LIBRA

EVENT DRIVEN ENTERPRISE

DESARROLLO LIBRA 6.5.0

Fecha: 12/01/2026

Introducción

Los programas de Libra están orientados hacia el uso de pestañas, incluso en programas muy sencillos se usa por lo menos una pestaña con el nombre del programa.

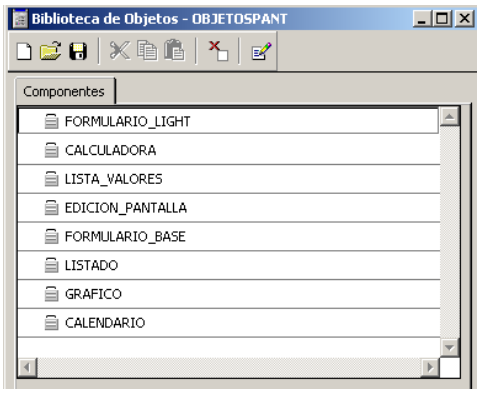
Grupo de Objetos

La principal ventaja de los grupos de objetos radica en que permite manipular con más facilidad varios componentes de un formulario como si se trataran de uno sólo. También permite de forma fácil reutilizar componentes en varios formularios, pero la principal ventaja es que si se añade un componente al grupo de objetos todos los formularios que hayan derivado una subclase (no copiado) de ese grupo de objetos, incluirán automáticamente dicho objeto componente cuando se vuelvan a compilar o a abrir en Forms.

Todos los componentes prefijados para ser usados en libra se almacenan en una librería de objetos llamada OBJETOSPANT.OLB.

Los componentes que incorpora la librería de objetos son:

- FORMULARIO_BASE: Incorpora todos los elementos básicos para desarrollar un programa.
- FORMULARIO_BASICO: Igual que FORMULARIO_BASE, pero sin ningún lienzo y ventana. Recomendado en programas que funcionen como plug-in.
- EDICIÓN_PANTALLA: Incorpora los elementos necesarios para permitir hacer dinámico el formulario.
- LISTA_VALORES: Elementos para mostrar listas de valores.
- LISTA_VALORES_GRUPO: Elementos para listas de valores por grupos.
- LISTADO: Incorpora lo necesario para realizar la llamada a un informe.
- GRAFICO. Incorpora lo necesario para realizar la llamada a un gráfico.
- CALENDARIO: Elementos para construir el calendario.
- CALCULADORA: Elementos para construir la calculadora. **Es importante que todos los programas contengan este grupo de objeto.**
- INSERTAR_IMAGEN: Obsoleto – NO USAR.
- FORMULARIO_LIGHT: Obsoleto – NO USAR.



Para incorporarlos en el programa que se está desarrollando simplemente abrimos la librería de objetos y arrastramos el componente a nuestro formulario y seleccionamos que **deseamos incorporarlo como una Subclase**, esto es muy importante para que futuros cambios de funcionalidad o estética no requieran modificar el programa y que se incorporen simplemente recompilando el programa.

Nombre de los bloques

Al arrancar el programa irá siempre al primer bloque navegable, por tanto, se puede usar cualquier nombre para los bloques, por ejemplo, los nombres de las tablas o abreviaturas.

Lienzo base

El lienzo base se llama CANVAS_TAB y tiene una pestaña llamada TAB0. Para añadir una nueva pestaña es tan simple como situarse sobre TAB0 y pulsar añadir.

Listados

Para poder usar listados desde un programa deberemos de cargar el componente LISTADOS de la librería de objetos "objetospant.olb"

Nota: No hace falta habilitar el botón de imprimir, ya que al inicializarse el programa se detecta si tiene un bloque BREPORT y en ese caso lo activa automáticamente.

Es recomendable, en la medida que sea posible, indicar el informe a ejecutar en el mantenimiento de programas en la pestaña "Informes", en vez de ponerlo de forma fija en el fuente del programa.

Mensajes

Todos los mensajes tienen que ser traducibles, para ello, para mostrar un mensaje al usuario siempre usaremos el paquete MSG de PKLIBPNT.PLL que contiene las siguientes funciones y procedimientos:

- **MENSAJE**: Muestra el mensaje y para la ejecución del código.
- **ALERTA**: Muestra el mensaje, pero no para la ejecución del código. Esta función devuelve el código del botón que pulsó el usuario.
- **MENSAJE_PERSONAL**. Igual que MENSAJE, pero le podemos añadir un texto personalizado. Trataremos de evitar su uso, ya que el texto personalizado no se puede traducir.
- **ALERTA_PERSONAL**: Igual que ALERTA, pero le podemos añadir un texto personalizado. Al igual que MENSAJE_PERSONAL, trataremos de evitar su uso, ya que el texto personalizado no se puede traducir.
- **REPLACE_TEXTO**: Se pasan como parámetros dos cadenas. El cometido de esta función es hacer que en el próximo mensaje o alerta que salte, sustituir la cadena1 por la cadena2, de esta forma si creamos el mensaje "Se han generado <> facturas" y hacemos la llamada MSG.REPLACE_TEXTO('<>', 10), al llamar a MSG.ALERTA el mensaje que se muestra al usuario es: "Se han generado 10 facturas".

Cuando usamos la función ALERTA o ALERTA_PERSONAL para comprobar cual de los botones ha pulsado el usuario se comparará con las siguientes constantes, ALERT_BUTTON1, ALERT_BUTTON2, ALERT_BUTTON3 para controlar si el usuario ha pulsado el botón 1, 2 ó 3 respectivamente.

IMPORTANTE: Las llamadas a MSG.ALERTA y MSG.ALERTA_PERSONAL ejecutan un RAISE Form_Trigger_Failure; por lo que si son llamados en un bloque PL/SQL que captura una excepción WHEN OTHERS hará que salte al código que maneja esa excepción.	**Ejemplo**:
	`DECLARE`
	`  v_boton_pulsado  NUMBER;`
	`BEGIN`
	`  v_boton_pulsado = MSG.ALERTA('ALERT', 'SALIR');`
	
	`  IF v_boton_pulsado = ALERT_BUTTON1 THEN`
	`  -- Ha pulsado el botón 1`
	`  ELSIF v_boton_pulsado = ALERT_BUTTON2 THEN`
	`  -- Ha pulsado el botón 2`
	`  ELSIF v_boton_pulsado = ALERT_BUTTON3 THEN`
	`  -- Ha pulsado el botón 3`
	`  END IF;`
	`END;`

Información detallada

Cuando se muestra un mensaje con 1 botón, puede recoger información detallada que se hubiese puesto en cola desde procedimientos almacenados en base de datos o en programas de Forms, de forma que, en el caso de haber información detallada en la cola, se añade un nuevo botón "Detalle" para que el usuario pueda visualizarla.

Se puede poner en la cola de ampliación códigos de mensaje definidos en la tabla MENSAJES, esta es la opción recomendada, ya que al estar codificado en MENSAJES se traduce al idioma del usuario:

```
PKPANTALLAS.TRAZA(<tipo mensaje>, <codigo_mensaje>, <texto adicional del mensaje', 'MSG');
```

Si el mensaje tiene parte variables, se pueden reemplazar ejecutando `PKPANTALLAS.SET_MSG_REPLACE_TEXTO(cadena a reemplazar, valor a reemplazar)` antes de llamar a `PKPANTALLAS.TRAZA`.

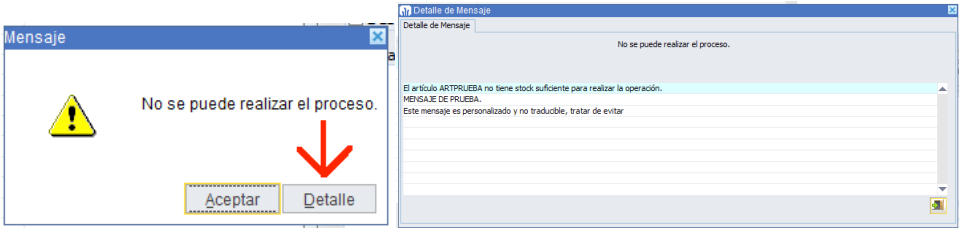
También se puede poner un texto fijo, pero no se intentará traducir: `PKPANTALLAS.TRAZA(NULL, NULL, <texto fijo>, 'MSG');`

Ejemplo:

```
CREATE OR REPLACE FUNCTION test_mensaje RETURN VARCHAR2 IS
  v_articulo  articulos.codigo_articulo%TYPE := 'ARTPRUEBA';
BEGIN
  pkpantallas.set_msg_replace_texto('<art>', v_articulo);
  pkpantallas.traza('TEST', 'NO_STOCK', NULL, 'MSG');
  pkpantallas.traza('TEST', 'PRUEBA', NULL, 'MSG');
  pkpantallas.traza(NULL, NULL, 'Mensaje no traducible, tratar de evitar', 'MSG');
  RETURN('GEN');
END;
/
```

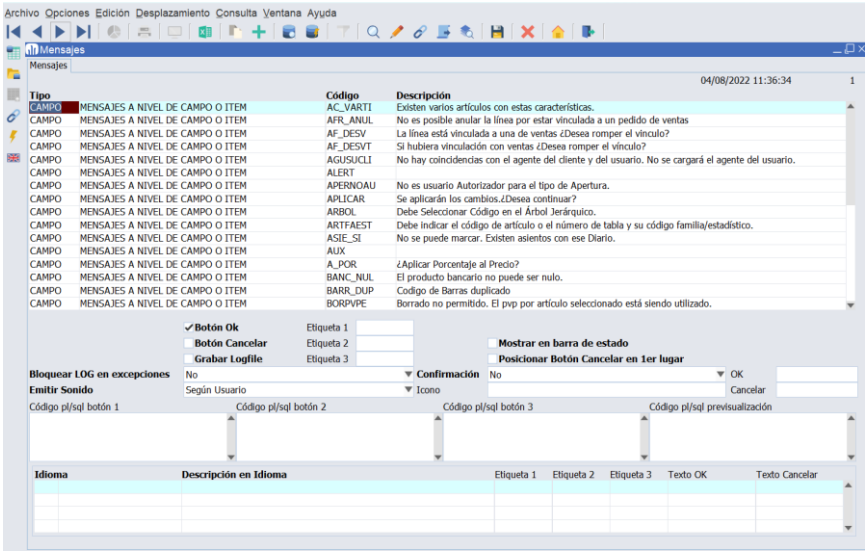
El programa llama a esa función:

```
DECLARE
  v_resultado VARCHAR2(30);
BEGIN
  v_resultado := test_mensaje();

  IF v_resultado != 'OK' THEN
    MSG.MENSAJE('PROCE', 'GENERAL');
  END IF;
END;
```



Crear Mensajes

Para crear los mensajes se hará desde el programa “MENSAJ”.



- **Grabar Logfile**: Si se activa cada vez que se muestre el mensaje dejará un registro de auditoría con que usuario y fecha se ha mostrado.
- **Botón Ok**: Muestra un botón con la etiqueta "Aceptar" o con el texto indicado en "Etiqueta 1". Si se desea un mensaje con tres botones, esta check deberá dejarse desmarcada.
- **Botón Cancelar**: Muestra un botón con la etiqueta “Cancelar” o con el texto indicado en "Etiqueta 2". Si se desea un mensaje con tres botones, esta check deberá dejarse desmarcada.
- **Confirmación**: Si se activa, al usuario se le pedirá que teclee el contenido de “Texto Confirmación OK” para aceptar el mensaje o “Texto Confirmación Cancelar” para cancelar el mensaje, de esta forma se evita que el usuario pase sin leer el mensaje. Valores posibles:
    - **No**: Se visualiza el mensaje de forma normal, no es necesario que el usuario teclee nada.
    - **Sí - Independiente de Mayúsculas / Minúsculas**: El usuario tiene que teclear los textos especificados en “Texto Confirmación OK” o “Texto Confirmación Cancelar”, pero puede teclear en mayúsculas o minúsculas, es indiferente.
    - **Sí - Texto Exacto**: El usuario tiene que teclear exactamente el texto del mensaje, con las mismas mayúsculas o minúsculas definidas en los textos de confirmación.
- **Emitir Sonido**: Permite configurar a nivel de cada mensaje si se debe o no de emitir un sonido al mostrarse el mensaje. Los valores posibles son:
    - **Sí**: Para el mensaje siempre se emitirá un sonido, independientemente de la parametrización del usuario.
    - **No**: Nunca se va a emitir un sonido para ese mensaje, independientemente de la parametrización del usuario.
    - **Según Usuario**: Dependerá de lo que tenga configurado el usuario en (U_MCONFG - Configurar usuario) o en (U_CONEM - Configurar grupo empresarial) en el campo "Sonido al mostrar un mensaje".
- **Icono**: (Requiere Forms 14c). Permite indicar un icono que se mostrará en el recuadro del mensaje. Para indicar un icono se dispone de lista de valores para poder seleccionar entre los iconos disponibles en Libra.
- **Posicionar Botón Cancelar en 1er lugar**: Si se activa, el botón “Cancelar” aparecerá seleccionado por defecto.
- **Mostrar en barra de estado**: Se puede activar para mensajes de poca importancia, de forma que únicamente se muestra el mensaje en la barra inferior y el usuario puede seguir trabajando normalmente sin tener que aceptar ningún mensaje.

- **Etiqueta 1, 2, 3**: Con esos campos se puede alterar el texto que aparece el cualquiera de los 3 botones que puede mostrar un mensaje.
- **Código pl/sql botón 1, 2, 3**: Código PL/SQL que se ejecutará cuando el usuario pulse en algunos de los botones del mensaje. Ver apartado "Código PL/SQL" para más información.
- **Código pl/sql previsualización**: Código PL/SQL que se ejecutará antes de lanzar el mensaje. Tiene acceso de lectura y modificación de la definición del mensaje, así como de los campos del programa que lo lanza. Para el acceso a las variables del mensaje utilizaremos el *pkpantallas.set_variable_env* y *pkpantallas.get_variable_env_xxx* (según corresponda) con estas constantes: *PKLIBPNT_MSG_DESCRIPCION, PKLIBPNT_MSG_BOTON_OK, PKLIBPNT_MSG_BOTON_CANCEL, PKLIBPNT_MSG_ETIQUETA_BOTON_1, PKLIBPNT_MSG_ETIQUETA_BOTON_2, PKLIBPNT_MSG_ETIQUETA_BOTON_3, PKLIBPNT_MSG_FORZAR_CONFIRMA, PKLIBPNT_MSG_VALOR_CONF_OK, PKLIBPNT_MSG_VALOR_CONF_CANCEL, PKLIBPNT_MSG_CODIGO_PL_SQL_B1, PKLIBPNT_MSG_CODIGO_PL_SQL_B2, PKLIBPNT_MSG_CODIGO_PL_SQL_B3, PKLIBPNT_MSG_POS_BOTON_CANCEL, PKLIBPNT_MSG_MOSTRAR_EN_BARRA*

Mensajes con 3 botones, si no se activa la check "Botón Ok" ni la check "Botón Cancelar" se mostrará un mensaje con 3 botones, el primer botón con la etiqueta "Si", el segundo con la etiqueta "No" y el tercero con la etiqueta "Cancelar".

Texto confirmación aleatorio (captcha): En el caso de activar forzar confirmación, es posible especificar que el texto sea el resultado de la ejecución de ***DBMS_RANDOM.string(opt,len)***, insertando en la sección de descripción el código ***<random:x:n>*** dónde "**x**" se corresponde con un valor válido para opt ('U','L','A','X','P') y "**n**" un entero entre 1 y 30 enviado a len.

Atributos Visuales

Es muy importante su uso para poder modificar la apariencia mediante parametrización. Normalmente las clases de propiedad ya llevan asociado el atributo visual correspondiente. Tenemos definidos los siguientes atributos visuales:

- **V1**: Atributo visual para el registro actual.
- **V2**: Atributo visual de campos editables numéricos o alfanuméricos.
- **VDISPLAY**: Atributo visual de campos no editables, normalmente descripciones.
- **VQUERY**: Atributo visual para modo entrada de consulta.
- **VQUERY**_DISPLAY: Atributo visual para modo entrada de consulta.
- **VCHECK**_BOX: Atributo visual para campos de tipo check.
- **VBOTON**: Atributo visual para botones
- **CAMPO_OBLIGATORIO_PROMPT**: Atributo visual para la etiqueta de campos obligatorios.
- **CAMPO_OPCIONAL_PROMPT**: Atributo visual para la etiqueta de campos opcionales.
- **VLIENZO**: Atributo visual para lienzos.
- **VENTANA**: Atributo visual de la ventana.
- **VLOGO**: Para los logos
- **VALERTA**: Para las alertas

V1: Se especificará a nivel de bloque en la propiedad Grupo de Atributos Visuales del Registro Actual.

V2, VCHECK_BOX, BOTÓN: Se especificarán a nivel de ítem en la propiedad Grupo de Atributos Visuales.

CAMPO_OBLIGATORIO_PROMPT, CAMPO_OPCION_PROMPT: Se especificarán a nivel de ítem en la propiedad Grupo de Atributos Visuales del prompt.

VLIENZO: Se aplicará a los lienzos en la propiedad Grupo de Atributos Visuales.

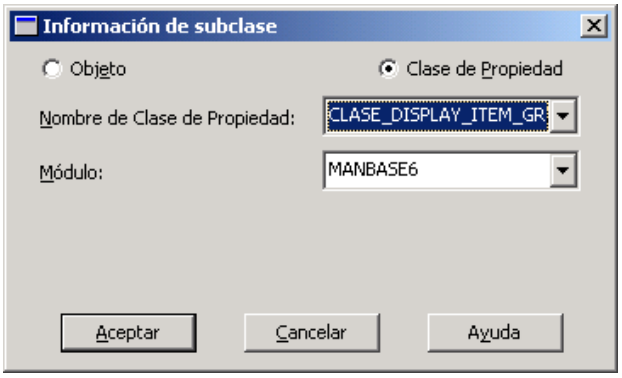
Cada usuario tiene un perfil de configuración de colores, fuentes, ..., para ello es muy importante usar las clases de propiedades y atributos visuales definidas en el componente FORMULARIO_BASE.

Clases de propiedad

Las clases de propiedad son agrupaciones de propiedades con un valor prefijado para los componentes del formulario (campos, lienzos, ventanas, …). Este valor se asignará automáticamente al componente que se le asigne la clase.

Para indicar a qué clase de propiedad pertenece un ítem se asignará en la propiedad *Información de Subclase*.

Seleccionamos en el botón de radio *Clase de Propiedad* y le damos el *Nombre de Clase de propiedad.*



Una propiedad de un ítem que tenga asociada una clase de propiedad puede estar en alguno de los siguientes estados:

- Heredado de la clase de propiedad.
- Rota la herencia con la clase de propiedad, es decir se ha modificado la propiedad para este ítem en concreto. , para volver a realizar la herencia simplemente pulsamos y el valor de la propiedad vuelve al estado original establecido en la clase.
- No heredado, pero no se ha modificado el valor de la propiedad, por tanto, el valor que tiene es el estándar para el Campo de Forms. . Si se añadiese esa propiedad a la clase de propiedad asignada al ítem la heredaría de forma automática.
- No heredado, pero se ha modificado el valor estándar de la propiedad, , para volver al valor original por defecto de Forms pulsaremos .

Es muy importante el uso de clases de propiedad y que rompamos la herencia sólo en los casos que sea estrictamente necesarios, de esta forma cualquier cambio masivo se alguna propiedad se limitará a modificar la clase en la librería de objetos y recompilar todos los programas.

Las clases de propiedad se dividen en las siguientes:

- Clases de propiedad para campos que se encuentren en un bloque multiregistro:
  - CLASE_DISPLAY_ITEM_GRID: Campos de visualización.
  - CLASE_ITEM_NUMBER_GRID: Campos numéricos.
  - CLASE_TEXT_ITEM_GRID: Campos alfanuméricos.
  - CLASE_ARCHIVO_GRID: Campo que va almacenar un archivo, ver apartado: Gestión de Archivos.
  - CLASE_DATE_ITEM_GRID: Campos de tipo fecha.
- Clases de propiedad para campos que no se encuentren en un bloque multiregistro:
  - CLASE_DISPLAY_ITEM: Campos de visualización.
  - CLASE_TEXT_ITEM: Campos alfanuméricos.
  - CLASE_TEXTAREA_ITEM: Campos que muestran varias líneas y permiten retornos de carro en el texto introducido en él.

  - CLASE_TEXT_ITEM_NUMBER: Campos numéricos.
  - CLASE_LIST_ITEM. Campos de lista desplegable.
  - CLASE_LIST_COMBO_BOX_ITEM: Campos de tipo list-item en donde el usuario puede teclear un nuevo valor distinto a los prefijados.
  - CLASE_TLIST_ITEM: Es similar al list-item, pero el usuario visualiza más de una opción.
  - CLASE_ARCHIVO: Campo que almacenará un archivo, ver apartado: Gestión de Archivos.
  - CLASE_DATE_ITEM: Campos de tipo fecha.
- Clases de propiedad válidas tanto para multiregistros como para items que no estén en un multiregistro:
  - CLASE_BUTTON. Botones.
  - CLASE_CHECK_BOX. Campos de tipo check.
  - CLASE_RADIO_GROUP. Campos de selección de tipo radio.
- Clases de para logos e imágenes:
  - CLASE_IMAGEN. Al asignar esta clase ya se incorpora el menú contextual para cargar la imagen.
- Clases de propiedad para objetos que no sean items.
  - CLASE_MARCO: Marcos de agrupación de objetos.
  - CLASE_FORM: Clase de formulario.
  - CLASE_BLOQUE: Clase para bloques que no sean multiregistro.
  - CLASE_BLOQUE_REG_UNICO: Clase para bloques que solo puedan tener un solo registro, por ejemplo, pantallas de filtros para consultas.
  - CLASE_BLOQUE_SCROLL: Clase para bloques multiregistro.
  - CLASE_PAGE. Clases para Lienzos.
  - CLASE_VENTANA. Clase para ventanas.
  - CLASE_LOGO: Clase para los logos (Obsoleta).

Nota: Los marcos tienen propiedad de título, pero trataremos de no ponerles nunca un título ya que no se puede modificar por código y por tanto no se puede traducir.

Campos de tipo elemento de lista

En Forms no existe la propiedad de bisel de un campo de tipo lista desplegable, por tanto, siempre va a tener apariencia de hundido, y no queda bien dentro de un grid de datos, por tanto, trataremos de ponerlo como registro único o evitar su uso.

Ejecución de programas por código

En Libra hay múltiples variantes de programas y no todas ellas tienen un ejecutable de forms asociado, por ejemplo, programas de tipo “Ejecuta Metadatos”, “Programas” …, no se pueden llamar usando CALL_FORM o OPEN_FORM y en su vez hay que utilizar FMENU.LLAMA_FORM.

FMENU.LLAMA_FORM tiene los mismos parámetros que CALL_FORM y por tanto se puede reemplazar uno por otro sin problema, de todas formas, los parámetros que se pasan a los programas no se pueden pasar con una lista de parámetros de Forms ya que esos parámetros en ocasiones tienen que ser convertidos al formato que acepte el programa que se quiere ejecutar, para ello hay que construir la lista de parámetros de tipo “pkpantallas.tabla_param_llamada_plug_in” y antes de llamar a FMENU.LLAMA_FORM hay que indicar que se quieren usar con el procedimiento FMENU. SET_PARAMETROS_LLAMA_FORM.

Para construir la lista de parámetros hay que tener una variable de tipo "pkpantallas.tabla_param_llamada_plug_in" y para ir cargando parámetros se llamará a FMENU.ADD_PARAMETRO(<código parámetro>, <tipo>, <valor>);

- **<código parámetro>**: Nombre del parámetro que espera el programa llamado.
- **<tipo>**: Forma de interpretar el valor indicado en <valor>, en el caso de pasar 'C' el valor indicado en <valor> se considera constante. Si se pasa 'R' considera que se está indicando BLOQUE.CAMPO y que en el momento de incorporar el parámetro debe de leer el valor de ese campo.
- **<valor>**. Dependiendo de <tipo> será un valor constante o BLOQUE.CAMPO.

Ejemplo:

```
DECLARE
  v_t_parametros  pkpantallas.tabla_param_llamada_plug_in;
BEGIN
   FMENU.ADD_PARAMETRO(v_t_parametros, 'P_CODIGO', 'C', '001');
   FMENU.ADD_PARAMETRO(v_t_parametros, 'P_CLIENTE', 'R', 'B1.CLIENTE');
   FMENU.SET_PARAMETROS_LLAMA_FORM(v_t_parametros);
   FMENU.LLAMA_FORM('programa', NO_HIDE, DO_REPLACE, QUERY_ONLY);
END;
```

Si en el parámetro "display" (es decir, el segundo parámetro de FMENU.LLAMA_FORM) se pasa el valor SESSION, en vez de ejecutarse el programa con un CALL_FORM se ejecutará con un OPEN_FORM, de manera que el programa llamador no se queda a la espera de que termine el programa llamado.

Campos de visualización de descripciones

Para los campos que visualicen la descripción almacenada en otra tabla, usaremos como nomenclatura el nombre del campo código con el prefijo D_.

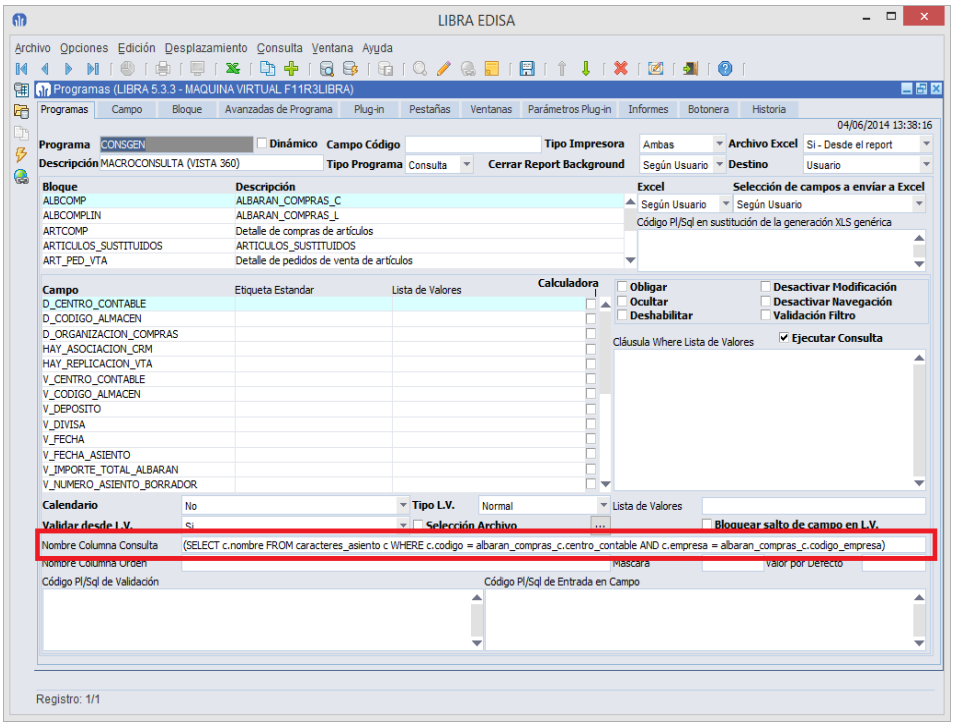
Por ejemplo, si tenemos el código del cliente y tenemos que visualizar su nombre, y el código del cliente se encuentra en un campo llamado CODIGO_CLIENTE, el nombre del cliente se mostrará en un campo llamado D_CODIGO_CLIENTE.

Al campo en donde queremos ver la descripción le aplicamos la clase de propiedad CLASE_DISPLAY_ITEM o CLASE_DISPLAY_ITEM_GRID.

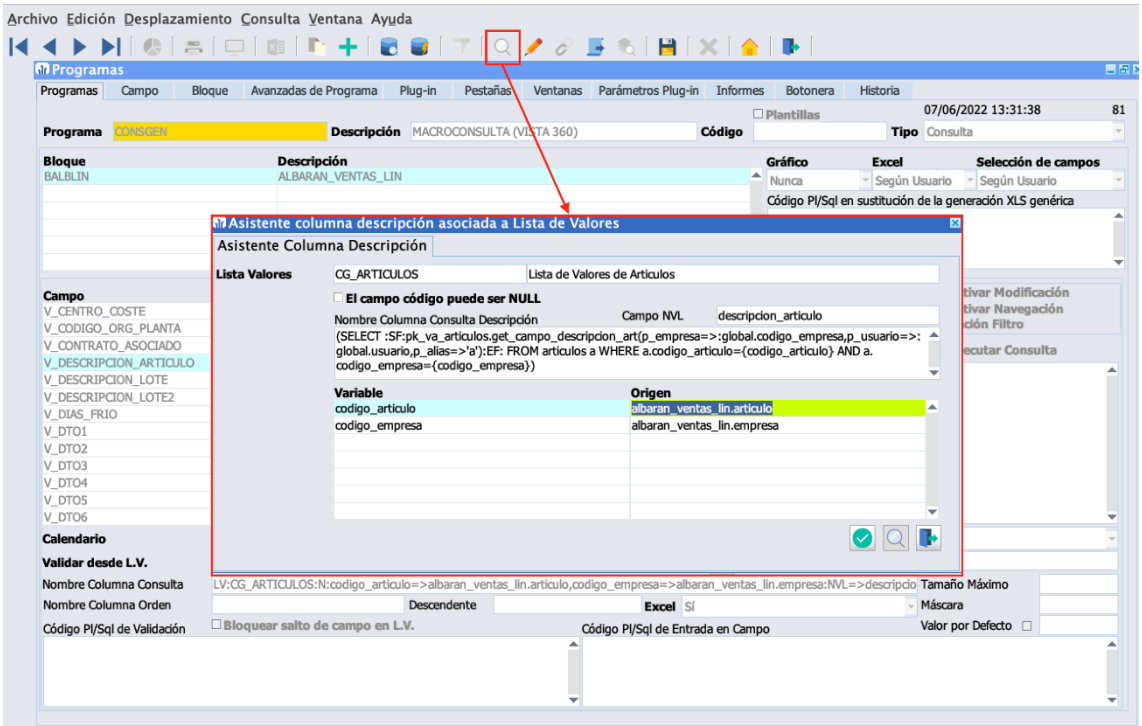
En el mantenimiento de programas y programas personalizados en la pestaña "Avanzadas de Campo" hay un campo "Nombre de Columna Consulta", en este campo se meterá la SQL para obtener la descripción.

Por ejemplo en un programa en el que se tenga que obtener el nombre del cliente en el campo D_CODIGO_CLIENTE, se introducirá en el mantenimiento de programas un campo D_CODIGO_CLIENTE y en "Nombre de Columna Consulta" se introducirá la siguiente sentencia `(SELECT nombre FROM clientes c WHERE c.codigo_rapido = clientes_prueba.cliente AND c.codigo_empresa = clientes_prueba.empresa)`.

Se puede observar que se usa CLIENTES_PRUEBA en la WHERE en vez de un alias, en esta consulta para hacer referencia a la tabla a la que está asociado el bloque lo hay que hacer por el nombre de la tabla, no se puede usar un alias.



Es posible indicar que la consulta se obtenga en tiempo de ejecución de una determinada lista de valores, para ello en el campo “Nombre Columna Consulta” en vez de introducir la consulta se pulsa sobre el botón de lista de valores o con F9 para ir a un asistente.



El asistente nos pedirá los siguientes datos:

- **Lista de Valores**: Código de la lista de valores que se utilizará para obtener la descripción.
- **El campo código puede ser null**: Si activamos esta check, le estamos indicando que el código puede ser nulo y por tanto debe optimizar la consulta para esa casuística.
- **Campo Código**: Sólo aparece si se activa "El campo código puede ser null" y ahí podemos indicar qué campo consideramos código, en el caso de no indicarlo se considerará la primera variable de la consulta de la lista de valores.
- **Campo NVL**: Indicamos que únicamente se realizará la consulta si el campo que se indica es NULL, por ejemplo, si es una consulta sobre ALBARAN_VENTAS_LIN, indicaremos el campo ALBARAN_VENTAS_LIN.DESCRIPCION_ARTICULO, ya que si el albarán tiene la descripción almacenada no hace falta ir al artículo a buscarla.

A continuación, hay que mapear cada variable de la consulta con el campo de la tabla. En el ejemplo anterior, si la tabla es ALBARAN_VENTAS_LIN la variable {codigo_articulo} se mapea con "albaran_ventas_lin.articulo" y la variable {codigo_empresa} con "albaran_ventas_lin.empresa".

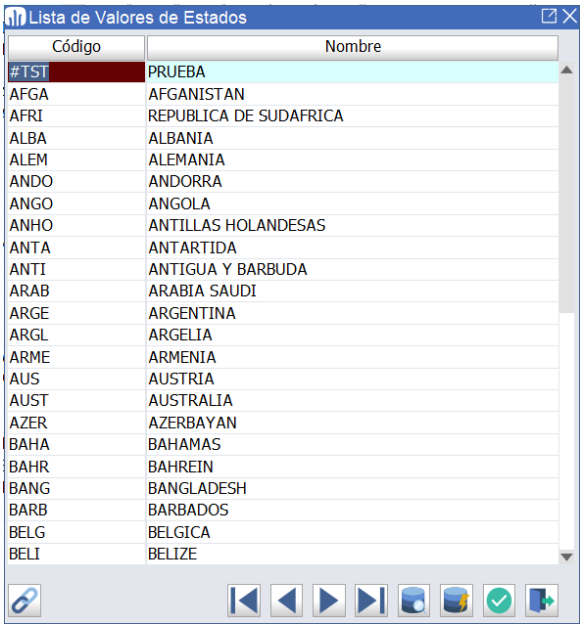
IMPORTANTE: En el caso de que se permita insertar, modificar o borrar registros hay que activar la propiedad de clave primaria a los campos que sean clave primaria. Si la tabla no tiene clave primaria no se puede usar esta opción, y habrá que usar el disparador POST-QUERY.

Listas de Valores

Como principal ventaja de este sistema de listas de valores es que se pueden personalizar de forma muy sencilla para una determinada instalación.

Para usar las listas de valores solo se necesita incorporar un grupo de objetos (LISTA_VALORES) a los programas.

Este sistema de listas de valores **también da soporte de validación del campo incluso si no se ha introducido el dato por la lista de valores**, de esta forma la validación de los programas que usan una misma lista de valores siempre va a ser exactamente igual, con lo que se optimiza el uso de la memoria SHARED_POOL de la base de datos.



MUY IMPORTANTE: Para el correcto funcionamiento de los programas, todo el código que se meta en los disparadores PRE-RECORD, PRE-BLOCK, WHEN-NEW-BLOCK-INSTANCE, WHEN-NEW-RECORD-INSTANCE debe de ir entre la condición **IF NOT lv.viene_de_lista THEN** a excepción de la llamada al DISPSTD correspondiente, por ejemplo:

```
DISPSTD.WHEN_NEW_BLOCK_INSTANCE;

IF NOT lv.viene_de_lista THEN
  <código del disparador>
END IF;
```

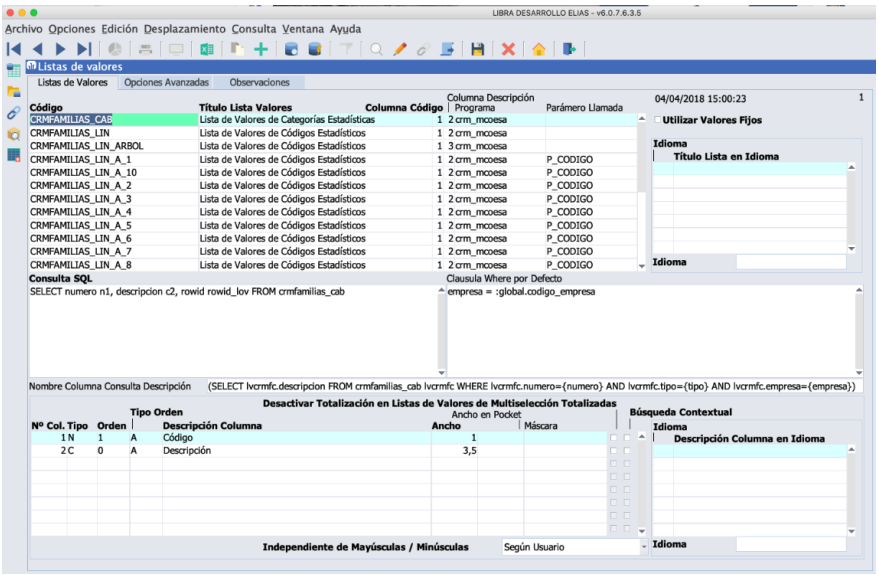
Limitaciones

Las listas de valores tienen como limitación principal el número máximo de columnas, que sólo pueden ser 20, estas pueden ser de tipo alfanumérico, numérico (también en Forms 14c se permite el tipo fecha). Si hace falta una lista de valores más compleja la deberemos implementar dentro del programa o mediante un plug-in (ver apartado de plug-in).

