



a3ERP

Eventos

Guía de uso

Sumario

Eventos a3ERP	4
Introducción	4
Conceptos básicos	4
Tipos de datos	4
Equivalencia entre entornos y lenguajes.....	5
Parámetros complejos	6
Ejemplos Delphi y C#.....	7
Estructura básica DLL.....	8
Registro en ERP	9
Nativa	9
COM	10
Eventos	11
Recomendaciones generales	11
Ciclo de Vida.....	11
Iniciar.....	13
IniciarConSistema	13
IniciarGeneral.....	14
SePuedeFinalizar	15
Finalizar	16
Entidades.....	17
Maestros	17
AntesDeGuardarMaestro.....	17
AntesDeGuardarMaestroV2	17
DespuesDeGuardarMaestro	18
DespuesDeGuardarMaestroV2.....	18
AntesDeBorrarMaestro.....	18
DespuesDeBorrarMaestro	19
Usuarios	19
AntesDeGuardarUsuario.....	19
DespuesDeGuardarMaestroV2.....	20
Documentos.....	20
Ciclo de vida	20
DespuesDeCargarDocumento.....	20
DespuesDeCargarDocumentoV2	21
Control de ejecución	22
ActivarMetodosAlServir	22
ActivarMetodosAlAnular.....	22
Sustitución de formularios.....	23

EsDocumentoExterno	23
HacerDocumentoExterno	23
HacerDocumentoExternoV2	24
Edición de documentos.....	24
SePuedeRealizarAccion.....	24
Impresión de documentos.....	25
SePuedeImprimirDocumento	25
Modificaciones.....	26
Campos	26
CamposAEscucharEnDocumento.....	26
AntesDeCambioDeCampoEnDocumento	27
AntesDeCambioDeCampoEnDocumentoV2	27
DespuesDeCambioDeCampoEnDocumento	28
AlValidarClienteProveedor	28
Líneas	29
AntesDeGuardarLinea	29
AntesDeGuardarLineaV2.....	29
AntesDeGuardarLineaConDetalle	30
AntesDeGuardarLineaConDetalleV2.....	31
DespuesDeGuardarLinea	31
DespuesDeGuardarLineaV2	31
SeProporcionaDetalle	32
ObtDetalle	32
ObtPrecioCompra	33
ObtPrecioVenta.....	34
AutorizarPrecioVenta.....	34
Guardado/ Cancelado	34
AntesDeGuardarDocumento	34
AntesDeGuardarDocumentoV2	35
DespuesDeGuardarDocumento	36
DespuesDeGuardarDocumentoV2.....	37
AntesDeAplicarCambios.....	38
DespuesDeCancelarDocumento	39
AntesDeActualizarStock.....	39
Eventos particulares de algunos documentos.....	41
Facturas.....	39
Ofertas	40
Expedientes.....	42
Comunes a documentos y maestros.....	45
RePintar.....	45

Configuración	46
Producción	47
Listados	49
Cartera	49
Asientos contables	51
Formularios y ventanas.....	52
Comisiones.....	53
Remesas	59
Desplegando la DLL en la oficina del cliente.....	60
Fichero LOG para las DLL	62
Control de excepciones procedentes de las DLL	62
Cómo llamar a una DLL desde MIMENU.MENU	62
Parámetros en programas externos	64
Lectura recomendada	65

Eventos a3ERP

Introducción

Los eventos de **a3ERP** 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 **a3ERP** 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 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 a3ERP.
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 a3ERP**.

Nombre	Delphi	.NET (C#)	VB6				
Cadena	Dependiendo del soporte Unicode: <table border="1" data-bbox="405 1240 778 1408"> <tr> <td>Anterior a Delphi 2009</td> <td>String</td> </tr> <tr> <td>Desde Delphi 2009</td> <td>AnsiString</td> </tr> </table>	Anterior a Delphi 2009	String	Desde Delphi 2009	AnsiString	System.String (String)	String
Anterior a Delphi 2009	String						
Desde Delphi 2009	AnsiString						
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, contruidos 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:

Nombre	Descripción															
Identificador	<p>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.</p> <table border="1"> <thead> <tr> <th>Índice</th> <th>Tipo</th> <th>Contenido</th> </tr> </thead> <tbody> <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> </tbody> </table>	Índice	Tipo	Contenido	0	Variante	Valor del primer campo de la llave del elemento al cual refiere el identificador.	...			n	Variante	Valor del n-ésimo campo de la llave.			
Índice	Tipo	Contenido														
0	Variante	Valor del primer campo de la llave del elemento al cual refiere el identificador.														
...																
n	Variante	Valor del n-ésimo campo de la llave.														
Campo	<p>Matriz de dos elementos que refieren al nombre de un campo de una entidad de a3ERP (maestro, documento...) y su valor:</p> <table border="1"> <thead> <tr> <th>Índice</th> <th>Tipo</th> <th>Contenido</th> </tr> </thead> <tbody> <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> </tbody> </table>	Índice	Tipo	Contenido	0	Cadena	Nombre del campo	1	Variante	Valor del campo						
Índice	Tipo	Contenido														
0	Cadena	Nombre del campo														
1	Variante	Valor del campo														
Registro	<p>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).</p> <table border="1"> <thead> <tr> <th>Índice</th> <th>Tipo</th> <th>Contenido</th> </tr> </thead> <tbody> <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> </tbody> </table>	Índice	Tipo	Contenido	0	Entero	Número de Campos	1	Campo	Primera Pareja campo/valor	...			n	Campo	N-ésima pareja campo/valor
Índice	Tipo	Contenido														
0	Entero	Número de Campos														
1	Campo	Primera Pareja campo/valor														
...																
n	Campo	N-ésima pareja campo/valor														

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 **a3ERP**. 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 **a3ERP** 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 **a3ERP** para comunicar los eventos. Esta será la que registremos en **a3ERP** 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 **a3ERP** 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 a3ERP

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, ORDENEJECUCION)
    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 **a3ERP** 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 librerías 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 “**nombredelSinExtension.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

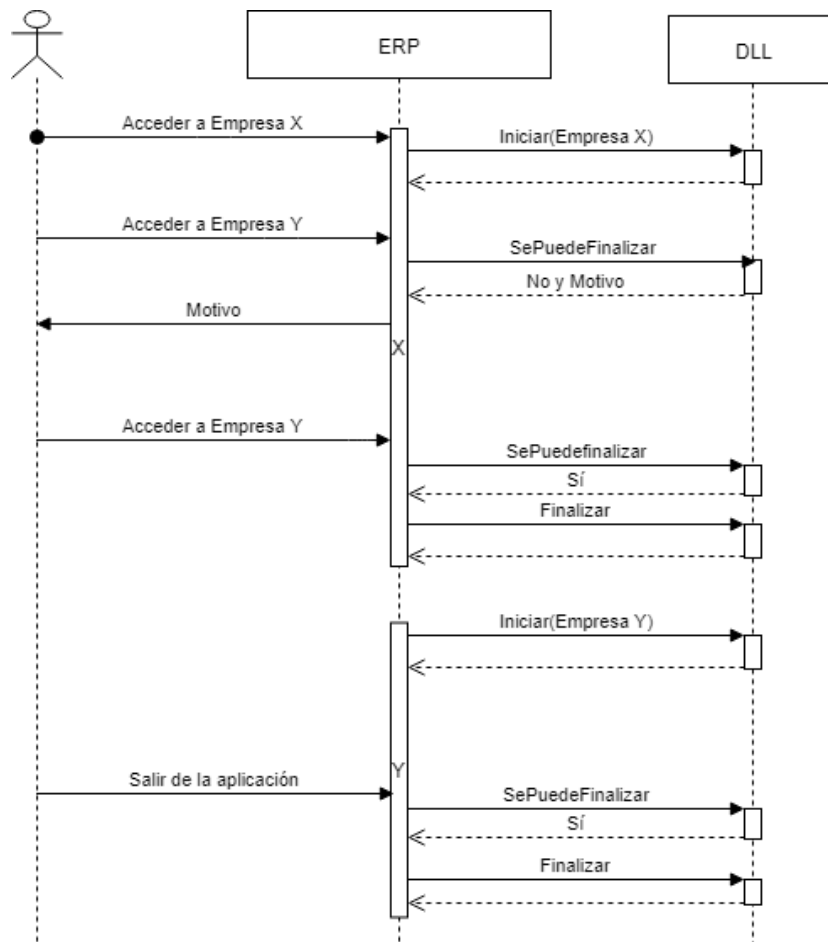
La implementación de los eventos permite, en muchos casos, acceder a los datos de a3ERP, alterarlos e incluso cambiar el comportamiento de la aplicación. Se recomienda un cuidado extremo a la hora de plantear y ejecutar tales acciones. Para dar una guía que ayude en tal tarea, recopilamos aquí algunas recomendaciones a la hora de implementar eventos de a3ERP:

- **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 a3ERP 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 a3ERP, 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 de a3ERP siempre y cuando, en el caso de una dll nativa, no se haya implementado también 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 autenticarse con otras credenciales y, por lo tanto, consumir una licencia adicional.

IniciarConSistema

Este evento se ejecuta cuando se accede a una determinada empresa de a3ERP 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 **a3ERP**, usando componentes de acceso a **datos compatibles con ADO**. Hoy día se puede usar IniciarGeneral para obtener la conexión que usa **a3ERP** y así evitar tener que autenticarse 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 **a3ERP** para acceder a las bases de datos de sistema y de empresa. Usando estas interfaces, la programación ejecutará sus sentencias dentro de las transacciones de **a3ERP**.

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 Connection a base de datos de sistema.

Posición 1.8: Interfaz ADO Connection a base de datos de empresa logado con usuario de trabajo **a3ERP**.

Posición 1.9: Interfaz ADO Connection 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 **a3ERP** 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 **a3ERP** 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

UltimoMotivo

Evento con el que a3ERP interrogará al módulo de terceros que canceló el último flujo de trabajo de a3ERP (devolvió false en un evento que permite cancelar la operación), para pedir el motivo a indicar al usuario.

Delphi: function ULTIMOMOTIVO: AnsiString; stdcall;

C#: string UltimoMotivo()

El resultado deberá ser o bien la cadena vacía (No aparecerá mensaje al usuario) o una cadena de texto que describa el motivo por el que se canceló la operación. Este mensaje aparecerá prefijado con el nombre con el que se ha registrado el módulo.

A tener en cuenta...

Por el momento, solo se dispara tras AntesDeGuardarDocumento y AntesDeGuardarDocumentoV2, en caso de devolver false.

