa, por ejemplo: desde una extensión que usara a3ERPActiveX o una sentencia SQL para modificar la entidad una vez guardada. Esto podría ocurrir si se implementa el DespuesDeGuardarDocumento (o DespuesDeGuardarMaestro) y se modificase algún campo. Si queremos que el usuario observe el cambio realizado, tendremos que implementar una respuesta a este evento.

IMPORTANTE: Se recomienda usar solo si no existiera otra manera de realizar el cambio antes de guardar la entidad. Una forma de evitarlo sería usar los eventos AntesDeGuardar... que permiten modificar valores de los campos de la entidad en edición.

En caso de que no se pueda evitar, se recomienda que solo se devuelva "True" en los casos en los que sea necesario, ya sea comprobando la tabla recibida o determinando si la ejecución se está realizando desde a3ERP o desde una programación no visual (en la que no haría falta).

Delphi: function REPINTAR(Tabla: string): Boolean;
C#: function bool Repintar(string tabla)

En el campo "**Tabla**" se recibe el nombre de la tabla del maestro, o de la tabla cabecera del documento, que se acaba de guardar.

El valor devuelto deberá ser "**True**" si se pretende que a3ERP recargue los valores de la entidad o "**False**" en caso contrario.

Configuración

Los siguientes eventos tienen como misión notificar el cambio en la configuración de algún elemento del **ERP**.

AntesDeGuardarDatosConfiguracionEmpresa

Este evento se lanza cuando se modifican datos de configuración de la empresa (Datos Generales \ parametrización empresa) almacenados en la tabla **DatosConfig**, permitiendo evitar dichos cambios.

Delphi: function ANTESDEGUARDARDATOSCONFIGURACIONEMPRESA(Id: Integer; var Datos: Variant; Estado: Integer);
C#: bool AntesDeGuardarDatosConfiguracionEmpresa(int id, object datos, int estado)

Donde cada campo refiere a:

- **Id:** Identificador de la empresa. Si no es multiempresa es un cero.
- **Datos**: Objeto dataset que contiene los campos y los valores que se están guardando.
- **Estado**: Indica si es un nuevo registro (0), si es una modificación (1) o un borrado (2)
- **Retorno:** True para permitir los cambios, false en caso contrario (aborta el proceso)

DespuesDeGuardarDatosConfiguracionEmpresa

Notifica que se han modificado los datos de configuración de la empresa y se han almacenado en base de datos.

Delphi: procedure DESPUESDEGUARDARDATOSCONFIGURACIONEMPRESA(Id: Integer; Estado: Integer);
C#: void DespuesDeGuardarDatosConfiguracionEmpresa(int id, int estado)

Donde cada campo refiere a:

- **Id**: Identificador de la empresa. Si no es multiempresa es un cero.
- **Estado**: Indica si es un nuevo registro (0), si es una modificación (1) o un borrado (2)
- Estos valores coincidirán con los del evento anterior.

AntesDeGuardarDatosEmpresa

Este evento se produce antes de que se hagan permanentes los cambios de los datos identificativos de la empresa (**Datos Generales\ parametrización empresa**) almacenados en la tabla **DatosEmp** permitiendo cancelar el proceso.

Delphi: function ANTESDEGUARDARDATOSEMPRESA(Id: Integer; var Datos: Variant; Estado: Integer);
C#: bool AntesDeGuardarDatosEmpresa(int id, object datos, int estado)

Donde cada campo refiere a:

- **Id**: Identificador de la empresa. Si no es multiempresa es un cero.
- **Datos:** Objeto dataset que contiene los campos y los valores que se están guardando.
- **Estado**: Indica si es un nuevo registro (0), si es una modificación (1) o un borrado (2)
- **Retorno:** True para permitir los cambios, False para abortar el proceso.

DespuesDeGuardarDatosEmpresa

Notifica que se han modificado los datos de la empresa y se han almacenado en base de datos.

Delphi: procedure DESPUESDEGUARDARDATOSEMPRESA(Id: Integer; Estado: Integer);
C#: void DespuesDeGuardarDatosEmpresa(int id, int estado)

Donde cada campo refiere a:

- **Id:** Identificador de la empresa. Si no es multiempresa es un cero.
- **Estado**: Indica si es un nuevo registro (0), si es una modificación (1) o un borrado (2)

Estos valores coincidirán con los del evento anterior.