No están soportadas sqls que hagan una UNION de varias SELECT, en caso de ser necesario se tendrá que crear una vista de base de datos. Tampoco están soportadas listas de valores que hagan un DISTINCT, en ese caso habrá que añadir al final de la where un GROUP BY por todos los campos que saca la consulta para conseguir el mismo efecto.

Pasos para crear una lista de valores

Para crear una lista de valores desde cero, iremos al mantenimiento de listas de valores de libra, (U_MLISVA).



En el código de la lista de valores se debe introducir siempre que sea posible el nombre de la tabla, solo usaremos otro nombre cuando queramos hacer una personalización específica de una lista de valores.

El título que queremos que aparezca en la ventana que abre la lista de valores lo introduciremos en el campo *Descripción*.

Cualquier columna puede contener el código que queremos devolver al campo desde donde que se llama a la lista de valores, esto se especifica en el campo *Columna Código*, por tanto, deberá de contener un valor entre 1 y 20 dependiendo de los campos que se visualicen. Este campo también se usará para la validación del campo cuando se introduce de forma manual por teclado. Internamente para hacer la validación le añadirá a la WHERE de la lista de valores la condición de que el campo especificado como código sea igual al valor introducido por el usuario.

Podemos definir en el campo *Columna Descripción* el número de columna de la que se ha de obtener la descripción a devolver al programa, por tanto, deberá de contener un valor entre 1 y 20 dependiendo de los campos que sean visualizados, la lista de valores devolverá esta descripción al campo **con el mismo nombre del campo desde el que se llama a la lista, pero con el prefijo D_.** Esto obliga a ser muy estrictos en la nomenclatura que se utiliza en los campos que muestran descripciones.

La base para mostrar la lista de valores es una consulta SQL de tipo SELECT ... con la siguiente particularidad: Los campos que queremos que se visualicen deberán de contener un alias del tipo:

- Columnas con valor alfanumérico: c1, c2, …, c20.
- Columnas con valor numérico: n1, n2, …, n20.
- Forms 14c: Columnas con valor fecha: f1, f2, …, f20.

Notas:

- Si hemos especificado una columna de un tipo no podemos especificar la misma de otro tipo, es decir, si hemos especificado el alias c1 no podremos tener otra columna con el alias n1.
- Si se quiere sacar una fecha y la lista de valores únicamente va ser utilizada en Forms 14c lo mejor es ya declararla como tipo fecha, pero si la lista de valores también se puede utilizar en instalaciones con Forms 12c la meteremos como un campo alfanumérico y para que ordene correctamente no se le aplicará ninguna conversión de tipo, es decir, si se pone como TO_CHAR(campo_fecha, 'DD/MM/YYYY') c4, cuando el usuario haga una ordenación la hará alfanumérica en vez de fecha, por lo que lo correcto sería simplemente poner *campo_fecha c4*.

MUY IMPORTANTE 1: Las columnas a visualizar han de tener el alias cx ó nx donde x es un número entre 1 y 20 y a continuación una coma y un espacio.

MUY IMPORTANTE 2: Si hay columnas que se calculan con una subquery (SELECT campo FROM xxxxx), es recomendable meter antes del FROM tabla_principal la etiqueta /*FRLV*/ para que el entorno busque de forma más precisa el punto en donde se encuentra la tabla principal. Si no se indica se intentará localizar, pero hay casos en los que no será capaz y puede producir un funcionamiento con errores.

Ejemplos:

Casos incorrectos:

- `SELECT nombre c1 , codigo c2,`: Incorrecto ya que hay un espacio después del alias c1.
- `SELECT nombre c1,codigo c2`: Incorrecto ya que no hay espacio que separa la coma del alias c1 y de la columna siguiente:

Lo correcto para los dos casos anteriores sería:

- `SELECT nombre c1, codigo c2`

Aparte de los campos que se visualizarán en la lista de valores también deberemos sacar siempre el rowid de la tabla principal de la SELECT con el alias **rowid_lov** y este campo debe de ser el último de los campos que selecciona la sentencia SQL.

Ejemplo:

```
SELECT codigo c1, nombre c2, rowid rowid_lov FROM agentes
```

Ejemplo más complejo:

```
SELECT codigo_articulo C1,
(SELECT          DECODE(TIPO_DESC_ART,'V',articulos.descrip_comercial,          'C',articulos.descrip_compra,
'T',articulos.descrip_tecnica,articulos.descrip_comercial)
FROM usuarios WHERE usuarios.usuario =:GLOBAL.USUARIO) C2, rowid rowid_lov
/*FRLV*/ FROM ARTICULOS
```

NOTA: véase que se ha añadido la etiqueta /*FRLV*/ para indicar en donde comienza el FROM principal de la consulta.

Para cada programa podremos especificar una cláusula WHERE específica para la lista de valores, pero en la misma lista de valores ya podemos especificar una por defecto. En la cláusula WHERE se pueden usar referencias a campos, variables globales, ... de Forms, un ejemplo típico sería *empresa = :global.codigo_empresa*

Es posible controlar desde funciones de base de datos la consulta que ejecutará una lista de valores, para ello entre las etiquetas **:SF:** y **:EF:** se puede introducir una función que devuelva la SELECT o parte de ella.

Por ejemplo, en el caso que se mostró en "Ejemplo más complejo", se podría simplificar la consulta para evitar que se haga uso de la tabla USUARIOS por cada fila que viene de la tabla quedando la consulta de la siguiente forma:

```
SELECT codigo_articulo c1, :SF:pk_va_articulos.get_campo_descripcion_art(:global.usuario):EF: c2, rowid
rowid_lov
FROM artículos
```

En este caso, `:SF:pk_va_articulos.get_campo_descripcion_art(:global.usuario):EF:` devolverá *descrip_comercial*, *descrip_compras* o *descrip_tecnica* en base a la parametrización de usuario validado. De esta forma si el usuario tuviese parametrizada la descripción comercial la consulta que tendría la lista de valores sería: `SELECT codigo_articulo c1, descrip_comercial c2, rowid rowid_lov FROM articulos`, al simplificar la consulta se consigue un mejor rendimiento en la ejecución de la lista de valores.

Mediante esta funcionalidad se podría cambiar el 100% el origen de la consulta, por ejemplo: `:SF:F_TEST(:global.codigo_empresa,:global.usuario):EF:`, ejecutará la función F_TEST que devuelve la consulta a utilizar. Ejemplo de F_TEST:

```
CREATE OR REPLACE FUNCTION F_TEST(p_empresa VARCHAR2, p_usuario VARCHAR2) RETURN VARCHAR2 IS
BEGIN
 RETURN ('SELECT codigo_articulo c1, ''PRUEBA'' c2, rowid rowid_lov FROM articulos');
END;
```

Para cada columna que se va a visualizar en la lista de valores hemos de introducir los siguientes campos:

- **Número**: Número correlativo entre 1 y 20 que identifica la columna, debe de corresponder con el número especificado en el alias en la SELECT.
- **Tipo**: Puede contener dos valores:
  - C: Columna alfanumérica.
  - N: Columna numérica.
- **Orden**: Cuando se lanza la lista de valores ya se puede forzar un ordenamiento inicial, en este campo se debe especificar el número de orden que va a ocupar este campo dentro del ORDER BY de la sentencia SQL. Si se introduce 0 se indica que no se quiere que ese campo forme parte de la ordenación inicial.
- **Tipo Orden**: Tipo de ordenación inicial del campo, puede contener dos valores:
  - A: Ascendente.
  - D: Descendente.
- **Descripción Columna**: Título que va a tener la columna.
- **Ancho**: Ancho en pulgadas de la columna.
- **Ancho Pocket**: Ancho en pulgadas de la columna cuando la lista se muestra en un puesto que es de tipo "Pocket".
- **Máscara**: Sólo se puede aplicar a columnas numéricas y se pondrá la máscara de formato con la que se debe de mostrar el número. En caso de que sean cantidades se puede poner CTD y aplica la máscara de formato correspondiente a los decimales establecidos en la empresa para las cantidades.
- **Desactivar totalización en Listas de Valores de Multiselección Totalizadas**: Hay un tipo de lista de valores en donde el usuario puede seleccionar los registros y las columnas numéricas son totalizadas, si se activa esta check para un campo numérico la columna no será totalizada.
- **Independiente de Mayúsculas / Minúsculas**: Si se indica "Sí", cuando se haga una búsqueda en ese campo, se hará independientemente de que en la tabla esté almacenado en mayúsculas / minúsculas e independientemente de que el patrón de búsqueda esté en mayúsculas o minúsculas. Indicando "No" las búsquedas serán sensibles a mayúsculas o minúsculas. Si se indica "Según Usuario" se utilizará el valor por defecto para este fin indicado a nivel de configuración de empresa / usuario.
- **Búsqueda Contextual**: Mediante esta check se pueden indicar las columnas sobre las que la búsqueda contextual debe realizar la búsqueda. En el supuesto de que no esté marcada ninguna columna, la búsqueda contextual se aplicará sólo sobre la columna que está marcada como "Columna Descripción", pero si se marca alguna columna como "Búsqueda Contextual", la "Columna Descripción" será ignorada, a no ser que se marque también esta columna.

Tanto el título de la ventana como el título de la columna se pueden cambiar por idioma, para ello se deberá cubrir las dos secciones de idiomas.

Existe una opción en la botonera vertical para exportar la lista de valores en formato SQL.

Asociar programa a una Lista de Valores

Es posible asociar un programa a una Lista de Valores. Esta funcionalidad es muy útil cuando el usuario saca la lista de valores y se da cuenta que el registro que necesita no existe, por tanto, se da la posibilidad de navegar al programa que se especifica para poder crear ese registro.

Para indicar el programa asociado a la lista de valores indicaremos el nombre del programa en el campo *Programa*. El botón de navegación únicamente se habilitará si este campo está cubierto y el usuario tiene permisos de altas en ese programa.

Aparte de navegar al programa asociado desde la lista de valores cuando el usuario entra en el campo que tiene la lista de valores puede hacer doble click sobre el campo o pulsar sobre el botón de hipervínculo del menú y lo llevaría también al programa sin necesidad de abrir la lista de valores.

Cuando se navega al programa se pueden pasar dos parámetros al programa llamado:

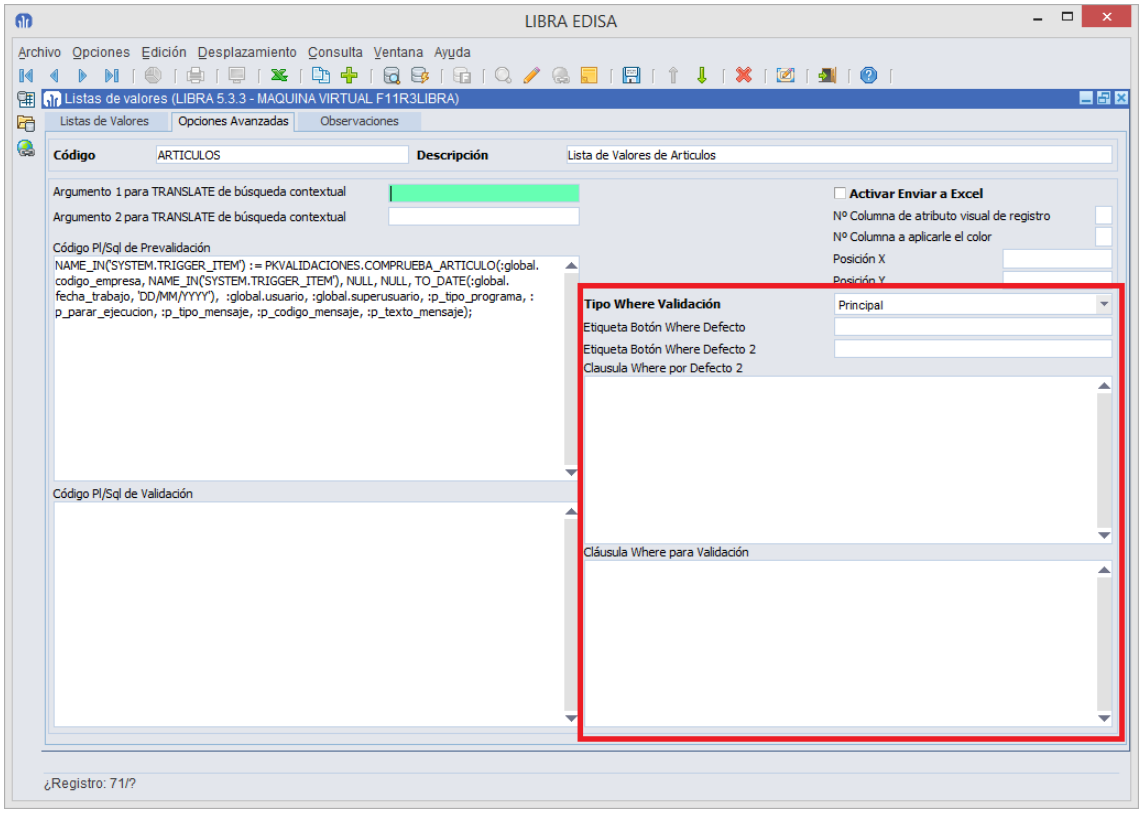
- **Valor que contiene el campo**: Para pasar el valor que tiene el campo hay que indicar el nombre del parámetro en "Parámetro Llamada"
- **Código de la lista de valores**: Se puede pasar el código de la lista de valores al programa para que pueda actuar de forma distinta según la lista de valores desde donde es llamado. Para ello hay que indicar en que parámetro del programa llamado hay que pasar el código de la lista de valores cubriendo el campo "Parámetro envío código de LV al llamar programa" en la pestaña "Opciones Avanzadas".

Segunda Cláusula Where para una lista de valores.

A una lista de valores podemos especificar una segunda cláusula where, y cuando se especifica al llamar a la lista de valores aparecerá un botón para poder conmutar la consulta entre la cláusula where normal y la segunda. Para el botón que aparece le indicaremos qué etiqueta debe de tener según la cláusula where que está aplicando, para eso son los campos "Etiqueta Botón Where Defecto" será la etiqueta que muestre cuando se está filtrando por la where por defecto y "Etiqueta Botón Where Defecto 2" será la etiqueta que muestre cuando está filtrando por la segunda lista de valores.

Se puede indicar cuál de las cláusulas where debe de usarse para efectuar la validación del campo, para ello se añadió el desplegable "Tipo Where Validación" que puede tener los siguientes valores:

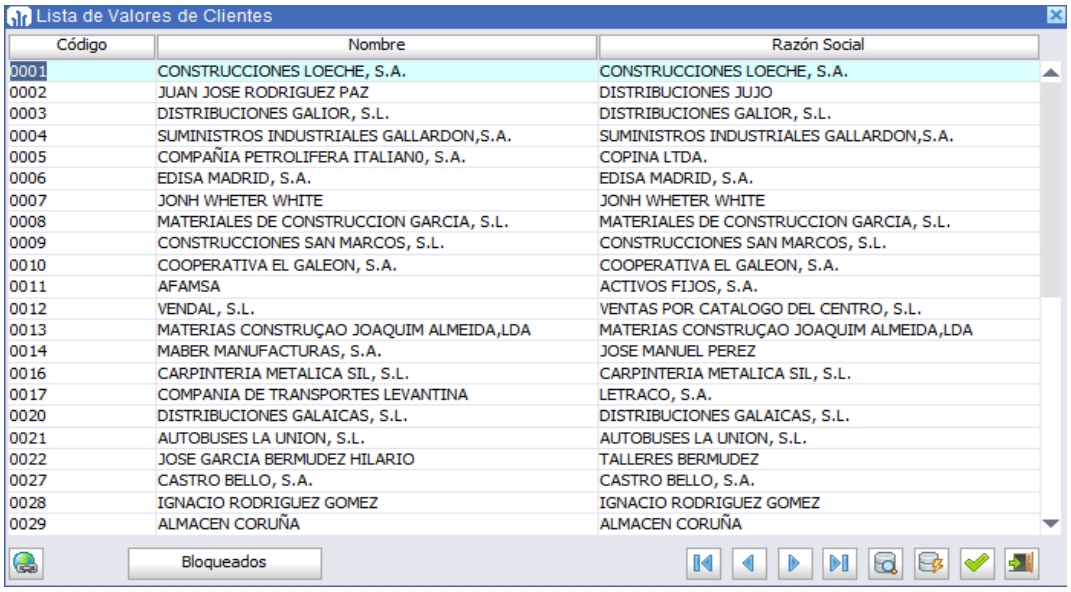
- **Principal**: Siempre se valida usando la cláusula where por defecto, no se usa la segunda cláusula where en la validación.
- **Secundaria**: Siempre se usa la segunda cláusula where, la cláusula where por defecto no será usada en la validación.
- **Personalizada**: Al seleccionar esta opción se habilita un nuevo campo donde indicar la una cláusula where específica para la validación independiente de las usadas en para mostrar los registros en la lista de valores.



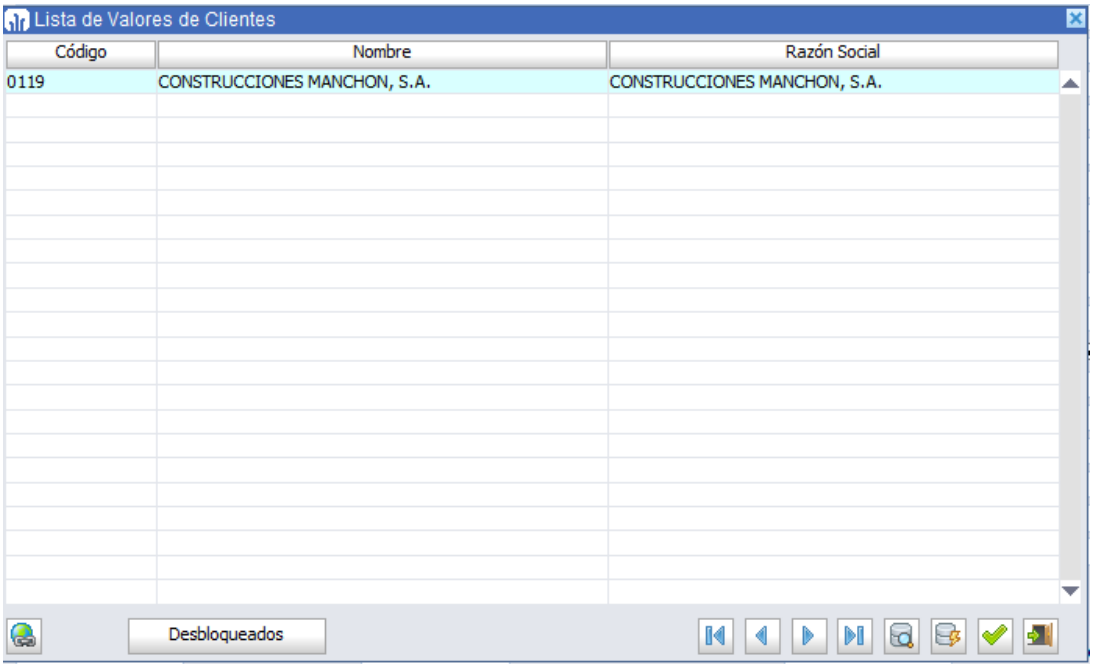
Por ejemplo podemos hacer que la lista de valores CLIENTES_GESTION habilite un botón donde ponga "Bloqueados" y al pulsar sobre él consultar los clientes que están bloqueados, para ello metemos en el campo Clausula Where por Defecto 2 la siguiente where: `WHERE codigo_empresa = :GLOBAL.codigo_empresa AND EXISTS (SELECT 1 FROM bloqueo_clientes WHERE empresa = :GLOBAL.codigo_empresa AND codigo_cliente = codigo_rapido AND (usuario = :GLOBAL.usuario OR usuario IS NULL) AND (TRUNC(SYSDATE) >= desde_fecha) AND (TRUNC(SYSDATE) <= hasta_fecha)) AND PKVALIDAR_ENTIDADES.CLIENTE(CODIGO_RAPIDO,:GLOBAL.CODIGO_EMPRESA,:GLOBAL.USUARIO,SYSDATE)='OK'`

Y en el campo "Etiqueta Botón Where Defecto" metemos el texto: Bloqueados y en "Etiqueta Botón Where Defecto 2" metemos el texto "Desbloqueados".

Al pulsar sobre la lista de valores saldrá de la siguiente forma:



Y al pulsar sobre el botón "Bloqueados" saldrá:



NOTA: Cuando se asigna una lista de valores a un programa se puede establecer para ese campo en concreto este funcionamiento sin alterar la lista de valores.

Cláusulas Where dinámicas.

El objetivo de generar condiciones “where” dinámicas sobre una lista de valores es **mejorar la velocidad de carga** de estas.

El principio de funcionamiento se basa en integrar una función que devuelva una where lo más adecuada posible a la instalación en base al usuario y la empresa a la que se ha conectado.

La idea es hacer una función que devuelva la where a añadir a la lista de valores, para ello, la función que devuelve la where irá entre las etiquetas :SF: y :EF:

Ejemplo: `empresa = :global.codigo_empresa :SF:F_MI_FUNCION(:global.usuario, :global.empresa, 'clientes.codigo_rapido'):EF:`

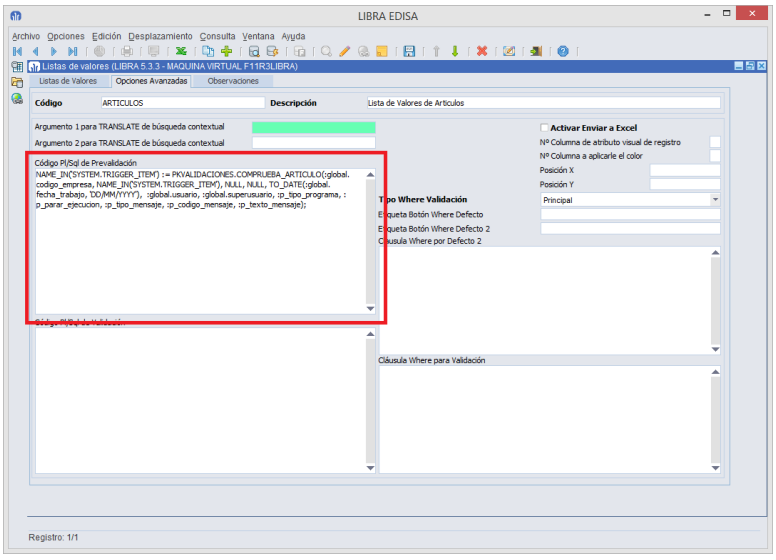
Código de ejemplo de F_MI_FUNCION que aplicará a la where el control de CLIENTES_PERMITIDOS únicamente cuando tiene datos:

```
CREATE OR REPLACE FUNCTION f_mi_funcion(p_usuario VARCHAR2, p_empresa VARCHAR2, p_campo VARCHAR2) RETURN VARCHAR2 IS
 v_cw      pkpantallas.type_max_plsql_varchar2;
 v_hay_reg VARCHAR2(1);

 CURSOR cur_existe(pc_usuario VARCHAR2, pc_empresa VARCHAR2) IS
   SELECT 'S'
     FROM clientes_permitidos cp
    WHERE cp.usuario = cur_existe.pc_usuario
      AND cp.empresa = cur_existe.pc_empresa;
BEGIN
 OPEN cur_existe(p_usuario, p_empresa);
 FETCH cur_existe INTO v_hay_reg;

 IF cur_existe%NOTFOUND THEN
   v_hay_reg := 'N';
 END IF;

 CLOSE cur_existe;

 IF v_hay_reg = 'S' THEN
   v_cw := 'AND EXISTS (SELECT 1 FROM clientes_permitidos cp WHERE cp.usuario = :global.usuario AND cp.empresa = :global.codigo_empresa AND
cp.cliente = ' || p_campo || ')';
 END IF;

 RETURN(v_cw);
EXCEPTION
 WHEN OTHERS THEN
   pkpantallas.log(sqlerrm, 'F_MI_FUNCION', 'OTHERS');
   RAISE;
END f_mi_funcion;
```

Si el usuario tiene registros en la tabla CLIENTES_PERMITIDOS aplicará la siguiente condición: *`empresa = :global.codigo_empresa AND EXISTS (SELECT 1 FROM clientes_permitidos cp WHERE cp.usuario = :global.usuario AND cp.empresa = :global.codigo_empresa AND cp.cliente = cl.codigo_rapido)`*

Si no hay registros aplicará simplemente: *`empresa = :global.codigo_empresa`*

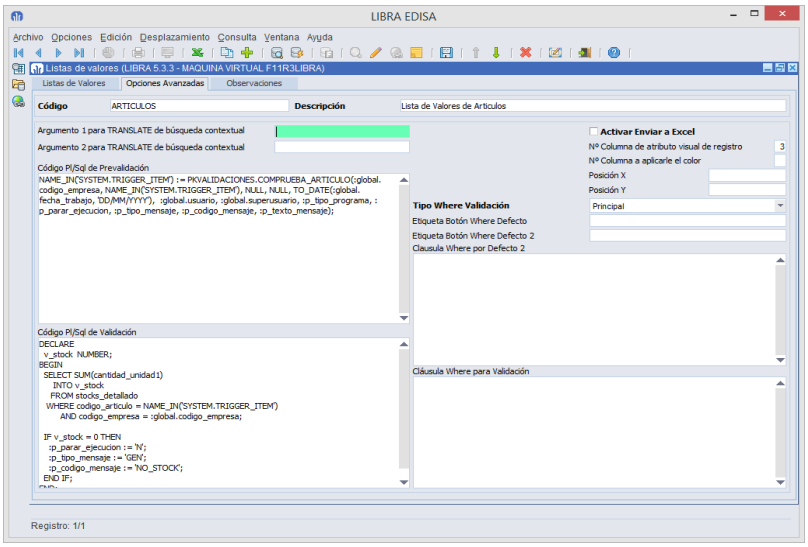
Código PL/SQL de Pre-Validación



Se puede utilizar el código de pre-validación para ser ejecutado antes de que se realice la validación con la SQL de la lista de valores. Esto se ve muy claro con el ejemplo de la tabla ARTICULOS de libra, ya que el usuario puede teclear el código (es lo que valida la lista de valores), el código sinónimo, código de barras, etc. Por tanto, este código puede cambiar la introducción del usuario haciendo una llamada a un paquete de base de datos.

Para ver con más detalle cómo se construye este código PL/SQL ver el apartado "Código PL/SQL".

Código PL/SQL de Validación

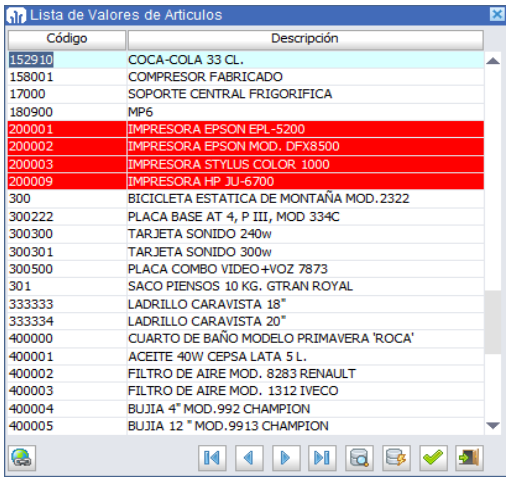


El código PL/SQL se ejecutará en todos los campos que tengan asociada la lista de valores, y se ejecuta después haberse realizado la validación contra los registros de la lista de valores, por lo tanto, ya se puede considerar que el dato que tiene el campo ya es válido, pero requiere una validación más exhaustiva usando PL/SQL. Este PL/SQL se ejecuta de forma adicional al código pl/sql de validación que pueda tener a nivel individual el campo.

Colorear determinados registros en Listas de Valores

Se puede especificar un determinado color para unos registros de una lista de valores, por ejemplo, para marcar artículos sin stock, clientes bloqueados, ...

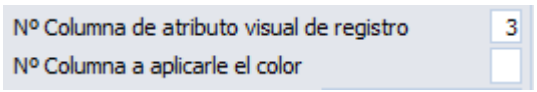
Ejemplo: Lista de Valores que colorea los artículos que el código comienza por 2.



Para parametrizar este funcionamiento hay que añadir una nueva columna a la SQL de la SELECT como si fuese una columna más de la lista de valores que devuelva el atributo visual a aplicar al registro y ponerle un alias cX, siendo X un número de columna de tipo carácter no usado en la lista de valores. El campo no hace falta introducirlo en la lista de campos de la lista de valores, con lo cual no lo muestra en la lista de valores.

En el ejemplo habría que añadir lo siguiente únicamente en la select de la lista de valores: `DECODE(SUBSTR(articulos.codigo_articulo, 1, 1), '2', 'ROJO', NULL) c3`

También hay que indicarle que la columna número 3 es la que va a tener la información del atributo visual a aplicar en el campo "Nº Columna de atributo visual de registro" de la pestaña "Opciones avanzadas".



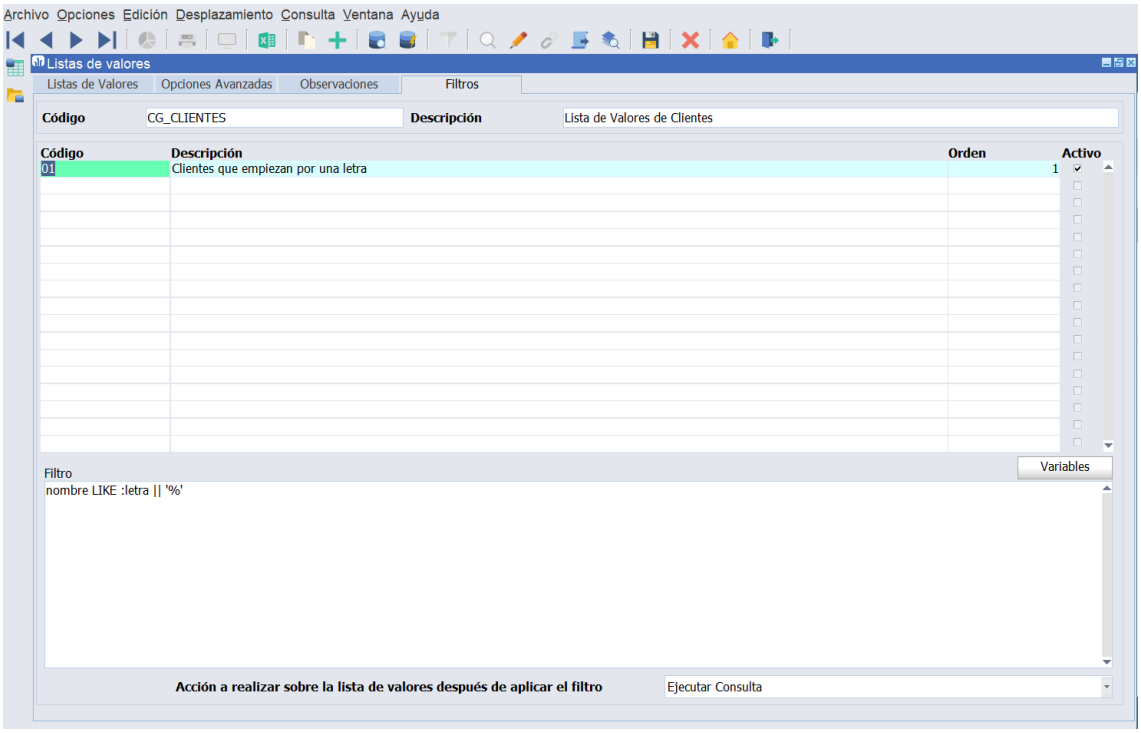
Si el atributo visual que se devuelve no existe en el programa no se le asignará ningún color. En programas compilados en una versión 6.4.8 pueden utilizar cualquier color de los definidos en la vista V_COLORES_ERP, los programas compilados en versiones anteriores tienen que limitarse a colores que se encuentren definidos en la librería de objetos OBJETOSPANT.OLB: MARRON, MARRON_OSCURO, MARRON_INTERMEDIO, MORADO, CYAN, AZUL_CLARO, AZUL_INTERMEDIO, VERDE_CLARO, NARANJA, BLANCO, MARRON_CLARO, FUXIA, AZUL_VERDOSO, MATE, NEGRO, AZUL_MORADO, VERDE_OSCURO, AMARILLO_NARANJA, AZUL_OSCURO, ROSACEO, ROJO, VERDE, AMARILLO, AZUL.

En vez de cambiar el color de todo el registro se puede indicar que únicamente se cambie el color de uno de los campos, para ello se puede indicar en "Nº Columna a aplicarle el color" el número de la columna que se ha de colorear. Esto es útil cuando van a ser muchos los registros coloreados para que el usuario no pierda la referencia del registro en el que se encuentra el cursor.

Filtros en listas de valores

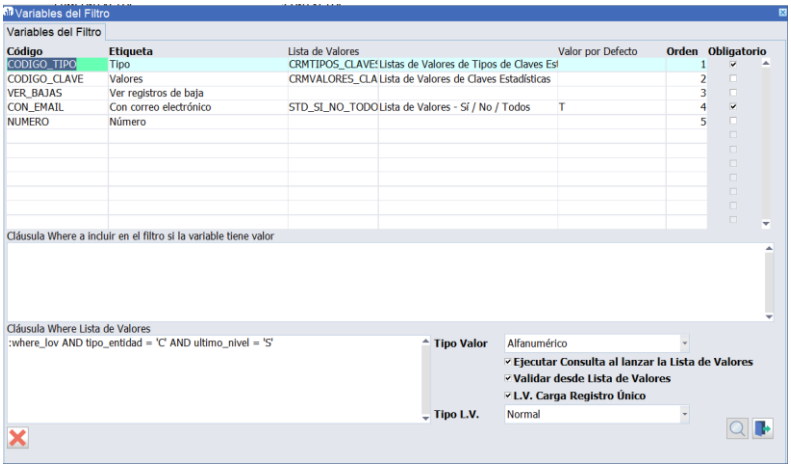
Para versiones de Libra 6.2.1 o superiores existe la posibilidad de que las listas de valores tengan filtros de forma similar a los filtros de los bloques. Para ello en el mantenimiento de Listas de valores y Listas de valores se dispone de la pestaña "Filtros" en donde se pueden configurar.

NOTA: Esta funcionalidad es incompatible con las listas de valores con doble cláusula where.



Para crear un filtro hay que darle un código, una descripción y el orden en el que se le mostrarán al usuario. Si se quiere desactivar temporalmente un filtro se puede desmarcar la check “Activo”.

En el campo “Filtro” filtro se pueden utilizar variables que le serán solicitadas al usuario, esas variables se introducirán directamente en la condición añadiendo : (dos puntos) delante, por ejemplo, si queremos al usuario un rango de fechas, se podría meter algo similar a esto: *av.empresa = :global.codigo_empresa AND av.fecha_pedido BETWEEN :p_desde_fecha AND :p_hasta_fecha*. En este caso se indica que se quieren usar las variables :p_desde_fecha y :p_hasta_fecha, a esas variables hay que indicar la forma en la que se van a solicitar al usuario, para ello hay que pulsar en el botón “Variables”.