Maestros

Los distintos maestros (Clientes, Proveedores, Artículos, Cuentas...) de a3ERP notifican mediante eventos cuándo se realiza modificaciones o borrados de los mismos, como notificación de dichas operaciones pudiendo impedirlos 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.

Delphi: 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 primaria 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.

Usuarios

La entidad **usuarios** tiene sus propios eventos, que permiten detectar su creación, modificación o borrado.

A tener en cuenta...

Los eventos de usuario no se disparan desde el selector de empresas si todavía no se ha entrado en alguna empresa.

ANTESDEGUARDARUSUARIO

Se produce justo antes de guardar el alta, modificación o borrado de un usuario de la aplicación, permitiendo continuar o abortar la operación mediante el parámetro de retorno.

```
Delphi: function ANTESDEGUARDARUSUARIO(const IdUsuario: Integer; var DatosUsuario: Variant;  
const Estado: Integer): Boolean; stdcall;  
C#: bool AntesDeGuardarUsuario(int idUsuario, object datosUsuario, int estado)
```

En el campo **IdUsuario** recibiremos el valor del campo identificador de la tabla Usuarios asignado al usuario sobre el que se ha realizado la modificación.

A tener en cuenta...

En alta el valor será -1.

DatosUsuario contendrá los valores del registro de la tabla Usuarios que se vayan a aplicar a B.D.

Estado es un valor entero que nos informa de que tipo de operación (alta, modificación o borrado) se ha producido. Los valores posibles son 0 para alta, 1 para modificación y 2 para borrado.

El valor de **retorno** indicará si se permite continuar (guardando los cambios en el sistema) o no con los valores booleanos "true" o "false" respectivamente.

DESPUESDEGUARDARUSUARIO

Notifica que se ha guardado en el sistema cambios (alta, modificación o borrado) en un usuario de la aplicación.

Delphi: procedure DESPUESDEGUARDARUSUARIO(const IdUsuario: Integer; const Estado: Integer); stdcall;

C#: void DespuesDeGuardarUsuario(int idUsuario, int estado)

En el campo **IdUsuario** recibiremos el valor del campo identificador de la tabla Usuarios asignado al usuario sobre el que se ha realizado la modificación.

A tener en cuenta...

En alta el valor será el del registro almacenado, no -1 como ocurre en el evento AntesDeGuardarUsuario).

Estado es un valor entero que nos informa de que tipo de operación (alta, modificación o borrado) se ha producido. Los valores posibles son 0 para alta, 1 para modificación y 2 para borrado.

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	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
CUOTA: Cuota	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;

Control de ejecución

Los siguientes eventos permiten decidir si se van a ejecutar los eventos de modificado en ciertas operaciones del programa.

ActivarMetodosAlServir

Esta función permite ejecutar o no los métodos 'AntesDeGuardarDocumento/V2' y 'DespuesDeGuardarDocumento/V2' cuando se está sirviendo un documento.

Delphi: function ActivarMetodosAlServir: Boolean; stdcall;

C#: bool ActivarMetodosAlServir()

Se aplica a **documentos comerciales**.

Si la función devuelve 'true', los métodos se ejecutan al servir.

Si la función devuelve 'false', o **no se implementa**, los métodos no se llaman.

A tener en cuenta...

El permitir la ejecución de los métodos dependerá de la funcionalidad que implemente la DLL en éstos. Si, por ejemplo, el proceso de la DLL solo es aplicable en altas o modificaciones, no debería ejecutarse.

En otros casos, puede que el código sea 'inocuo' al servir, pero por cuestiones de rendimiento podría ser interesante inhibir la ejecución de los métodos.

ActivarMetodosAlAnular

Esta función permite ejecutar o no los métodos 'AntesDeGuardarDocumento/V2' y 'DespuesDeGuardarDocumento/V2' cuando se está anulando un documento.

Delphi: function ActivarMetodosAlAnular: Boolean; stdcall;

C#: bool ActivarMetodosAlAnular()

También se llama si estamos revirtiendo una anulación. Se aplica a **documentos comerciales**.

Si la función devuelve 'true', los métodos se ejecutan al servir.

Si la función devuelve 'false', o **no se implementa**, los métodos no se llaman.

La nota del evento 'ActivarMetodosAlServir' es aplicable a este caso.

Sustitución de formularios

Estos eventos permiten sustituir ciertos formularios de documentos por formularios a medida en la programación de terceros. Se puede utilizar para extender o para simplificar la edición de documentos.

EsDocumentoExterno

Método para indicar a **a3ERP** si el tipo de documento indicado se edita mediante una pantalla o edición externa (true) o, por el contrario, usa el mantenimiento estándar (false) para el tipo de documento indicado (TipoDocumento).

Delphi: function EsDocumentoExterno (TipoDocumento: string): Boolean; stdcall;

C#: bool EsDocumentoExterno(string tipoDocumento)

Este método permite que se pueda editar, mediante el método que elija el programador de la DLL de terceros que la implemente, un documento concreto del tipo especificado (TipoDocumento).

En caso de respuesta afirmativa (true) posteriormente se llamará a '**HacerDocumentoExterno**' para permitir que la programación de terceros realice la edición de un documento concreto del tipo especificado.

En caso contrario, se llamará a la edición de **a3ERP** del tipo de documento para mostrar/editar un documento concreto.

Se aplica a **documentos comerciales y a expedientes**.

A tener en cuenta...

Hoy en día este método puede ser invocado por **a3ERP** varias veces a lo largo de su ejecución para el mismo tipo de documento, generalmente llamando primero a EsDocumentoExterno y posteriormente a HacerDocumentoExterno.

HacerDocumentoExterno

Método para invocar una edición, implementada por la DLL de terceros, de un documento que no sea la estándar de **a3ERP**.

Delphi: procedure HacerDocumentoExterno(TipoDocumento: string; IdDocumento: Double); stdcall;

C#: void HacerDocumentoExterno (string tipoDocumento, double idDocumento)

Este método permite, a una programación a medida, presentar una edición de un documento (IdDocumento) concreto de un determinado tipo (TipoDocumento) distinta a la proporcionada por **a3ERP**.

Sólo se llamará a este método si previamente se devolvió true en EsDocumentoExterno, para el mismo tipo de documento pasado por parámetro y si no se encuentra implementado HacerDocumentoExternoV2.

Se aplica a **documentos comerciales y a expedientes**.

HacerDocumentoExternoV2

Método para invocar una edición, implementada por la DLL de terceros, de un documento que no sea la estándar de **a3ERP**.

Delphi: procedure HacerDocumentoExternoV2 (TipoDocumento: string; IdDocumento: Double; Parametros: Variant); stdcall;

C#: void HacerDocumentoExternoV2 (string tipoDocumento, double idDocumento, object parametros)

Este método permite, a una programación a medida, presentar una edición de un documento (IdDocumento) concreto de un determinado tipo (TipoDocumento) distinta a la proporcionada por **a3ERP**.

Sólo se llamará a este método si previamente se devolvió true en EsDocumentoExterno, para el mismo tipo de documento pasado por parámetro.

A diferencia del método anterior, se reciben los parámetros de la pantalla.

Si el parámetro "Modal" es igual a 'T', en **Parametros.Devolucion.Valor** y **Parametros.Devolución.Estado** dejaremos el resultado de la devolución.

La programación externa debe devolver un valor de resultado cuando es modal; si el resultado es "Vacío" entendemos que se ha cancelado la edición.

Se aplica a **documentos comerciales y a expedientes**.

Edición de documentos

SePuedeRealizarAccion (disponible a partir de la Versión 13.01.0)

Este evento se ejecuta justo antes de abrir un documento. Se envía por parámetro el tipo de documento a abrir, el identificador del documento en la base de datos y la aplicación a realizar. Se devuelve como dos últimos parámetros de salida la acción permitida con el documento y un mensaje en forma de texto.

Delphi: procedure SePuedeRealizarAccion (const TipoDocumento: AnsiString; const IdDoc: Double; const Accion: Integer; out aRespuesta: integer; out aMotivo: AnsiString)

C#: void SePuedeRealizarAccion (string TipoDocumento, double idDoc, int Accion, ref int aRespuesta, ref string aMotivo);

El parámetro '**TipoDocumento**' 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.
- **'Accion'** es un entero que envía a3ERP. Informa qué tipo de operación se quiere realizar con el documento. Los valores posibles son:

Valor	Significado
0	Alta
1	Edición
2	Borrado

- **'aRespuesta'** es un entero. Informa de las acciones permitidas realizar sobre el documento. En caso de tener varias DLLs, la acción permitida siempre será la más restrictiva. Los posibles valores son:

Valor	Significado
0	Acción no permitida
1	Sólo lectura
2	Acción permitida

- **'aMotivo'** es un string dirigido al usuario en el que se indica el motivo de la respuesta dada.

Cuando se aconseja utilizar este evento.

- Cuando se quiere poder decidir las acciones que se pueden realizar sobre un documento dado, por ejemplo, el borrado o edición de una factura ya confirmada.

Impresión de documentos

SePuedeImprimirDocumento

Método para indicar si se puede imprimir o no un documento concreto, en cuyo caso permite devolver una cadena con el motivo para que a3ERP la muestre al usuario que lo pidió.

Delphi: function SEPUEDEIMPRIMIRDOCUMENTO(TipoDocumento: Ansistring; IdDocumento: Double; var Mensaje: string): Boolean; stdcall;

C#: bool SePuedeImprimirDocumento(string tipoDocumento, float idDocumento, ref string mensaje)

En **TipoDocumento** recibiremos una cadena de texto que describirá el tipo del documento que se pretende imprimir.

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	AR: Regularización
AV: Albarán de venta	AT: Traspasos
OP: Orden de producción	

IdDocumento albergará el identificador del documento en cuestión.

En el valor de **retorno** se deberá indicar si se continúa con la impresión o no (valores true o false respectivamente).

En caso de que el valor de retorno sea false (impedir impresión), se puede rellenar el parámetro **Mensaje**, con un mensaje que indique el motivo al usuario el motivo por el que no se continúa.

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 a3ERP 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	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
CUOTA: Cuota	IN: Inventario
TQ: Tique de TPV	

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 **a3ERP** 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 := unassigned; // 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 **a3ERP**, 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 a3ERP. Nos puede servir para trazar el cambio o realizar operaciones con datos propios.

Delphi: procedure DESPUESECAMBIODECAMPOENDOCUMENTO(ClaseDocumento: AnsiString;
IdDoc: Double; Campo: string; ValorAnterior, NuevoValor: OleVariant); stdcall;

C#: void DespuesDeCambioDeCampoEnDocumento(string claseDocumento, float idDoc, string
campo, object valorAnterior, object nuevoValor);

AlValidarClienteProveedor

A pesar de su nombre, este evento se dispara para validar únicamente el cliente recientemente indicado en un documento de venta, cuota o expediente. Permite abortar el cambio de cliente, mediante el resultado de la llamada.

Delphi: function ALVALIDARCLIENTEPROVEEDOR(const TipoDocumento, Codigo: string): Boolean;
stdcall;

C#: bool AlValidarClienteProveedor(string tipoDocumento, string codigo);

En **TipoDocumento** se recibirá los siguientes valores:

- EXPEDIENTE para un expediente.
- Cuota para una cuota
- OV para Oferta de venta
- PV para Pedido de venta
- DV para depósito de venta
- AV para albarán de venta
- FV para factura de venta

En **Codigo**, el código del cliente que se acaba de asignar al documento, cuota o expediente.

El **resultado**, True o False determinará si el programa permite aplicar el cambio al campo o no.

A tener en cuenta...

Se puede implementar esta funcionalidad con el evento de “AntesDeCambioEnCampoDeDocumento”, con más información.

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	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
CUOTA: Cuota	EXPEDIENTE: Expediente
TQ: Tique de TPV	

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 (Disponible desde 13.02.01)
AV: Albarán de venta	EXPEDIENTE: Expedientes
	TR: Traspaso (disponible a partir de versión 13.02)

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 AntesDeGuardarLineaV2, es decir, de implementarse los dos solo se llamará a esa versión (AntesDeGuardarLineaV2)

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 (disponible a partir de versión 13.02.01)
AV: Albarán de venta	TR: Traspaso (disponible a partir de versión 13.02)

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);

SeProporcionaDetalle

Esta función informa a **a3ERP** que la DLL externa se encargará de informar el detalle de los artículos con lotes, fechas de caducidad, números de serie o ubicaciones.

Delphi: function SEPROPORCIONADETALLE(Documento: String; Cabecera: Variant; Linea: Variant): Boolean; stdcall;

C#: bool SeProporcionaDetalle(string documento, object cabecera, object linea)

Está relacionada con el procedimiento 'ObtDetalle'.

Disponible en **documentos de compra, venta, traspasos y regularizaciones**.

Cuando se devuelve el valor true, **a3ERP** hace una llamada al método 'ObtDetalle'. Si se devuelve false, o ni siquiera se implementa, no se efectúa la llamada.

Este evento se llama cada vez que se da de alta una nueva **línea de detalle**. Por ejemplo, en un albarán se entra una línea de un artículo con lotes, 7 unidades. El usuario pulsa el botón 'detalle', e informa en una línea 3 unidades y en otra 4. Cada una de estas líneas provocará una llamada a 'SeProporcionaDetalle'.

El parámetro '**Documento**' puede contener uno de los siguientes valores:

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: Traspasos

En el parámetro **cabecera** se enviará el conjunto de campos (registro) y sus valores guardado en la tabla cabecera del documento (ejemplo con factura de ventas: CABEFACV). En el parámetro **línea**, el conjunto de campos (registro) con los valores de los campos de la línea del documento (siguiendo el mismo ejemplo, el contenido para el documento de la tabla LINEFACT).

ObtDetalle

Informa el detalle de los artículos con lotes, fechas de caducidad, números de serie o ubicaciones.

Delphi: procedure OBTDETALLE(Documento: String; Cabecera: Variant; Linea: Variant; var Detalle: Variant); stdcall;

C#: void ObtDetalle(string documento, object cabecera, object línea, ref object detalle)

Este evento se ejecuta sólo si el evento SeProporcionaDetalle ha respondido 'true'.

Disponible en **documentos de compra, venta, traspasos y regularizaciones**.

El parámetro **Documento** puede contener uno de los siguientes valores:

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: Traspasos

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).

En el parámetro **detalle** se devolverá una estructura dataset con un único registro que contiene las parejas campo/valor de los campos detalle que se deseen informar al ERP. Sólo es necesario incluir los campos aplicables para el artículo.

Por ejemplo, si no usa fechas de caducidad, no es necesario devolver un par ['FecCaduc', '']. Los campos podrán ser: **NumSerie**, **Lote**, **FecCaduc** (como cadena) y **Ubicacion**.

Si no se quiere pasar ningún valor, hay que devolver el valor equivalente a Variant.NULL (En C# System.DBNull.Value o Null en Delphi) o un dataset vacío (ejemplo en C#: new object[] { 0, new object[] { 0 } }).

ObtPrecioCompra

Este evento notifica del cálculo del precio (precio y descuentos) para un artículo en un documento de compra, permitiendo devolver otro precio (no los descuentos) distinto.

Delphi: function OBTPRECIOCOMPRA(const Precio: Double; Cabecera, Linea: Variant): Double; stdcall;

C#: float ObtPrecioCompra(float precio, object cabecera, object linea)

En **Precio** se obtiene el precio de compra calculado para las condiciones actuales de la línea del documento.

En **Cabecera** se recibe el registro correspondiente a la cabecera del documento (correspondiente a la estructura en B.D. de la tabla de cabecera del tipo de documento en cuestión).

En **Linea**, registro correspondiente a la línea del documento de la cual se ha calculado el precio (correspondiente a la estructura en B.D. de la tabla de líneas del tipo de documento en cuestión).

El **retorno** debe tener el precio que se quiere aplicar a la línea (Por ejemplo, el mismo precio recibido).

A tener en cuenta...

- Se ejecuta al cambiar el artículo o las unidades de una línea.
- Esta funcionalidad hoy se podría implementar con 'Escuchar campo V2'.

ObtPrecioVenta

Este evento notifica del cálculo del precio (precio y descuentos) para un artículo en un documento de venta, permitiendo devolver otro precio (no los descuentos) distinto.

Delphi: Function OBTPRECIOVENTA(const Precio: Double; Cabecera, Linea: Variant): Double; stdcall;

C#: float ObtPrecioVenta(float precio, object cabecera, object linea)

En **Precio** se obtiene el precio de venta calculado para las condiciones actuales de la línea del documento.

En **Cabecera** se recibe el registro correspondiente a la cabecera del documento (correspondiente a la estructura en B.D. de la tabla de cabecera del tipo de documento en cuestión).

En **Linea**, registro correspondiente a la línea del documento de la cual se ha calculado el precio (correspondiente a la estructura en B.D. de la tabla de líneas del tipo de documento en cuestión).

El **retorno** debe tener el precio que se quiere aplicar a la línea (Por ejemplo, el mismo precio recibido).

A tener en cuenta...

- Se ejecuta al cambiar el artículo o las unidades de una línea
- Esta funcionalidad hoy se podría implementar con 'Escuchar campo V2'.

AutorizarPrecioVenta

Este evento permite aceptar o no el precio de una línea de documento de venta, de un artículo con precio mínimo indicado (es decir, no cero) al guardar una línea o al repartir precio por los componentes.

Delphi: function AUTORIZARPRECIOVENTA(const CodArt: string; const Unidades, Precio, Desc1, Desc2, Desc3, Desc4: Double): Boolean; stdcall;

C#: bool AutorizarPrecioVenta(string codArt, float unidades, float precio, float desc1, float desc2, float desc3, float desc4)

De la línea se recibe en **CodArt** el código de artículo, **precio** y descuentos (**Desc1 a Desc4**). Debe **devolverse** True si se acepta o False en caso contrario (con lo que se abortará el proceso de guardado de la línea).

A tener en cuenta...

- Se dispara al guardar una línea, repartir precio por componentes.
- No se dispara cuando se indican cero unidades, el artículo es el genérico, se sirve o se copian documentos.

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	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	

- '**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.

Si se devuelve false, posteriormente se llamará a *UltimoMotivo* (si está implementado), para obtener un mensaje que presentar al usuario para la cancelación.

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	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	TQ: tique (TPV)
CUOTA: Cuotas	EXPEDIENTE: expedientes
OPORTUNIDAD: Oportunidad	

El resto de la funcionalidad del evento es igual (incluyendo llamada a UltimoMotivo) 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	OPORTUNIDAD: Oportunidad

- '**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.

Cuando 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.

AntesDeAplicarCambiosDoc (A partir de Versión 13.01.0)

Este evento se ejecuta justo antes de aplicar los cambios definitivamente en la base de datos, cuando los cambios en la base de datos aun se encuentran en transacción SQL.

Delphi: function ANTESDEAPLICARCAMBIOSDOC(Documento: AnsiString; const IdDoc: Double; Estado: integer; var Motivo: AnsiString): Boolean; stdcall;

C#: bool ANTESDEAPLICARCAMBIOSDOC(string documento, double idDoc, int estado, ref string motivo);

La función deberá devolver true si se permite continuar con el proceso de aplicar los cambios a base de datos o false en caso contrario. Para esta última circunstancia, se deberá indicar en el parámetro Motivo un literal que se pueda presentar al usuario y le indique la razón de la cancelación del proceso.

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

- '**Motivo**' es una cadena que debe rellenarse en caso de devolver False. Debe indicarse un mensaje que informe al usuario del motivo de la cancelación del proceso.

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

Cuando se aconseja utilizar este evento.

- Cuando se necesita realizar alguna acción con el documento justo antes que este quede guardado, condicionando el guardado del documento a una posible excepción en la ejecución del método de la DLL, por ejemplo, el firmado y encadenamiento de una factura. 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.

DespuesDeAplicarCambiosDocumento (A partir de Versión 14.00.0)

Este evento se ejecuta justo después de aplicar los cambios definitivamente en la base de datos, ya fuera de la transacción que los guardó.

Delphi: procedure DESPUEDEAPLICARCAMBIOSDOCUMENTO(TipoDocumento: AnsiString; const IdDoc: Double; const Aplicado: integer; const Motivo: AnsiString); stdcall;

C#: void DespuesDeAplicarCambiosDocumento (string tipoDocumento, double idDoc, int aplicado, string motivo);

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

El parámetro '**TipoDocumento**' 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.
- '**Aplicado**' es un entero. Informa si los cambios se han aplicado (commit) o no (rollback):

Valor	Significado
0	No (Se ha realizado Rollback)
1	Sí (Se ha realizado Commit)

'**Motivo**' es una cadena que solo vendrá indicada si Aplicado tuviera valor 1 (rollback), con la descripción del error que lo hubiese provocado.