- **Código**: Identificador de la variable, si en a condición se usó :p_desde_fecha, el código debe de ser P_DESDE_FECHA.
- **Etiqueta**: Texto que aparecerá junto al campo al generarse la pantalla de filtros del usuario.
- **Lista de Valores**: Código de la lista de valores que tendrá el campo del filtro.
- **Valor por Defecto**: Permite indicar un valor que aparecerá inicialmente al usuario y que podrá ser modificado.
- **Obligatorio**: Si se activa no se dejará realizar la consulta mientras el usuario no proporcione un valor para el filtro.
- **Cláusula Where a incluir en el filtro si la variable tiene valor**: Esta cláusula where únicamente será añadida cuando el usuario introduce algún valor en la variable y permite simplificar la consulta, sobre todo cuando la lista de valores des de tipo multiselección. En principio el contenido del filtro se añadirá al filtro principal añadiendo al final: "AND (+ la cláusula where de la variable +)", pero puede ser que interese que esa condición sea añadida a una parte en concreto de la where principal ya podría estar por ejemplo dentro de una subconsulta, en ese caso en la where principal se añadirá :CODIGO_VARIABLE y en el caso de que el usuario no cubra ese filtro :CODIGO_VARIABLE se quita y si el usuario cubre el filtro se reemplaza.
- **Cláusula Where Lista de Valores**: Filtro para los registros que visualizará la lista de valores. Este campo está asociado al campo “Lista de valores”. Toda lista de valores puede tener asociada una cláusula “WHERE” para todos los programas, pero esa “WHERE” quedará anulada si en este campo se introduce una específica, de forma que se puede llegar a tener condiciones “WHERE” distintas en cada programa. Es muy recomendable añadir la expresión “:where_lov” que se reemplazará en tiempo de ejecución por la cláusula where que tenga la lista de valores, de forma que un cambio de la where en la lista de valores será traslada a todos los programas. Para hacer referencia en esta cláusula where a otra variable que se pida antes (que tenga un orden inferior) debe de usarse ":CODIGO_VARIABLE". No debe de usarse el

campo interno usado, por ejemplo "BFILTROS.FILTRO_ALFA2" ya que si por algún motivo se cambia el orden o se añaden nuevas variables ese campo va a cambiar.

- **Tipo Valor**: Permite indicar si el dato es “Alfanumérico”, “Numérico”, “Fecha” o de tipo “Check”.
- **Ejecutar Consulta al lanzar la Lista de Valores**: Si se indica “Lista de Valores” y se desactiva esta check al lanzar la lista de valores se iniciará en modo de entrada consulta, es decir, se inicia esperando que el usuario proporcione un filtro inicial.
- **Validar desde Lista de Valores**: Si se indica “Lista de Valores” y esta check está activa, únicamente se podrá introducir un valor de los que se puedan visualizar en la lista de valores.
- **Tipo L.V.**: Si el filtro tiene lista de valores permite indicar el tipo de lista de valores a utilizar. Si se indica un tipo de multiselección en la cláusula where del filtro hay que usar el operador IN o NOT IN. Si se selecciona la opción "Rellenar List-Item" el campo se mostrará en forma de List-Item con los valores que devuelva la lista de valores cargados.

Mediante el desplegable “**Acción a realizar sobre la lista de valores después de aplicar el filtro**”, se puede indicar:

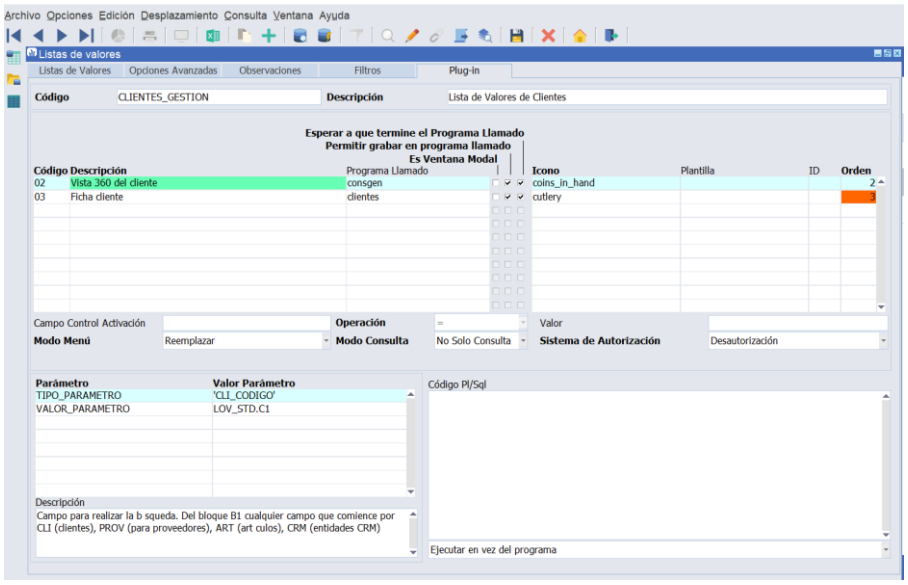
- **Ejecutar Consulta**: Una vez el usuario selecciona el filtro se ejecutará consulta en la lista de valores mostrando los registros que cumplan la condición del filtro.
- **Activar el Modo de Entrada Consulta**: En vez de consultar los registros, el bloque de la lista de valores se limpiará y se pondrá en modo de entrada consulta para que el usuario pueda filtrar más en base a aplicar patrones de búsqueda sobre los campos visibles, una vez indicados los patrones de búsqueda puede pulsar en el botón “Ejecutar Consulta” o F8 para mostrar los registros que cumplan la condición de búsqueda.

Plug-ins en listas de valores

Para versiones de Libra 6.2.1 o superiores existe la posibilidad de que las listas de valores tengan plug-ins forma similar a los plug-ins de los programas. Para ello se dispone de la pestaña "Plug-in" en donde se pueden configurar. **El número máximo de plug-ins que puede tener una lista son 5**.

Ejemplos de uso:

- Visualizar la foto de artículos.
- Llamar a un programa con detalle de información del registro de la lista de valores.
- En el control de fuentes permitirá abrir los metadatos del archivo o un programa con la historia de cambios.
- etc.



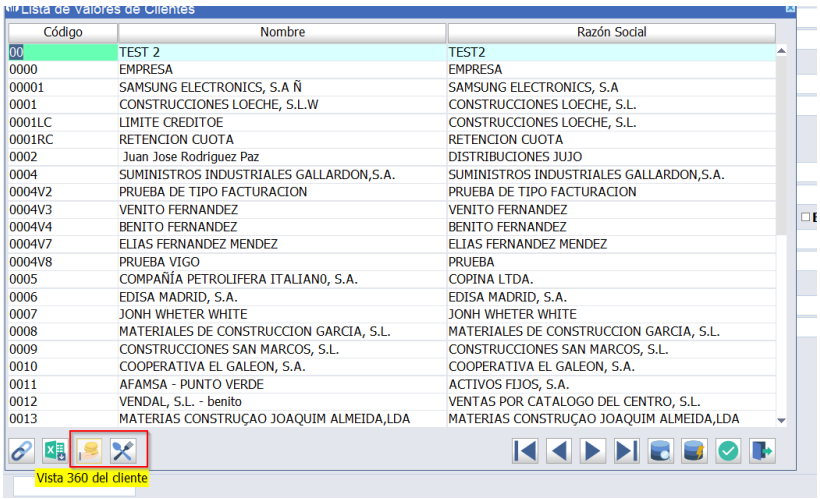
Para obtener información de como configurar un plug-in, el funcionamiento es similar a la funcionalidad de los plug-ins de los bloques del mantenimiento de programas. Ver el apartado Plug-ins para más información.

Las diferencias con los plug-ins de bloques de programas son las siguientes:

- No se les puede asignar una tecla rápida.
- No se pueden asignar al menú contextual de botón derecho del ratón.
- Tiene la opción "Es Ventana Modal" que no está disponible en los bloques.
- En caso de indicar un Código PL/SQL únicamente se puede ejecutar sobre el registro en el que se encuentra el cursor.

Es Ventana Modal: Esta check es muy importante al llamar a un programa para darle información al entorno del tipo de programa al que se llama, si se indica que es "Ventana Modal" no hace falta que se cierre la lista de valores antes de llamar al programa, ya que el programa al ser modal que dará por encima de la propia lista de valores, pero si se llama a un programa que no se ejecuta en una ventana modal y se activa esta check, Libra se quedará bloqueado ya que quedará la lista de valores por encima del programa y el usuario no podrá interactuar con ninguno de los dos programas.

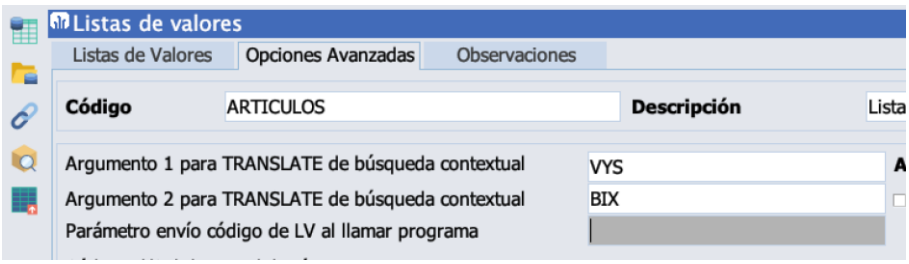
Ejemplo de lista de valores con plug-ins:



Mejoras de búsquedas contextuales

Se puede mejorar la búsqueda contextual, haciendo que busque por el sonido de la pronunciación de una palabra. Por ejemplo, búsquedas independientes de si una palabra se escribe con B o con V, etc.

Para activar este tipo de búsquedas contextuales en una lista de valores hay que cubrir en la pestaña de "Opciones avanzadas" los campos "Atributo 1 para TRANSLATE de búsqueda contextual" y "Atributo 2 para TRANSLATE de búsqueda contextual". Lo que especifiquemos en el atributo1 lo sustituirá letra a letra por el del atributo2, por ejemplo, si en el atributo1 ponemos VYS y en atributo 2 ponemos BIX, la V la sustituirá en la búsqueda por B, la Y la sustituirá por I y la S por la X.



Envío del contenido de una lista de valores a Hoja de Cálculo.

En la pestaña “Opciones Avanzadas” existe la check “Activar Enviar a Excel” que si está activada permite al usuario que envíe el contenido que está viendo en la lista de valores a Excel

Para que se active la Excel el usuario debe tener activado que puede enviar datos de pantalla a Excel en el programa de personalizar estética.

Posicionado de una lista de valores en Pantalla

Por norma general una lista de valores se centrará en la pantalla del programa que la llama. Si el puesto que está ejecutando el programa está marcado como de tipo “Pocket” la lista de valores se posicionará en la posición 0,0. En el caso de que una determinada lista de valores se quiera posicionar en un área específica de la pantalla se pueden indicar las coordenadas mediante los campos “Posición X” y “Posición Y”.

Consulta para obtener la descripción

La lista de valores puede contener la consulta que deben de ejecutar los programas para obtener la descripción a mostrar en los programas. En el campo “Nombre Columna Consulta Descripción” se introducirá la consulta necesaria para calcular la descripción. Debido a que en este punto no se conocen los campos con los que tiene que enlazar de la tabla del programa hay que utilizar variables entre los comodines {}.

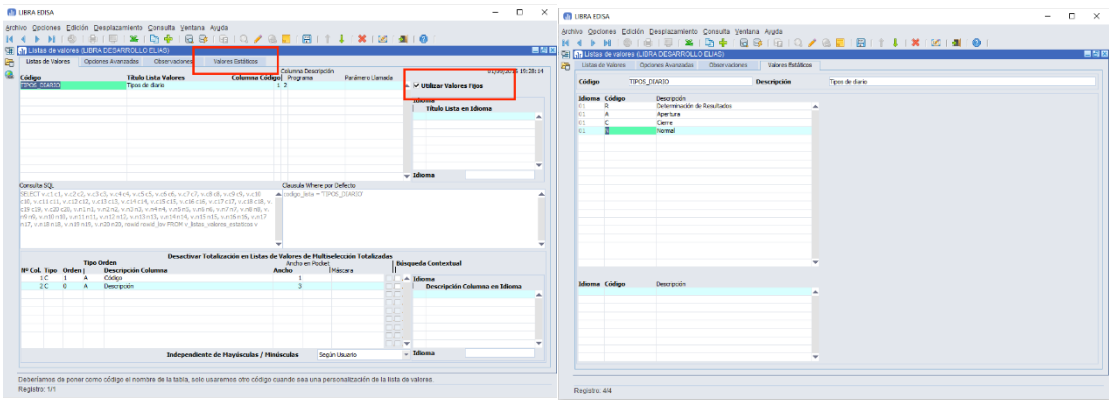
Ejemplo: `(SELECT f.descripcion FROM crmfamilias_lin f WHERE f.codigo={codigo_estad4} AND f.numero=4 AND f.tipo='X' AND f.empresa={empresa})`

En el programa hay que enlazar las variables *{codigo_estad4}* y *{empresa}* con los campos correspondiente, por ejemplo “*crmexpedientes_cab.codigo_estad4*” y “*crmexpedientes_cab.empresa*”. La forma de realizar este enlace se explica en el apartado *Campos de visualización de descripciones*.

Listas de valores con valores estáticos

Se pueden crear listas de valores que lleven los registros a mostrar incorporados en la propia lista, para lelo, hay que la check "Utilizar Valores Fijos" y aparecerá una pestaña donde indicar los valores posibles de la lista de valores para cada una de sus columnas.

En este tipo de lista de valores no se podrá modificar la consulta SQL ni la Cláusula Where, para ello esos campos se deshabilitan.



Listas de Valores por grupos.

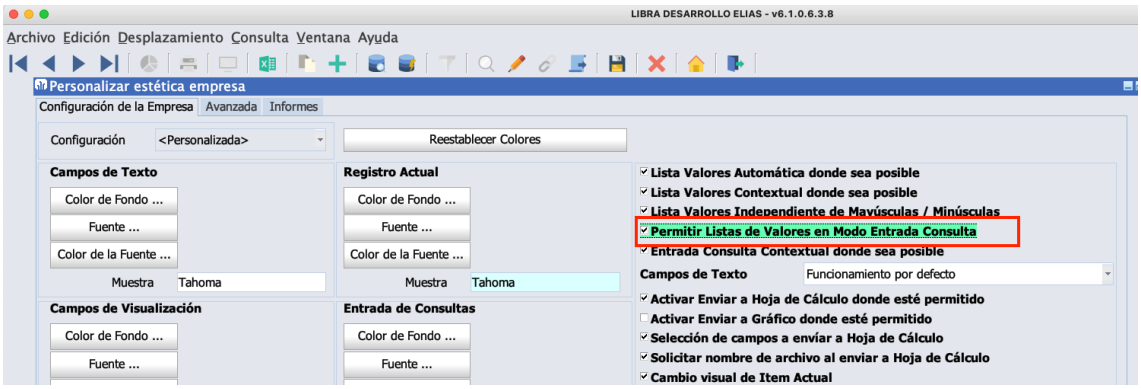
Son un tipo de listas de valores que muestran los registros en grupos de 9 en 9 ó de 5 en cinco y se puede seleccionar el registro que se quiere pulsando un número entre 1 y 9 ó 1 y 5 dependiendo del tipo. Ejemplo:

Para activarlas hay que meter en el fuente el componente "LISTA_VALORES_GRUPO" de la librería de objetos y en el mantenimiento de programas o programas personalizados indicar el tipo de lista de valores por grupo en el desplegable "Tipo Lista Valores" en la pestaña "Campo".

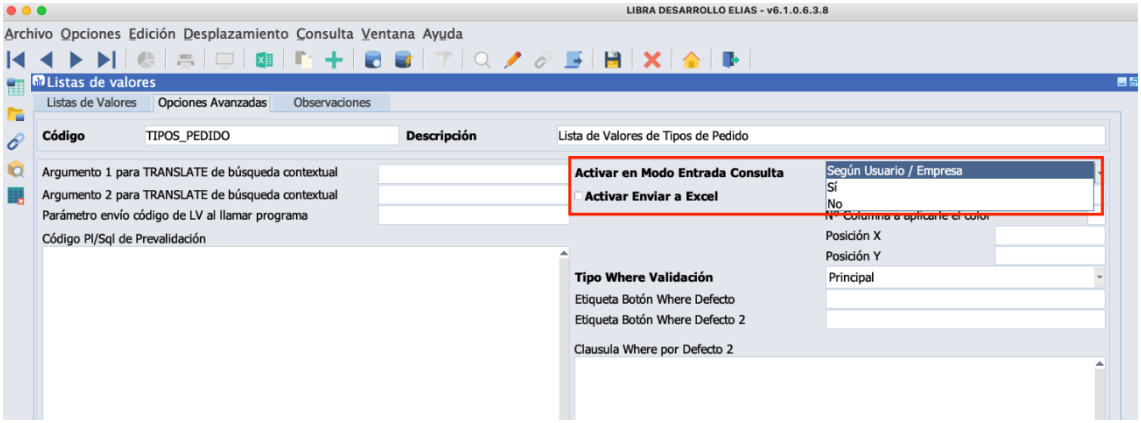
Listas de valores en modo Entrada Consulta

Cuando se está en el modo de entrada consulta de un bloque y se lanza la lista de valores únicamente se está aplicando la cláusula where definida en la lista de valores siendo ignorada la where indicada en el campo. Esto es debido a que en modo entrada consulta no hay ningún tipo de validación y por lo tanto se puede ir a un campo en dónde la where filtra por campos anteriores y al estar estos a NULL nunca mostraría nada.

Se puede desactivar a nivel de empresa / usuario



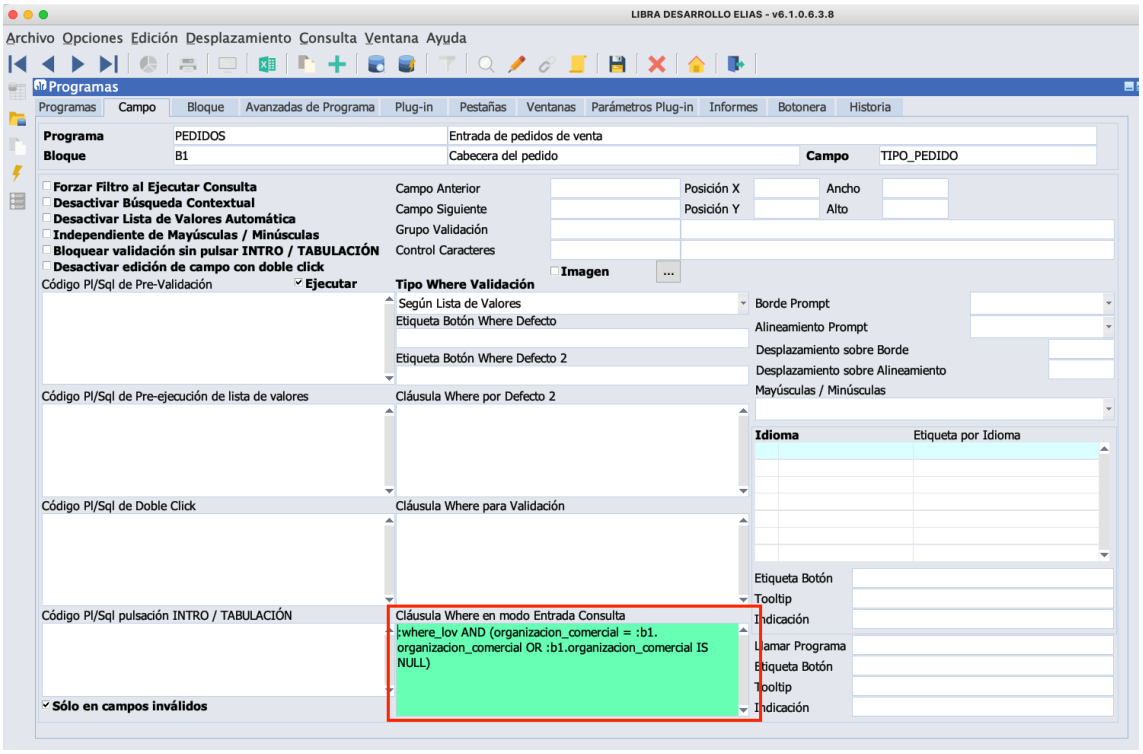
También se puede deshabilitar a nivel de lista de valores:



Dónde:

- **Según Usuario / Empresa**: Asumirá la parametrización por Usuario / Empresa.
- **Sí**: Se permite siempre independientemente de lo que esté parametrizado por Usuario / Empresa.
- **No**: Nunca se permite, independientemente de lo que esté parametrizado por Usuario / Empresa.

También a nivel de campo se puede indicar la cláusula where a aplicar en el caso de estar en modo entrada consulta. En esta cláusula where ya se podría controlar si los campos anteriores tienen valor o no para filtrar por ellos o no según se quiera. Si se filtra por un bloque padre no hay necesidad de hacer ese tipo de control ya que Forms entra en modo entrada consulta por bloque y no a nivel de programa completo. Por ejemplo, en el campo TIPO_PEDIDO del programa PEDIDOS se podría meter la siguiente where: "*:where_lov AND (organizacion_comercial = :b1.organizacion_comercial OR :b1.organizacion_comercial IS NULL)*" de forma que si el usuario indicó una organización comercial se filtrará por ella y en caso de que vaya directamente al campo tipo de pedido se mostrarán todos los tipos de pedido independientemente de la organización comercial a la que pertenezcan.



Funciones para gestión de la Lista de Valores

Todas estas funciones están implementadas en el paquete LV

- **LV.ACTIVA**(<lista de valores>, <ejecutar consulta> [,cláusula where], [consulta sql]): Se llama cada vez que se llega a un campo con lista de valores, para habilitar el botón y el menú correspondiente. En este paso ya se indica la lista de valores asociada al campo, si se ejecuta automáticamente consulta al ejecutarla o si tiene una cláusula WHERE específica.
  - <lista de valores>: Código de la lista de valores.
  - <ejecutar consulta>: Hay dos opciones posibles:
    - S: Cuando se lanza la lista de valores se ejecuta consulta automáticamente.
    - N: No se ejecuta la consulta y se queda en modo de entrada consulta.
  - [cláusula WHERE]: Este parámetro es opcional, si no se especifica se asume que dejamos que la lista de valores use la cláusula WHERE por defecto.
  - [Consulta sql]: Este parámetro también es opcional, si no se especifica asume que dejamos la sql que tiene la lista de valores.
- **LV.DESACTIVA**: Deshabilita opción de menú, iconos, ...
- **LV.LLAMADA**: Ejecuta la lista de valores, configura la ventana con los campos necesarios, construye la sentencia SQL, se la asigna al bloque, ...
- **LV.ROW_ID**: Si el usuario seleccionó una fila esta función nos devuelve el rowid de la fila seleccionada, si el usuario canceló la lista de valores devolverá NULL.
- **LV.VIENE_DE_LISTA**: Devolverá un dato de tipo BOLEAN, con el valor TRUE si desde que entramos en el campo se ha ejecutado la lista de valores, en caso contrario devolverá FALSE.
- **LV.ESTABLECER_BOTON_LISTA**: La llamada a este procedimiento es opcional. Ver "Indicar el botón de llamada a la lista de valores".

Indicar el botón de llamada a la lista de valores

Hay casos, como por ejemplo en ventanas flotantes, en los que el botón que tiene que pulsar el usuario es un botón que se encuentra en la ventana flotante en vez del botón de la botonera principal.

Para indicar el botón que ejecutará la lista de valores se usará **LV.ESTABLECER_BOTON_LISTA('<bloque>.<botón>);** antes de la llamada a **DISPSTD.WHEN_NEW_ITEM_INSTANCE**, por lo que en los bloques en que suceda este caso habrá que personalizar el disparador WHEN-NEW-ITEM-INSTANCE.

Ejemplo: Si tenemos un botón que lanza la lista de valores en el bloque B8 y el nombre del botón es LISTA_VALORES, personalizaremos en el bloque B8 el disparador WHEN-NEW-ITEM-INSTANCE con el siguiente código:

```
LV.ESTABLECER_BOTON_LISTA('B8.LISTA_VALORES');
DISPSTD.WHEN_NEW_ITEM_INSTANCE;
```

Un botón que se utilice para llamar a una lista de valores debe tener las siguientes propiedades:

- **Activado**: No
- **Teclado de Navegación**: No
- **Navegación del Ratón**: No

Disparadores

Solo se deberán de usar la llamada fija a la lista de valores en el programa en casos muy excepcionales, donde por cualquier motivo no sea efectiva la definición para el campo de la lista de valores en el mantenimiento de programas.

Trataremos de definir **todos los disparadores a nivel de bloque** en vez de a nivel de ítem, y usaremos la variable **:system.trigger_item** para saber a qué ítem nos estamos refiriendo. Esto nos da muchas más posibilidades para poder añadir código genérico y facilita enormemente la depuración.

WHEN_NEW_ITEM_INSTANCE

Para indicar de forma fija la lista de valores que va a usar un ítem usaremos el procedimiento LV.ACTIVA (Se puede ver la definición en la sección Funciones para gestión de la lista de valores). Siempre que sea posible definir la lista de valores en el mantenimiento de programas se indicará ahí la lista de valores a usar.

```
DISPSTD.WHEN_NEW_ITEM_INSTANCE;
IF :system.trigger_item = 'CAMPOS.ESTADO' THEN
  LV.ACTIVA('ESTADOS', 'S');
ELSIF :system.trigger_item = 'CAMPOS.PROVINCIA THEN
  LV.ACTIVA('PROVINCIAS', 'S', 'estado = campos.estado');
END IF;
```

ATENCIÓN: Nunca deberíamos de cerrar la posibilidad de que para campos que no tengan actualmente lista de valores se le pueda asignar una desde el mantenimiento de programas, para eso es imprescindible ejecutar el procedimiento estándar DISPSTD.WHEN_NEW_ITEM_INSTANCE. Deberemos ejecutarlo antes que los LV.ACTIVA para que el disparador estándar no cambie la activación de la lista de valores.

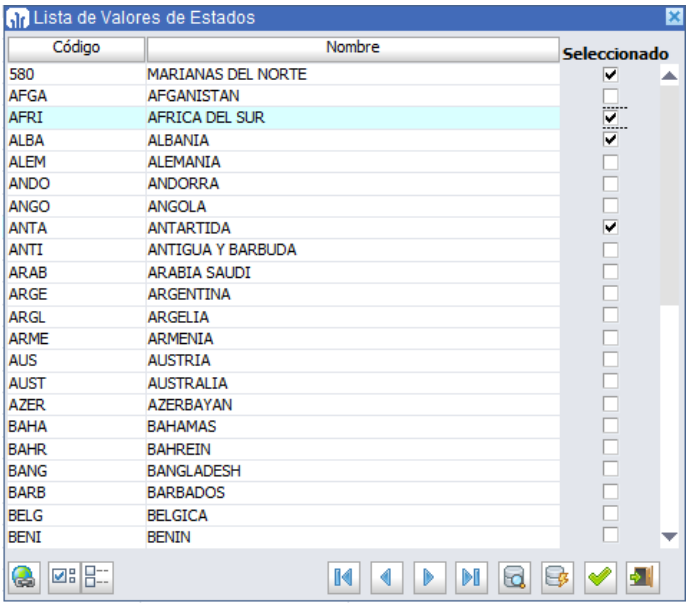
WHEN-VALIDATE-ITEM

```
DISPSTD.WHEN_VALIDATE_ITEM;
IF :system.trigger_item = 'CAMPOS.ESTADO' THEN
  IF :campos.estado IS NOT NULL THEN
    BEGIN
     SELECT nombre
       INTO :campos.d_estado
       FROM estados
      WHERE codigo = :campos.estado;
    EXCEPTION
      WHEN NO_DATA_FOUND THEN
        :campos.d_estado := NULL;
        MSG.MENSAJE('CAMPO', 'NO_VALID');
    END;
  ELSE
    :campos.d_estado := NULL;
  END IF;
END IF;
```

Al igual que en WHEN_NEW_ITEM_INSTANCE no se debería de anular la posibilidad de usar el mantenimiento de programas para realizar la validación de otros campos. Podríamos desde el mantenimiento de programas asignar la lista de valores, pero desactivar la opción de validar desde lista y dejar fijo el código de validación en el fuente del programa. También se podría usar la validación de la lista de valores y luego en el código fuente del programa añadir una restricción a mayores, etc.

Listas de valores de Multiselección

Las listas de valores de multiselección se diferencian de las listas de valores normales en que el usuario puede seleccionar varios registros de la lista. En este tipo de lista de valores aparece un campo de tipo check para que el usuario marque las filas que quiere seleccionar.

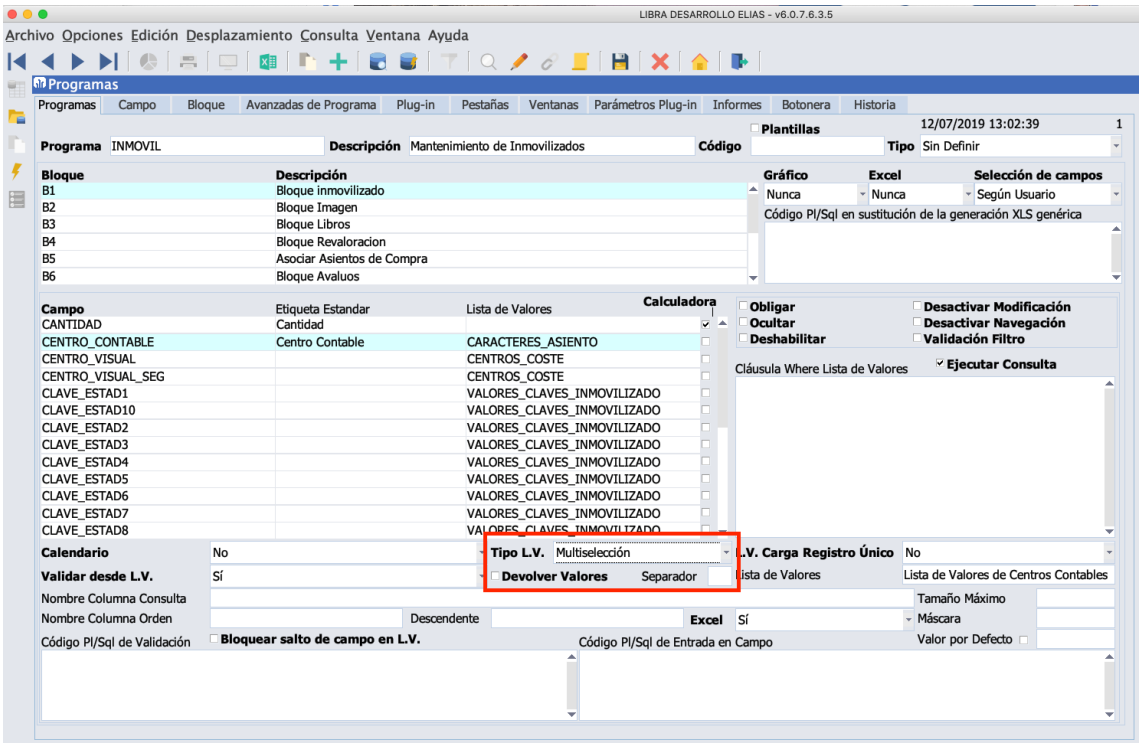


Para activar la multiselección hay que indicar en la pestaña Campo en el campo “Tipo Lista Valores” una de estas opciones:

- **Multiselección**:
- **Multiselección Totalizada**: Se diferencia de la anterior de que las columnas numéricas se les añade un total en donde se suman los registros que ha seleccionado el usuario.

Cuando se indica en el campo que es una lista de valores de multiselección aparecen 2 campos nuevos:

- **Separador**: Carácter que se utiliza para separar los valores, si no se indica nada se asume “,” (coma). En caso de poner el código de separador **R**, asumirá que tiene que devolver el valor en registros nuevos, es decir, en vez de concatenarlos en el campo que se llama a la lista de valores, por cada valor seleccionado creará un registro nuevo. Es importante que en caso de usar esta opción el campo que tiene la lista de valores sea el único campo obligatorio y si hay campos obligatorios se carguen por código PL/SQL de validación de registro o del campo que tiene la lista de valores, en caso de dar un error de validación se cancelará en ese momento y no se crearán más registros.
- **Devolver Valores**: Si se activa, cuando el usuario valida la lista de valores los códigos de los registros seleccionados se devuelven concatenados y separados por el carácter indicado en “Separador”.



Observaciones para el caso de activar la lista de valores en el mantenimiento de programas:

- Si el campo que llama a la lista de valores de multiselección, es validado desde lista de valores, los valores introducidos manualmente también serán validados y únicamente, si todos los valores son válidos se deja salir del campo.
- Si se valida desde lista de valores y el campo tiene asociado un campo descripción "D_XXX" en ese campo se meterán las descripciones de los valores y se utilizará el mismo separador.
- Si se valida desde lista de valores al llamar a la lista de valores desde un campo que ya tiene algún valor se marcarán las checks de los registros correspondientes.
- No se va a permitir seleccionar más valores de los que pueden entrar en el campo que llama a la lista de valores.

Las listas de valores también se pueden gestionar por código dentro del programa para ello se dispone de las siguientes funciones y procedimientos:

Procedimiento para activar una lista de multiselección.

- **LV.ACTIVAR_MULTISELECCION**: Modifica el funcionamiento de la próxima lista de valores que se active para que esta sea de multiselección. Esta llamada se pondrá antes de DISPSTD.WHEN_NEW_ITEM_INSTANCE.

Funciones para procesar los registros seleccionados por el usuario.

- **LV.PRIMER_REG_MULTISELECCION**: Devuelve el *rowid* del primer registro seleccionado por el usuario, si no ha seleccionado ninguno devolverá NULL.
- **LV.SIGUIENTE_REG_MULTISELECCION**: Devuelve el *rowid* del siguiente registro seleccionado por el usuario al anterior que nos ha devuelto esta función o la función PRIMER_REG_MULTISELECCION. Si ya no quedan más registros devolverá NULL.
- **LV.BORRAR_MULTISELECCION**: Borra toda la selección realizada por el usuario.
- **LV.CAMPO_LOV**: Devuelve el campo desde el que se ha llamado la lista de valores.

Ejemplo de lista de valores de multiselección.