DespuesDeCancelarDocumento

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

Delphi: procedure DESPUEDECANCELARDOCUMENTO(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	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	AV: Albarán de venta
CUOTAS: Cuotas	

- **'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.

AntesDeActualizarStock

Este evento notifica que se va a proceder a actualizar el stock de un artículo en un determinado almacén. Permite cancelar la operación con el valor de retorno.

Delphi: Function ANTESDEACTUALIZARSTOCK(const CodArt, CodAlm: string): Boolean;
C#: bool AntesDeActualizarStock(string codArt, string codAlm)

En **CodArt** se recibe el código del artículo para el cual se está actualizando el stock.

En **CodAlm** el almacén afectado.

El valor de retorno informará a la aplicación si debe continuar con el proceso (true) o no (false), en cuyo caso se abortará.

Eventos particulares de algunos documentos

Facturas

TrasGenerarFacturaE

Delphi: procedure TRASGENERARFACTURAE(IdFac: Double; XMLDocument: Variant; var XMLText: string; var TextModified: Boolean); stdcall;

C#: void TrasGenerarFacturaE(float idFac, object xmlDocument, ref string xmlText, ref bool textModified)

Se ejecuta cuando se ha generado un documento e-factura, pero todavía no se ha firmado. Lo que permite ampliar la información del documento con nodos adicionales, que pueden ser obligatorios para el destinatario de la e-factura.

Después de este evento, el proceso sigue su curso y la e-factura se firma.

- **IdFac** es el número interno de identificación de la Factura (IDFACV).
- **XMLDocument** el documento generado en formato XML. Cumple una interface IDOMDocument. A través de esta interfaz se puede manipular el DOM correspondiente a la factura (por ejemplo, añadir nuevos nodos, eliminar o modificar nodos generados por la aplicación).
- **XMLText** la facturaE generada en formato texto. Es un parámetro de retorno, por lo que permitiría la modificación del documento indicando un texto XML bien formado.

AntesDeAnadirFacturaRectificada (V13.02.02)

Delphi: function ANTESDEANADIRFACTURARECTIFICADA(TipoFactura: AnsiString; IdFacRectificada: Double; DatosCabecera: Variant; DetalleRectificadas: Variant; var Motivo: AnsiString): Boolean; stdcall;

C#: bool AntesDeAnadirFacturaRectificada (string tipoFactura, double idFacRectificada, object datosCabecera, object detalleRectificadas, ref string motivo)

Se ejecuta justo antes de añadir una factura al detalle de facturas rectificadas de una factura rectificativa de venta o compra. Permite abortar la inserción (rectificado de dicha factura), devolviendo false, en cuyo caso permite indicar el motivo a mostrar al usuario.

- **TipoFactura** es el tipo de factura (V para venta, C para compra).
- **IdFacRectificada** será el identificador de la factura que se pretende rectificar en la factura rectificativa.
- **DatosCabecera** contendrá los datos de cabecera (registro) de la cabecera de factura rectificativa con la que se quiere rectificar la factura indicada en IdFacRectificada.,
- **DetalleRectificadas** alberga un conjunto de datos con la lista de facturas actualmente rectificadas por la factura cuyos datos recibimos en DatosCabecera (campos de la tabla __DETALLERECTIFICADAS_VENTAS o __DETALLERECTIFICADAS_COMPRAS, según tipo factura)
- **Motivo** se deberá indicar cuando devolvamos false en el evento (es decir, queramos impedir la rectificación de la factura con IdFacRectificada) y se quiera indicar un motivo al usuario.

AntesDeAnadirFacturaSustituída (V13.02.02)

Delphi: function ANTESDEANADIRFACTURASUSTITUIDA(TipoFactura: AnsiString; IdFacSustituída: Double; DatosCabecera: Variant; DetalleRectificadas: Variant; var Motivo: AnsiString): Boolean; stdcall;

C#: bool AntesDeAnadirFacturaSustituída(string tipoFactura, double idFacSustituída, object datosCabecera, object detalleRectificadas, ref string motivo)

Se ejecuta justo antes de añadir una factura al detalle de facturas sustituidas de una factura sustitutiva de venta. Permite abortar la inserción (rectificado de dicha factura), devolviendo false, en cuyo caso permite indicar el motivo a mostrar al usuario.

- **TipoFactura** es el tipo de factura (siempre V, venta).
- **IdFacSustituída** será el identificador de la factura que se pretende sustituir con la factura sustitutiva.
- **DatosCabecera** contendrá los datos de cabecera (registro) de la cabecera de factura sustitutiva con la que se quiere sustituir la factura indicada en IdFacSustituída.,
- **DetalleRectificadas** alberga un conjunto de datos con la lista de facturas actualmente rectificadas por la factura cuyos datos recibimos en DatosCabecera (campos correspondientes a __DETALLESUSTITUIDAS_VENTAS).
- **Motivo** se deberá indicar cuando devolvamos false en el evento (es decir, queramos impedir la sustitución de la factura con IdFacSustituída) y se quiera indicar un motivo al usuario.

Ofertas

AntesDeAceptarOferta

Este método se invoca antes de guardar una oferta que acaba de ser aceptada.

Delphi: function AntesDeAceptarOferta(TipoDocumento: string; IdDoc: Double): Boolean; stdcall;

C#: bool AntesDeAceptarOferta(string tipoDocumento, double idDoc)

Si se devuelve 'true', o no se implementa el evento, se acepta el cambio de estado y se guarda la oferta.

Si se devuelve 'false', se revierte el cambio de estado. A continuación, se guardan los demás cambios que pueda tener la oferta.

TipoDocumento puede tomar los valores 'OV' para oferta de ventas y 'OC' para oferta de compras.

IdDoc informa el id de la oferta en cuestión.

AntesDeRechazarOferta

Este método se invoca antes de guardar una oferta que acaba de ser rechazada.

Delphi: function AntesDeRechazarOferta(TipoDocumento: string; IdDoc: Double): Boolean; stdcall;

C#: bool AntesDeRechazarOferta(string tipoDocumento, double idDoc)

Si se devuelve 'true', o no se implementa el evento, se acepta el cambio de estado y se guarda la oferta.

Si se devuelve 'false', se revierte el cambio de estado. A continuación, se guardan los demás cambios que pueda tener la oferta.

Es responsabilidad de la DLL informar al usuario del motivo del rechazo cuando sea necesario o conveniente.

TipoDocumento puede tomar los valores 'OV' para oferta de ventas y 'OC' para oferta de compras.

IdDoc informa el id de la oferta en cuestión.

AntesDePasarAPendienteOferta

Método que se llama antes de guardar una oferta cuando se ha cambiado el estado a pendiente.

Delphi: function ANTESDEACEPTAROFERTA(TipoDocumento: AnsiString; IdDoc: Double): Boolean; stdcall;

C#: bool AntesDePasarAPendienteOferta(string tipoDocumento, double idDoc)

Si se devuelve 'true', o no se implementa el evento, se acepta el cambio de estado y se guarda la oferta.

Si se devuelve 'false', se revierte el cambio de estado. A continuación, se guardan los demás cambios que pueda tener la oferta.

Si se cancela el cambio en una oferta nueva, el valor que tomará es el de pendiente de aceptar.

TipoDocumento puede tomar los valores 'OV' para oferta de ventas y 'OC' para oferta de compras.

IdDoc informa el id de la oferta en cuestión.

DespuesDeAceptarOferta

Método que se llama tras guardar una oferta cuando se ha cambiado el estado a aceptada. Recibe si dicho cambio fue aceptado o no.

Delphi: procedure DESPUESDEACEPTAROFERTA(const TipoDocumento: AnsiString; const IdDoc: Double; const OperationDone: BOOL); stdcall;

C#: void DespuesDeAceptarOferta(string tipoDocumento, double idDoc, bool aceptada)

TipoDocumento puede tomar los valores 'OV' para oferta de ventas y 'OC' para oferta de compras.

IdDoc informa el id de la oferta en cuestión.

OperationDone informa si la oferta fue aceptada o no.

DespuesDeRechazarOferta

Método que se llama tras guardar una oferta cuando se ha cambiado el estado a rechazada. Recibe si dicho cambio fue aceptado o no.

Delphi: procedure DESPUESDEREHAZAROFERTA(const TipoDocumento: AnsiString; const IdDoc: Double; const OperationDone: BOOL); stdcall;

C#: void DespuesDeRechazarOferta(string tipoDocumento, double idDoc, bool rechazada)

TipoDocumento puede tomar los valores 'OV' para oferta de ventas y 'OC' para oferta de compras.

IdDoc informa el id de la oferta en cuestión.

OperationDone informa si la oferta fue aceptada o no.

DespuesDePasarAPendienteOferta

Método que se llama tras guardar una oferta cuando se ha cambiado el estado a pendiente de aceptar. Recibe si fue aceptado dicho cambio o no.

Delphi: procedure DESPUESDEPASARAPENDIENTEOFERTA(const TipoDocumento: AnsiString; const IdDoc: Double; const OperationDone: BOOL); stdcall;

C#: void DespuesDePasarAPendienteOferta(string tipoDocumento, double idDoc, bool pasadaAPendiente)

TipoDocumento puede tomar los valores 'OV' para oferta de ventas y 'OC' para oferta de compras.

IdDoc informa el id de la oferta en cuestión.

OperationDone informa si la oferta fue aceptada o no.

A tener en cuenta...

Este evento puede no generarse si el sistema impide el cambio de estado (por ejemplo, por encontrarse servida).

Expedientes

AntesDeFacturarDocumento

Notificación previa al facturado de un expediente. Permite impedir el facturado.

A tener en cuenta...

Esta notificación no se disparará si se implementa AntesDeFacturarDocumentoV2.

Delphi: function ANTESDEFACTURARDOCUMENTO(Documento: AnsiString; IdDoc: Double; const Cabecera: Variant; const Lineas: Variant; var Motivo: AnsiString): boolean; stdcall;

C#: bool AntesDeFacturarDocumento(string tipoDocumento, float idDocumento, object cabecera, object lineas, ref string motivo)

El parámetro **tipoDocumento** recibiremos "EXPEDIENTE".

En **IdDocumento** dispondremos del identificador del expediente (campo IDEXPE).

En **Cabecera**, el dato de tipo Dataset con el contenido de la cabecera (estructura tabla __CABEEXPE)

En **Lineas**, el dato de tipo Dataset con las líneas del expediente (estructura __LINEEXPE).

En **retorno** deberemos indicar si permitimos la facturación o no con true y false respectivamente. En caso de no permitirlo, en **motivo**, se deberá indicar un texto que se notificará al usuario.

AntesDeFacturarDocumentoV2

Versión extendida del evento anterior. Añade la posibilidad de refrescar el expediente facturado (recargarlo de base de datos) antes de facturar, por si se produce un cambio desde una programación externa de los datos del mismo.

A tener en cuenta...

De presentarse junto con el anterior evento, solo se disparará esta versión.

Delphi: function ANTESDEFACURARDOCUMENTOV2(TipoDocumento: string; IdDocumento: Double; const Cabecera: Variant; const Lineas: Variant; var Motivo: string; var Repintar: Boolean): boolean; stdcall;

C#: bool AntesDeFacturarDocumentoV2(string tipoDocumento, float idDocumento, object cabecera, object lineas, ref string motivo, ref bool repintar)

Aparte de los parámetros del evento anterior, el nuevo parámetro **Repintar** indicará a **a3ERP** si debe recargar el expediente con el valor True o False para continuar sin más.

DespuesDeFacturarDocumento

Notifica la finalización del facturado de un expediente, informando de la factura generada.

Delphi: Procedure DESPUESDEFACURARDOCUMENTO(TipoDocumento: string; IdDocumento, IdFactura: Double); stdcall;

C#: void DespuesDeFacturarDocumento(string tipoDocumento, float idDocumento, float idFactura);

En **TipoDocumento** recibiremos "EXP"

En **IdDocumento** el identificador del expediente facturado.

IdFactura dispondrá el identificador de la factura de venta generada.

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 de **a3ERP**.

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 ANTESDEGUARDAR DATOSEMPRESA(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 DESPUESDEGUARDAR DATOSEMPRESA(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.

Producción

OrdenNuevoComponente

Esta función permite informar los datos de un componente cada vez que añade uno a cualquier fase de la orden de producción. Sustituye al diálogo visual de a3ERP de nuevo componente.

Delphi: function ORDENNUEVOCOMPONENTE(Lineas: Variant; Cabecera: Variant): Variant; stdcall;

C#: object OrdenNuevoComponente(object lineas, object cabecera)

Disponible sólo en **órdenes de producción**.

Devuelve un objeto registro, conteniendo los valores por defecto que se pretendan para un componente nuevo. No es necesario informar todos los campos.

Líneas y Cabecera son objetos de tipo registro.

OrdenNuevoOperario

Esta función permite informar los datos de un operario que cada vez que añade uno a cualquier fase de la orden de producción. Sustituye al diálogo visual de **a3ERP** de nuevo operario.

Delphi: function ORDENNUEVOOPERARIO(Lineas: Variant; Cabecera: Variant): Variant; stdcall;

C#: object OrdenNuevoOperario(object lineas, object cabecera)

Disponible sólo en **órdenes de producción**.

Devuelve un objeto registro, conteniendo los valores por defecto que se pretendan para un operario nuevo. No es necesario informar todos los campos.

Líneas y Cabecera son objetos de tipo registro.

OrdenNuevaMaquina

Esta función permite informar los datos de una máquina cada vez que añade uno a cualquier fase de la orden de producción. Sustituye al diálogo visual de **a3ERP** de nueva máquina.

Delphi: function ORDENNUEVAMAQUINA(Lineas: Variant; Cabecera: Variant): Variant; stdcall;

C#: object OrdenNuevaMaquina(object lineas, object cabecera)

Disponible sólo en **órdenes de producción**.

Devuelve un objeto registro, conteniendo los valores por defecto que se pretendan para una máquina nueva. No es necesario informar todos los campos.

Líneas y Cabecera son objetos de tipo registro.

OrdenNuevoProducto

Esta función permite informar los datos de un producto cada vez que se añade uno nuevo a la orden de producción. Sustituye al diálogo visual de **a3ERP** de nuevo producto.

Delphi: function ORDENNUEVOPRODUCTO(Lineas: Variant; Cabecera: Variant): Variant; stdcall;

C#: object OrdenNuevoProducto(object lineas, object cabecera)

Disponible sólo en **órdenes de producción**.

Devuelve un objeto registro, conteniendo los valores por defecto que se pretendan para el nuevo producto. No es necesario informar todos los campos.

Líneas y Cabecera son objetos de tipo registro

Listados

Los siguientes eventos informan de los listados, con su parametrización, que se lanzan en la aplicación permitiendo interceptar la ejecución de cualquiera de ellos.

EsListadoExterno

Método para indicar si el listado será externo a a3ERP o no, con los valores true o false respectivamente.

Delphi: function ESLISTADOEXTERNO(IdListado: AnsiString): Boolean; stdcall;

C#: bool EsListadoExterno(string idListado)

En caso de indicarse que será externo, el módulo de extensión de a3ERP al que se pregunta deberá encargarse de lanzar un listado (o realizar otras acciones que tome oportunas), como por ejemplo uno usando otra herramienta (ejemplo: cristal reports), cuando se le notifique el evento HacerListado.

En caso contrario, a3ERP lanzará el listado estándar asociado al identificador de listado pasado.

A tener en cuenta...

Puede dispararse varias veces para el mismo listado.

HacerListado

Delphi: Procedure HACERLISTADO(IdListado: String; Parametros: Variant); stdcall;

C#: void HacerListado(string idListado, object parametros)

Método para realizar el listado externo al a3ERP con los parámetros de la pantalla del a3ERP.

- **IdListado:** el identificador del listado, igual que en el objeto 'Listado'
- **Parámetros:** Los parámetros que utiliza el listado es un array tipo Registro. La posición 0 es un entero, que indica cuántos de parámetros. La posición 1 a la Parametros[0], es a su vez un array de dos elementos, en el que el primero es una cadena con el nombre del parámetro y el segundo, otra cadena con el valor del mismo.

AntesDelImprimir

Este método se ejecuta cuando se va a imprimir cualquier listado/impreso estándar de a3ERP.

A tener en cuenta...

No se dispara para listados externos.

Delphi: procedure ANTESDEIMPRIMIR(IdListado: String; Modelo: string; Destino: Integer; Parametros: Variant); stdcall;

C#: void AntesDelImprimir(string idListado, string modelo, int destino, object parametros)

Los parámetros son:

- **IdListado:** el identificador del listado, igual que en el objeto 'Listado'.
- **Modelo:** la definición que el usuario ha escogido. Se compone de dos partes, separadas por el signo igual (=). En primer lugar, se informa el nombre físico de la definición. Después se concatena el nombre que figura en el desplegable en pantalla.

Por ejemplo:

LstCli.000=Cientes con foto.

Si el usuario imprime el listado original, se informa: LstCli.dfm=Original.

- **Destino:** dónde se envía el listado. Puede ser cualquier valor del enumerado 'destino' utilizado en el objeto 'Listado' de a3ActiveX.

destImpresora	0	Impresora
destHTML	1	Fichero, formato HTML
destPDF	2	Fichero, formato PDF
destRTF	3	Fichero, formato RTF
destJPG	4	Fichero, formato JPG
destGIF	5	Fichero, formato GIF
destBMP	6	Fichero, formato BMP
destEMF	7	Fichero, formato EMF
destWMF	8	Fichero, formato WMF
destPantalla	9	Vista previa
destExcel	10	Fichero, formato XLS.
destFacturaE	11	Fichero en formato EFACTURA

- **Parámetros:** Los parámetros que utiliza el listado es un array tipo 'Dataset', igual a la que se usa en eventos de documentos. La posición 0 es un entero, que indica cuántos elementos tipo 'row' vienen

detrás. En este caso, siempre vale 1, todos los parámetros se pasan en una sola 'row'. La posición 1, es a su vez un array.

En este último, la posición 0 es un entero que indica cuántos campos tiene la 'row'.

A partir de la posición 1, hay tantos arrays de dos posiciones como campos haya. La primera posición es el nombre del parámetro, la segunda el valor.

DespuesDelImprimir

Este método se ejecuta cuando se ha impreso o guardado cualquier listado/impreso.

A tener en cuenta...

Se dispara para listados externos.

Delphi: procedure DESPUEDEIMPRIMIR(IdListado: String; Modelo: string; Destino: Integer; Parametros: Variant); stdcall;

C#: void DespuesDelImprimir(string idListado, string modelo, int destino, object parametros)

Los parámetros son:

- **IdListado:** el identificador del listado, igual que en el objeto 'Listado'.
- **Modelo:** la definición que el usuario ha escogido. Se compone de dos partes, separadas por el signo igual (=). En primer lugar, se informa el nombre físico de la definición. Después se concatena el nombre que figura en el desplegable en pantalla.

Por ejemplo:

LstCli.000=Cientes con foto.

Si el usuario imprime el listado original, se informa: LstCli.dfm=Original.

- **Destino:** dónde se envía el listado. Puede ser cualquier valor del enumerado 'destino' utilizado en el objeto 'Listado' de a3ActiveX.

destImpresora	0	Impresora
destHTML	1	Fichero, formato HTML
destPDF	2	Fichero, formato PDF
destRTF	3	Fichero, formato RTF
destJPG	4	Fichero, formato JPG
destGIF	5	Fichero, formato GIF
destBMP	6	Fichero, formato BMP
destEMF	7	Fichero, formato EMF
destWMF	8	Fichero, formato WMF

destPantalla	9	Vista previa
destExcel	10	Fichero, formato XLS.
destFacturaE	11	Fichero en formato EFACTURA

- **Parametros:** Los parámetros que utiliza el listado es un array tipo 'Dataset', igual a la que se usa en eventos de documentos. La posición 0 es un entero, que indica cuántos elementos tipo 'row' vienen detrás. En este caso, siempre vale 1, todos los parámetros se pasan en una sola 'row'. La posición 1, es a su vez un array.

En este último, la posición 0 es un entero que indica cuántos campos tiene la 'row'.

A partir de la posición 1, hay tantos arrays de dos posiciones como campos haya. La primera posición es el nombre del parámetro, la segunda el valor.

ConsiderarImpresionComoCopia

Este método se ejecuta cuando se va a generar cualquier listado/impreso estándar de a3ERP, para determinar si debe llevar la marca de copia devolviendo los valores true (el listado se considerará copia) o false.

A tener en cuenta...

No se dispara para listados externos.