- **WHEN-NEW-ITEM-INSTANCE**: Hay dos formas de indicar que un campo tiene lista de valores de multiselección
  - Activamos la lista de valores en el código del programa, por ejemplo.

```
DISPSTD.WHEN_NEW_ITEM_INSTANCE.
IF :system.trigger_item = 'PRUEBA.CODIGO' THEN
  LV.ACTIVAR_MULTISELECCION;
  LV.ACTIVA('ESTADOS', 'S');
END IF;
```

  - La lista de valores se establece en el mantenimiento de programas (siempre que se pueda trataremos de usar este método), en el código solo indicamos que es de multiselección, por ejemplo:

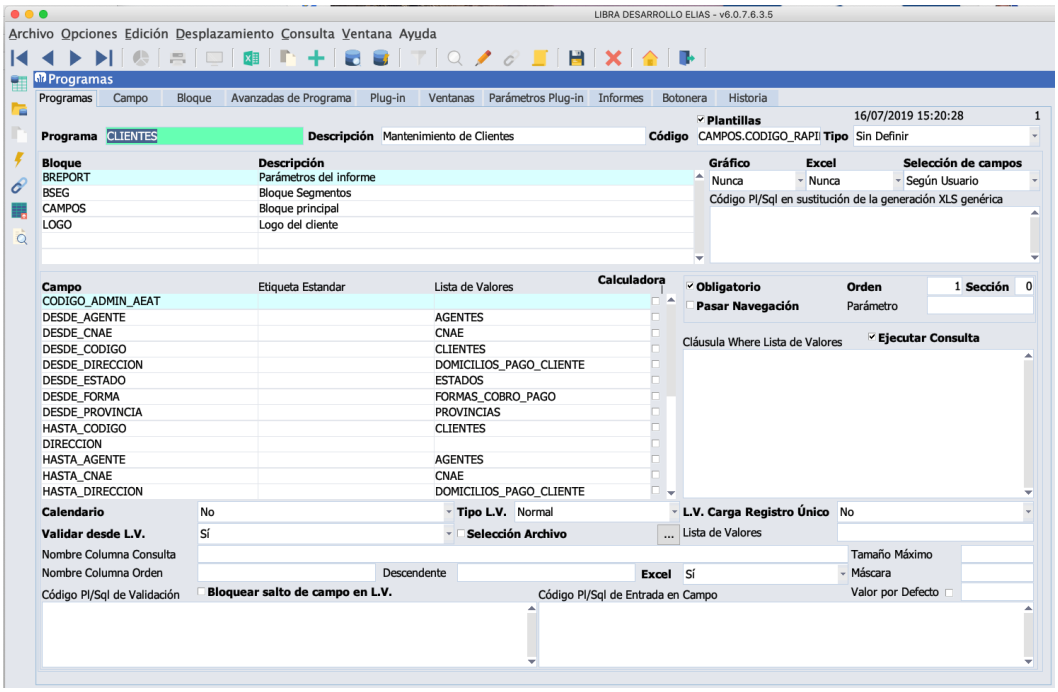
```
DISPSTD.WHEN_NEW_ITEM_INSTANCE.
IF :system.trigger_item = 'PRUEBA.CODIGO' THEN
  LV.ACTIVAR_MULTISELECCION;
END IF;
```

- **WHEN-NEW-RECORD-INSTANCE**: Procesamos la selección del usuario, sólo cuando se viene de la lista de valores del campo sin ser cancelada.

```
DECLARE
  v_rowidVARCHAR2(30);
BEGIN
  DISPSTD.WHEN_NEW_RECORD_INSTANCE;
  IF lv.viene_de_lista AND lv.row_id IS NOT NULL
          AND lv.campo_lov = 'PRUEBA.CODIGO' THEN
    v_rowid := LV.PRIMER_REG_MULTISELECCION;

      WHILE v_rowid IS NOT NULL LOOP
          SELECT codigo
            INTO :prueba.codigo
            FROM estados
           WHERE rowid = v_rowid;

      NEXT_RECORD;
      v_rowid := LV.SIGUIENTE_REG_MULTISELECCION;
    END LOOP;

    LV.BORRAR_MULTISELECCION;
  END IF;
END;
```

Programa para mantener programas !!!!!!

Aunque parezca un juego de palabras es la mejor descripción para el Mantenimiento de Programas del E.R.P.



Desde este programa se pueden realizar operaciones que pueden modificar el comportamiento de los programas sin necesidad de modificar ni compilar, simplemente saliendo y volviendo a entrar en el programa modificado ya se asumen los nuevos cambios.

NOTA: No es necesario introducir todos los campos de un bloque, únicamente aquellos que tengan la necesidad de activar alguna funcionalidad en particular.

El valor de la check "Plantillas" a nivel de programa, "Campo Código" y el apartado Específicos Programas Dinámicos de cada campo, son para uso exclusivo de programas que sean dinámicos (basados en plantillas).

Operaciones que se pueden realizar a nivel de programa

- **Tipo**: Define la tipología del programa:
  - **Consulta**: Es un programa de consulta, por tanto, algunas listas de valores cambian el comportamiento de los registros a mostrar. Por ejemplo, en una consulta de artículos se podrán ver artículos inactivos, mientras que si es un movimiento de almacén los artículos inactivos no pueden visualizarse.
  - **Plug-In:** El programa está pensado exclusivamente para ser utilizado como plug-in, por tanto, no es necesario que se encuentre en el menú del usuario para ser utilizado.
  - **Web:** Programa de movilidad.
  - **Ejecuta Metadatos:** No necesita ejecutable compilado para ejecutarse, con la información que se encuentra definida a nivel de metadatos es suficiente para ser ejecutado.
  - **Plug-In Ejecuta Metadatos**: Igual que "Ejecuta Metadatos" y no se requiere una entrada en el menú para que el usuario pueda ejecutarlo.
  - **Filtro Metadatos**: Es un programa que no está asociado a tabla y no necesita ejecutable. Este programa permite pedir datos al usuario por pantalla para luego se utilizados por algún proceso.
  - **Plug-In Filtro Ejecuta Metadatos**: Igual que "Filtro Metadatos" y no se requiere una entrada en el menú para que el usuario pueda ejecutarlo.

- **Código PL/SQL de Inicialización**: Sólo se ejecuta al entrar en el programa, es interesante para establecer propiedades que no se pueden cambiar de otra forma.
- **Código PL/SQL de Finalización**: Sólo se ejecuta al salir del programa.
- **Permitir Informes por...**: Mediante estas checks se puede configurar el destino que puede tomar un informe, si se deshabilitan todas al entrar en el programa también se deshabilita el botón de impresión.
- **Selección de Directorio en Informes**: En la pantalla de selección de destino de la impresión, si el usuario indica que desea el informe en archivo, el usuario puede indicar la ruta + el nombre del archivo, si se activa esta check quiere decir que el nombre de archivo se genera de forma automática dentro del programa, por lo tanto, el usuario únicamente debe de indicar el directorio en donde quiere obtener los archivos.
- **Grabar automáticamente al salir**: Si se activa, cuando hay cambios que están pendientes de ser grabados, al pulsar el botón de salir esos cambios se graban automáticamente sin preguntar al usuario.
- **Ejecutar Consulta al Entrar**: Si se activa, en le primer bloque navegable del programa se ejecutará consulta al entrar en él por primera vez. Esa consulta se ejecutará siempre y cuando no se hubiese ejecutado ya consulta por la lógica del programa en el disparador INICIO y que el bloque tenga la propiedad de QUERY_ALLOWED a TRUE.
- **Icono**: Cuando el programa se utiliza como plug-in de otro se propone este icono para usarlo como icono del plug-in.
- **Selección Manual de Plantilla Inicial**: Únicamente es visible cuando el programa es "dinámico". Permite indicar que al entrar en el programa se debe de solicitar al usuario la plantilla a usar en vez de aplicar de forma automática la última plantilla que seleccionó el usuario.
- **Código PL/SQL de grabación**: Este código se ejecuta cuando el usuario graba o sale del programa grabando las modificaciones. No se ejecuta por cada registro, se ejecuta por cada grabación, por tanto, si queremos que el usuario grabe por registro para que se ejecute este código hay que usarlo en conjunto con el bloqueo de salida hasta grabar a nivel de bloque.
- **Código PL/SQL Botón Impresión**: Se ejecuta cuando el usuario pulsa sobre el botón de imprimir, y se puede indicar en qué punto exacto se ejecuta en la lista "Punto ejecución PL/SQL Botón Impresión":
  - **Antes de navegar a la pantalla de Impresión**: Es lo primero que ejecuta, antes de ir a la pantalla de filtros de informe, de esta forma se podría anular esa pantalla y llamar a un plug-in con otro frontal de informes distinto.
  - **Después de Navegar a la pantalla de Impresión**: Se ejecuta después de mostrar la pantalla de filtros de informes, después de poner los valores por defecto de destino, impresora, ..., por lo que desde el código PL/SQL se pueden alterar.
  - **Antes y Después de Navegar a la pantalla de Impresión**: Se ejecuta dos veces, una antes de navegar y otra después, si se quiere que haga cosas distintas se puede usar la variable :system.cursor_block, después de navegar contendrá el valor BREPORT.

  Si el programa no tiene informe y se activa mediante la check de Forzar Botón de Impresión el especificar que se ejecute Antes o Después de Navegar va a dar el mismo resultado ya que en ningún momento va a hacer esa navegación. Si se especifica que se ejecute Antes y Después se va a ejecutar dos veces.
- **Código PL/SQL para antes de ejecución del informe**: Se ejecuta cuando el usuario pulsa el botón de imprimir en la pantalla de filtros del informe, una vez seleccionado el destino, impresora, ... y se ejecuta justo antes de hacer la llamada al report, por lo que se podría usar para hacer una carga de una tabla temporal, modificar la impresora por la que se va imprimir o alterar el informe que se va a imprimir por ejemplo. Si se especifica prevalecerá sobre el código PL/SQL que tenga el usuario o la empresa.

Se puede establecer que se ejecute cuando el usuario selecciona un determinado destino, mediante los list-item que tiene debajo (Al ir a Pantalla, Impresora, Archivo, eMail, Fax y Gestión Documental) y tienen los siguientes valores:

- **Usuario**: Se ejecuta dependiendo de la parametrización del usuario en "Personalización de Estética" y si no tiene registro ahí según la parametrización de "Personalización de Estética por Empresa", en caso de no haber tampoco registro se asume que Si se ejecuta.
- **Sí**: Se ejecuta independientemente de si el usuario/empresa tiene que no se ejecuta.
- **No**: No se ejecuta independientemente de si el usuario/empresa tiene que no se ejecuta.

Como particularidades de este código PL/SQL es que se pueden leer las siguientes variables:

- **PKPANTALLAS.GET_VARIABLE_ENV_VARCHAR2('IMP_INFORME')**: Devuelve el nombre del informe que se va a ejecutar.
- **PKPANTALLAS.GET_VARIABLE_ENV_VARCHAR2('IMP_DESNAME')**: Devuelve a donde se enviará el informe, en caso de que el destino sea impresora, devolverá el nombre de la impresora a la que se envía, en caso de ser algún tipo de archivo devolverá el nombre del archivo que se va a generar con la ruta completa.
- **PKPANTALLAS.GET_VARIABLE_ENV_VARCHAR2('IMP_DESTINO_EXCEL')**: Si se envía a Excel indica el nombre de archivo con la ruta completa que se va a generar.
- **PKPANTALLAS.GET_VARIABLE_ENV_VARCHAR2('IMP_REPORTS60_TMP')**: Valor de la variable REPORTS60_TMP del libra6.ini.
- **PKPANTALLAS.GET_VARIABLE_ENV_VARCHAR2('IMP_DISPOSITIVO')**: Dispositivo de salida del informe, posibles valores SCREEN, PRINTER, FILE, MAIL, FAX, GESTODOC.

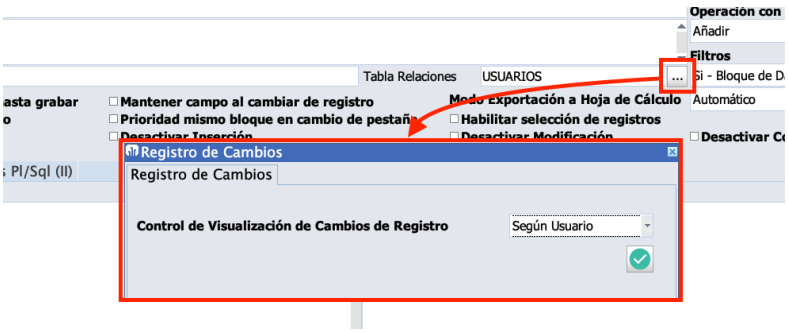
  Otras particularidades de este código PL/SQL es que se puede establecer la impresora o el nombre del archivo a generar (dependiendo del destino del informe) estableciendo PKPANTALLAS.SET_VARIABLE_ENV('IMP_DESNAME', 'valor');. También se puede cambiar el informe a ejecutar estableciendo PKPANTALLAS.SET_VARIABLE_ENV('IMP_INFORME', 'valor');

- **Código PL/SQL para después de ejecución del informe**: Se ejecuta una vez se ha finalizado la ejecución del informe, y al igual que en el en "Código PL/SQL para antes de Ejecutar Informe" se puede indicar que se ejecute cuando el destino sea uno en concreto. En este código es especialmente importante ya que si se va a ejecutar invalida la impresión en segundo plano en caso de estar activada.

  Como particularidades de este código PL/SQL es que se pueden leer las siguientes variables:
  - **PKPANTALLAS.GET_VARIABLE_ENV_VARCHAR2('IMP_INFORME')**: Devuelve el nombre del informe ejecutado.
  - **PKPANTALLAS.GET_VARIABLE_ENV_VARCHAR2('IMP_DESNAME')**: Devuelve a donde se mandó el informe, en caso de que el destino sea impresora devolverá el nombre de la impresora a la que se envío, en caso de ser algún tipo de archivo devolverá el nombre del archivo generado con la ruta completa.
  - **PKPANTALLAS.GET_VARIABLE_ENV_VARCHAR2('IMP_DESTINO_EXCEL')**: Si se envía a Excel indica el nombre de archivo con la ruta completa que se generó.
- **Grupos de Repetición**: Los grupos de repetición permiten definir campos que el usuario deberá teclear tantas veces como indique el campo "Nº". Luego a nivel de campo (en la pestaña campo), a los campos que se quieran incluir en el grupo de repetición hay que cubrir el campo "Grupo Validación".
  - **Código**: Código del grupo de repetición.
  - **Descripción**: Descripción del grupo.
  - **Nº**: Número de veces que el usuario tiene que introducir la información para validar el grupo de repetición.

Operaciones que se pueden realizar a nivel de bloque

- **Cambiar el origen de datos del bloque para consulta, campo "Consulta"**: En este campo se puede indicar de donde tiene que obtener Oracle los datos para mostrar en el bloque. Puede ser una tabla o una consulta. En caso de ser una consulta se debe de poner entre paréntesis.
- **Cambiar la condición de visualización de registros de un bloque**: A los bloques se les puede añadir una condición a mayores de la que tiene el programa, para ello se cubrirá el campo Where Inicial de la sección de bloques. La forma en que se introduce la condición al bloque depende del list-item "Operación con Where Inicial":
  - **Añadir**: Se añade la condición a la que tenga el programa en el fuente.
  - **Sustituir**: Se ignora la condición que tenga el programa en el fuente y se utiliza únicamente la del mantenimiento de programas.
- **Cambiar la ordenación de un bloque**: A los bloques se les puede modificar el tipo de ordenación que realiza, para ello se cubrirá el campo Ordenación Inicial. NOTA: Esta ordenación sustituye a la que tenga el programa. Si dentro del programa se vuelve a asignar la propiedad ORDER_BY del bloque se perderá esta asignación.
- **Tabla Relaciones**: Por defecto se controla el borrado de registros o modificación de campos clave primaria en base a las relaciones asociadas a la tabla del bloque, pero puede haber casos donde el bloque está asociado a una vista, pero el control de las relaciones interesa gestionarlas por las definidas en una tabla, en este caso se puede indicar en este campo la tabla con la que se tienen que verificar las relaciones. Esta tabla también será utilizada para gestionar las auditorías de cambios. Al introducir una tabla en este campo se puede configurar los permisos de visualización de esas auditorías, por lo general lo más adecuado es dejar el valor por defecto "Según Usuario".



- **Enviar a Excel el contenido del bloque**: Si se activa en el bloque la opción "Excel", cuando el cursor entre en ese bloque se activa en el menú una opción para enviar el contenido directamente a Excel.
  - **Según Usuario**: Solo se activará esta posibilidad si en la personalización del usuario tiene activado el envío a hoja de cálculo.
  - **Nunca**: No se permite volcar el contenido de bloque a hoja de cálculo independientemente de la configuración del usuario.
  - **Siempre**: Se permite el volcado a hoja de cálculo independientemente de la configuración del usuario.
- **Selección de campos a enviar a Excel**: Permite configurar el comportamiento que tendrá cuando el usuario pulse el botón de envío a hoja de cálculo, pudiéndose indicar que no se soliciten los campos a enviar a hoja de cálculo, de forma que se envían todos.
- **Código PL/SQL en sustitución de la generación XLS genérica**: Este código PL/SQL, en el caso de introducirse, se ejecutará cuando el usuario pulse sobre el botón de envío a hoja de cálculo, pudiéndose meter las llamadas necesarias a PKXLSBD para realizar la hoja de cálculo de forma totalmente específica para ese bloque.
- **Código PL/SQL de validación**: Código PL/SQL que se ejecuta cuando se valida un registro completo.

- **Código PL/SQL de Post Inserción**: Se ejecuta después de haberse insertado el registro en la base de datos.
- **Código PL/SQL de Post Actualización**: Se ejecuta después de haberse modificado el registro en la base de datos.
- **Código PL/SQL de Pre borrado**: Se ejecuta cuando se intenta borrar un registro. Si el resultado de la ejecución termina con :p_parar_ejecucion con el valor 'S' se evita el borrado del registro.
- **Código PL/SQL de Post Borrado**: Se ejecuta una vez se ha borrado el registro.
- **Código PL/SQL de inicialización**: Se ejecuta cada vez que se crea un registro, de esta forma se puede asignar valores por defecto en base a determinadas condiciones. Se debería utilizar siempre y cuando el valor por defecto no se pueda asignar usando el campo "Valor por defecto", ya que el uso del campo "Valor por defecto" no hace intervenir a la base de datos.
- **Código PL/SQL de Entrada en Bloque**: Se ejecuta cuando el cursor entra en el bloque.
- **Código PL/SQL de Entrada en Registro**: Se ejecuta cada vez que se entra en un registro nuevo o se cambia de registro en el bloque.
- **Código PL/SQL – Antes de consultar registros**: Se ejecuta antes de realizar la consulta que realiza el bloque para rellenarlo de información.
- **Código PL/SQL de Consulta de Registro**: Código que se ejecuta por cada registro que se consulta de la base de datos. **IMPORTANTE**: Al ejecutarse por cada registro que se trae de la base de datos y al ejecutarse el código PL/SQL en la base de datos se va a incrementar el tráfico de red ralentizando la consulta.
- **Bloquear salida hasta grabar**: Si se activa la check y el usuario modifica algo, este bloque va a bloquear la salida del registro en que se encuentra el usuario hasta que no grabe o no lo borre o borre el registro de un bloque padre.

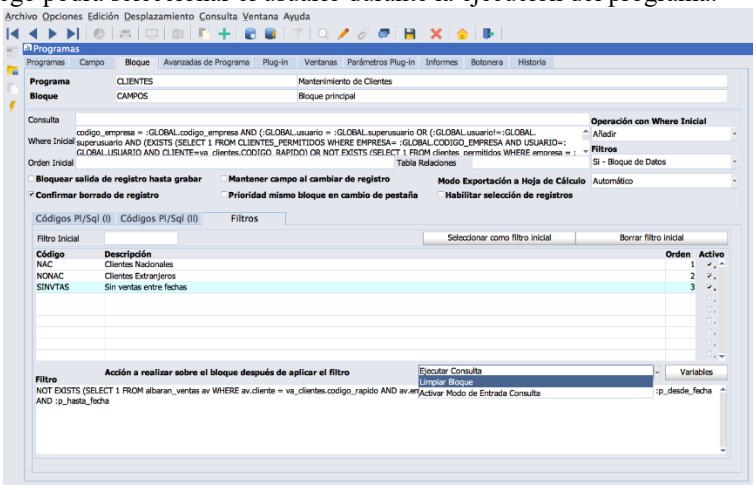
  Por ejemplo, si lo activamos en la entrada de pedidos en el bloque B1 (cabecera), en cuanto el usuario modifique algo no se le va a dejar salir del pedido mientras no lo grabe o no lo borre.

  Si lo activamos para un bloque que tiene padres, por ejemplo, líneas de pedido, habría que activarlo también para sus padres ya que si no lo hacemos evitamos que salga de las líneas, pero puede ir a la cabecera y cambiarla lo que produce un cambio en las líneas.

  Si se activa en un bloque multilínea y el usuario intenta salir con el ratón a otro registro se navega de nuevo al registro en que estaba.
- **Prioridad mismo bloque en cambio de pestaña**: El funcionamiento normal de cuando el usuario pulsa en una pestaña es buscar el primer campo navegable de la pestaña de destino, ese campo podría ser de otro bloque. Si se activa esta check primero mira si en la pestaña de destino hay un campo navegable del bloque en que se encuentra el cursor, si lo hay va a ese campo y si no hay busca el primer campo navegable de la pestaña de destino sea del bloque que sea.
- **Mantener campo al cambiar de registro**: Si se navega al registro anterior o al registro siguiente se intentará mantener el cursor en el mismo campo. Si el registro al que se navega es nuevo se irá al primer campo navegable.
- **Confirmar borrado de registro**: Si está activado antes de borrar un registro se pide confirmación al usuario, si no está activado se borra directamente.
- **Modo Exportación a Hoja de Cálculo**:
  - **Automático**: Dependiendo de los campos que desea exportar el usuario a Hoja de Cálculo, el entorno de Libra puede determinar que el mejor camino para realizarlo es volver a ejecutar la misma consulta que hizo el bloque para rellenarse.

- **Forzar recorrer bloque**: Esta opción le indica al entorno que debe de recorrer el bloque para obtener los datos para generar la hoja de cálculo en vez de realizar otra consulta contra la base de datos.
- **Forzar conexión directa**: Si el bloque no usa tablas temporales y paquetes con variables de sesión y generalmente el número de registros a exportar va a ser alto, con esta opción pude mejorar la velocidad de exportación.
- **Bloquear conexión directa:** Debe de utilizarse en el caso de que el bloque esté asociado a una tabla temporal o que utilice funciones que dependan de variables de sesión.

- **Búsqueda Contextual**: En Forms 14 indica si se debe de habilitar o no la búsqueda contextual del bloque.
  - **Según Usuario**: El bloque permite la búsqueda contextual, pero será la parametrización de grupo empresarial / usuario quien determine si se debe de activar o no.
  - **Siempre**: Independientemente de la configuración del grupo empresarial / usuario el usuario podrá ejecutar la búsqueda contextual.
  - **Nunca**: Independientemente de la configuración del grupo empresarial / usuario el usuario no podrá ejecutar la búsqueda contextual.
- **Habilitar selección de registros**: Esta check únicamente debe de ser activada en caso de bloques de tipo multiregistro. Permite al usuario a seleccionar con el ratón (mientras mantiene la tecla Control o Mayúsculas pulsada) varios registros. Una vez seleccionados varios registros se pueden borrar todos a la vez o lanzar un plug-in que se ejecute sólo para los registros seleccionados.
- **Filtros**: Permite configurar el bloque para que gestione dos tipos de filtros:
  - **Sí – Bloque de Filtro**: Se utiliza para indicar que el bloque es donde se introducen los filtros para luego ejecutar una consulta, al indicar esto el usuario puede grabar los filtros utilizados para ser recuperados luego de forma fácil.
  - **Sí – Bloque de Datos**: Se mostrará una pestaña anidada donde se pueden definir filtros que luego podrá seleccionar el usuario durante la ejecución del programa.



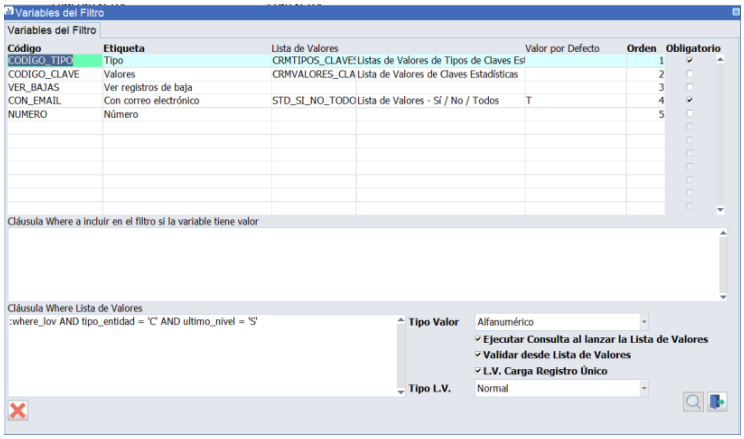
Se pueden definir tantos filtros como se quiera, el usuario cuando ejecute el programa y entre en el bloque va a tener un botón para abrir una ventana en donde indicar el filtro que quiere aplicar, y si tiene ya aplicado algún filtro una de las opciones será “Limpiar Filtros” para volver al estado original.

Se puede hacer que según se entre en el programa arranque el bloque con un filtro aplicado, para ello situamos el cursor en el registro que se quiera que sea inicial y se pulsa en el botón “Seleccionar como filtro inicial”, para desactivar el filtro inicial se pulsará en el botón “Borrar filtro inicial”. Un filtro que use variables no podrá ser utilizado como filtro inicial.

- **Código**: Identificador único del filtro, no será visible por el usuario.

- **Descripción**: Será lo que vea el usuario para identificar el filtro.
- **Orden**: Orden en que se mostrará al usuario para seleccionar el filtro a aplicar.
- **Activo**: El usuario solo podrá seleccionar aquellos filtros que tengan la check "Activo" marcada.
- **Acción a realizar sobre el bloque después de aplicar el filtro**: Indica que es lo que se debe de hacer sobre el bloque en el caso de que el usuario aplique un determinado filtro. Las acciones que se pueden indicar son:
    - **Ejecutar Consulta**: Es lo que se hacía siempre hasta esta versión y consiste en refrescar la consulta del bloque con el filtro aplicado.
    - **Ejecutar Consulta y no mantener bloque filtrado**: Con esta opción se hace la consulta según el filtro y se muestran los datos que cumplen la condición, pero el bloque no queda atado a ese filtro y si el usuario pulsa F7 o usa el botón de ejecutar consulta verá todos los registros, es decir, es equivalente a la opción "Ejecutar Consulta" y luego que el usuario vaya a "Limpiar Filtros"
    - **Limpiar Bloque**: Se aplica el filtro y se limpia todos los registros visualizados, el usuario tendrá que ejecutar consulta para ver los registros.
    - **Activar Modo de Entrada Consulta**: El bloque se queda en modo de entrada consulta esperando a que el usuario introduzca algún criterio a mayores a aplicar sobre alguno de los campos.
- **Filtro**: Será la condición que se añada al bloque, esa condición se añade de la siguiente forma: *AND (<condición>)*

    En el filtro se pueden utilizar variables que le serán solicitadas al usuario, esas variables se introducirán directamente en la condición añadiendo : (dos puntos) delante, por ejemplo, si queremos al usuario un rango de fechas, se podría meter algo similar a esto: *av.empresa = :global.codigo_empresa AND av.fecha_pedido BETWEEN :p_desde_fecha AND :p_hasta_fecha*. En este caso se indica que se quieren usar las variables :p_desde_fecha y :p_hasta_fecha, a esas variables hay que indicar la forma en la que se van a solicitar al usuario, para ello hay que pulsar en el botón "Variables".



- **Código**: Identificador de la variable, si en a condición se usó :p_desde_fecha, el código debe de ser P_DESDE_FECHA.
- **Etiqueta**: Texto que aparecerá junto al campo al generarse la pantalla de filtros del usuario.
- **Lista de Valores**: Código de la lista de valores que tendrá el campo del filtro.
- **Valor por Defecto**: Permite indicar un valor que aparecerá inicialmente al usuario y que podrá ser modificado.
- **Obligatorio**: Si se activa no se dejará realizar la consulta mientras el usuario no proporcione un valor para el filtro.

- **Cláusula Where a incluir en el filtro si la variable tiene valor**: Esta cláusula where únicamente será añadida cuando el usuario introduce algún valor en la variable y permite simplificar la consulta, sobre todo cuando la lista de valores des de tipo multiselección. En principio el contenido del filtro se añadirá al filtro principal añadiendo al final: "AND (+ la cláusula where de la variable +)", pero puede ser que interese que esa condición sea añadida a una parte en concreto de la where principal ya podría estar por ejemplo dentro de una subconsulta, en ese caso en la where principal se añadirá :CODIGO_VARIABLE y en el caso de que el usuario no cubra ese filtro :CODIGO_VARIABLE se quita y si el usuario cubre el filtro se reemplaza.
- **Cláusula Where Lista de Valores**: Filtro para los registros que visualizará la lista de valores. Este campo está asociado al campo "Lista de valores". Toda lista de valores puede tener asociada una cláusula "WHERE" para todos los programas, pero esa "WHERE" quedará anulada si en este campo se introduce una específica, de forma que se puede llegar a tener condiciones "WHERE" distintas en cada programa. Es muy recomendable añadir la expresión ":where_lov" que se reemplazará en tiempo de ejecución por la cláusula where que tenga la lista de valores, de forma que un cambio de la where en la lista de valores será traslada a todos los programas. Para hacer referencia en esta cláusula where a otra variable que se pida antes (que tenga un orden inferior) debe de usarse ":CODIGO_VARIABLE". No debe de usarse el campo interno usado, por ejemplo "BFILTROS.FILTRO_ALFA2" ya que si por algún motivo se cambia el orden o se añaden nuevas variables ese campo va a cambiar.
- **Tipo Valor**: Permite indicar si el dato es "Alfanumérico", "Numérico", "Fecha" o de tipo "Check".
- **Ejecutar Consulta al lanzar la Lista de Valores**: Si se indica "Lista de Valores" y se desactiva esta check al lanzar la lista de valores se iniciará en modo de entrada consulta, es decir, se inicia esperando que el usuario proporcione un filtro inicial.
- **Validar desde Lista de Valores**: Si se indica "Lista de Valores" y esta check está activa, únicamente se podrá introducir un valor de los que se puedan visualizar en la lista de valores.
- **Tipo L.V**.: Si el filtro tiene lista de valores permite indicar el tipo de lista de valores a utilizar. Si se indica un tipo de multiselección en la cláusula where del filtro hay que usar el operador IN o NOT IN. Si se selecciona la opción "Rellenar List-Item" el campo se mostrará en forma de List-Item con los valores que devuelva la lista de valores cargados.

Operaciones que se pueden realizar a nivel de campo

- **Cambiar etiquetas de campos**: En los programas, sean dinámicos o no, se puede modificar el texto de las etiquetas de los campos para posibilitar la traducción del E.R.P a otros idiomas y permitir en una instalación que estén usuarios con las pantallas en un idioma y otros usuarios con otro idioma. Los pasos que realizan los programas para obtener la etiqueta de un campo son los siguiente:
  - Programas dinámicos:
    - Busca la etiqueta en la personalización por idioma de los campos para la plantilla.
    - Si la etiqueta no está personalizada para la plantilla en el idioma del usuario se busca igual que en los programas no dinámicos.
  - Programas no dinámicos:
    - Buscar la etiqueta para el campo en el idioma del usuario. Sección Etiquetas por Idioma.
    - Si no tiene etiqueta en el idioma del usuario usa la etiqueta Estándar.
    - Si no tiene ninguna de las anteriores se mostrará la introducida en el código fuente del programa.
- **Habilitar hipervínculos a otros programas**: Si en el campo *Llamar programa* introducimos el nombre del fichero de un programa, cuando un usuario que tenga permisos para entrar en ese programa y se posicione en el campo se habilitará el botón de llamada directa y podrá navegar al

programa especificado. Si se especifica en este campo el programa a llamar prevalecerá sobre el programa que tenga asociado la lista de valores.

- **Calculadora**: Activando o desactivando la check del campo *Calculadora* haremos que cuando el usuario se encuentre en ese campo y pulsa sobre la lista de valores se abrirá una calculadora. **Solo se debería de activar en campos numéricos**.
- **Calendario**: Indicando "Sí" en cualquiera de las dos modalidades existentes, cuando el usuario se encuentre en ese campo y pulsa sobre la lista de valores se abrirá un calendario.

  El indicar que un campo tiene calendario desde el mantenimiento de programas lleva asociado que la validación se realizará como una fecha.
  - **Sí - Proponer fecha de trabajo si es obligatorio**: Si el campo es obligatorio y el usuario intenta dejarlo en blanco se cubrirá de forma automática con la fecha de trabajo (:global.fecha_trabajo).
  - **Sí - Sin proponer fecha de trabajo**: Si el campo es obligatorio y el usuario intenta dejarlo en blanco le obligará a introducir un valor manualmente.
- **Cambiar / Asignar lista de valores asociada a un campo**: Si en el campo Lista de Valores introducimos el código de una lista de valores, se activará la posibilidad de usar la lista de valores especificada en ese campo.
- **Ejecutar Consulta al Lanzar la L.V.**: Si está activada la check, indicamos que cada vez que se lance la lista de valores se ejecutará automáticamente consulta de la misma, en caso contrario se lanzará la lista de valores y se quedará a la espera de que el usuario introduzca un filtro y pulse F8.
- **Cláusula WHERE Lista de Valores**: La lista de valores puede tener asociada una cláusula WHERE para todos los programas, pero esa WHERE quedará anulada si en este campo introducimos una específica para el campo. Si se cubre este campo, esta where prevalecerá sobre la where especificada en la lista de valores. Si dentro de la where ponemos el identificador **:where_lov**, este será sustituido por la cláusula where original de la lista de valores, con lo que se logra una especie de herencia. **:where_lov2** será sustituido por la cláusula where 2 definida en la lista de valores, **:where_lovv** será sustituido por la cláusula where de validación definida en la lista de valores. Ejemplo:
  - **Where lista de valores**: empresa=:global.codigo_empresa
  - **Where programa**: tabla=1 AND :where_lov
  - **Resultado**: tabla=1 AND empresa=:global.codigo_empresa

NOTA: Se debería usar siempre que sea posible la herencia de la where de la lista de valores al programa, para que un cambio en la lista de valores original se propague a la where de todos los programas en donde se usa.

Se puede gestionar la cláusula where de la lista de valores en tiempo de ejecución al través del resultado de una función de base de datos. El comportamiento es muy parecido al explicado en el apartado: Listas de valores con las etiquetas :SF: y :EF:, pero para indicar que la función debe de evaluarse en cada ejecución vez ve de una única vez al entrar en el programa, se usan las etiquetas :SDF: y :EFD:

Si sólo se utiliza :global.usuario y :global.codigo_empresa, debería de utilizarse :SF: y :EF:, pero si se necesita alterar la where de la lista de valores según el dato de un campo anterior debe de utilizarse :SFD: y :EFD:

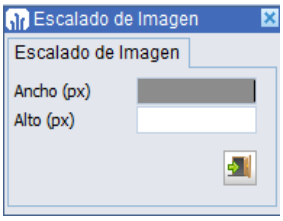
Ejemplo: Si en el programa PEDIDOS en la where de B1.CLIENTE cambiamos: *(:b1.oc_por_actividades = 'N' OR (codigo_actividad IS NULL OR EXISTS (SELECT 1 FROM org_comer_actividades oca WHERE oca.codigo_actividad = clientes.codigo_actividad AND oca.org_comercial = :b1.organizacion_comercial AND oca.codigo_empresa = :global.codigo_empresa)))* **por** *:SFD:PKVALIDAR_ENTIDADES.CW_VALIDA_ACTIVIDADES(:global.usuario, :global.codigo_empresa, :b1.organizacion_comercial, 'clientes.codigo_actividad', ':b1.organizacion_comercial'):EFD:*

Al ejecutar la lista de valores o la validación del cliente, únicamente hará el AND EXISTS sobre ORG_COMER_ACTIVIDADES si la organización comercial tiene actividades.