Delphi: function CONSIDERARIMPRESIONCOMOCOPIA(const IdListado: AnsiString; const Modelo: string; const Destino: Integer; Parametros: Variant): Boolean; stdcall;

C#: bool ConsiderarImpresionComoCopia(string idListado, string modelo, int destino, object parametros)

Los parámetros son:

- **IdListado:** el identificador del listado, igual que en el objeto 'Listado'.
- **Modelo:** la definición que el usuario ha escogido. Se compone de dos partes, separadas por el signo igual (=). En primer lugar, se informa el nombre físico de la definición. Después se concatena el nombre que figura en el desplegable en pantalla.

Por ejemplo:

LstCli.000=Cientes con foto.

Si el usuario imprime el listado original, se informa: LstCli.dfm=Original.

- **Destino:** dónde se envía el listado. Puede ser cualquier valor del enumerado 'destino' utilizado en el objeto 'Listado' de a3ActiveX.

destImpresora	0	Impresora
destHTML	1	Fichero, formato HTML

destPDF	2	Fichero, formato PDF
destRTF	3	Fichero, formato RTF
destJPG	4	Fichero, formato JPG
destGIF	5	Fichero, formato GIF
destBMP	6	Fichero, formato BMP
destEMF	7	Fichero, formato EMF
destWMF	8	Fichero, formato WMF
destPantalla	9	Vista previa
destExcel	10	Fichero, formato XLS.
destFacturaE	11	Fichero en formato EFACTURA

- **Parametros:** Los parámetros que utiliza el listado es un array tipo 'Dataset', igual a la que se usa en eventos de documentos. La posición 0 es un entero, que indica cuántos elementos tipo 'row' vienen detrás. En este caso, siempre vale 1, todos los parámetros se pasan en una sola 'row'. La posición 1, es a su vez un array.

En este último, la posición 0 es un entero que indica cuántos campos tiene la 'row'.

A partir de la posición 1, hay tantos arrays de dos posiciones como campos haya. La primera posición es el nombre del parámetro, la segunda el valor.

Cartera

Los eventos que se producen al modificar la cartera de la aplicación se detallan a continuación.

AntesDeGuardarEfecto

Evento que se produce inmediatamente antes de realizar una operación sobre la cartera de la empresa. Permite impedir continuar la operación.

Delphi: function ANTESDEGUARDAREFECTO(const Operacion: string; Datos: variant): Boolean; stdcall;

C#: bool AntesDeGuardarEfecto(string operacion, object datos)

- **Operación** se refiere a la operación que se está realizando sobre la cartera, pudiendo ser alguno de los siguientes valores:

MO – Alta / Modificar

BO – Borrar

CO – Cobro

ACO - Anulación de cobro

PA – Pago

APA - Anulación de pago

RE – Recibir

ARE – Anular recepción
ACT – Actualizar riesgo
AACT – Anular riesgo
IMP – Imputar gastos
AIMP – Anular imputación de gastos
DEV - Devolución
ADEV – Anular devolución
ENV – Enviar
AENV – Anular enviar
BLO – Bloquear
ABLO – Anular bloqueo
MANT – Anticipo
BANT – Anular anticipo
DCE – Dudoso cobro
ADE – Anular dudoso cobro
IME – Impagar
AIE – Anular impagar
ADDAG - Añadir efecto a agrupación
MAG – Nueva agrupación
BAG – Borrar agrupación

- **Datos** es un Registro con los datos del efecto (tabla cartera).
- El **resultado** booleano determina si se continuará con la operación (true) o no (false).

DespuesDeGuardarEfecto

Evento que se produce tras realizar algún cambio sobre la cartera de la empresa.

Delphi: procedure DESPUESDEGUARDAREFECTO(const Operacion: string; const NumCartera: Double; const NumVen: integer); stdcall;

C#: void DespuesDeGuardarEfecto(string operacion, float numCartera, int numVen)

En **Operación** recibiremos el tipo de cambio (véase AntesDeGuardarEfecto).

En **NumCartera** el número de cartera del efecto sobre el que se ha trabajado.

En **NumVen** el número de vencimiento del efecto de la cartera afectado.

AntesDeGuardarRemesa

Notificación previa al registro en b.d. de alta/cambios/borrado en una remesa. Permite impedir que se continúe con la operación.

Delphi: function ANTESDEGUARDARREMESA (const Operacion, Tipo: string; Cabecera, Lineas: variant): boolean; stdcall;

C#: bool AntesDeGuardarRemesa(string operacion, string tipo, object cabecera, object lineas)

- **Operación** recibe el tipo de modificación sobre una remesa.

MOD – Alta / Modificación de una remesa

BOR – Borrado de una remesa

CO – Cobro

COD – Cobro al descuento

ACO - Anulación de cobro

PA – Pago

APA - Anulación pago

- **Tipo** contiene C para remesas de cobro y P para remesas de pagos.
- **Cabecera** es un Registro con los datos de la cabecera de la remesa (registro de tabla Cartera).
- **Lineas** es un Dataset con los datos de los efectos (registros de tabla Cartera subordinados) que componen la remesa.
- El valor de **retorno** booleano True permite continuar con la operación, mientras que False lo impide.

DespuesDeGuardarRemesa

Notifica el registro en b.d. de alta/cambios/borrado en una remesa.

Delphi: function DESPUESDEGUARDARREMESA (const Operacion, Tipo: string; const IdRemesa: Double); stdcall;

C#: void DespuesDeGuardarRemesa(string operacion, string tipo, float idRemesa)

- **Operación** recibe el tipo de cambio realizado (véase AntesDeGuardarRemesa).
- **Tipo** contiene C para remesas de cobro y P para remesas de pagos.
- **IdRemesa** contendrá el identificador asignado a la remesa.

EsConfirmingExterno

Método en el que la DLL informa a a3ERP si el Confirming se gestiona o no desde la DLL. Actualmente a3ERP, configura los confirmings de CaixaBank, BBVA y Bankinter, el resto de confirmings deben gestionarse mediante el método HacerConfirmingExterno (detallado en este documento)

Delphi: function EsConfirmingExterno (codban, entidadBancaria: string):boolean; stdcall;

C#: bool EsConfirmingExterno (string codban, string entidadBancaria)

- **CodBan** recibe la cuenta bancaria del banco seleccionado para el confirming.
- **entidadBancaria** código de la entidad bancaria (los 4 dígitos siguientes al IBAN).

HacerConfirmingExterno

Método en el que la DLL se encarga de realizar Confirming no implementado en a3ERP (distinto a CaixaBank, BBVA y Bankinter). Solo se llama a este procedimiento cuando

Delphi: function HacerConfirmingExterno (IdRemesa: double); stdcall;

C#: void HacerConfirmingExterno (Double IdRemesa)

- **IdRemesa** recibe el IdRemesa de la remesa de la que se va a realizar el confirming.

Asientos contables

Detallaremos a continuación las notificaciones de cambios contables en la aplicación.

AntesDeGuardarApunte

Durante la edición de un asiento, notifica el añadido o modificación (no el borrado) de una de sus líneas, permitiendo impedir la aplicación de los cambios en memoria.

Delphi: function ANTESDEGUARDARAPUNTE (Apunte: variant): boolean; stdcall;

C#: bool AntesDeGuardarApunte(object apunte)

Con el parámetro **Apunte** (tipo Dataset) recibiremos el registro de la línea que se está añadiendo o modificando, con la estructura de un registro de la tabla __ASIENTOS.

El valor booleano de **retorno** permite continuar o impedir los cambios (valores true y false respectivamente).

A tener en cuenta...

Puede llamarse varias veces para el mismo registro.

AntesDeGuardarAsiento

Realiza una notificación del guardado o borrado del asiento previo a su almacenamiento en base de datos, permitiendo cancelar la operación.

Delphi: function ANTESDEGUARDARASIENTO(IdAsiento: Double; Asiento: variant): boolean; stdcall;

C#: bool AntesDeGuardarAsiento(float idAsiento, object asiento)

El parámetro **IdAsiento** contendrá el identificador interno del asiento (NumApunte) que se está guardando o borrando.

En **Asiento** (tipo Dataset) todos los registros (apuntes) del asiento que se está guardando (tabla __ASIENTOS). Si es un borrado, vendrá vacío (es decir, un dataset con cero registros).

El valor de **retorno** permitirá la operación o no (valores true o false respectivamente).

DespuesDeGuardarAsiento

Notifica que el alta, los cambios o el borrado realizados en un asiento ya se encuentran en base de datos.

Delphi: procedure DESPUESDEGUARDARASIENTO(IdAsiento: Double; Asiento: variant); stdcall;

C#: void DespuesDeGuardarAsiento(float idAsiento, object asiento)

El parámetro **IdAsiento** contendrá el número de asiento (NumAsiento) del asiento que se ha guardado.

En **Asiento** (tipo Dataset) todos los registros (apuntes) del asiento que se ha guardado (tabla __ASIENTOS). Si es un borrado, vendrá vacío (un dataset con cero registros).

Formularios y ventanas

Las siguientes notificaciones se producen al presentar, reactivar o cerrar pantallas de la aplicación.

DespuesDeNuevoFormulario

Avisa de la aparición de una nueva pantalla en la aplicación.

Delphi: procedure DESPUESDENUEVOFORMULARIO(ClaseFormulario: AnsiString; Handleformulario: HWND); stdcall;

C#: void DespuesDeNuevoFormulario(string claseFormulario, int handleFormulario)

En el parámetro **ClaseFormulario** obtendremos una cadena que informará del tipo de formulario que se va a mostrar.

Con **HandleFormulario** recibiremos el handle de windows (identificador único) de la nueva ventana. Este handle se puede usar con las funciones de la API de windows, pero bajo la responsabilidad del autor del desarrollo ya que puede afectar al comportamiento de la nueva ventana y la aplicación, como podría ser interceptando eventos, incluyendo sus propios controles o modificando los existentes.

AntesDeDestruirFormulario

Avisa del cierre de una pantalla de la aplicación.

Delphi: procedure ANTESDEDESTRUIRFORMULARIO(ClaseFormulario: AnsiString; Handleformulario: HWND); stdcall;

C#: void AntesDeDestruirFormulario(string claseFormulario, int handleFormulario)

En el parámetro **ClaseFormulario** obtendremos una cadena que informará del tipo de formulario que se va a cerrar.

Con **HandleFormulario** recibiremos el handle de windows (identificador único) de la nueva ventana. Este handle se puede usar con las funciones de la API de windows.

DespuesDeActivarFormulario

Notifica la reactivación de una pantalla no cerrada de la aplicación que previamente había pasado a segundo plano.