- **Validar desde L.V.**: Este desplegable puede tener los siguientes valores:
    - **Sí:** Fuerza a que cuando se introduzca un dato manualmente en el campo se valide que ese código está en los registros que se mostrarían en la lista de valores.
    - **No:** Permite introducir cualquier valor. No realiza ninguna validación contra la lista de valores.
    - **Sí – Llamar programa asociado:** Cuando se introduce manualmente un dato que no se puede validar contra la lista de valores y la lista de valores tiene asociado un programa se abrirá el programa automáticamente para que el usuario pueda crear el registro.
    - **Sí – Forzar Lista de Valores Contextual:** Indica que en la validación del campo siempre se lance la lista de valores, incluso aunque el valor introducido por el usuario en el campo únicamente devuelva un registro. Si se quiere forzar que la validación siempre sea a través de la lista de valores se deberá activar también la check "Bloquear validación sin pulsar INTRO / TABULACIÓN".
- **Forzar Filtro al Ejecutar Consulta**: Cuando se activa, en el bloque no se permitirá ejecutar consulta sin antes introducir un valor de filtro de este campo, es decir, si el usuario pulsa F7, mientras no introduzca un valor para filtrar en el campo no se le permitirá ejecutar consulta con F8. **NOTA**: Para que funcione totalmente esta opción es necesario que a nivel de bloque en el fuente tenga el disparador KEY-EXEQRY con la llamada a DISPSTD.KEY_EXEQRY como mínimo.
- **Desactivar búsqueda contextual** (En opciones avanzadas): Hay campos donde la búsqueda contextual puede ser incompatible con ellos, ya que puede haber varios registros válidos y al hacer la validación va a estar saltando la lista de valores en bucle mientras no se salga del campo con el ratón. Ver apartado *búsqueda contextual* para más información.
- **Bloquear salto de campo en L.V.**: Si se activa la check cuando se selecciona un registro de la lista de valores el cursor se mantendrá en el campo que ha llamado a la lista de valores, si está desmarcado saltará al siguiente campo navegable.
- **Tipo L.V.**: Permite indicar el formato que va a tener la lista de valores:
    - **Normal**: Lista de valores simple.
    - **Grupos 9 Registros**: Ver apartado *Listas de Valores por Grupos*.
    - **Grupos 5 Registros**: Ver apartado *Listas de Valores por Grupos*.
    - **Multiselección**: Ver apartado *Listas de Valores de Multiselección*.
    - **Multiselección Totalizada**: Ver apartado *Listas de Valores de Multiselección*.
    - **Rellenar List-Item**: Se usa en campos de tipo List-Item, para que sean rellenados al iniciarse el programa con los valores proporcionados por la lista de valores.
    - **Selección Icono:** Al pulsar sobre el botón de lista de valores al usuario le aparecerá una ventana mostrando todos los iconos que van incluidos en el estándar de Libra. Al indicar este tipo, no hace falta indicar nada en el campo "Lista de Valores".
    - **Selección Color:** Al pulsar sobre el botón de lista de valores al usuario se le abrirá una ventana en la que puede seleccionar un color de forma visual. El valor que retorna es el RGB en formato hexadecimal.
- **L.V. Carga Registro Único**: Se puede indicar que un campo asuma de forma automática el valor de la lista de valores en el caso de que la lista de valores sólo devuelve un único registro. Se puede indicar el punto en donde debe de realizarse la carga:
    - **No**: No se comprueba si la lista de valores solo tiene un registro válido.
    - **Sí - Cargar al entrar en campo**: Se hace en el WHEN-NEW-ITEM-INSTANCE, en los registros nuevos, al entrar en un campo que está vacío y tiene activado este parámetro, se realiza la comprobación, si sólo hay un registro se utiliza y se salta al siguiente campo. Esta opción marca el registro como inicializado, por lo que no es una buena opción para el primer campo del registro.
    - **Sí - Cargar al inicializar el registro**: Se hace en WHEN-CREATE-RECORD. Es especialmente útil para inicializar el primer campo del registro o aquellos campos que no dependen en absoluto de valores de campos anteriores.

- **Obligar**: Pone como obligatorio el campo y pone en negrita la descripción del campo.
- **Ocultar**: Hace invisible el campo.
- **Desactivar Modificación**: No permite la modificación del campo a los usuarios.
- **Desactivar Navegación**: Hace que en la navegación normal por teclado no se pase por ese campo, se podría ir con el ratón.
- **Validación Filtro**: Si el dato introducido por el usuario existe en la tabla correspondiente pone la descripción de forma normal, si no existe no carga la descripción, pero deja continuar, esto es útil cuando tenemos campos DESDE/HASTA y queremos poner desde AAAAAA a ZZZZZZ.
- **Desactivar Búsqueda Contextual**: Si el usuario tiene marcado que se utilice la búsqueda contextual y la lista de valores para un mismo código puede tener dos registros válidos entrará en un bucle en donde el usuario no puede salir de la lista de valores ya que se la vuelve abrir, en ese caso es necesario desactivar la lista de valores contextual para evitar que suceda esto.
- **Independiente de mayúsculas / minúsculas**: Si se activa para un campo, cuando se haga una búsqueda en ese campo con entrada / ejecución consulta hará la búsqueda independientemente de que en la tabla esté almacenado en mayúsculas / minúsculas e independientemente de que el patrón de búsqueda esté en mayúsculas o minúsculas.
- **Mayúsculas / Minúsculas**:
    - **Si está en blanco** no hace nada, lo que tenga el programa es lo que vale.
    - **Forzar Mayúsculas**: Obliga a que ese campo todo lo que se introduzca esté en mayúsculas.
    - **Forzar Mayúsculas sin Espacios**: El valor introducido se convierte en mayúsculas, y si el campo contiene algún espacio, no será validado.
    - **Forzar Minúsculas**: Obliga a que ese campo todo lo que se introduzca esté en minúsculas.
    - **Forzar primera letra Mayúscula**: Lo que introduzca el usuario será cambiado a minúsculas y la primera letra de cada palabra se pondrá en mayúsculas.
    - **Forzar Mayúsculas y Minúsculas**: Obliga a que ese campo se pueda meter tanto mayúsculas o minúsculas, se diferencia de la primera opción en que en el programa puede estar puesto que se fuerzan mayúsculas y de esta forma se permiten tanto mayúsculas como minúsculas´
- **Etiqueta Botón:** Se utiliza para botones y campos de tipo check box. En estos se puede utilizar junto a la etiqueta estándar o por idioma para el prompt.
- **Tooltip:** Etiqueta que se muestra al pasar el ratón por encima del campo.
- **Indicación:** Texto que se muestra en la barra de estado cuando el cursor entra en el campo.
- **Etiqueta Excel:** Etiqueta que se utilizará para la columna al exportar a hoja de cálculo el contenido del bloque. En el caso de que no tenga etiqueta se utilizará la etiqueta del prompt y en su defecto el código del campo.
- **Nombre Columna Consulta**: Nombre de la consulta que se enviará a la base de datos en vez del nombre de campo. Para más información ver apartado: *Campos de visualización de descripciones*.
- **Nombre Columna Orden**: Si especificamos algo en esta columna cuando el usuario pulse con el botón derecho sobre el campo e indique que desea orden ascendente o descendente va a ordenar por lo que esté especificado en este campo. Esto nos permite por ejemplo cuando tenemos una columna alfanumérica, pero en la instalación han metido valores como estos, 1, 2, 9, 10, 15, el orden lo va a hacer de forma alfanumérica, va a poner el 15 antes del 9, esto se soluciona poniendo en este campo: LPAD(campo, 15, '0').
- **Descendente (Nombre Columna Orden Descendente)**: Permite indicar el criterio de ordenación cuando se ordena de forma descendiente. Si está vacío el criterio que se utiliza es el indicado en "Nombre Columna Orden" añadiendo DESC al final. Este campo sólo se utiliza si se ha alimentado también el campo "Nombre Columna Orden"
- **Excel**: Se puede indicar el comportamiento del campo a la hora de ser exportado a hoja de cálculo.
    - **Sí**: El campo puede ser exportado.
    - **Sí – Totalizando si es posible**: Si el campo es numérico se intentará totalizar.

  - **No**: El campo no puede ser exportado.
- **Campo Anterior**: Cambia el campo al que salta el cursor cuando pulsamos MAYS+TAB, es decir, retrocedemos de campo.
- **Campo Siguiente**: Cambia el campo al que salta el cursor cuando pulsamos INTRO o ENTER.
- **Máscara**: Permite cambiar la máscara de formato del campo, por ejemplo, si fuese fecha se podría poner DD/MM/YY. También admite una constante para indicar que se le aplique la máscara de cantidades con los decimales que se tengan parametrizados en libra para cantidades poniendo CTD y DF para el formato de fecha que está parametrizado en el libra.env. Si se quiere la máscara de formato con únicamente 2 dígitos para el año se puede utilizar DFYY.
- **Tamaño Máximo**: Tamaño máximo en caracteres que va a aceptar el campo. Únicamente se puede reducir el tamaño sobre el que tenga el fuente, nunca aumentarlo.
- **Imagen**: Si el campo tiene CLASE_IMAGEN o CLASE_ARCHIVO, se le puede activar que se realice un escalado de la imagen subida, normalmente para reducir tamaño que ocupe menos. Al activar la check se abrirá la siguiente ventana en donde indicar las dimensiones.



En esta ventana se indicará las medidas en pixeles a la que se quiere reescalar la imagen, si se indica Ancho y Alto la imagen será redimensionada a esas medidas pudiendo perder las proporciones, por lo que se recomienda únicamente indicar una de las unidades, de forma que la otra será calculada para que la imagen continúe manteniendo las mismas proporciones.

Cuando el campo es CLASE_IMAGEN, para que Libra almacene también la imagen original, el bloque debe de tener un campo asociado a la tabla que se llame ID_ARCHIVO_<NOMBRE_CAMPO_IMAGEN>, por ejemplo, si el campo que contiene la foto se llama IMAGEN_FOTO el campo para almacenar la foto original sin procesar deberá llamarse ID_ARCHIVO_IMAGEN_FOTO. En el disparador PRE-DELETE del bloque hay que llamar a pk_blob2bd.borra_archivo en el caso de que el campo ID_ARCHIVO_X tenga valor.

- **Permitir Consulta**: Permite configurar a nivel de campo si puede ser utilizado para realizar el filtrado del bloque, tanto con F7/F8 como con la búsqueda contextual del bloqueo. Los valores posibles son:
  - **Sí**: El campo se puede utilizar tanto para filtrar por F7/F8 y en la búsqueda contextual de bloque.
  - **Sólo en modo entrada consulta**: Se puede utilizar para filtrar por F7/F8 pero no se utilizará para filtrar el bloque en la búsqueda contextual.
  - **No**: No se utiliza ni para filtrar por F7/F8 ni en la búsqueda contextual de bloque. Este es el valor recomendable para indicar a los campos no asociados a tabla para que el usuario no intente filtrar por ellos, ya que le llevará a confusión ya que no filtrará nada con el valor introducido.
- **Tipo Editor**: Permite definir qué editor va a tener el campo, puede ser uno de los siguientes:
  - **Automático**: Es la opción por defecto de todos los campos. Si el campo permite más de 1.000 caracteres se habilitará el nuevo editor de texto plano y en caso contrario el editor nativo de Oracle Forms.
  - **Nativo**: Se fuerza a que se utilice el editor nativo de Oracle Forms.
  - **Texto Plano**: Nuevo editor de texto.
  - **HTML**: Editor en formato enriquecido. En el campo se mostrará con las etiquetas HTML y para poder verlo bien hay que pulsar en el botón del editor o (Ctrl + E).

- **Posición X**: Mueve el campo a la posición x que indiquemos de la pantalla.
- **Posición Y**: Mueve el campo a la posición y que indiquemos de la pantalla.
- **Ancho**: Cambia el tamaño de ancho del campo.
- **Alto**: Cambia el tamaño de alto del campo.
- **Borde Prompt**: Permite indicar a qué esquina del campo se va a anclar la etiqueta del campo.
- **Alineamiento Prompt**: Permite indicar el alineamiento de la etiqueta dentro de la esquina a la que se encuentra anclada.
- **Desplazamiento sobre borde**: Posición relativa a la situación del campo del texto de la descripción del campo.
- **Desplazamiento sobre alineamiento**: Parecido a "Desplazamiento sobre borde" pero en la otra coordenada.
- **Valor por Defecto**: Lo que se introduzca lo meterá en el campo de forma inicial cada vez que se cree un registro nuevo. La check que se encuentra a la izquierda de este campo indica que el valor introducido debe de ser parseado en tiempo de ejecución, por ejemplo, si en "Valor por Defecto" se introduce GLOBAL.CODIGO_EMPRESA y se activa la check, el valor por defecto que se introducirá al crear el registro es el valor que contenga la variable GLOBAL.CODIGO_EMPRESA en ese momento. Si no se activa la check se introducirá el literal "GLOBAL.CODIGO_EMPRESA".
- **Ejecutar código de Pre-Validación**: Indica si se ejecutará código PL/SQL antes de realizarse la validación desde la lista de valores, este código puede estar definido en el mantenimiento de programas en el campo Código PL/SQL de validación o en la lista de valores, prevaleciendo el primero sobre el de la lista de valores.
- **Código PL/SQL de Pre-Validación**: Código que se ejecutará antes de realizar la validación con la SQL de la lista de valores. Solo se ejecutará si está marcada la check Ejecutar.
- **Código PL/SQL pulsación INTRO / TABULACIÓN**: Se ejecuta cuando el usuario pulsa intro en el campo. Si está activa la check "Sólo en campos inválidos" únicamente se ejecutará si el campo está pendiente de validar y se pulsa INTRO o TABULACIÓN, si no está activada la check se ejecuta siempre.
- **Bloquear validación sin pulsar INTRO / TABULACIÓN**: Si está activada y el usuario intenta validar el campo sin pulsar INTRO / TABULACIÓN, por ejemplo, saliendo del campo con el ratón, se le mostrará un mensaje de que debe de pulsar INTRO.
- **Código PL/SQL de validación**: El programa ya tiene que incorporar en el fuente las validaciones de integridad necesarias que nunca podrán ser alteradas sin modificar el fuente, como por ejemplo, que un artículo no pueda tener más de dos unidades de almacén. Pero mediante la introducción de código PL/SQL de validación asociado al campo se pueden añadir restricciones específicas en una determinada instalación o qué su cambio no suponga una alteración del diseño de la aplicación.

  Este código PL/SQL se ejecutará después de haberse realizado la validación desde lista de valores. En caso de que se tenga marcada la check *Validar desde Lista de Valores* (si esta check no está marcada este código se ejecutará igual). Por tanto, si se ha activado la validación desde lista de valores se supone que en el momento de ejecutarse este código el campo ya tiene un valor válido.

- **Código PL/SQL de pre-ejecución de lista de valores**: Este código se ejecuta en el momento en que el usuario solicita una lista de valores, y tiene como principal característica que dependiendo del resultado de su ejecución podemos hacer que salte una lista de valores u otra.
- **Código PL/SQL de Doble click**: Se ejecuta cuando el usuario hace doble click con el ratón sobre el campo.
- **Código PL/SQL de entrada en campo**: Se ejecuta cada vez que entra el cursor en el campo, siempre y cuando no se venga de una lista de valores.
- **Cláusula Where por Defecto 2**: Si se especifica habilita un botón en la lista de valores para poder conmutar entre condiciones. Para más información ver el apartado: Listas de valores.

- **Tipo Where Validación**: Permite indicar la cláusula where que se usará para realizar la validación en caso de listas de valores de doble where. Puede tener los siguientes valores:
    - **Según Lista de Valores**: Se usa el tipo de validación indicado en la lista de valores.
    - **Principal**: Se usa la cláusula where principal.
    - **Secundaria**: Se usa la cláusula where secundaria.
    - **Personalizada**: Se puede indicar una cláusula where personalizada en el campo "Cláusula Where para Validación".
- **Etiqueta Botón Where Defecto y Etiqueta Botón Where Defecto 2**: Se usa cuando se especifica una Cláusula Where por Defecto 2 e indican el texto que va a contener el botón que se habilita para conmutar entre condiciones. Para más información ver el apartado: Listas de valores.
- **Cláusula Where en modo Entrada Consulta:** Permite indicar una Cláusula Where que se aplicará cuando el bloque se encuentre en modo de entrada consulta. Para más información ver el apartado: Listas de valores.
- **Selección de archivo**: En casos de campos en que el usuario deba de introducir una ruta a un archivo se activará la check "Selección archivo", una vez marcada aparecerá la siguiente ventana:

NOTA: La activación de selección de archivo únicamente implica que en ese campo se puede abrir mediante el botón de lista de valores el diálogo del sistema operativo de selección de archivo, en el caso de querer almacenar un archivo en la base de datos, simplemente es necesario que en el bloque exista un campo oculto llamado ID_ARCHIVO de tipo NUMBER y al campo donde el usuario va a subir el archivo (normalmente se llamará NOMBRE_ARCHIVO) se le asigne la clase CLASE_ARCHIVO.

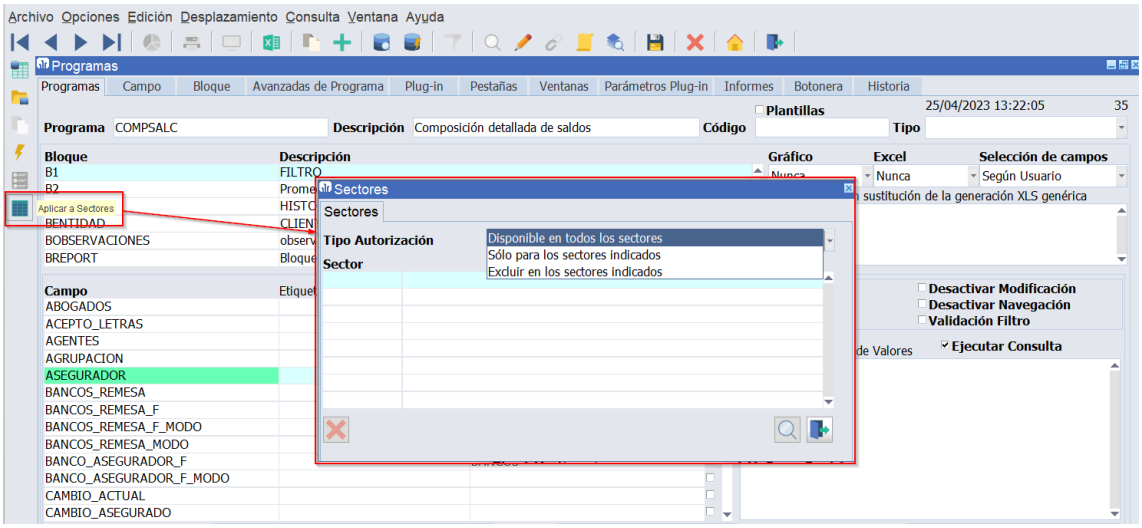
    - **Selección**: Se indica cual es el objetivo de la búsqueda:
        - **Selección de archivo para abrir**: Solo se pueden seleccionar archivos existentes.
        - **Selección de archivo para guardar**: Se pueden seleccionar archivos existentes y no existentes, si se selecciona uno existente se mostrará el mensaje de que se va a sobreescribir y se pedirá aceptación al usuario.
        - **Selección de directorio**: Abre la ventana de selección de directorios.
    - **Campo Archivo**: Campo al que se le va a asignar el nombre del archivo. Para indicar el campo se introducirá BLOQUE.CAMPO. Si no se indica se asignará al campo en que se encuentre el cursor.
    - **Campo Directorio**: Campo al que se le va a asignar el directorio del archivo seleccionado. Para indicar el campo se introducirá BLOQUE.CAMPO.
    - **Título**: Texto que aparecerá en la ventana de selección de archivo.
    - **Filtro**: Se usará para filtrar los tipos de archivos que se visualizarán en la selección, como separador de los tipos de archivos se usará el carácter "|". Ejemplo para seleccionar archivos de tipo texto y todos los archivos: "*Archivos de Texto (*.txt)|*.txt|Todos los archivos (*.*)|*.*|*"

NOTAS:

- Si a un campo se le asocia más de un elemento (lista de valores, calendario, calculadora) solo funcionará uno y este será en el que indica el siguiente orden: primero lista de valores, segundo calendario y en tercer lugar la calculadora.
- Para más información sobre Código PL/SQL y su estructura, ver el apartado: Código PL/SQL.

Control de visualización de campo según el sector del grupo empresarial

Se puede indicar a los sectores a los que aplica el campo, de forma que se puede hacer que dependiendo de los sectores activados a nivel de grupo empresarial el campo esté o no disponible. Además, esa información es muy valiosa para el programa de chequeo de versión, ya que si un campo únicamente aplica al sector de "Extrusión de Aluminio" al chequear la versión sabe que únicamente tendrá que tenerlo en cuenta cuando se está empaquetando la versión de Extrusión de Aluminio. Si hay configuración de sectores para el campo el campo "Calculadora" se mostrará con fondo rojo.

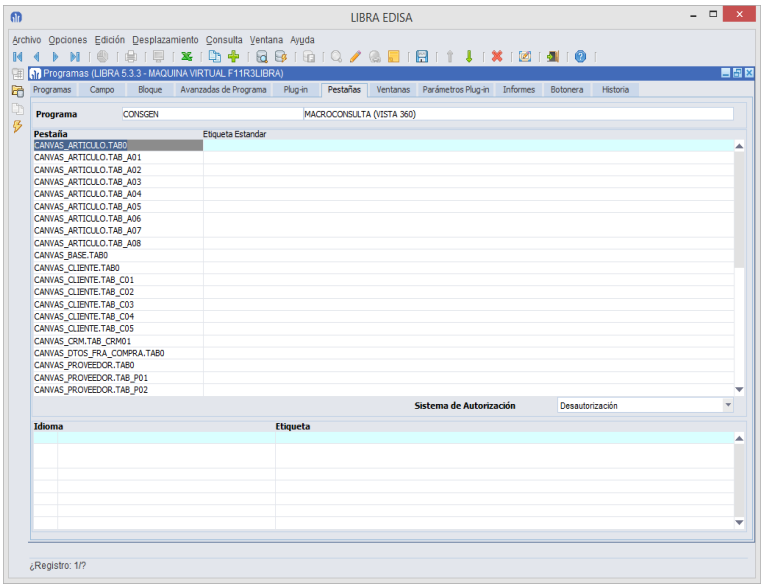


En el campo "Tipo Autorización" se indica como debe de considerar los sectores que se indican en el multiregistro:

- **Disponible en todos los sectores**: Se ignoran los sectores indicados
- **Sólo para los sectores indicados**: Se aplicará en los grupos empresariales que tengan configurados alguno de los sectores indicados.
- **Excluir en los sectores indicados**: Al contrario que el valor anterior, se mostrará únicamente cuando el usuario esté validado en Libra en un grupo empresarial que no tenga activado alguno de los sectores que se indican en el multiregistro.

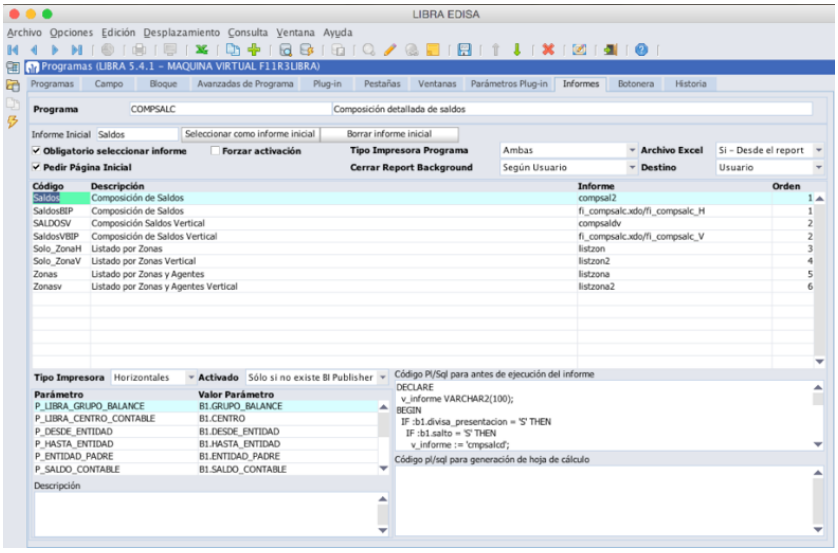
Pestañas

Permite indicar que pestañas tiene el programa (siempre que no sea dinámico), para la traducción de su etiqueta por idioma y la autorización por perfiles de usuario.



En el código de pestaña introduciremos LIENZO.PESTAÑA, es decir, para la pestaña TAB1 del lienzo CANVAS_BASE introduciremos CANVAS_BASE.TAB1.

- **Etiqueta Estándar**: Introducimos la etiqueta que se usará por defecto en caso de que en el idioma del usuario no exista etiqueta personalizada. Si no se introduce nada en este campo se mostrará la etiqueta que contenga el fuente del programa.
- **Sistema de Autorización**: Indicamos si perfiles introducidos en el bloque de Perfiles Autorizados / Desautorizados pueden visualizar o no la pestaña, es decir, si este campo contiene el valor Autorización solo los usuarios con los perfiles especificados podrán visualizar la pestaña y si este campo contiene el valor Desautorización todos los usuarios podrán visualizar la pestaña excepto los de los perfiles indicados.
- **Bloque Etiquetas por Idioma**: Introducimos por idioma la etiqueta correspondiente para la pestaña.



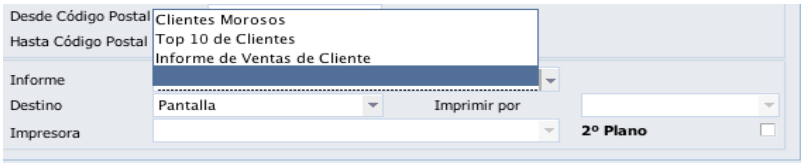
- **Archivo Excel**:
  - **Sí - Desde el report**: Se ejecuta la salida nativa de Oracle para generar la hoja de cálculo.
  - **Sí – Usando Rep2Excel**: El informe no debe tener código específico, se genera un HTML y se llama a la herramienta rep2excel que lo convierte en Excel.
  - **No dar opción**: No aparece el tipo de archivo Excel en la pantalla de selección de parámetros del report.
- **Tipo Impresora Programa**: Puede ser modificado por informe:
  - **Verticales**: Sólo se mostrarán impresoras que no tengan marcado el parámetro de carro ancho en impresoras lógicas.
  - **Horizontales**: Sólo se mostrarán impresoras que tengan marcado el parámetro de carro ancho en impresoras lógicas.
  - **Ambas**: Aparecerán todas las impresoras independientemente del parámetro de impresoras lógicas.
- **Destino**: También está relacionado con los informes. Se indica el destino por defecto que va a proponer el programa. Hay las siguientes opciones:
  - Pantalla.
  - Impresora.
  - Archivo.
  - Email.
  - Fax
  - Gestión Documental
  - Usuario: Buscará en la parametrización de la configuración del usuario, que puede ser cualquiera de las anteriores.
- **Apertura automática hoja de cálculo**: Indica como debe de comportarse cuando en la impresión se selecciona impresión a archivo de tipo hoja de cálculo. Si se indica "Según Usuario" se tomará el valor por defecto que tenga el usuario en personalización por usuario / empresa. Si tiene el valor "Sí", todas las impresiones que generen archivo de hoja de cálculo abrirán el archivo de forma automática independientemente de la parametrización del usuario. Si tiene el valor "No", nunca se abrirán los archivos una vez generados.
- **Pedir Página Inicial**: Si se activa, permite al usuario indicar en que número de página desea comenzar la numeración de páginas del informe, en vez de comenzar siempre en 1. Al report se le pasará el parámetro P_LIBRA_PAGINA_INICIAL que deberá contemplar.

Se pueden configurar otros informes que se ejecutarán en sustitución del que tenga el fuente del programa en caso de que el usuario lo seleccione, a esos informes se les pasarán los mismos parámetros que recibiría el listado que tenga el fuente, más los parámetros indicados en la sección de parámetros en los campos "Parámetro" y "Valor Parámetro".

Configuración de los informes

- **Tipo Impresora**: Permite indicar a nivel de informe el tipo de impresora (Horizontal / Vertical / Según programa).
- **Código PL/SQL para antes de ejecución del informe**: Este código se ejecuta antes de lanzar el informe seleccionado, de esta forma, se puede alterar el informe que se va a ejecutar en el momento que el usuario pulsa el botón de imprimir. Para modificar el informe a ejecutar indicará con PKPANTALLAS.SET_VARIABLE_ENV('IMP_INFORME', 'informe');
- **Código pl/sql para generación de hoja de cálculo**: Desde los códigos PL/SQL se puede configurar la generación de hojas de cálculo, en este código PL/SQL se puede introducir el método de generación de la hoja de cálculo, de forma de que se evite usar Rep2Excel. (Ver apartado: Generación de hojas de cálculo mediante PKXLSBD).

En la pantalla de selección se vería de esta forma:



- **Informe**: Archivo de reports o BI-Publisher a ejecutar. También Se pueden incluir informes implementados mediante el generador de informes, para ello en el campo "Informe" se indicará "GI:" y a continuación el código del informe. Por ejemplo "GI:CLIVTAS" ejecutara el informe CLIVTAS.
- **Nombre archivo sugerido (sin extensión):** Nombre del archivo que se le propondrá al usuario antes de generar el informe como destino archivo.

Se puede establecer que aparezca un informe seleccionado por defecto, para ello hay que posicionarse en el informe y pulsar "Seleccionar como informe inicial". Para quitar el informe inicial hay que pulsar "Borrar informe inicial".

Si se activa la check "Obligatorio seleccionar informe" no se permitirá dejar en blanco el desplegable de Informes, de esta forma se anula el informe que es llamado dentro del programa.

IMPORTANTE: En los programas que se rompe la herencia del tamaño y posición del campo BREPORT.INFORME se deshabilita el funcionamiento de estos informes. En ese caso se puede forzar activando la check "Forzar activación".

Por código se puede interactuar con este campo con las siguientes funciones:

- IMP.GET_PROPIEDAD('BREPORT_INFORME_CODIGO_INFORME'): Devuelve el código del informe seleccionado.
- IMP.GET_PROPIEDAD('BREPORT_INFORME_NOMBRE_INFORME'): Devuelve el nombre del archivo del informe seleccionado.
- IMP.GET_PROPIEDAD('BREPORT_INFORME_TITULO_INFORME'): Devuelve la etiqueta del informe seleccionado.

El report también recibirá los datos del informe seleccionado en los siguientes parámetros: P_INFORME_NOMBRE_INFORME, P_INFORME_CODIGO_INFORME, P_INFORME_TITULO_INFORME.

Generación / Impresión Múltiples

En la pestaña "Informes" del mantenimiento de programas cuando se posiciona el cursor sobre uno de los informes definidos se habilita un nuevo plug-in "Informes Adicionales" que abre una ventana en donde se pueden indicar los informes a imprimir a mayores del indicado. En esta misma ventana hay que indicar el orden en el que tienen que ser impresos / generados.

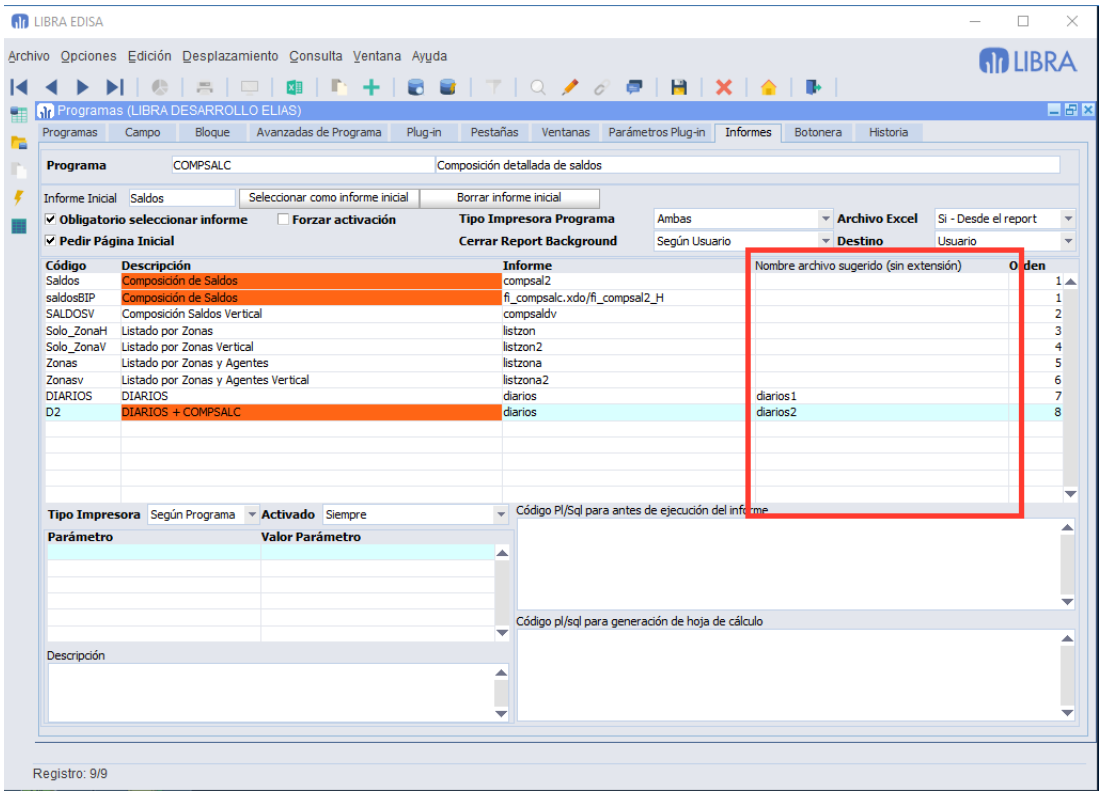
Los informes adicionales también tienen que estar definidos como informes antes de ser usados como informe adicional. Si se quiere que el informe no esté visible en el desplegable, de forma que sólo pueda ser utilizado como informe adicional de otro, se marcará orden 0 y de esa forma se carga al abrir el programa, pero no se muestra en el desplegable.

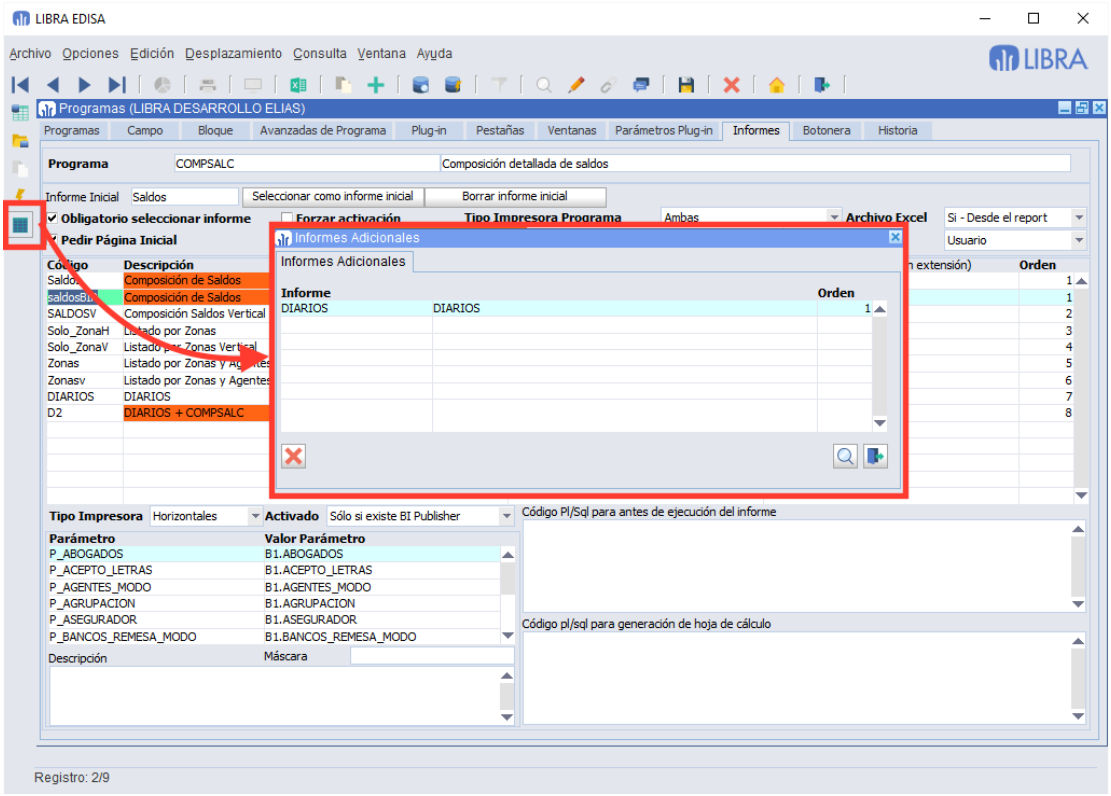
Si la descripción del informe aparece en naranja indica que ese informe tiene configurados informes adicionales.

También se puede indicar para cada informe el nombre de archivo sugerido. Es importante indicar nombres de archivos distintos para que cuando durante la impresión de los informes adicionales genere archivos distintos y no se sobrescriban.

Funcionamiento según destino:

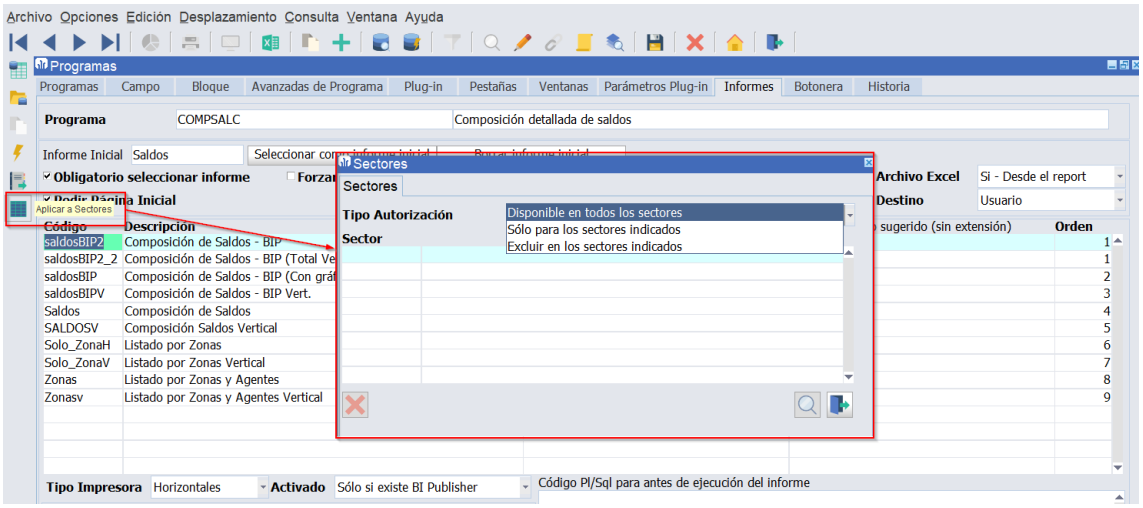
- **Pantalla**: Se abre una pestaña del navegador por cada informe.
- **Impresora**: Se imprimen en orden todos a la misma impresora.
- **Fichero**: Se genera en el directorio indicado un archivo por cada informe.
- **Correo Electrónico**: Depende de la configuración que se tenga por Usuario / Empresa
    - **Método nativo de Reports**: Será reports quien mande por mail los informes y mandará un mail por cada informe.
    - **Java**: Todos los informes se adjuntan en el mismo por correo.
    - **Microsoft Outlook**: El método que tenemos de lanzar Outlook con el informe adjunto sólo permite adjuntar un único archivo, por lo que se meten todos los archivos en un ZIP y se adjunta ese archivo comprimido con los informes.
    - **Aplicación asociada a Tipo de Archivo**: Se abren cada uno de los archivos con la aplicación que tenga asociada en el sistema operativo del usuario y ya desde esa aplicación será el usuario quien manualmente ejecute la acción de envío de correo electrónico.


LIBRA EDISA
Archivo Opciones Edición Desplazamiento Consulta Ventana Ayuda
LIBRA
Programas (LIBRA DESARROLLO ELIAS)
Programas Campo Bloque Avanzadas de Programa Plug-in Pestañas Ventanas Parámetros Plug-in Informes Botonera Historia
Programa COMPSALC Composición detallada de saldos
Informe Inicial Saldos Seleccionar como informe inicial Borrar informe inicial
Obligatorio seleccionar informe Forzar activación Tipo Impresora Programa Ambas Archivo Excel Si - Desde el report
Pedir Página Inicial Cerrar Report Background Según Usuario Destino Usuario
Código Descripción Informe Nombre archivo sugerido (sin extensión) Orden
Saldos Composición de Saldos compsal2 1
saldosBIP Composición de Saldos fi_compsalc.xdo/fi_compsal2_H 1
SALDOSV Composición Saldos Vertical compsaldv 2
Solo_ZonaH Listado por Zonas listzon 3
Solo_ZonaV Listado por Zonas Vertical listzon2 4
Zonas Listado por Zonas y Agentes listzona 5
Zonasv Listado por Zonas y Agentes Vertical listzona2 6
DIARIOS DIARIOS diarios diarios1 7
D2 DIARIOS + COMPSALC diarios diarios2 8
Tipo Impresora Según Programa Activado Siempre
Código Pl/Sql para antes de ejecución del informe
Parámetro Valor Parámetro
Código pl/sql para generación de hoja de cálculo
Descripción
Registro: 9/9


LIBRA EDISA
Archivo Opciones Edición Desplazamiento Consulta Ventana Ayuda
LIBRA
Programas (LIBRA DESARROLLO ELIAS)
Programas Campo Bloque Avanzadas de Programa Plug-in Pestañas Ventanas Parámetros Plug-in Informes Botonera Historia
Programa COMPSALC Composición detallada de saldos
Informe Inicial Saldos Seleccionar como informe inicial Borrar informe inicial
Obligatorio seleccionar informe Forzar activación Tipo Impresora Programa Ambas Archivo Excel Si - Desde el report
Pedir Página Inicial Usuario
Informes Adicionales
Informes Adicionales
Informe Orden
DIARIOS DIARIOS 1
Código Descripción Orden
Saldos Composición de Saldos 1
saldosBIP Composición de Saldos 1
SALDOSV Composición Saldos Vertical 2
Solo_ZonaH Listado por Zonas 3
Solo_ZonaV Listado por Zonas Vertical 4
Zonas Listado por Zonas y Agentes 5
Zonasv Listado por Zonas y Agentes 6
DIARIOS DIARIOS 7
D2 DIARIOS + COMPSALC 8
Tipo Impresora Horizontales Activado Sólo si existe BI Publisher
Código Pl/Sql para antes de ejecución del informe
Parámetro Valor Parámetro
P_ABOGADOS B1.ABOGADOS
P_ACEPTO_LETRAS B1.ACEPTO_LETRAS
P_AGENTES_MODO B1.AGENTES_MODO
P_AGRUPACION B1.AGRUPACION
P_ASEGURADOR B1.ASEGURADOR
P_BANCOS_REMESA_MODO B1.BANCOS_REMESA_MODO
Descripción Máscara
Código pl/sql para generación de hoja de cálculo
Registro: 2/9

Control de visualización de informe según sector del grupo empresarial

Se puede indicar a los sectores a los que aplica el informe, de forma que se puede hacer que dependiendo de los sectores activados a nivel de grupo empresarial el informe esté o no disponible. Si hay configuración de sectores para el informe el campo "Orden" se mostrará con fondo rojo.

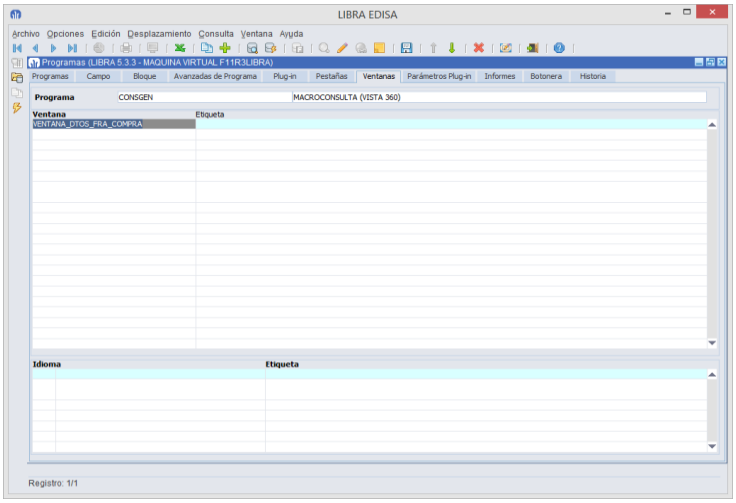


En el campo "Tipo Autorización" se indica como debe de considerar los sectores que se indican en el multiregistro:

- **Disponible en todos los sectores**: Se ignoran los sectores indicados
- **Sólo para los sectores indicados**: Se aplicará en los grupos empresariales que tengan configurados alguno de los sectores indicados.
- **Excluir en los sectores indicados**: Al contrario que el valor anterior, se mostrará únicamente cuando el usuario esté validado en Libra en un grupo empresarial que no tenga activado alguno de los sectores que se indican en el multiregistro.

Ventanas

Nos permite indicar qué ventanas tiene el programa, para la traducción de su etiqueta por idioma.



- **Etiqueta**: Introducimos la etiqueta que se usará por defecto en caso de que en el idioma del usuario no exista etiqueta personalizada. Si no se introduce nada en este campo se mostrará la etiqueta que contenga el fuente del programa.
- **Bloque Etiquetas por Idioma**: Introducimos por idioma la etiqueta correspondiente para la ventana.

Plug-in

Un plug-in consiste en parametrizar llamadas a otros programas desde la botonera vertical.

Lo vamos a ver con un ejemplo: Imaginemos que un cliente nos pide que desde las líneas de los albaranes de compra quiere consultar las tarifas de compra del artículo.

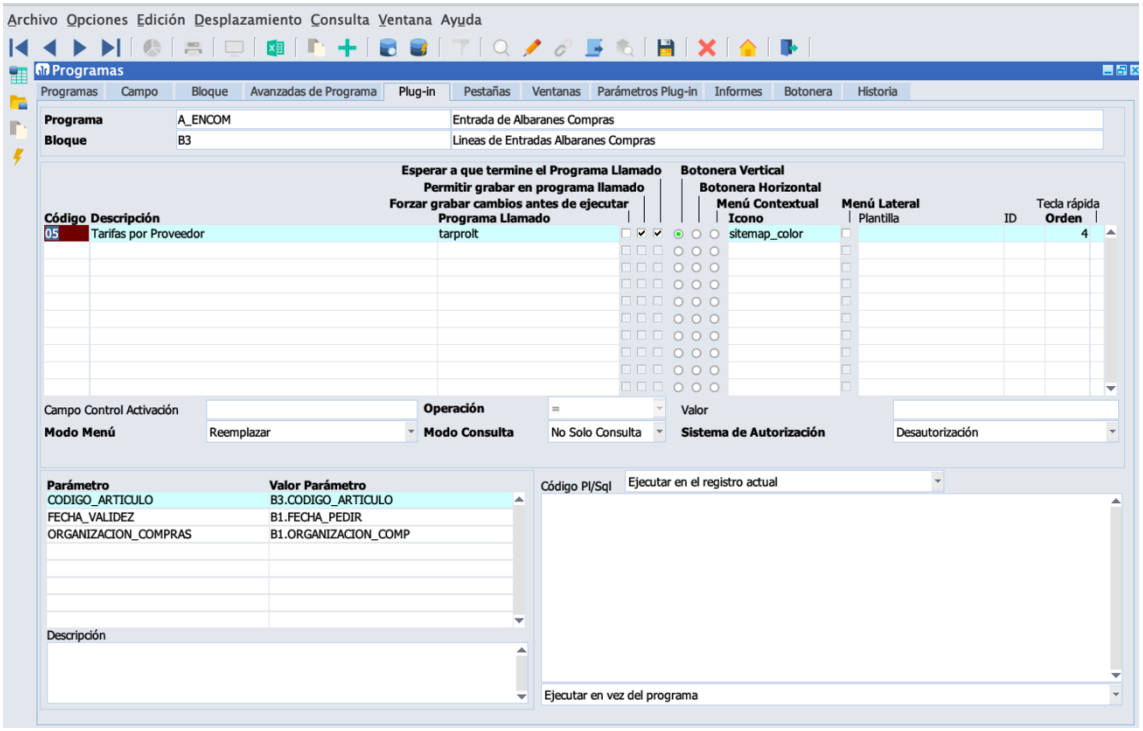
Para configurar la llamada iremos al Mantenimiento de programas y en la sección de bloques hay que posicionarse en el bloque en el que se quiere añadir un plug-in.

En el ejemplo que estamos viendo el bloque en el que queremos añadir el plug-in es el B3, en la descripción normalmente ya indica a que se corresponde "Líneas de Entradas Albaranes Compras", y vamos a la pestaña "Plug-in".

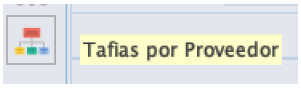
Llamaremos a una consulta ligera llamada ***tarprolt***, que recibe los siguientes parámetros:

- CODIGO_ARTICULO: Código del artículo que queremos consultar.
- ORGANIZACION_COMPRAS: Código de la organización de compras en la que queremos consultar los precios.
- FECHA_VALIDEZ: Fecha a la que queremos que estén vigentes los precios.

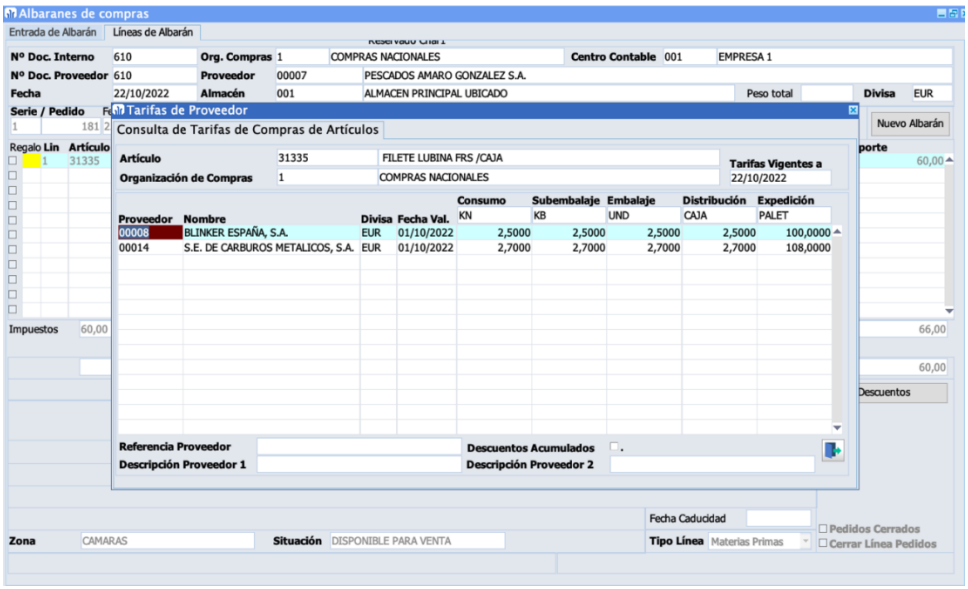
El resultado de la parametrización del ejemplo sería el siguiente:



Al ejecutar el programa, al entrar en las líneas la botonera aparece el siguiente botón:



Al pulsar el nuevo botón después de haber introducido el código del artículo aparece la ventana del programa llamado, que al ser una ventana modal parece que está totalmente integrada en el programa.



La pantalla de configuración de plug-ins consta de dos bloques, el primero es en el que especificamos los plug-ins que se quieren activar en el bloque anteriormente seleccionado. Campos para configurar plug-ins:

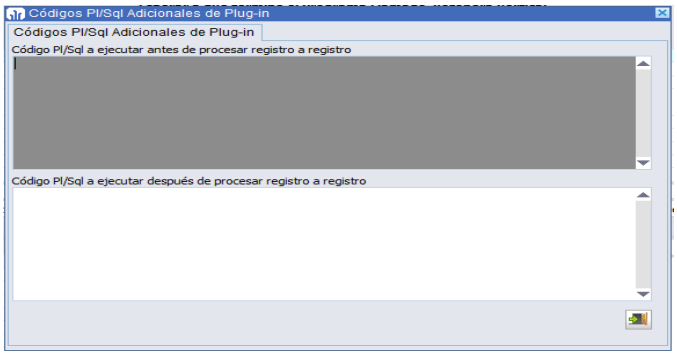
- **Código**: Será un código que le daremos según el criterio que se quiera. El usuario no lo va a visualizar.
- **Descripción**: Será el texto que aparecerá al pasar el ratón por encima del icono del plug-in y en el menú en el apartado "Opciones".
- **Programa Llamado**: Nombre del programa, informe, o disparador que se llamará cuando el usuario pulse el botón correspondiente al plug-in. Si se especifica código PL/SQL indicado que se ejecute en vez del programa se puede poner aquí cualquier cosa ya que lo va a ignorar en el caso de que list-item que se encuentra debajo del código PL/SQL tenga el valor "Ejecutar en vez del programa".
    - **Ejecutar informe del Generador de Informes**: Se indicará el código del informe con el prefijo GI: (ejemplo: GI:VENTAS).
    - **Ejecutar un disparador definido dentro del propio fuente**: Se indicará el nombre del disparador a ejecutar con el prefijo TG: (ejemplo: TG:MI_TRIGGER)
    - **Ejecutar un programa forzando una determinada opción de menú**: Se indicará el código de la opción de menú con el prefijo MN: (ejemplo: MN:2P102323).
- **Forzar grabar cambios antes de ejecutar**: Si el usuario tiene cambios que todavía no han sido grabados en la base de datos, antes de ejecutar el plug-in se le mostrará un mensaje en el que se le pide si desea grabar los cambios. Si el usuario indica que "No" se cancela la ejecución del plug-in.
- **Esperar a que termine el Programa Llamado**: En caso de activar esa opción el programa llamado funciona igual que si el usuario fuese por ventanas y lo abriese desde el menú, es decir, el programa llamador continúa su ejecución, la conexión a la base de datos es distinta para cada programa. El inconveniente principal es que el programa llamado no puede devolver valores al programa llamador y tampoco puede compartir variables de sesión de la base de datos, y como ventaja se evita el mensaje "No se puede iniciar otra llamada a pantalla" cuando el usuario tiene varios programas llamados de forma concurrente.
- **Permitir grabar en programa llamado**: Únicamente tiene sentido cuando está activada la check "Esperar a que termine el Programa Llamado". Cuando el programa llamador espera al programa llamado se comparte la misma conexión con la base de datos, por lo que si el programa llamado hace un commit también afectará a lo que tenga pendiente de grabar el programa llamador. Si se

activa esta check al usuario no se le permite grabar, los cambios que haga tienen que ser grabados una vez regrese al programa llamador.

- **Botonera**: Se puede indicar si el botón se quiere que aparezca en la botonera vertical o en la horizontal. La botonera vertical tiene capacidad para más botones.
- **Icono**: Nombre del icono a utilizar. Este campo dispone de una lista de valores de todos los iconos disponibles en Libra.
- **Menú Lateral**: Esta check únicamente aplica cuando en "Programa Llamado" es un programa de la movilidad de Libra. Al activar esta check el programa se abrirá en navegador del menú lateral reemplazando el menú del usuario. El usuario para cerrar ese programa y volver a visualizar el menú tiene que pulsar sobre los tres puntos del menú y seleccionar "Regresar al Menú".
- **Plantilla**: En caso de ser un programa dinámico el que se indica en "Programa llamado" se puede forzar a que se ejecute con una determinada plantilla.
- **ID**: En caso de que el programa indicado en "Programa llamado" tenga varias personalizaciones se puede forzar a que se ejecute con una en concreto.
- **Orden**: Si el bloque tiene varios plug-in indicará en que orden se muestran en la botonera vertical. Lo normal será indicar el orden de 1 a 20, siendo el menor número el que primero saldrá, justo después de los botones que ya tenga el programa definido de forma fija.
- **Tecla rápida**: Se puede asignar a un plug-in una tecla de función de manera que al ser pulsada se ejecute dicho plug-in. La paridad entre el número de tecla rápida y la tecla real del teclado dependerá del fichero de recursos de Oracle Forms que tenga el pc instalado. Por defecto son las siguientes:
  - 1: CONTROL+SHIFT+F1
  - 2: CONTROL+SHIFT+F2
  - 3: CONTROL+SHIFT+F3
  - 4: F11
  - 5: F5
  - 6: F12
  - 7: CONTROL+SHIFT+F7
  - 8: CONTROL+SHIFT+F8
  - 9: CONTROL+SHIFT+F9
- **Control de activación y desactivación del plug-in**: El plug-in se puede activar o desactivar en base a un valor de un campo del programa:
  - **Campo Control Activación**: Campo en el que se va a comprobar el valor que tiene a la hora de decidir si se activa o desactiva el plug-in. Se indicará BLOQUE.CAMPO. Cada vez que el usuario haga un cambio en el campo indicado se realizará la evaluación del control. Se pueden configurar campos adicionales sobre los que aplicar la evaluación en el botón "Campos en los que evaluar el control de activación" de la botonera vertical.
  - **Operación**: Lista de operaciones soportadas para hacer la comparación.
  - **Valor**: Valor sobre el que se evaluará la Operación, si el resultado de la operación es TRUE se activa el campo y si es FALSE se desactiva. Si en "Operación" se ha indicado "Expresión" en este campo se indicará una expresión booleana, es decir, que devuelva TRUE o FALSE. Ejemplo: (:bloque.campo1 = 'XXX' OR :b3.campo2 = 'YYYY').
- **Modo Menú**: Indica si al llamarse el programa se debe de mantener el menú del programa llamador en el programa llamado o que este inicialice su propio menú. Valores posibles:
  - **No reemplazar**: Se mantiene el menú del programa llamador en el programa llamado.
  - **Reemplazar**: Se inicializa el menú del programa llamado.
- **Modo Consulta**: Indica si al llamarse al otro programa se va a hacer en modo de solo consulta o no.
  - **Sólo Consulta**: En el programa llamado solo se podrán ejecutar consultas, nunca modificación de datos.
  - **No sólo consulta**: En el programa llamado se pueden modificar datos.

- **Código PL/SQL**: Si tiene contenido ejecuta este código y dependerá de lo que tenga "Tipo de ejecución de código PL/SQL" tiene o no en cuenta los campos *Programa llamado*, *Modo menú*, *Modo consulta* y *del contenido del bloque de parámetros*. Ver sección de Código PL/SQL para más detalles.
    - **Ámbito de ejecución del Código PL/SQL**: Sobre el Código PL/SQL hay un List-Item que permite indicar sobre qué registros se debe de ejecutar ese código. Los valores que puede tomar son los siguientes:
        - **Ejecutar en el registro actual**: Únicamente se ejecuta una vez y la ejecución se realiza en el registro en el que se encuentra el cursor.
        - **Ejecutar para todos los registros**: Se va ejecutar ese código para todos los registros que existan en ese momento en el bloque. Por defecto se ejecuta para todos los registros que se encuentran en el bloque en el que se encuentra el cursor, pero a partir de la versión 6.0.8 de entorno se puede indicar un bloque en concreto sobre el que ejecutar el código PL/SQL en cada uno de sus registros, para ello hay que indicar el bloque en el campo "**En bloque**".
        - **Sólo en registros seleccionados**: Si el bloque tiene activada la check "Habilitar selección de registros", el usuario puede seleccionar varios registros y el plug-in se ejecutará por cada uno de los registros que tenga seleccionados el usuario, si el usuario no ha seleccionado ninguno se ejecutará sobre el registro en el que se encuentra el cursor.
        - **Sólo seleccionados o para todos si no hay selección**: Se comporta igual que "Sólo en registros seleccionados", con la diferencia de que si el usuario no ha seleccionado ningún registro se ejecutará para todos los registros que tenga el bloque en ese momento en vez de hacerlo únicamente para el registro actual.

> **NOTA**: Si se selecciona una opción diferente a "Ejecutar en el registro actual", se habilita un botón "...", al pulsar en este botón se abre una ventana en donde se pueden indicar dos "Códigos PL/SQL adicionales".



        - **Código PL/SQL a ejecutar antes de procesar registro a registro**: Este código PL/SQL se ejecuta antes de comenzar a ejecutarse el código PL/SQL parametrizado en el plug-in, en este PL/SQL se podría cancelar la ejecución, mediante ":p_parar_ejecucion".
        - **Código PL/SQL a ejecutar después de procesar registro a registro**: Este código PL/SQL se ejecuta al terminar de procesar por cada registro el código PL/SQL del plug-in.
    - **Tipo de ejecución del código PL/SQL**:
        - **Ejecutar en vez del programa**: No se llama al programa indicado en "Programa llamado", solo se ejecuta el código PL/SQL.
        - **Ejecutar antes del programa**: Primero se ejecuta el código PL/SQL y luego se ejecuta el programa indicado en "Programa Llamado". En el código PL/SQL se

puede saber si se está ejecutando antes del programa si pkpantallas.get_variable_int_varchar2('PKLIBPNT.PUNTO_EJEC_CODIGO_PLSQL') devuelve el valor 'A'.

- **Ejecutar después del programa**: Primero se ejecuta el programa indicado en "Programa Llamado" y una vez se sale de ese programa se ejecuta el código PL/SQL. En el código PL/SQL se puede saber si se está ejecutando antes del programa si pkpantallas.get_variable_int_varchar2('PKLIBPNT.PUNTO_EJEC_CODIGO_PLSQL') devuelve el valor 'D'.
- **Ejecutar antes y después del programa**: Se ejecuta antes de ejecutarse el programa indicado en "Programa Llamado" y otra vez después de salir de ese programa.

Con solo especificar estos datos ya tiene que aparecer en la botonera del programa al entrar en el bloque el botón del plug-in, pero lo único que hará al pulsarlo es llamar al programa, pero sin ningún parámetro, en el ejemplo que estamos siguiendo el programa llamado no sabe el artículo que tiene el usuario en la línea del albarán de compras, por tanto, no podrá mostrar el precio, para pasar los parámetros dependerá del caso y del programa que se llame.

Los campos que tenemos que indicar para pasar parámetros son los siguientes:

- **Parámetro**: Nombre del parámetro que recibe el programa llamado, este dato depende del programa al que llamemos, posiblemente el consultor tenga que consultar a un técnico para que le diga los parámetros que tiene que pasar.
- **Valor Parámetro**: Permite configurar de donde va a obtener el valor del programa principal para ser pasado al parámetro del programa plug-in. Este valor se puede ser de cuatro tipos:
  - **Variable**: Se puede obtener de:
    - **Campo**: Se especifica en formato BLOQUE.CAMPO del que queremos obtener el valor. Este será la opción más común.
    - **Variable global**: Se introduce GLOBAL.VARIABLE.
    - **Parámetro**: Parámetro local del programa principal, se introduce PARAMETER.NOMBRE_PARAMETRO
  - **Constante**: Será un valor fijo y lo indicaremos entre comillas simples, por ejemplo '10002'.
  - **Fórmula**: Las fórmulas serán evaluadas en el momento de la ejecución del plug-in. Para indicar una fórmula hay que indicar el prefijo F: y a continuación la fórmula. **Ejemplo**: `F:'01/01/' || TO_CHAR(f_current_date, 'YYYY')`
  - **Propiedad**: Se pasará :XXX:<objeto>:<propiedad>, donde XXX, indica el tipo de objeto del que se quiere obtener la propiedad. **Ejemplo**: :GBP:CAMPOS:DEFAULT_WHERE se corresponde con la propiedad DEFAULT_WHERE del bloque CAMPOS. Tipos:
    - **GBP**: Bloque (Get_Block_Property).
    - **GIP**: Item (Get_Item_Property).
    - **GWP**: Window (Get_Window_Property).
    - **GFP**: Form (Get_Form_Property).
    - **GCP**: Canvas (Get_Canvas_Property).
    - **GTP**: Tab (Get_Tab_Page_Property).
    - **GMP**: Menú (Get_Menu_Item_Property).
    - **GII**: Item Instance (Get_Item_Instance_Property).
    - **GLL**: Get_List_Element_Label. Si la propiedad es '0' devolverá el texto del elemento seleccionado en ese momento.
    - **PIP**: DISPSTD.GET_PROPIEDAD del Item.
    - **PBP**: DISPSTD.GET_PROPIEDAD del Bloque.
    - **PFP**: DISPSTD.GET_PROPIEDAD del Programa.

Devolver valor desde el plug-in al programa llamador

Para devolver valores a campos del programa llamador desde el programa llamado usaremos la siguiente instrucción:

VALIDACIONES.RETORNO_PLUG_IN(<valor a devolver>, <destino>);

- **<valor a devolver>**: Valor que envía el programa llamado al programa llamador.
- **<destino>**: Campo o parámetro del programa llamado en donde se va asignar el valor.

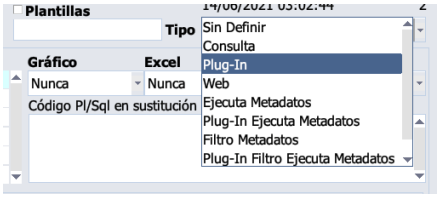
Ejemplo:

```
VALIDACIONES.RETORNO_PLUG_IN(:bsustituidos.codigo_articulo_sus, 'B8.REFERENCIA');
```

Este código asigna en B8.REFERENCIA del programa llamador el valor contenido en bsustituidos.codigo_articulo del programa llamado.

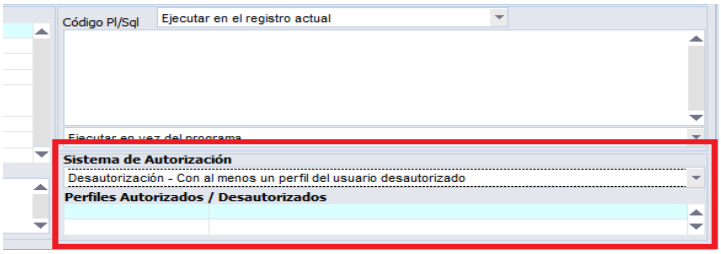
Permisos

Cuando se desarrolla un programa para que funcione únicamente como plug-in, para evitar tener que ponerlo en los menús por perfil se puede indicar en el tipo del programa que es "Plug-in".



Autorizar / Desautorizar plug-in

En el mantenimiento de programas personalizados se puede especificar que perfiles tienen acceso a un determinado plug-in o qué perfiles lo tienen desautorizado.



Si en el campo "Sistema de Autorización" se indica "Desautorización" los perfiles indicados en "Perfiles Autorizados / Desautorizados" no podrán ejecutar el plug-in, si no se especifica ninguno todos los usuarios pueden usar el plug-in.

Si en "Sistema de Autorización" se indica "Autorización" solo los perfiles indicados en "Perfiles Autorizados / Desautorizados" podrán ejecutar el plug-in, si no se especifica ninguno ningún usuario podrá ejecutar el plug-in.

Plug-ins globales a un programa

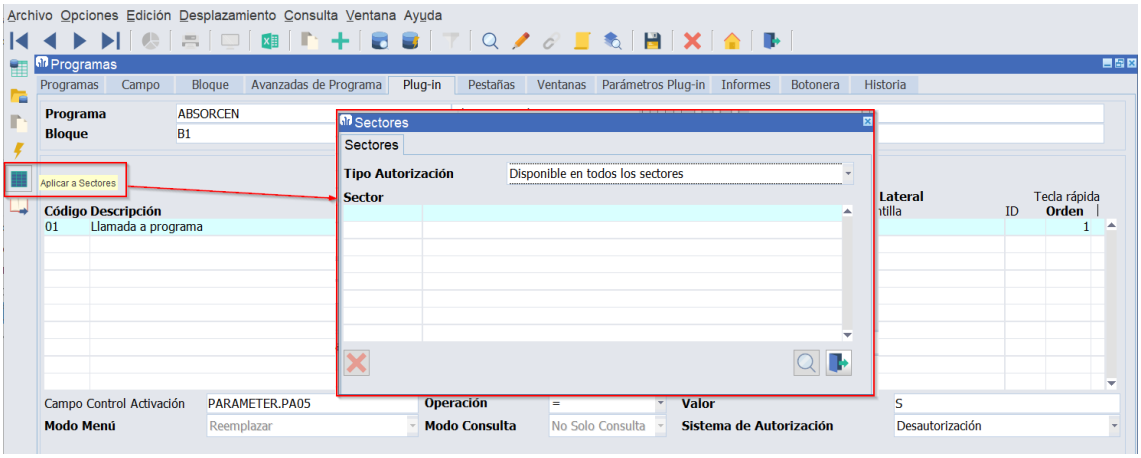
Se pueden crear plug-ins globales a todos los bloques de un programa, el funcionamiento es similar a cuando se crean para un determinado bloque, lo único que hay que hacer es asociarlos al bloque .GLOBALPLUGINS (importante, tiene un punto al principio del texto GLOBALPLUGINS).

Crear plug-ins globales a todos los programas de Libra.

También se pueden crear plug-ins globales a todos los programas y sus bloques, para ello hay que dar de alta el programa con código .GLOBALPLUGINS (importante, tiene un punto al principio del texto GLOBALPLUGINS) con un único bloque .GLOBALPLUGINS (también con un punto al principio del texto).

Control de visualización del plug-in según sector del grupo empresarial

Se puede indicar a los sectores a los que aplica el plug-in, de forma que se puede hacer que dependiendo de los sectores activados a nivel de grupo empresarial el plug-in esté o no disponible. Si hay configuración de sectores para el plug-in el campo "Orden" se mostrará con fondo rojo.



En el campo "Tipo Autorización" se indica como debe de considerar los sectores que se indican en el multiregistro:

- **Disponible en todos los sectores**: Se ignoran los sectores indicados
- **Sólo para los sectores indicados**: Se aplicará en los grupos empresariales que tengan configurados alguno de los sectores indicados.
- **Excluir en los sectores indicados**: Al contrario que el valor anterior, se mostrará únicamente cuando el usuario esté validado en Libra en un grupo empresarial que no tenga activado alguno de los sectores que se indican en el multiregistro.

Duplicado automático de tablas detalle al duplicar registro de bloque

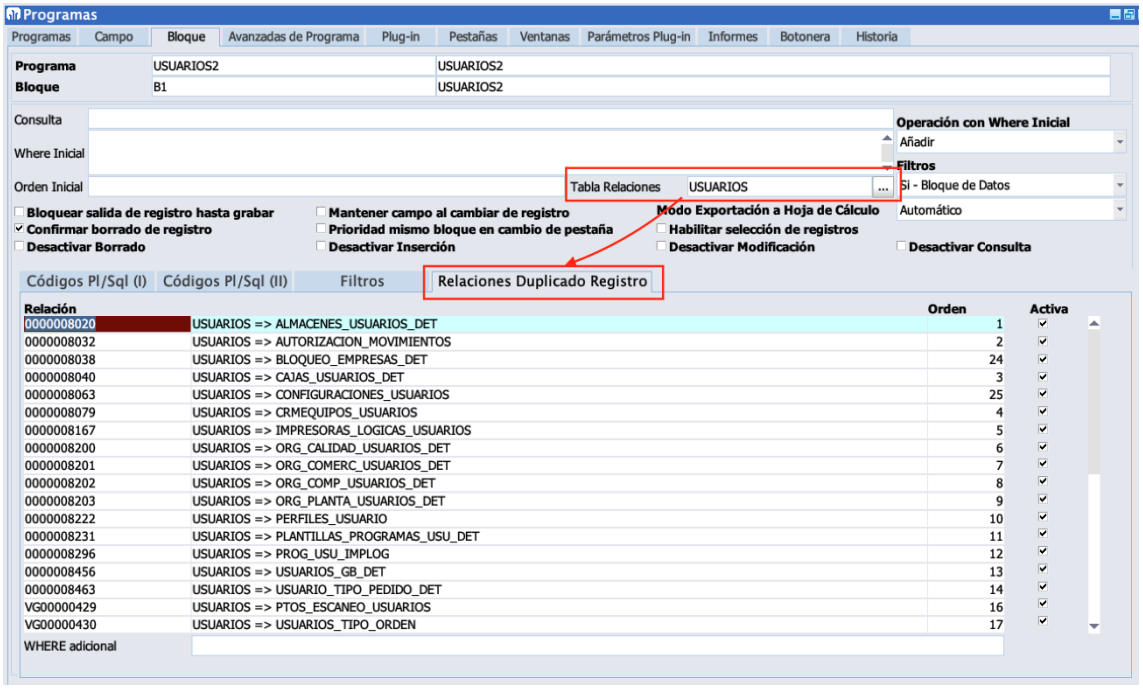
A los bloques se les puede asociar una lista de relaciones que se usarán para duplicar los registros hijos de la tabla asociada al bloque cuando se duplique un registro. Al indicar los códigos de relación ya es el entorno quien se encarga de todo, crear los registros de detalle y consultar los bloques hijos en el caso de ser necesario. Las tablas que se duplican pueden o no estar en el mismo programa, por ejemplo, en el programa de "Organizaciones Comerciales" se puede indicar la relación "0000007310" y con eso al duplicar una organización comercial también duplicará los tipos de pedido.