Delphi: procedure DESPUESDEACTIVARFORMULARIO(ClaseFormulario: AnsiString; Handleformulario: HWND); stdcall;

C#: void DespuesDeActivarFormulario(string claseFormulario, int handleFormulario)

En el parámetro **ClaseFormulario** obtendremos una cadena que informará del tipo de formulario que se va a cerrar.

Con **HandleFormulario** recibiremos el handle de windows (identificador único) de la nueva ventana. Este handle se puede usar con las funciones de la API de windows, pero bajo la responsabilidad del autor del desarrollo ya que puede afectar al comportamiento de la nueva ventana y la aplicación.

AntesDeMensaje

Notifica la visualización de un mensaje de aviso desde ERP. Desde ERP se puede controlar la visualización de mensajes utilizando el parámetro/variable **OmitirMensajes**. Recibiremos esta notificación antes de que un aviso se muestre y con el parámetro **OmitirMensaje** de este procedimiento, el autor del desarrollo podrá decidir si finalmente el mensaje se muestra por pantalla o no.

Delphi: procedure ANTESDEMENSAJE (const Msg: AnsiString; const TipoMensaje: int; var OmitirMensaje: boolean); stdcall;

C#: void AntesDeMensaje(string Msg, int TipoMensaje, ref bool OmitirMensaje)

La descripción de los parámetros del procedimiento es la siguiente:

En **Msg** se recibe el texto completo del mensaje.

En **TipoMensaje**, se recibe un entero que identifica el tipo de mensaje que se recibe. Los valores posibles para este parámetro son los siguientes:

- **Warning:** TipoDeMensaje = 0
- **Error:** TipoDeMensaje = 1
- **Information:** TipoDeMensaje = 2
- **Confirmation:** TipoDeMensaje = 3
- **Custom:** TipoDeMensaje = 4

El parámetro **OmitirMensaje** permite al autor de desarrollo decidir si el mensaje finalmente se muestra o no en la aplicación.

SkinAplicado

Notifica el cambio del skin activo en ERP. Este evento se lanza al abrir la aplicación para notificar el Skin activo y cuando se modifica el skin activo desde dentro de ERP.

Delphi: procedure SKINAPLICADO (const NombreSkin: AnsiString; const NombrePaleta: AnsiString; EstiloNativo: Boolean); stdcall;

C#: void AntesDeMensaje(string NombreSkin, string NombrePaleta, bool EstiloNativo)

La descripción de los parámetros del procedimiento es la siguiente:

El parámetro **NombreSkin** indica el nombre del skin aplicado en ERP. La aplicación utiliza componentes de DevExpress (<https://www.devexpress.com/products/vcl/>) para la gestión de Skins, así que este nombre corresponde al skin dentro de los que utiliza esta biblioteca de componentes.

Algunos skins, permiten diferentes variantes de colores dentro del propio estilo del skin. Esas variantes se definen con el parámetro **NombrePaleta**, que indica el nombre de la paleta de colores que se aplica a un determinado skin. Si el skin no permite variantes, se devuelve el valor “Default”.

El parámetro **EstiloNativo** define si la librería de DevExpress utiliza el estilo nativo de Windows para dibujar determinados controles.

Comisiones

Los eventos de comisiones notifican del procesado del listado de comisiones de la aplicación (pestaña detalle).

AntesDeProcesarComisiones

Avisa del inicio de un proceso de liquidación de comisiones, permitiendo modificar los valores inicialmente calculados por la aplicación.

Delphi: procedure ANTESDEPROCESARCOMISIONES (var Datos: Variant); stdcall;

C#: void AntesDeProcesarComisiones(object datos)

Los datos de la liquidación de comisiones calculada inicialmente por la aplicación se proveen en la estructura Datos, del tipo Dataset que se recuperan de la tabla FactComis (Totales de comisiones de las facturas de venta).

El contenido:

Idfacv, Sumanum, Fecha, NumDoc (Tipo/Serie/NumDoc), Base, Margen, TotDoc, NumCartera de la factura

Codigo, Nombre, del cliente de la factura,

Codrep, NomRep, FactCobr, Devueltos, Atrasados, diasatraso, DudosoCobro del representante que comisiona de dicha factura

IdFactComis (Id), Comtotal (TotComis), Comabonadas (TotPagComis), comadescontar, comapagar, totcomapagar de FactComis

(no necesariamente en ese orden).

DespuesDeProcesarComisiones

Notifica que ha terminado el proceso de liquidación de comisiones con el id que se provee en la llamada.

Delphi: procedure DESPUESDEPROCESARCOMISIONES (IdLiqCom: integer); stdcall

C#: void DespuesDeProcesarComisiones(int idLiqCom)

En el único parámetro que recibimos, **IdLiqCom**, obtendremos el Id de la liquidación de comisiones (tabla __LIQUIDACIONCOMISIONES) que se inició con la llamada anterior AntesDeProcesarComisiones.

Remesas

AntesDeGuardarConfirming

Notifica la generación de un fichero de confirming de pagos SEPA, permitiendo modificar su contenido o evitar su guardado.

Delphi: function ANTESDEGUARDARCONFIRMING (IdRemesa: Double; var DataConfirming: string): Boolean; stdcall;

C#: bool AntesDeGuardarConfirming (double idRemesa, ref string dataConfirming)

El parámetro **IdRemesa** contendrá el identificador (tabla __REMESAS) de la remesa de pagos generada.

DataConfirming dispondrá del contenido del fichero antes de realizar su guardado en disco. Permite realizar modificaciones en el contenido del fichero, siendo el valor que contenga al finalizar el evento el que se guarde.

El resultado del evento (valor booleano True) permite continuar con el guardado del texto almacenado en DataConfirming o, por el contrario (valor booleano False), cancelará la operación.

EsConfirmingExterno (Ver. 13.3 o superior)

Sirve para indicar a a3ERP que el módulo implementa el confirming de una determinada entidad bancaria.

Delphi: function ESCONFIRMINGEXTERNO (const CodBanco, Entidad: AnsiString): Boolean; stdcall;

C#: bool EsConfirmingExterno (string codBanco, string entidad)

El parámetro **CodBanco** contendrá el código del banco de la remesa.

Entidad será el código de entidad (4 primeros dígitos de una cuenta bancaria española).

El resultado del evento (valor booleano True) le indicará a a3ERP que el módulo implementa (mediante la llamada HacerConfirmingExterno) el confirming para la entidad.

HacerConfirmingExterno (Ver. 13.3 o superior)

A3ERP llamará a este procedimiento del módulo cuando necesite generar un confirming para la entidad sobre la que le interrogó mediante la llamada EsConfirmingExterno.

Delphi: function HACERCONFIRMINGEXTERNO (const IdRemesa: Double); stdcall;

C#: void HacerConfirmingExterno (double idRemesa)

El parámetro **IdRemesa** contendrá el el identificador (__REMESAS.IDREMESA) de la remesa.

Desplegando la DLL en la oficina del cliente

A continuación, te indicamos los pasos:

1. En el servidor, copia la dll en la carpeta 'Binarios' de la aplicación. Recuerda la estructura de carpetas: "Fabricante\ Producto\ Binarios".
2. Abre al directorio C:\ Program Files\ A3\ ERP\ Sistema.Custom\Sistema.
3. Copia archivo regdlls.ini y pega en carpeta donde está ubicada su dll.
4. Edita este nuevo el fichero regdlls.ini.
5. Aparecerá contenido como este:



6. En este ejemplo estamos registrando una dll que se llama: ejemplo_dll_con_framework4.dll y que está ensamblada para trabajar con el Framenwork 4.0.
7. Si la DLL utiliza otro framework, deberás incluir la línea debajo del REGASM que corresponda.
8. Una vez añadida la línea, guarda el archivo.
9. Después, pasa una vista SQL desde el Managment Studio, insertando la DLL. Esto se debe hacer para cada una de las bases de datos que se quiera utilizar esta DLL.

Los valores de la tabla DLL son:

- **DLL:** Si trabajas con Delphi, C++, etc. debes escribir el nombre de la dll. Ejemplo: dll.dll.
Si es un Active hecha en .NET como C# o Vd.NET, se debe poner en formato NAMESPACE.NOMBRECLASE (generalmente el namespace coincide con el nombre de la DLL).

A tener en cuenta...

En caso de duda, consulta la documentación de tu plataforma de desarrollo.

- **Fabricante:** Nombre carpeta creada en Extensiones \ Fabricante
- **IdDll:** Identificador numérico de la DLL
- **Producto:** Nombre de la carpeta creada por usted en \Extensiones\ Empresa\Producto

```
Insert into dlls(dll, fabricante, iddll, producto)
Values('ejemplo_dll_con_framework4.dll', 'fabricante', '2', 'ejemplo_dll_con_framework4')
```

Después, debes ejecutar el asistente del servidor, así los clientes se actualizarán automáticamente y registrará las DLL modificadas o nuevas.

Fichero LOG para las DLL

Otro de los mecanismos que se han implementado para ayudar en la depuración y control de errores en las DLL es la posibilidad de generar un log de todas las llamadas a los eventos.

Este control permite saber a qué métodos de DLL se ha invocado por fecha y hora, así como conocer si se ha entrado y salido del evento o por el contrario se ha generado una excepción durante su ejecución.

Para ello, es necesario **añadir las siguientes entradas en el fichero CONFIGURACION.INI ubicado en %APPDATA%\A3:**

[DLLS]

Log=1

Con esto conseguimos que automáticamente **a3ERP** genere ficheros de LOG en la siguiente ruta:

%APPDATA%\A3\LOGS\DLLS

Control de excepciones procedentes de las DLLS

Por definición, una dll jamás deberá generar una excepción. En cada caso, los distintos manejadores de eventos programados deberán realizar la operación oportuna ante un error en su código (como puede ser devolver un parámetro que impida continuar, si está disponible para el evento en cuestión).

De todas formas, A3ERP implementa dentro de lo que es la gestión de dlls un mecanismo para informar al usuario de "Quién" y "Que" ha generado una excepción. Este se basa en tratar de interceptar la excepción (dependerá del lenguaje en el que esté implementada la dll) y no tiene porque funcionar.

En caso de sospecharse un error en el tratamiento de un evento, es recomendable activar el log de dlls comentado en el punto anterior.

Cómo llamar a una DLL desde MIMENU.MENU

Se pueden añadir opciones de menú que llamen a una DLL. Debemos diferenciar entre dos tipos de DLL's:

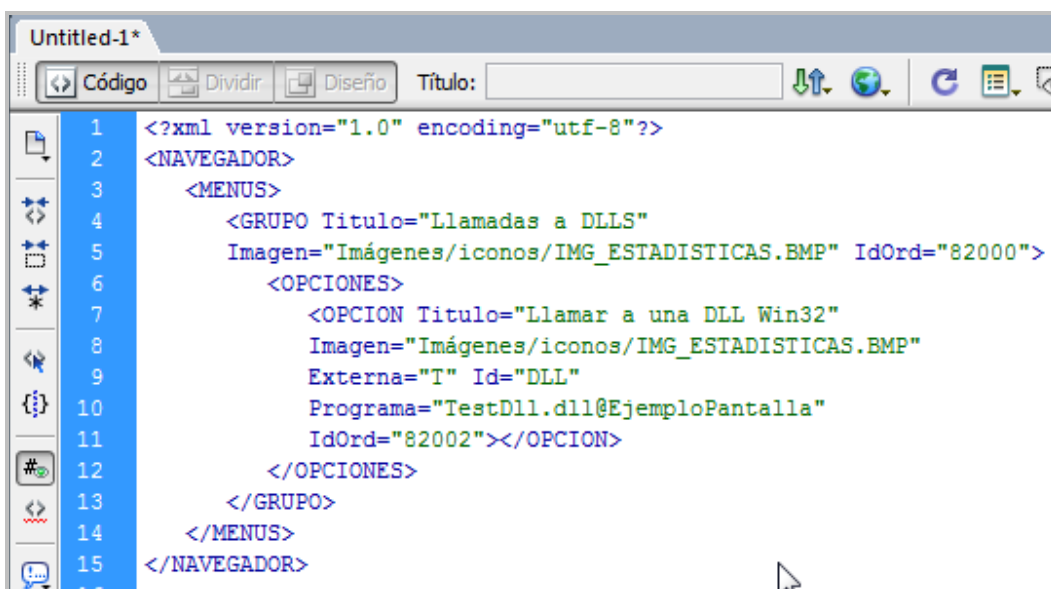
- DLL's win32
- DLL's ActiveX

Llamando a una DLL WIN32

Lo primero sería generar nuestro propio menú con la opción que queremos añadir.

Consultar el capítulo **Personalizar Menús**

Este es un ejemplo típico:



```
1 <?xml version="1.0" encoding="utf-8"?>
2 <NAVEGADOR>
3   <MENUS>
4     <GRUPO Titulo="Llamadas a DLLS"
5       Imagen="Imágenes/iconos/IMG_ESTADISTICAS.BMP" IdOrd="82000">
6       <OPCIONES>
7         <OPCION Titulo="Llamar a una DLL Win32"
8           Imagen="Imágenes/iconos/IMG_ESTADISTICAS.BMP"
9           Externa="T" Id="DLL"
10          Programa="TestDll.dll@EjemploPantalla"
11          IdOrd="82002"></OPCION>
12       </OPCIONES>
13     </GRUPO>
14   </MENUS>
15 </NAVEGADOR>
```

Como se ve, en la propiedad PROGRAMA debe ir indicada tanto la DLL, como el método publicado al que queremos que llame esa opción.

En el ejemplo, la DLL se llama TestDLL y el método al que llamamos es EjemploPantalla. Hay que notar que para separar la DLL del método usaremos el símbolo arroba (@).

Para nuestro ejemplo copiamos la DLL en la misma ruta donde esté el a3ERP.exe y arrancamos a3ERP. Nos aparecerá la nueva categoría “Llamadas a DLL’s”, que contiene una opción que se llama “Llamar a una DLL win32”.

A partir de la **versión 14.03** es obligatorio registrar en la tabla DLL’s de a3ERP, las dlls que se quieran invocar. Para más información de cómo se registran dlls en a3erp consultar el capítulo “Registro en ERP”.

Llamando a una DLL ACTIVEX

Como en el caso de las DLL’s de tipo WIN32, hay que generar nuestro propio menú (o usar uno ya creado).

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <NAVEGADOR>
3   <MENUS>
4     <GRUPO Titulo="Llamadas a DLLS" Imagen="Imágenes/iconos/IMG_
5       <OPCIONES>
6         <OPCION Titulo="Llamar a una DLL Win32"
7           Imagen="Imágenes/iconos/IMG_ESTADISTICAS.BMP"
8           Externa="T" Id="DLL"
9           Programa="@Nexus.AddIn.Nexus@EjemploPantalla"
10          IdOrd="82002"></OPCION>
11        </OPCIONES>
12      </GRUPO>
13    </MENUS>
14  </NAVEGADOR>
15

```

En este caso, lo único que cambia es que, en la propiedad PROGRAMA, se debe estructurar la llamada de la siguiente manera: **@ClaseDelActiveX@ProcedimientoQueSePasaraComoParametro**

En el caso de los ActiveX es necesario saber que a3ERP siempre invocará el método “OPCION” (que debemos haber implementado y exportado nosotros en nuestro ActiveX) y que somos nosotros quienes, mediante comparaciones, deberemos llamar a un método u otro, según el parámetro que nos llegue.

Ejemplo en C# de este método

```

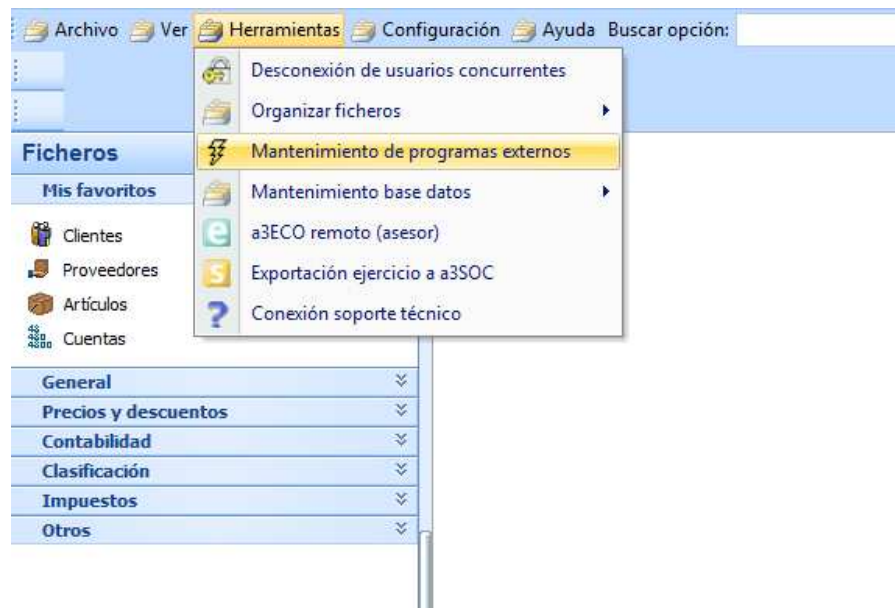
//Método para llamar a otras opciones desde el menú o desde programas externos
//En caso de llamarse la opción desde programas externos se rellenará el segundo parámetro
//con la clave del mantenimiento asociado
public void Opcion(string IdOpcion, string parámetro) {
    if (IdOpcion.ToUpper() == " EJEMPLOPANTALLA ")
        MessageBox.Show("se ha llamado a la pantalla");
    else if (IdOpcion.ToUpper() == "MICLEIENTE")
        MessageBox.Show(parámetro);
}

```


Otra cosa más a tener en cuenta es que, en este caso, la DLL ha de estar registrada con **regasm**, en el caso de ser un ActiveX que haya sido desarrollado usando **.NET**, o con **regsvr32** en el caso de estar desarrollada con **Delphi** o **VB 6**.

Parámetros en programas externos

Seleccionar **“Configuración/ Datos Generales/ Preferencias”** y activar la opción de **“Programas externos”**. En ese momento, aparecerá una nueva opción en el menú **“Herramientas/ Mantenimiento de programas externos”**.



En el caso de que tengamos activada la opción de **Programas Externos** desde **Datos Generales**, será en el **“Mantenimiento de programas externos”** donde definiremos qué parámetros se van a pasar.

Veamos un ejemplo:

Tabla: Artículos				
Programas:	Orden	Programa o librería	Descripción	Procedimiento de la DLL
*	1	W:\MI PROGRAMA\MIPROGRAMA.EXE :usuario :password :empresa :carpeta :tipocontable :defecto		

Si nos fijamos, se ha puesto el ejecutable y los parámetros en la misma línea, separados por un espacio. En los programas externos es donde tiene sentido el parámetro: DEFECTO. ¿Por qué?

Si nosotros definimos en **Artículos** una llamada a un ejecutable y no indicamos ningún parámetro **a3ERP**, por defecto, pasará el código del maestro (codcli, en el caso de artículos). Pero cuando pasamos un parámetro al ejecutable, **a3ERP** no sabe en qué posición queremos que pase el valor del parámetro del código del maestro por lo que, en este caso, somos nosotros los encargados de indicarle, en primer lugar, si queremos que se pase y, en segundo lugar, en qué posición.

Por lo tanto, si añadimos un parámetro y no ponemos el :DEFECTO, a3ERP dejará de pasar el código del maestro.

Significado de cada parámetro

- **DEFECTO:** es el valor del maestro que se pasa de forma automática cuando no hay parámetros.
- **TIPOCONTABLE:** es el tipo contable actual.
- **CARPETA:** contiene la ruta donde está instalado a3ERP.
- **EMPRESA:** contiene el nombre de la empresa desde la que se llama.
- **USUARIO:** contiene el usuario logado que llama al programa externo.
- **PASSWORD:** contiene el password ENCRIPTADO del usuario logado.
- **USUARIOANALYTICS:** contiene el valor de la propiedad "Perfil" en la opción de seguridad para las opciones de A3ERP Analytics.

Lectura recomendada

1. [Introducción a DLLs y eventos disponibles](#)
2. [Nuestra primera DLL win32 nativa o .net](#)
3. [Haciendo que a3ERP conozca nuestra dll](#)
4. [Tipos de datos en Delphi y C#. Dlls, tipos "Registro" y "Conjunto de datos"](#)
5. [Diferencias y aclaraciones entre a3erp v7 y v8](#)
6. [Documentos y refresco de la pantalla](#)
7. [Disparando eventos de documentos también cuando se sirve](#)
8. [Eventos de impresión](#)
9. [Ayuda para tratar con las Cabeceras y líneas \(Variant\) desde Delphi](#)
10. [Menús](#)



Soluciones integrales de
gestión para Despachos
Profesionales y
Empresas

902 330 083 tel
www.wolterskluwer.es