Para que aparezca la pestaña "Relaciones Duplicado Registro" hay que indicar el campo "Tabla Relaciones", que por lo general coincidirá (salvo por ejemplo cuando hay vistas actualizables) con la tabla asociada al bloque.

Se puede utilizar cualquier relación donde la tabla maestra sea la indicada en tabla relaciones, incluso se pueden utilizar las relaciones desactivadas para el chequeo de integridad. En el caso de que no se quiera tener en cuenta una relación por personalización habrá que darla de alta en programas personalizados y desactivar la check "Activa".

En "WHERE adicional" se incluirá la condición que deben de cumplir los registros a duplicar, por lo general no hace falta meter ahí ningún tipo de where, únicamente en aquellos casos que existan ciertos tipos de registro no se quieran duplicar.

El campo "Orden" indica el orden en el que han de procesarse las relaciones. Es importante el orden cuando hay claves foráneas en las tablas ya que debe duplicarse antes la cabecera que las líneas ya que si se duplicasen líneas sin cabeceras fallaría el proceso al violarse la integridad de la clave foránea.



Documentación de modificaciones en programas

Cuando alguien haga una modificación deberá de indicar quién la hizo, la fecha y un comentario, para que esté disponible esa información a consultores.

Personalizar programas

Cualquier modificación que se realice en el cliente en el mantenimiento de programas **será sobrescrita y por tanto perdida en el próximo cambio de versión**, para solucionar este problema se ha ideado un mantenimiento de programas personalizado. Este programa es similar al mantenimiento de programas anterior, pero con varias diferencias.

- Las modificaciones que se realicen en este mantenimiento sobre un programa prevalecerán sobre la información que exista para el mismo objeto (bloque, campo, pestaña, ventana) del mantenimiento de programas estándar. Estas modificaciones no serán nunca sobrescritas al realizar un cambio de versión.
- En las pestañas “Botonera” y “Pestañas” tiene un bloque de Perfiles Autorizados / Desautorizados, en donde se introducirán los perfiles de usuario que podrán visualizar o no visualizar la pestaña, dependiendo del parámetro de Sistema de Autorización.
- Hay una pestaña para poder añadir plug-ins personalizados a un programa, los plug-in se describen en un apartado específico.

Se puede crear cualquier personalización sin que tenga que existir en el estándar, incluso se podría crear un programa directamente en programas personalizados sin que tenga que haber nada en las tablas estándar.

Modificar por código las propiedades cargadas del mantenimiento de programas.

DISPSTD.SET_PROPIEDAD

Una vez se han cargado las propiedades del mantenimiento de programas, se pueden alterar durante la ejecución desde el código fuente del programa.

Para modificar una propiedad se llamará a:

DISPSTD.SET_PROPIEDAD(<código>, <código_propiedad>, <valor_propiedad>, <tipo>);

- **<código>**: Dependerá de <tipo>, si se quiere modificar la propiedad e un campo será BLOQUE.CAMPO, en caso de un bloque será el nombre del bloque y en caso de un plugin es BLOQUE.CODIGO_PLUGIN.
- **<código_propiedad>**: Código de la propiedad a modificar (ver tabla de propiedades).
- **<valor_propiedad>**: Valor que se va a asignar a la propiedad.
- **<tipo>**: Puede contener los siguientes valores:
  - **C**: Propiedad de campo, es el valor por defecto que se asume si no se pasa este parámetro.
  - **B**: Propiedad de bloque.
  - **PI**: Propiedad de Plug-in

DISPSTD.GET_PROPIEDAD

Al igual que se puede establecer una propiedad para un campo se puede leer con la función:

DISPSTD.GET_PROPIEDAD(<código>, <código_propiedad>, <tipo>) RETURN <valor_propiedad>;

A nivel de campo

PROPIEDAD	***Descripción***	***Valores posibles***
ACTIVAR_CALCULADORA	Indica si el campo tiene calculadora.	S, N
ACTIVAR_CALENDARIO	Indica si el campo tiene calendario, si se asocia calendario implica que va a validar el campo como si fuese una fecha.	S, N
CODIGO_LISTA	Código de la lista de valores que tiene asociado el campo.	Código de una lista de valores que exista en LISTAS_VALORES_CAB o en LISTAS_VALORES_PERS_CAB.
LV_CODIGO_LISTA	Igual que en CODIGO_LISTA, pero aparte de establecer esta propiedad también carga de la lista de valores las propiedades: LV_COLUMNA_DESCRIPCION LV_COLUMNA_CODIGO LV_WHERE_DEFECTO LLAMAR_PROGRAMA PARAMETRO_LLAMAR_PROGRAMA CODIGO_PL_SQL_PREVALIDACION TRANSLATE_BC_ARG1 TRANSLATE_BC_ARG2 CASE_INSENSITIVE_QUERY_DESC	Igual que en CODIGO_LISTA. IMPORTANTE: Se debe de tener cuidado en que parte del código se pone la asignación de esta propiedad ya que implica acceso a la base de datos para cargar los datos de la lista de valores.
LV_EJECUTAR_CONSULTA	Indica si al llamar a la lista de valores va a ejecutar consulta o va a entrar directamente en modo de entrada de consulta.	S, N
LV_VALIDAR_DESDE_LISTA	Indica si se va a realizar la validación del campo usando la SELECT y la where de la lista de valores.	S, N, L (Forzar Lista de Valores Contextual), P (Llamar programa asociado)
LV_COLUMNA_DESCRIPCION	Número de columna de la lista de valores que contiene la descripción para trasladar al campo D_XXXXX si el usuario selecciona algo de la lista de valores.	1 .. 10
LV_COLUMNA_CODIGO	Número de la columna de la lista de valores que contiene el código que va a trasladar al campo desde el que se llama a la lista de valores si el usuario selecciona algo de la lista de valores.	1 .. 10

LV_CONSULTA_BD	SQL que debe de cumplir la normativa de formato para una lista de valores. Ver apartado: Listas de Valores.	Ejemplo: SELECT codigo_rapido c1, nombre c2, rowid rowid_lov FROM clientes
LV_WHERE_DEFECTO	Cláusula Where a aplicar a la lista de valores.	Ejemplo: codigo_empresa = :global.codigo_empresa
LV_WHERE_ENTER_QUERY	Cláusula Where a aplicar a la lista de valores cuando el bloque está en modo de entrada consulta	
LV_CODIGO_PL_SQL_PRE_EJECUCION	Código PL-SQL que se ejecutará cuando el usuario llama a la lista de valores, ver sección de mantenimiento de programas y de código PL/SQL.	Código PL/SQL
CODIGO_PL_SQL_VALIDACION	Código PL-SQL que se ejecutará después de la validación por lista de valores, ver sección de mantenimiento de programas y de código PL/SQL.	Código PL/SQL
CODIGO_PL_SQL_PREVALIDACION	Código PL-SQL que se ejecutará antes de la validación por lista de valores. Ver sección de mantenimiento de programas y de código PL/SQL.	Código PL/SQL
CODIGO_PL_SQL_ENTRADA	Código PL-SQL que se ejecutará cuando el cursor entra en el campo. Ver sección de mantenimiento de programas y de código PL/SQL.	Código PL/SQL
LLAMAR_PROGRAMA	Programa al que se llama cuando el usuario hace doble click en el campo o cuando pulsa en el botón de hipervínculo de la botonera.	Ejemplo: CLIENTES
PARAMETRO_LLAMAR_PROGRAMA	Parámetro que se pasa al programa	Ejemplo: CODIGO_CLIENTE
DESACTIVA_BUSQUEDA_CONTEXTUAL	Si se desactiva y el usuario tiene activada búsqueda contextual en el campo que se desactiva no funciona.	S, N
VALIDACION_FILTRO	Indica si se va a validar como si fuese un filtro, es decir, si lo que introduce el usuario no existe en la validación desde lista de valores se permite continuar y si existe se carga la descripción.	S, N
VALOR_POR_DEFECTO	Valor por defecto que se va a asignar al programa en los registros nuevos.	
LISTA_VALORES_GRUPO	Indica si la lista de valores va a ser normal o se va a ir mostrando por grupos.	S: Si con 9 registros. 5: Si con 5 registros. N: No. M: Multiselección. T: Multiselección Totalizada.
TRANSLATE_BC_ARG1	Texto reemplazado en la búsqueda contextual. Para más información ver el apartado: Listas de valores.	
TRANSLATE_BC_ARG2	Texto a reemplazar en la búsqueda contextual. Para más información ver el apartado: Listas de valores.	
CASE_INSENSITIVE_QUERY_DESC	Indica si al hacer la búsqueda contextual por la descripción se va a ignorar la diferencia entre mayúsculas y minúsculas.	S, N
COLUMNA_ORDER	Campo o SQL por la que se va a ordenar cuando se indica con el botón derecho sobre el campo que se quiere ordenación ascendente o descendente.	Ejemplo: LPAD(codigo_rapido, 8, '0')
LV_WHERE_DEFECTO2	Segunda cláusula where de la lista de valores, si se especifica se habilita un botón en la lista de valores para conmutarlas.	
ETIQUETA_BOTON_WHERE	En caso de especificar una segunda cláusula where a la lista de botón será la etiqueta que tendrá el botón para conmutar a la segunda.	
ETIQUETA_BOTON_WHERE2	En caso de especificar una segunda cláusula where a la lista de botón será la etiqueta que tendrá el botón para conmutar a la primera.	

TAMANO_MAXIMO	Número máximo de caracteres admitidos.	
SELECCION_ARCHIVO	Indica si se habilita lista de valores para selección de archivos para el campo.	N: Sin selección de archivo. D: Selección de directorio. G: Selección de archivo para grabar. A: Selección de archivo para abrir.
SEL_ARCHIVO_CAMPO_DIRECTORIO	En caso de estar activada la selección de archivos en qué BLOQUE.CAMPO devuelve el directorio seleccionado.	
SEL_ARCHIVO_CAMPO_ARCHIVO	En caso de estar activada la selección de archivos en qué BLOQUE.CAMPO devuelve el archivo sin ruta seleccionado.	
SEL_ARCHIVO_TITULO	Titulo que aparecerá en la ventana de selección de archivo.	
SEL_ARCHIVO_FILTRO	Filtro de los archivos a mostrar.	
BLOQUEA_VALIDA_SIN_INTRO	Bloquear que no se pueda validar el campo sin pulsar INTRO o TABULACIÓN	S: bloqueado. N: Sin bloquear.
CODIGO_PL_SQL_VALIDA_ENTER	Código PL/SQL que se ejecutará cuando el usuario pulse INTRO en el campo.	
LV_BLOQUEAR_SALTO_CAMPO	Bloquear que salte de campo cuando se selecciona un valor de una lista de valores.	S: Bloqueado el salto de campo. N: Salta de campo.
PL_SQL_VALIDA_ENTER_INVALID	Indica si el código de CODIGO_PL_SQL_VALIDA_ENTER se ejecuta sólo cuando el campo esté inválido.	S: Se ejecuta sólo cuando el campo está inválido. N: Se ejecuta siempre que el usuario pulse INTRO o TABULACIÓN
DESACTIVAR_LV_AUTOMATICA	Si el usuario / empresa tiene marcado que en campos obligatorios con lista de valores si se intentan dejar en blanco se lance automáticamente la lista de valores, se puede desactivar mediante esta opción.	S: Se desactiva la lista de valores automática. N: No se desactiva.
DEVOLVER_VALORES_SELECCIONADOS	En caso de que el campo tenga en LISTA_VALORES_GRUPO el valor M (Multiselección) o T (Multiselección totalizada) se puede indicar que los valores que se han seleccionado automáticamente se devuelvan al campo que llama a la lista de valores.	S: Se devuelven los valores seleccionados separados por SEPARADOR_MULTISELECCIONADOS. N: No se devuelve, en el código fuente del programa se tiene que contemplar.
SEPARADOR_MULTISELECCIONADOS	Carácter que separará los distintos valores que se devuelven de la selección en una lista de valores de multiselección.	
FILTRO_OBLIGATORIO_EJE_CONS	Indica si para hacer F8 o pulsar en botón el campo tiene que tener valor, es decir, se obligaría a entrar en modo entrada de consulta (F7) meter un valor en ese campo y luego ejecutar la consulta.	S: Obligatorio. N: Opcional.
CODIGO_PL_SQL_DOBLE_CLICK	Código PL/SQL que se ejecutará al hacer doble click en el campo.	
ETIQUETA_EXCEL	Etiqueta a usar como título de la columna cuando se exporten los valores del campo a hoja de cálculo.	
EDITOR_CAMPO	A: Automático, F: Nativo de Forms, P: Texto Plano, H: HTML	
APLICA_VA_CURRENT_RECORD	Únicamente se utiliza en SET_PROPIEDAD, permite aplicar el atributo visual indicado a un campo del registro actual. De esta forma se puede aplicar cualquier color definido en la vista V_COLORES_ERP. Ejemplo: dispstd.set_propiedad('BLOQUE.CAMPO', 'APLICA_VA_CURRENT_RECORD', 'BROWN_500');	

Ejemplo: Asignar al campo B1. CODIGO la lista de valores CLIENTES añadiendo la condición de que el estado del cliente sea el especificado en el campo B1.ESTADO, habilitando la validación por lista de valores.

```
IF NVL(DISPSTD.GET_PROPIEDAD('B1.CODIGO', 'LV_CODIGO_LISTA'), '.') != 'CLIENTES' THEN
  DISPSTD.SET_PROPIEDAD('B1.CODIGO', 'LV_CODIGO_LISTA', 'CLIENTES');
  DISPSTD.SET_PROPIEDAD('B1.CODIGO', 'LV_VALIDAR_DESDE_LISTA', 'S');
  DISPSTD.SET_PROPIEDAD('B1.CODIGO', 'LV_WHERE_DEFECTO', ':where_lov AND estado = :b1.estado');
END IF;
```

A nivel de bloque

PROPIEDAD	*Descripción*	*Valores posibles*
ENVIAR_EXCEL	Activa la posibilidad e envío a Excel cuando el cursor está en el bloque	S, N
BLOQUEAR_SALIDA_HASTA_GRABAR	Si tiene S, si se modifica un registro el usuario no va a poder salir de él mientras no grabe	S, N
CODIGO_PL_SQL_INICIALIZACION	Código PL/SQL que se ejecuta cada vez que se crea un registro nuevo.	
CODIGO_PL_SQL_VALIDACION	Código PL/SQL que se ejecuta cada vez que se valida el registro.	
CODIGO_PL_SQL_BORRADO	Código PL/SQL que se ejecuta cada vez que se borra un registro.	
CODIGO_PL_SQL_ENTRADA	Código PL/SQL que se ejecuta cada vez que el cursor entra en un registro.	
CODIGO_PL_SQL_POST_INSERT	Código PL/SQL que se ejecuta después de cada inserción de registro.	
CODIGO_PL_SQL_POST_UPDATE	Código PL/SQL que se ejecuta después de cada modificación de registro.	
CODIGO_PL_SQL_POST_DELETE	Código PL/SQL que se ejecuta después de cada borrado de registro.	
CODIGO_PL_SQL_CONSULTA	Código PL/SQL que se ejecuta por cada registro que viene de la base de datos al rellenarse el bloque.	
CODIGO_PL_SQL_ENTRADA_BLOQUE	Código PL/SQL que se ejecuta por cada vez que el cursor entra en el bloque.	
CODIGO_PL_SQL_PRE_DUPLICADO	Código PL/SQL que se ejecuta antes de duplicar un registro.	
CODIGO_PL_SQL_POST_DUPLICADO	Código PL/SQL que se ejecuta después de duplicar un registro.	
CONFIRMAR_BORRADO_REGISTRO	Se usa para indicar si se debe pedir confirmación al usuario para borrar el registro.	S: Se pide confirmación N: No se pide confirmación.
SELECCIONAR_CAMPOS_EXCEL	Indica si se va a pedir al usuario cuando envíe los datos del bloque a Excel los campos a ser exportados.	S: Se pide al usuario los campos a exportar. N: No se le piden los campos al usuario.
BLOQUEAR_SALIDA_HASTA_GRABAR	Si tiene S, si se modifica un registro el usuario no va a poder salir de él mientras no grabe	S, N
HABILITAR_SELECCION_REGISTROS	Si se pasa el valor S, se habilita la selección de registros múltiple.	S, N
MANTENER_CAMPO_CURSOR	Si se pasa el valor S, al cambiar de registro mantiene el cursor en el mismo campo mientras no se navegue con el ratón o a un registro nuevo.	S, N

A nivel de programa

PROPIEDAD	*Descripción*	*Valores posibles*
CODIGO_PL_SQL_GRABACION	Código PL/SQL que se ejecuta antes de grabar.	
CODIGO_PL_SQL_FINALIZACION	Código PL/SQL que se ejecuta antes de salir.	

SALIR_GRABAR_CAMBIOS_AUT	Indica si se graban automáticamente los cambios que estén pendientes de grabar sin preguntar al usuario cuando se sale del programa	S, N
PLT_VALIDA_RANGOS_CODIGO	Si se pasa el valor N, deshabilita la validación del rango de código de las plantillas.	S, N
ORDER_BY_NULLS	Si se pasa L, al ordenar los nulos se mostrarán al final	L, D
ACTIVA_TRAZA_SILENCIOSA	Permite activar a partir de se momento una traza silenciosa del programa, es decir, el usuario no recibirá en los mensajes los códigos que se añaden en una traza normal. Mediante este método se permite activar traza de forma controlada por programa.	BASICO WARN INFO DEBUG TRACE
DUPLICANDO	Devuelve si el usuario ha pulsado duplicar en un registro que tiene relaciones de detalle para ser duplicadas. N: No hay nada pendiente, S: El usuario pulsó duplicar y se está esperando a que el usuario cubra todos los datos de la clave primaria. F: El usuario ha cubierto los datos de la clave primaria, en cuanto se navegue a un nuevo campo se procederá al duplicado del detalle, una vez hecho el duplicado se vuelve a pasar a valor N.	S, N, F

A nivel de plug-in

PROPIEDAD	***Descripción***	***Valores posibles***
ACTIVA_PLUG_IN	Activa o desactiva un plug-in	S: Activa plug-in N: Desactiva plug-in
ICONO	Cambia el icono asignado al plug-in	
PROGRAMA_LLAMADO	Permite cambiar el programa que se va a ejecutar en el plug-in.	Si se cambia el programa y hay parámetros, el nuevo programa tiene que ser compatible con los parámetros indicados en el plug-in, en caso contrario la ejecución dará error.

A nivel de Informe, mediante IMP.SET_PROPIEDAD

PROPIEDAD	***Descripción***	***Valores posibles***
PL_SQL_PRE_EJECUCION_REP	Código PL/SQL para ejecutar antes del informe.	
PL_SQL_POST_EJECUCION_REP	Código PL/SQL para ejecutar después del informe.	
EJE_PL_SQL_PRE_REP_SCREEN	Indica si se ejecuta el código de preejecución del informe cuando el destino es pantalla.	S, N
EJE_PL_SQL_PRE_REP_PRINTER	Indica si se ejecuta el código de preejecución del informe cuando el destino es Impresora.	S, N
EJE_PL_SQL_PRE_REP_FILE	Indica si se ejecuta el código de preejecución del informe cuando el destino es Archivo.	S, N
EJE_PL_SQL_PRE_REP_MAIL	Indica si se ejecuta el código de preejecución del informe cuando el destino es Correo electrónico.	S, N
EJE_PL_SQL_PRE_REP_FAX	Indica si se ejecuta el código de preejecución del informe cuando el destino es Fax.	S, N
EJE_PL_SQL_PRE_REP_GESTDOC	Indica si se ejecuta el código de preejecución del informe cuando el destino es Gestión Documental.	S, N
EJE_PL_SQL_PRE_POST_SCREEN	Indica si se ejecuta el código de postejecución del informe cuando el destino es pantalla.	S, N
EJE_PL_SQL_PRE_POST_PRINTER	Indica si se ejecuta el código de postejecución del informe cuando el destino es Impresora.	S, N
EJE_PL_SQL_PRE_POST_FILE	Indica si se ejecuta el código de postejecución del informe cuando el destino es Archivo.	S, N
EJE_PL_SQL_PRE_POST_MAIL	Indica si se ejecuta el código de postejecución del informe cuando el destino es Correo electrónico.	S, N
EJE_PL_SQL_PRE_POST_FAX	Indica si se ejecuta el código de postejecución del informe cuando el destino es Fax.	S, N
EJE_PL_SQL_PRE_POST_GESTDOC	Indica si se ejecuta el código de postejecución del informe cuando el destino es Gestión Documental.	S, N
FICHERO_REPORT	Informe a ejecutar	
CODIGO_IMPRESORA	Código de la impresora que se usará como destino del informe.	

SERVIDOR_BIP	Dirección del servidor de BI-Publisher configurado	
NOMBRE_ARCHIVO_FIJADO	A partir de ese momento se propondrá como nombre de archivo el indicado, para reestablecerlo habrá que volver a establecer esta propiedad con el valor NULL. Se puede utilizar las siguientes variables de sustitución: • <usuario>: Se reemplaza por el código de usuario. • <fecha>: Se reemplaza por la fecha en formato YYYYMMDD • <hora>: Se reemplaza por la hora en formato HH24MISS El nombre de archivo debe de indicarse sin extensión ya que será añadida de forma automática. Si en el mantenimiento de programas, para el informe en cuestión se ha configurado un nombre de archivo ese prevalecerá sobre el indicado en esta propiedad.	
SALIR_AUT_BREPORT	Si se pasa S, después de cada impresión se cerrará la ventana de impresión BREPORT	S, N

A nivel de Lista de Valores, mediante LV.SET_PROPIEDAD

PROPIEDAD	***Descripción***	***Valores posibles***
CENTRAR_LOV	Se indica si al abrir la lista de valores se va a centrar en la pantalla o se va a quedar en la posición 0,0 de la pantalla.	S, N
WHERE_LOV	Cláusula Where que se va aplicar a la lista de valores	
WHERE_LOV2	Segunda Cláusula Where de la lista de valores, al especificarla se va habilitar un botón para poder conmutar.	
ETIQUETA_BOTON_WHERE	Cuando se especifica WHERE_LOV2, será la etiqueta que tenga el botón de conmutación cuando se está aplicando la where normal de la lista de valores	
ETIQUETA_BOTON_WHERE2	Cuando se especifica WHERE_LOV2, será la etiqueta que tenga el botón de conmutación cuando se está aplicando la segunda where de la lista de valores	
MULTISELECCION	Permite activar o desactivar la multiselección S: Activa la multiselección. T. Activa la multiselección con totalización de las columnas numéricas. N: Desactiva la multiselección.	S, N, T

Código PL/SQL

El código PL/SQL tiene las siguientes características:

- Se ejecuta en la base de datos, por tanto, **no se puede hacer referencia a funciones y procedimientos almacenados en librerías o en el fuente del programa**.
- Al ejecutarse en la base de datos, se pueden crear funciones, procedimientos y packages en la propia base de datos e introducir el código en cualquier lenguaje reconocido por la base de datos, ya sea PL/SQL, Java, XML, ...

Hay dos métodos de ejecución de código PL/SQL:

- **V0** (Legacy - Se usa en versiones de Libra anteriores a la 6.1.2).
  - Permite usar dentro del código hasta un máximo de 20 variables del propio programa de la misma forma que si se ejecutase en un bloque PL/SQL dentro de Forms, es decir, se pueden hacer referencia a campos de un bloque con :bloque.campo, a parámetros con :parameter.parametro, ... Estas referencias son de entrada / salida, es decir, si en el código PL/SQL de validación se hace una asignación a un campo del programa esta asignación se verá reflejada en el programa, por ejemplo si introducimos :bloque.campo := '10', después de la ejecución el campo 'campo' del bloque 'bloque' contendrá el valor 10. **IMPORTANTE**: Los campos de tipo DATE o DATETIME se enlazan como texto.
  - Se pueden usar variables de sólo lectura que no cuentan dentro de la limitación de las 20 variables máximas. Para indicar que la variable es de sólo lectura hay que hacer referencia a ella con ::, por ejemplo *::b1.codigo*
  - No se puede parar la ejecución a mitad del código, es decir, el código se ejecutará íntegramente, pero se pueden encadenar códigos PL/SQL usando :p_codigo_sql.
- **V1**: A diferencia de V0 (Legacy) tiene los siguientes cambios:
  - El número de variables a enlazar del programa es ilimitado.
  - Los campos de los bloques que sean DATE o DATETIME se enlazan como DATE en vez de como texto.

En versiones anteriores a 6.1.2 se ejecutarán en modo V1 y en 6.1.2 o posteriores en modo V2. Se puede forzar que un determinado código PL/SQL se ejecute en una versión u otra añadiendo al principio de todo uno de los siguientes comentarios:

- /*PLSQLV0*/: Fuerza la ejecución en modo V0 (Legacy) en versiones 6.1.2 o posterior.
- /*PLSQLV1*/: Fuerza la ejecución en modo V1 en versiones anteriores a la 6.1.2.

Existen los siguientes parámetros fijos que solo son de salida de resultados:

- **:p_parar_ejecucion**: Se le puede asignar 'S' , 'N', 'O', 'V', 'T', 'P' durante la ejecución del código, esto no implica que si asignamos el valor 'S' no se ejecute el código que hay a continuación, implica que si asignamos el valor 'S' después de haberse ejecutado la validación PL/SQL se va a realizar un RAISE Form_Trigger_Failure y por tanto, si es un código de validación, el campo va a continuar con el estado INVÁLIDO.

  Si asignamos el valor 'O' solo se parará la ejecución si el usuario pulsa el botón de cancelar en el mensaje que se le muestre.

  Si asignamos el 'V' y el usuario pulsa cancelar en el mensaje se restablecerá el valor que introdujo originalmente el usuario en el campo (puede ser que el código pl/sql lo modificase) y el campo queda inválido, es decir, se ejecuta un RAISE Form_Trigger_Failure. En caso de que acepte el mensaje continuará con el valor devuelto por el código PL/SQL y se valida el campo.

Con el valor 'T' funciona exactamente igual que 'V', pero si el usuario pulsa cancelar en el mensaje se restablece el valor introducido por el usuario originalmente en el campo, pero con la diferencia de que el campo queda validado.

Si se asigna el valor 'P', el campo o registro se valida independientemente si el usuario pulsa el botón de aceptar o el de cancelar, pero si hay algo en p_codigo_pl_sql únicamente se ejecuta si el usuario pulsa aceptar.

En el caso de recuperar el valor anterior, por defecto se recupera el valor que tenía el campo antes de ejecutar el PL/SQL, si se quiere reestablecer el valor que tenía el campo a un momento anterior a que el usuario lo modificase, se puede añadir un segundo carácter en :p_parar_ejecucion con el valor P o D. Con el valor P recupera el valor de antes de entrar en el campo y con B el valor que devuelve la propiedad DATABASE_VALUE para ese campo. Ejemplo: :p_parar_ejecucion := 'VP';

- **:p_tipo_mensaje** y :**p_codigo_mensaje**: Una vez finalizada la ejecución de la validación si tienen valor se mostrará el mensaje correspondiente de la tabla MENSAJES. Se puede personalizar el mensaje asignando el texto del mismo en la variable de salida **:p_texto_mensaje**. Es posible reemplazar cadenas dentro del mensaje que se va a mostrar mediante llamadas a **pkpantallas.set_msg_replace_texto(cadena1, cadena2)**, de forma que en el próximo mensaje que se muestre se cambia el texto indicado en "cadena1" por el que tenga "cadena2".
- **:p_valor_campo_ok**: Se usa en combinación con :p_tipo_mensaje y :p_codigo_mensaje. En esta variable se puede introducir el valor que deberá de asignarse al campo que está lanzando el mensaje en caso de que el usuario pulse el botón "Aceptar".
- **:p_valor_campo_cancel**: Se usa en combinación con :p_tipo_mensaje y :p_codigo_mensaje. En esta variable se puede introducir el valor que deberá de asignarse al campo que está lanzando el mensaje en caso de que el usuario pulse el botón "Cancelar".
- **:p_lv_lista_valores**: Una vez finalizada la ejecución del código, si este campo termina con valor, se ejecutará la lista de valores que indique el valor, independientemente de la lista de valores que tenga asociado el campo en el mantenimiento de programas. Si se usa en un código de validación (pre validación o validación) fallará si el usuario sale del campo usando el ratón, para evitar este problema debemos comprobar antes de asignarle valor si se está ejecutado desde un disparador KEY-NEXT-ITEM con el parámetro **:p_ejecutado_desde_kni** que contendrá el valor 'S' si se puede lanzar correctamente la lista de valores. En caso de querer evitar que se valide el campo sin lanzar la lista de valores deberemos usar :p_ejecutado_desde_kni en combinación con **:p_tipo_mensaje**, **:p_codigo_mensaje** y **:p_parar_ejecucion** como se muestra en el siguiente ejemplo:

```
IF :p_ejecutado_desde_kni = 'N' THEN
  :p_parar_ejecucion := 'S';
  :p_tipo_mensaje := 'CAMPO';
  :p_codigo_mensaje := 'ENTER';
ELSE
  :p_lv_lista_valores := 'CLIENTES';
  :p_lv_ejecutar_consulta := 'S';
END IF;
```

- **:p_lv_ejecutar_consulta**: Se usa en el caso de que *:p_lv_lista_valores* devuelva valor, si se indica el valor 'S' la lista de valores se ejecutará mostrando registros, mientras que si tiene el valor 'N' la lista de valores se ejecutará en modo ENTER-QUERY, es decir, en modo de entrada de filtro.
- **:p_lv_where**: Se usa en el caso de que *:p_lv_lista_valores* devuelva valor, indicará la condición con la que se debe ejecutar la lista de valores. Si se deja en blanco usará la where que tenga por defecto la lista de valores.
- **:p_lv_consulta_bd**: Se usa en el caso de que *:p_lv_lista_valores* devuelva valor, permite modificar la SELECT que va a enviar la lista de valores a la base de datos, si este parámetro no se especifica se usará la SELECT que tenga por defecto la lista de valores.

IMPORTANTE: "**:p_lv_lista_valores**", "**:p_lv_ejecutar_consulta**", "**:p_lv_where**" y "**:p_lv_consulta_bd**" no afectan a las reglas de validación del campo, es decir, únicamente se utilizan en la ventana de la lista de valores a la hora de visualizar los valores, pero internamiente no cambia el comportamiento del campo, para modificar la lista de valores hay que usar comandos plug-in, por ejemplo:

```
PKPANTALLAS.INICIALIZAR_CODIGO_PLUG_IN;
PKPANTALLAS.COMANDO_PLUG_IN('PKLIBPNT_SIP', 'BLOQUE.CAMPO', 'LV_CODIGO_LISTA', 'CLIENTES');
PKPANTALLAS.COMANDO_PLUG_IN('PKLIBPNT_SIP', 'BLOQUE.CAMPO', 'LV_EJECUTAR_CONSULTA', 'S');
PKPANTALLAS.COMANDO_PLUG_IN('PLUGIN', 'BLOQUE.CAMPO', 'LV_VALIDAR_DESDE_LISTA', 'S');
PKPANTALLAS.COMANDO_PLUG_IN('PKLIBPNT_SIP', 'BLOQUE.CAMPO', 'LV_WHERE_DEFECTO', ':where_lov AND <condición>');
```

- **:p_codigo_pl_sql**: Permite que la ejecución de un código PL/SQL devuelva otro PL/SQL a ejecutar en caso de que el parámetro :p_parar_ejecucion sea N, O u P y el usuario pulse aceptar el mensaje. De esta manera podemos lanzar una pregunta y en base a la respuesta del usuario ejecutar un proceso o no. El uso es ilimitado, es decir un :p_codigo_pl_sql podría devolver otro código y así ir encadenando preguntas al usuario. **Ejemplo**: Se pregunta al usuario si quiere borrar el registro, si pulsa en "Aceptar" se ejecutará lo que tenga :p_codigo_pl_sql, si pulsa "Cancelar" no se hará nada.

```
:p_parar_ejecucion := 'O';
:p_tipo_mensaje := 'COMPR';
:p_codigo_mensaje := 'TEXTOLIB';
:p_texto_mensaje := '¿Desea borrar el registro?';
:p_codigo_pl_sql := 'DELETE FROM tabla WHERE condicion';
```

- **:p_ejecutar_programa**: Una vez finalizada la ejecución del código, si este campo termina con valor, se ejecutará el programa que indique el valor. Para pasar parámetros al programa se usaremos las siguientes instrucciones:
  - **PKPANTALLAS.INICIALIZAR_PARAMETROS_PLUG_IN**: Se ejecuta sin ningún parámetro y solo lo ejecutaremos una vez antes de pasar ningún parámetro.
  - **PKPANTALLAS.PARAMETRO_PLUG_IN(<parámetro>, <tipo>, <valor>)**: Se llamará una vez por cada parámetro a pasar.
    - **<parámetro>**: Nombre del parámetro que recibe el programa llamado, este dato depende del programa al que llamemos.
    - **<tipo>**: Le indicamos si el parámetro es una constante o es una referencia a un campo del programa. 'C': Constante, 'R': Referencia.
    - **<valor>**: De donde va a obtener el valor del programa principal para ser pasado al parámetro del programa plug-in. Este valor depende del tipo:
      - Por referencia, se puede obtener de:
        - Campo, especificaremos BLOQUE.CAMPO del que queremos obtener el valor. Este será la opción más común.
        - Variable global, especificaremos GLOBAL.VARIABLE.
        - De un parámetro local del programa principal, especificaremos PARAMETER.NOMBRE_PARAMETRO
      - Constante: Será un valor fijo, por ejemplo '10002'.

Se puede hacer que al llamar a un programa con :**p_ejecutar_programa** se ejecute con una determinada personalización asignando la personalización a la variable **:global.id_personalizacion**:

```
:global.id_personalizacion := '1';
:p_ejecutar_programa := 'CONSGEN';
:p_modo_menu_prog_llamado := 'DO_REPLACE';
:p_modo_consulta_prog_llamado:= 'NO_QUERY_ONLY';
```

Se puede hacer que al llamar a un programa dinámico se ejecute con una determinada plantilla haciendo que se ignoren los permisos del usuario sobre esa plantilla asignado el código de la plantilla a **:global.forzar_plantilla**:

```
:global.forzar_plantilla := 'NACIONALES';
:p_ejecutar_programa := 'CLIENTES';
:p_modo_menu_prog_llamado := 'DO_REPLACE';
:p_modo_consulta_prog_llamado:= 'NO_QUERY_ONLY';
```

El usuario puede tener un mismo programa en varias opciones de menú y esas opciones de menú pueden tener algunas opciones particulares que las diferencian unas de otras, por ejemplo, en las impresoras disponibles, para forzar que un programa se ejecute asumiendo la parametrización de una determinada opción de menú se puede hacer asignando el código de la opción de menú a **:p_ejecutar_programa**, con el prefijo **MN:**

```
:p_ejecutar_programa := 'MN:2V2500000510';
:p_modo_menu_prog_llamado := 'DO_REPLACE';
:p_modo_consulta_prog_llamado:= 'NO_QUERY_ONLY';
```

Se puede forzar la carga de una plantilla de valores del bloque de filtro mediante la variable **:global.id_plantilla_valores_defecto**.
También se puede hacer que al ejecutar el programa ejecute algo de código, por lo que se podría hacer una llamada por plug-in a un programa que no está preparado para ser llamado como plug-in, el código que se quiere ejecutar en el programa destino hay que pasárselo al procedimiento: **pkpantallas.set_codigo_pl_sql_inicio(<codigo>);**

Ejemplo: Desde la vista 360º, en la pantalla de cliente, el código que habría que usar en un plug-in para llamar al programa CONPED1 (consulta de pedidos) para que al entrar haga la consulta del cliente que tenemos en pantalla, y además quite la pestaña de filtros para que no se pueda cambiar la consulta.

```
:p_ejecutar_programa := 'conped1';
:p_modo_menu_prog_llamado := 'DO_REPLACE';
:p_modo_consulta_prog_llamado:= 'NO_QUERY_ONLY';
PKPANTALLAS.SET_CODIGO_PL_SQL_INICIO('PKPANTALLAS.INICIALIZAR_CODIGO_PLUG_IN;
PKPANTALLAS.COMANDO_PLUG_IN(''COPY'',    '''    ||    :bcliente.v_codigo_rapido    ||    ''',
''B1.DESDE_CLIENTE'');
PKPANTALLAS.COMANDO_PLUG_IN(''COPY'',    '''    ||    :bcliente.v_codigo_rapido    ||    ''',
''B1.HASTA_CLIENTE'');
PKPANTALLAS.COMANDO_PLUG_IN(''VALIDATE'', ''RECORD_SCOPE'');
PKPANTALLAS.COMANDO_PLUG_IN(''SYNCHRONIZE'');
PKPANTALLAS.COMANDO_PLUG_IN(''EXECUTE_TRIGGER'', ''CONSULTAR'');
PKPANTALLAS.COMANDO_PLUG_IN(''STPP'',          ''CANVAS_BASE.TAB0'',          ''VISIBLE'',
''PROPERTY_FALSE'');');
```

También se puede utilizar para llamar a un Report, para ello hay que indicar la extensión .REP, por ejemplo: **:p_ejecutar_programa := 'informe.rep';.** Con PKPANTALLAS.INICIALIZAR_PARAMETROS_PLUG_IN y PKPANTALLAS.PARAMETRO_PLUG_IN se le pueden pasar parámetros al informe.

Para indicar por donde imprimir el informe se puede utilizar (ver apartado: Impresión Multidestino) PKPANTALLAS.INICIALIZA_MULTIDESTINO_REPORT y PKPANTALLAS.ADD_MULTIDESTINO_REPORT. En el caso de no indicar el destino se abrirá la pantalla típica de selección de destino de impresión.

Si no se indica un destino se abrirá una pantalla donde el usuario deberá indicar si quiere el informe por pantalla, impresora, etc. Al seleccionar por impresora saldrán tanto las impresoras marcadas como horizontales como las marcadas verticales, este comportamiento se puede modificar:

- Para forzar que únicamente salgan las verticales hay que ejecutar: PKPANTALLAS.SET_VARIABLE_ENV('IMP_TIPO_IMPRESORA', 'V');
- Para forzar que únicamente salgan las horizontales hay que ejecutar: PKPANTALLAS.SET_VARIABLE_ENV('IMP_TIPO_IMPRESORA', 'H');

- **:p_modo_menu_prog_llamado**: Se usa sólo si se ha especificado valor para *:p_ejecutar_programa*. Indica si al llamarse el programa se debe de mantener el menú del programa llamador en el programa llamado o que este inicialice su propio menú. Valores posibles:
  - **NO_REPLACE**: (Valor por defecto, si no se especifica esta variable asumirá este valor). Se mantiene el menú del programa llamador en el programa llamado.
  - **DO_REPLACE**: Se inicializa el menú del programa llamado.
- **:p_modo_consulta_prog_llamado**: Se usa sólo si se ha especificado valor para *:p_ejecutar_programa*. Indica si al llamarse al otro programa se va a hacer en modo de solo consulta o no.
  - **QUERY_ONLY**: (Valor por defecto, si no se especifica esta variable asumirá este valor). En el programa llamado solo se podrán ejecutar consultas, nunca modificación de datos.
  - **NO_QUERY_ONLY**: En el programa llamado se pueden modificar datos.
- **:p_esperar_fin_programa_llamado**: Valores posibles:
  - *N: (:p_esperar_fin_programa_llamado := 'N';)*. Se utiliza para indicar que el programa llamador no debe de quedar a la espera de que termine el programa llamado. En caso de activar esa opción el programa llamado funciona igual que si el usuario fuese por ventanas y lo abriese desde el menú, es decir, el programa llamador continúa su ejecución, la conexión a la base de datos es distinta para cada programa. El inconveniente principal es que el programa llamado no puede devolver valores al programa llamador y tampoco puede compartir variables de sesión de la base de datos, y como ventaja se evita el mensaje "No se puede iniciar otra llamada a pantalla" cuando el usuario tiene varios programas llamados de forma concurrente.
  - *S: (:p_esperar_fin_programa_llamado := 'S';)*. El programa llamador se queda a la espera de que termine la ejecución del programa llamado. Si no se indica nada en esta variable este es el valor por defecto que asume. Es importante tener en cuenta que cuando se deshabilita que se espere por el programa llamado se ejecuta OPEN_FORM y Oracle Forms no ejecutará nada más a partir de ese momento ya que el control pasa totalmente al programa llamado, si se requiere que una vez se ejecute el programa se continúe ejecutando código no se puede utilizar esta opción.
  - *H: (:p_esperar_fin_programa_llamado := 'H';)*. Funciona de la misma forma que el valor S, pero con la diferencia de que se le indica al programa llamador que debe de ocultarse. Esto es especialmente útil cuando el código PL/SQL se ejecuta cuando hay una ventana modal abierta en el programa llamador, ya que si se indica el valor S la ventana quedará por encima del programa llamado y quedará Libra bloqueado.
- **:global.call_form_modo_post**: Si se pasa el valor S, en el programa llamado no se podrá realizar un COMMIT, el usuario podrá modificar datos pero estos no serán grabados hasta que no salga del programa y grabe en el programa llamador.

Hay parámetros fijos que sólo son de entrada. Estos parámetros son los siguientes:

- **:p_tipo_programa**: Tendrá el valor del campo Tipo de Programa del mantenimiento de programas.
- **:p_validar_desde_lista**: Tendrá el valor del campo Validar desde Lista de Valores.
- **:p_ejecutado_desde_kni**: Contendrá el valor 'S' si el código se está ejecutando por la acción de usuario de pulsar ENTER o TAB sobre el campo, en otro caso tendrá el valor 'N'.
- **NAME_IN('SYSTEM.TRIGGER_ITEM')**: Es sustituido por el contenido de la variable de sistema :system.trigger_item. NOTA: Tiene que estar escrito todo en mayúsculas, es decir, name_in('SYSTEM.TRIGGER_ITEM'); no será sustituido.
- Se pueden pasar variables entre códigos PL/SQL, mediante funciones SET_VARIABLE_ENV y GET_VARIABLE_ENV definidas en el paquete PKPANTALLAS, para más información ver en el apartado: Variables y parámetros globales, la sección: Definibles dinámicamente.

En los bloques PL/SQL se puede controlar el resultado de la ejecución de las listas de valores con las siguientes funciones:

- **pkpantallas.get_valor_ultima_ejecucion_lov('CAMPO')**: Devuelve el nombre del último campo que ha ejecutado una lista de valores en formato BLOQUE.CAMPO.
- **pkpantallas.get_valor_ultima_ejecucion_lov('ROWID')**: Devuelve el rowid del último registro seleccionado por lista de valores, si la última ejecución de la lista de valores se canceló devolverá NULL.
- **pkpantallas.get_valor_ultima_ejecucion_lov('VALOR_RETORNADO')**: Devuelve el último código del último registro seleccionado por lista de valores, si la última ejecución de la lista de valores se canceló devolverá NULL.

Ejemplos:

```
IF :campos.divisa_etiqueta IS NOT NULL AND :campos.doble_etiquetaje IS NOT NULL THEN
  IF :campos.divisa_etiqueta = 'EUR' AND :campos.doble_etiquetaje = '1' THEN
    :campos.doble_etiquetaje := '2';
    :p_parar_ejecucion := 'S';
    :p_tipo_mensaje := 'CAMPO';
    :p_codigo_mensaje := 'TEXTOLIB';
    :p_texto_mensaje := 'Si la divisa es EUR no se permite el primer valor';
  ELSIF :campos.divisa_etiqueta <> 'EUR' AND :campos.doble_etiquetaje <> '1' THEN
    :campos.doble_etiquetaje := '1';
    :p_parar_ejecucion := 'S';
    :p_tipo_mensaje := 'CAMPO';
    :p_codigo_mensaje := 'TEXTOLIB';
  :p_texto_mensaje := 'Si la divisa no es EUR el único valor permitido es el primero';
  END IF;
END IF;
```

Ejemplo con NAME_IN('SYSTEM.TRIGGER_ITEM'):

```
NAME_IN('SYSTEM.TRIGGER_ITEM')          :=          PKVALIDACIONES.COMPRUEBA_ARTICULO(:global.codigo_empresa,
NAME_IN('SYSTEM.TRIGGER_ITEM'),  TO_DATE(:global.fecha_trabajo,  :global.nls_date_format),  :p_tipo_programa,
:global.usuario,     :global.superusuario,     :p_parar_ejecucion,     :p_tipo_mensaje,     :p_codigo_mensaje,
:p_texto_mensaje);
```

Lo que realmente ejecutará este código si el contenido de :system trigger_item es :b3.codigo_articulo, y el tipo de programa es CONSULTA:

```
:b3.codigo_articulo    :=    PKVALIDACIONES.COMPRUEBA_ARTICULO(:global.codigo_empresa,    :b3.codigo_articulo,
TO_DATE(:global.fecha_trabajo,  :global.nls_date_format),  'CONSULTA',  :global.usuario,  :global.superusuario,
:p_parar_ejecucion, :p_tipo_mensaje, :p_codigo_mensaje, :p_texto_mensaje);
```

Ejemplo de lanzamiento de lista valores.

```
IF :b1.tipo_entidad = 'PR' THEN
  :p_lv_lista_valores := 'PROVEEDORES';
  :p_lv_ejecutar_consulta := 'S';
  :p_lv_where := 'codigo_empresa = :global.codigo_empresa AND nivel_legal = ''S''';
ELSE
  :p_lv_lista_valores := 'CLIENTES';
  :p_lv_ejecutar_consulta := 'S';
  :p_lv_where := 'codigo_empresa = :global.codigo_empresa AND nivel_legal = ''S''';
END IF;
```

Generación de hojas de cálculo desde códigos PL/SQL

Se permite generar desde los códigos PL/SQL, para ello se usa el paquete PKXLSBD de la misma forma que el paquete PKXLS que se explica en el apartado: **Generación** de hojas de cálculo.

Ejecutar operaciones de Forms desde PL/SQL de Libra.

Un código PL/SQL puede devolver ciertas operaciones que se ejecutarán en el programa, como por ejemplo cambiar propiedades de campos, mover el cursor de campo, ejecutar triggers, ...

Las operaciones que se ejecutarán en Forms serán ejecutadas secuencialmente, no hay opción a tomar decisiones (salvo alguna pequeña excepción) por el medio de ellas, ni realizar bucles.

Para ello en el PL/SQL hay que ejecutar las siguientes instrucciones:

- **PKPANTALLAS.INICIALIZAR_CODIGO_PLUG_IN**: Sólo se ejecutará una vez, e indicamos donde comenzamos a introducir las instrucciones a ejecutar en el programa.
- **PKPANTALLAS.COMANDO_PLUG_IN**(<operación>, [parametro1], [parametro2], [parametro3]);
  - <operación>: Ver tabla de operaciones.
  - [parametro1, 2 y 3]: Opcional, y es obligatorio especificarlo si en la tabla de operaciones lo usa la instrucción que se ejecuta.

Estas llamadas también se pueden hacer en un plug-in y se ejecutarán en el programa llamador cuando se cierre el programa llamado.

Puede haber operaciones que den problemas si se usan en un código de validación (prevalidación o validación) si el usuario sale del campo usando el ratón, para evitar este problema (cuando se use una de esas operaciones) se puede comprobar si se está ejecutando desde un disparador KEY-NEXT-ITEM con el parámetro **:p_ejecutado_desde_kni** que contendrá el valor 'S'. En caso de querer evitar que se valide el campo sin ejecutar nada deberemos usar :p_ejecutado_desde_kni en combinación con **:p_tipo_mensaje**, **:p_codigo_mensaje** y **:p_parar_ejecucion**.

Operaciones soportadas:

Operación	***Ejecuta***
COPY	COPY(parametro1, parametro2)
DEFAULT_VALUE	DEFAULT_VALUE(parametro1, parametro2)
GO_ITEM	GO_ITEM(parametro1)
GO_BLOCK	GO_BLOCK(parametro1)
GO_RECORD	GO_RECORD(parametro1)
DO_KEY	DO_KEY(parametro1)
EXECUTE_TRIGGER	EXECUTE_TRIGGER(parametro1)
MSG_REPLACE_TEXTO	MSG.REPLACE_TEXTO(parametro1, parametro2)
ALERTA	MSG.ALERTA_PERSONAL(parametro1, parametro2, parametro3)
VALIDATE	VALIDATE(parametro1). Ver en la ayuda de Forms, los valores que recibe el built-in VALIDATE.
ERASE	ERASE(parametro1)

IF_FF_RFTF	IF Form_Failure THEN
	RAISE Form_Trigger_Failure;
	END IF;

FF	RAISE Form_Trigger_Failure;
HOST	HOST(parametro1) ó HOST(parametro1, parametro2);
HOST_CLIENT	Ejecuta la aplicación indicada en parametro1 en el equipo cliente. Si se utiliza en cliente / servidor, tiene el mismo funcionamiento que el comando "HOST".

REFRESCAR_BLOQUE	Vuelve a ejecutar consulta en el bloque en que se encuentra el cursor, respetando los filtros aplicados por el usuario y vuelve a posicionarse en el mismo registro y campo en el que se encontraba el cursor.
SELECT_ALL	Selecciona todo el texto del campo. Se podría meter en el PL/SQL de entrada en campo de los campos multilínea para forzar que se seleccione todo el texto siempre al entrar.
CURSOR_STYLE	Set_Application_Property(CURSOR_STYLE, parametro1);
SIP	Set_Item_Property(parametro1, parametro2, parametro3).Ver en la ayuda de Forms, los valores que recibe el built-in Set_Item_Property.
SBP	Set_Block_Property(parametro1, parametro2, parametro3).Ver en la ayuda de Forms, los valores que recibe el built-in Set_Block_Property.
SFP	Set_Form_Property(parametro1, parametro2, parametro3).Ver en la ayuda de Forms, los valores que recibe el built-in Set_Form_Property.
SIIP	Set_Item_Instance_Property(parametro1, parametro2, parametro3).Ver en la ayuda de Forms, los valores que recibe el built-in Set_Item_Instance_Property.
SRBP	Set_Radio_Button_Property(parametro1(*), parametro1(*), parametro2, parametro3). (*) En el parámetro 1 tiene que ponerse de la forma BLOQUE.CAMPO.ELEMENTO, BLOQUE.CAMPO se usará en el primer parámetro de Set_Radio_Button_Property y ELEMENTO en el segundo. Ver en la ayuda de forms los valores que recibe el built-in Set_Radio_Button_Property.
STPP	Set_Tab_Page_Property(parametro1, parametro2, parametro3).Ver en la ayuda de Forms, los valores que recibe el built-in Set_Tab_Page_Property.
SWP	Set_Window_Property(parametro1, parametro2, parametro3).Ver en la ayuda de Forms, los valores que recibe el built-in Set_Window_Property.
SMIP	Set_Menu_Item_Property(parametro1, parametro2, parametro3).Ver en la ayuda de Forms, los valores que recibe el built-in Set_Menu_Item_Property.
SCP	Set_Canvas_Property(parametro1, parametro2, parametro3).Ver en la ayuda de Forms, los valores que recibe el built-in Set_Canvas_Property.
SVAP	Set_Va_Property(parametro1, parametro2, parametro3). Ver en la ayuda de Forms, los valores que recibe el built-in Set_Va_Property.
FITEM	Parámetro1 puede ser uno de los siguientes: VISIBLE, ACTIVADO, MODIFICABLE, OBLIGATORIO. Parámetro2 el campo que se quiere modificar. Parámetro3: S o N.
POSTEAR	POSTEAR;
CENTRA_VENTANA	CENTRA_VENTANA(parametro1);
PKLIBPNT_SIP	DISPSTD.SET_PROPIEDAD(parametro1, parametro2, parametro3). Ver apartado Modificar por código las propiedades cargadas del mantenimiento de programas.
PKLIBPNT_SBP	DISPSTD.SET_PROPIEDAD(parametro1, parametro2, parametro3, 'B'). Ver apartado Modificar por código las propiedades cargadas del mantenimiento de programas.
PKLIBPNT_SPP	DISPSTD.SET_PROPIEDAD(parametro1, parametro2, parametro3, 'P'). Ver apartado Modificar por código las propiedades cargadas del mantenimiento de programas.
PKLIBPNT_SPI	DISPSTD.SET_PROPIEDAD(parametro1, parametro2, parametro3, 'PI'). Ver apartado Modificar por código las propiedades cargadas del mantenimiento de programas.
PKLIBPNT_IMP	IMP.SET_PROPIEDAD(parametro1, parametro2). Ver apartado Modificar por código las propiedades cargadas del mantenimiento de programas.
PKLIBPNT_LV	LV.SET_PROPIEDAD(parametro1, parametro2). Ver apartado Modificar por código las propiedades cargadas del mantenimiento de programas.
KEY_F	Ejecuta el plug-in que tenga asignada la tecla rápida indicada en parametro1
EJECUTA_PLUG_IN	Ejecuta el plug-in con código indicado en parámetro2 en el bloque indicado en parametro1. NOTA: Se ejecuta independiente de si el usuario tiene permisos o no sobre el PLUG-IN. Si se utiliza esta opción quiere decir que se ya se ha validado que el usuario puede realizar la ejecución.
SYNCHRONIZE	SYNCHRONIZE
GET_XML	Genera en el ordenador cliente un archivo XML con ruta y nombre indicada en parametro1 (se pueden utilizar los mismos modificadores de nombre de archivo que GET_FILE_TXT) con el XML inicializado anteriormente con pk_xml.init_linea_xml o con pk_xml.init_linea_xml_sql.
GET_XML_IAS	Genera en el servidor de aplicaciones un archivo xml con ruta y nombre indicada en parametro1 con el xml inicializado anteriormente con pk_xml.init_linea_xml o con pk_xml.init_linea_xml_sql.

GET_FILE_TXT	Genera en el ordenador cliente los archivos de texto con sus líneas correspondiente que se han ido almacenando mediante PKPANTALLAS.ADD_LINEAS_FICHERO. Ver apartado: Generar archivos de texto en ordenador cliente desde código PL/SQL.
GET_FILE_TXT_IAS	Genera en el servidor de aplicaciones los archivos de texto con sus líneas correspondiente que se han ido almacenando mediante PKPANTALLAS.ADD_LINEAS_FICHERO. Ver apartado: Generar archivos de texto en ordenador cliente desde código PL/SQL.
ACTIVA_PLUG_IN	Se usa para activar o desactivar un plug-in de un bloque, en parametro1 se indicará BLOQUE.CODIGO_PLUGIN y en parámetro2 se indicará S para activarlo y N para desactivarlo.
DELETE_RECORD	DELETE_RECORD;
CLEAR_RECORD	CLEAR_RECORD;
CLEAR_BLOCK	CLEAR_BLOCK(parametro1);
CLEAR_LIST	CLEAR_LIST(parametro1);. Permite borrar el contenido de un LIST-ITEM. Parametro1 identifica el campo a borrar con BLOQUE.CAMPO
DLE	Delete_List_Element(parametro1, parametro2);. Permite borrar únicamente un elemento del list-item. Parametro1 identifica el campo a borrar con BLOQUE.CAMPO y parametro2 es el número de elemento a borrar. Si se borran más de uno hay que tener en cuenta que por cada borrado los elementos se renumeran.
ALE	Add_List_Element(parametro1, NVL(TO_NUMBER(Get_List_ElementCount(parametro1)), 0) + 1, parametro2, parametro3);. Añade al final de la lista un nuevo elemento. Parametro1 identifica el con BLOQUE.CAMPO y parametro2 es el texto que se le mostrará al usuario cuando seleccione la opción y parametro3 es el valor interno que contendrá el campo cuando el usuario seleccione la opción.
PL	POPULATE_LIST. Parametro1 identifica el campo a borrar con BLOQUE.CAMPO y parametro2 es la sql que debe de usarse para rellenar la lista de valores. La sql debe de sacar únicamente 2 campos, el primero será el texto que verá el usuario al seleccionar el elemento y el segundo campo el código interno que tendrá el campo cuando el usuario seleccione el elemento.
PLSQL	Ejecuta el código pl/sql pasado en Parametro 1.
SCUSP	Set_Custom_Property(parametro1, 1, parametro2, parametro3)
WWW	Abre la web pasada por parámetro en v_parametro1 en el navegador
TXT2VOZ	Exclusivo menú nuevo, es decir no disponible en menú legacy: Usa el sintetizador de voz del navegador para reproducir el texto que se pasa en parametro1.
START	Visualiza el archivo indicado en parámetro1 y que debe de encontrarse en el ordenador que ejecuta Libra
SHOW_VIEW	SHOW_VIEW(parametro1)

DESCARGA_ARCHIVO	Permite especificar tanto el **nombre del archivo** como el **filtro de tipo de archivo** que se mostrará en el selector, mediante sus dos primeros parámetros. Su principal utilidad es habilitar la descarga de archivos generados desde código PL/SQL, especialmente en programas de mantenimiento donde no es posible incluir esta funcionalidad directamente en el FMB.
	
	Ejemplo:
	
	-- Generar y cargar el archivo SQL en PK_BLOB2BD
	pk_gal_inarchstrd_bbdd.generar_sql(:b1.codigo, FALSE);
	
	-- Inicializar y ejecutar la descarga del archivo al equipo del usuario
	pkpantallas.inicializar_codigo_plug_in;
	pkpantallas.comando_plug_in('DESCARGA_ARCHIVO', :b1.codigo || '.sql', 'SQL (*.sql)|*.sql|');

Ejemplo de activación/desactivación de plug-in desde PL/SQL

```
PKPANTALLAS.INICIALIZAR_CODIGO_PLUG_IN;
IF :b1.cli_codigo = '0005' THEN
  PKPANTALLAS.COMANDO_PLUG_IN('ACTIVA_PLUG_IN', 'BCLIENTE.CL', 'N');
ELSE
  PKPANTALLAS.COMANDO_PLUG_IN('ACTIVA_PLUG_IN', 'BCLIENTE.CL', 'S');
END IF;
```

Generar archivos de texto en ordenador cliente o servidor de aplicaciones desde PL/SQL

Para generar un archivo de texto desde código PL/SQL primero hay que almacenar los datos de cada archivo que se quiere generar en la base de datos mediante las siguientes funciones del paquete pkpantallas:

- **pkpantallas.incializa_lineas_fichero**: Se ejecuta una sola vez e inicializa las estructuras internas del paquete pkpantallas para almacenar los datos para generar los ficheros.
- **pkpantallas.add_lineas_fichero(<tipo>, <archivo_o_linea>)**: Se usa tanto para indicar el nombre del archivo y la ruta como las líneas de texto que va a contener. Recibe dos parámetros, en el primer parámetro **<tipo>** indica si en el segundo parámetro **<archivo_o_linea>** se está pasando el nombre del archivo o de la línea de texto que va a contener el archivo. Obligatoriamente la primera vez que llama se tiene que pasar un nombre del archivo y las siguientes líneas que se añadirán a ese archivo, una vez se cambie el archivo se cierra el anterior y las nuevas líneas se añaden al nuevo.
  Al indicar el nombre de archivo se le pueden añadir modificadores (exclusivo versión Forms 12c) concatenándolos al nombre del archivo con el separador ":MOD:". Puede haber más de un modificador, en ese caso se separan por comas. (ver ejemplo). Los modificadores disponibles son:
  - **GET_FILE_NAME**: Se abre el selector de archivo para indicar en dónde y con que nombre se grabará el archivo. En el selector se propone el nombre de archivo indicado. Por defecto se considera que el archivo tiene extensión .txt, si fuese otra extensión se puede indicar la cadena de extensiones del selector de archivos añadiendo : y la cadena, por ejemplo, para archivos .log sería: LOG (*.log)|*.log|All Files (*.*)|*.*|
  - **DOS**: Fuerza a que los saltos de línea sean para equipos Windows.
  - **CODIFICACION:WE8ISO8859P1**: Codifica en formato ANSI el archivo de texto en vez de UTF-8.

Para generar finalmente el archivo hay que llamar a GET_FILE_TXT. Ver apartado: Ejecutar operaciones de Forms. **Ejemplo**:

```
pkpantallas.inicializa_lineas_fichero;
pkpantallas.add_lineas_fichero('F',     'fichero1.log:MOD:GET_FILE_NAME:LOG     (*.log)|*.log|All     Files
(*.*)|*.*|,DOS,CODIFICACION:WE8ISO8859P1');
pkpantallas.add_lineas_fichero('L', 'LINEA1');
pkpantallas.add_lineas_fichero('L', 'LINEA2');
pkpantallas.add_lineas_fichero('F', 'c:\temp\fichero2.txt');
pkpantallas.add_lineas_fichero('L', 'LINEA1');
PKPANTALLAS.INICIALIZAR_CODIGO_PLUG_IN;
PKPANTALLAS.COMANDO_PLUG_IN('GET_FILE_TXT');
```

Si el archivo se quiere almacenar en el servidor de aplicaciones en vez del equipo del usuario que ejecuta Libra, en vez de usar el comando GET_FILE_TXT se usará el comando GET_FILE_TXT_IAS, en este caso los modificadores del nombre de archivo serán ignorados y en el nombre de archivo debe de indicarse la ruta completa en el servidor de aplicaciones.

Generar archivos XML en ordenador cliente o servidor de aplicaciones desde código pl/sql.

Para generar un archivo de texto desde código PL/SQL primero hay que inicializar en el paquete PK_XML el archivo XML a descargar pasando una variable de tipo XMLTYPE a PK_XML.INIT_LINEA_XML o pasando una consulta que devuelva un XMLTYPE en PK_XML.INIT_LINEA_XML_SQL.

Para ejecutar la descarga se realizará con el comando plug-in GET_XML (Ver apartado: Ejecutar operaciones de Forms), en el primer parámetro del comando se indicará el nombre de archivo que puede tener los mismos indicadores que los indicados para GET_FILE_TXT.

Ejemplo:

```
pk_xml.init_linea_xml_sql('SELECT          XMLELEMENT("elementoraiz",          XMLAGG(XMLELEMENT("cliente",
XMLELEMENT("codigo", codigo_rapido), XMLELEMENT("nombre", nombre), XMLELEMENT("direccion", direccion)))) FROM
clientes WHERE ROWNUM <= 3');
PKPANTALLAS.INICIALIZAR_CODIGO_PLUG_IN;
PKPANTALLAS.COMANDO_PLUG_IN('GET_XML', '3clientes.xml:GET_FILE_NAME');
```

Si el archivo se quiere almacenar en el servidor de aplicaciones en vez del equipo del usuario que ejecuta Libra, en vez de usar el comando GET_XML se usará el comando GET_XML_IAS, en este caso los modificadores del nombre de archivo serán ignorados y en el nombre de archivo debe de indicarse la ruta completa en el servidor de aplicaciones.

Leer propiedades de objetos del programa desde código PL/SQL.

Desde el código PL/SQL se pueden leer las propiedades de Item, Block, Window, Form, Canvas, Tab, Menú, Item Instance. Para ello se usará lo siguiente:

:XXX:<objeto>:<propiedad>

- **XXX**: Indica el tipo de objeto del que se quiere obtener la propiedad:
  - **GBP**: Bloque (Get_Block_Property).
  - **GIP**: Item (Get_Item_Property).
  - **GWP**: Window (Get_Window_Property).
  - **GFP**: Form (Get_Form_Property).
  - **GCP**: Canvas (Get_Canvas_Property).
  - **GTP**: Tab (Get_Tab_Page_Property).
  - **GMP**: Menú (Get_Menu_Item_Property).
  - **GII**: Item Instance (Get_Item_Instance_Property).
  - **GCU**: Get_Custom_Property.
  - **GLL**: Get_List_Element_Label. Si el <propiedad> es '0' o no se indica, devolverá el texto del elemento actualmente seleccionado.

Para saber las propiedades disponibles en cada uno de los objetos es recomendable consultar la ayuda de Forms Builder para más información.

NOTA: Los valores de las propiedades son las que tenía en objeto justo antes de ejecutarse el código PL/SQL.

Ejemplo: Se lee la propiedad DEFAULT_WHERE del bloque CAMPOS y luego se le asigna al mismo bloque, pero añadiendo la condición AND estado = 'ESPA'

```
PKPANTALLAS.INICIALIZAR_CODIGO_PLUG_IN;
PKPANTALLAS.COMANDO_PLUG_IN('SBP', 'CAMPOS', 'DEFAULT_WHERE', :GBP:CAMPOS:DEFAULT_WHERE || ' AND estado = ''ESPA'');
```

Gestionar los registros seleccionados por el usuario.

A nivel de bloque se puede activar la check “Habilitar selección de registros”, si está activada esa check el usuario puede seleccionar varios registros de forma simultánea en un bloque. Desde los códigos PL/SQL se puede acceder a los registros seleccionados por el usuario mediante las siguientes funciones:

- **PKPANTALLAS.GET_REGISTROS_SELECCIONADOS**: Devuelve un array de tipo PKPANTALLAS.TABLA_REGISTROS_BLOQUE, en este array habrá una entrada por cada registro que seleccionó el usuario.
- **PKPANTALLAS.GET_VALOR_CAMPO_SELEC_VARCHAR2**: Devuelve el valor de un campo VARCHAR2 de uno de los registros seleccionados, para ello recibe los siguientes parámetros:
  - **p_id**: Número de registro dentro del array devuelto por PKPANTALLAS.GET_REGISTROS_SELECCIONADOS.
  - **p_campo**: Nombre del campo del que se quiere recoger el valor. IMPORTANTE: Únicamente se guarda el contenido de aquellos campos que están en el grid, si hay campos que tienen el nº de registros visualizados a 1 no se guardará su valor.
- **PKPANTALLAS.GET_VALOR_CAMPO_SELEC_NUMBER**: Igual que PKPANTALLAS.GET_VALOR_CAMPO_SELEC_VARCHAR2, pero para campos numéricos.
- **PKPANTALLAS.GET_VALOR_CAMPO_SELEC_DATE**: Igual que PKPANTALLAS.GET_VALOR_CAMPO_SELEC_VARCHAR2, pero para campos de tipo fecha.

- **PKPANTALLAS.GET_VALORES_CAMPO_SELEC_NUMBER**: Devuelve el valor de un determinado campo numérico en un array de tipo PKPANTALLAS.NUMBER_TABLE para todos los registros. Es útil para cuando sólo interesa el valor de un campo de todos los registros.
- **PKPANTALLAS.GET_VALORES_CAMPO_SELEC_VCH2**: Devuelve el valor de un determinado campo en un array de tipo PKPANTALLAS.VARCHAR2_TABLE para todos los registros. Es útil para cuando sólo interesa el valor de un campo de todos los registros.

Ejemplo:

```
DECLARE
  v_registros  PKPANTALLAS.TABLA_REGISTROS_BLOQUE;
  v_id         PLS_INTEGER;
BEGIN
  v_registros := PKPANTALLAS.GET_REGISTROS_SELECCIONADOS();

  IF v_registros.COUNT = 0 THEN
    PKPANTALLAS.LOG('NO HAY REGISTROS SELECCIONADOS');
  ELSE
    v_id := v_registros.FIRST;

    WHILE v_id IS NOT NULL LOOP
      PKPANTALLAS.LOG('CAMPO SELECCIONADO: ' ||
              PKPANTALLAS.GET_VALOR_CAMPO_SELEC_VARCHAR2(v_id, 'CAMPO') || ', LV: ' ||
                   PKPANTALLAS.GET_VALOR_CAMPO_SELEC_VARCHAR2(v_id, 'LV_CODIGO_LISTA'));
      v_id := v_registros.NEXT(v_id);
    END LOOP;
  END IF;
END;
```

Búsqueda contextual

En el programa de personalizar estética por usuario (u_mconfig) y empresa (u_mconem) hay dos checks:

- Lista de valores automática donde sea posible: Si se activa esta check cuando el usuario pulsa ENTER en un campo obligatorio que está vacío y que tiene lista de valores en el mantenimiento de programas la lanza de forma automática.
- Lista de valores contextual donde sea posible: Si activamos esta check y el campo tiene lista de valores y validar desde lista de valores en el mantenimiento de programas se intentará validar (solo cuando el usuario sale del campo pulsando ENTER, si lo hace con el ratón se sigue haciendo como hasta ahora) el campo de la siguiente forma:
    - Primero haciendo una búsqueda exacta por código, es decir, si es una organización comercial, buscando aquella que el código coincida exactamente con lo que ha tecleado el usuario.
    - Búsqueda con LIKE por código sustituyendo espacios por % (también se pueden poner %) y añadiendo siempre un % al final, en el ejemplo de la organización comercial, si el usuario teclea 1 y no hay ninguna organización comercial con código 1 (ya se validaría por el punto anterior) buscaría con LIKE codigo '1%', si solo hay una que cumpla la condición ya la validaría y si hay varias lanzará la lista de valores con los coincidentes.
    - Búsqueda con LIKE por descripción o por los campos de la lista de valores que tengan marcada la check "Búsqueda Contextual". Si hay un único registro que cumpla la condición ya lo valida y si hay varios que la cumplen lanza la lista de valores con los coincidentes.

Para que esto funcione bien cuando se personalice el disparador KEY-NEXT-ITEM en vez de usar VALIDATE(ITEM_SCOPE) usaremos DISPSTD.VALIDATE_ITEM, por ejemplo, si queremos que al validar un campo con el teclado el cursor nos salte a otro bloque haremos lo siguiente:

```
DISPSTD.VALIDATE_ITEM;
GO_BLOCK('BLOQUE');
```

Habilitar y Deshabilitar opciones de menú (Paquete FMENU)

Mediante el código genérico del formulario base ya se realizan las siguientes operaciones con el menú:

- Si el programa tiene bloque BREPORT, es decir que tiene listado se activa el botón de imprimir, en caso contrario se deshabilita.
- Si nos situamos un registro cuyo bloque que tenga la propiedad Borrado permitido a NO, deshabilita el botón de borrar registro, en caso contrario lo habilita.
- Si nos situamos en un bloque que tenga en la propiedad Inserción Permitida a NO se deshabilitan los botones de crear registro y duplicar registro.
- Si nos situamos en un bloque que tenga en la propiedad Consulta Permitida a NO se deshabilitan los botones de entrada consulta y ejecución consulta, en caso contrario se habilitan.
- Si nos situamos en un bloque que tenga bloques detalle se habilita el botón de bloque siguiente.
- Si nos situamos en un bloque que tenga un bloque padre se habilita el botón de bloque anterior.
- Si el campo tiene asociado programa para llamada directa se habilita el botón de llamada directa, en caso contrario se deshabilita.
- Si el campo tiene lista de valores, calendario o calculadora se habilita el botón de lista de valores, en caso contrario se deshabilita.
- Si nos situamos en un bloque que tenga asociada la clase BLOQUE_BLOQUE_REG_UNICO y por tanto tenga como atributo visual de registro actual el BLOQUE_REGISTRO_UNICO se deshabilitan los botones de primer y último registro y de siguiente y anterior registro. También lleva asociado que se deshabilitan los disparadores KEY_UP y KEY_DOWN.

Con estos casos se cubre la práctica totalidad de los casos en donde hay que habilitar y deshabilitar opciones de menú y botones de la botonera, pero en caso de ser necesario actuar sobre alguno de ellos para habilitar o deshabilitar se usará el paquete FMENU de la librería PKLIBPNT.

NOTA: Para cambiar la propiedad de borrado permitido, modificación permitida según una condición, es mejor usar el disparador PRE-RECORD en vez del WHEN-NEW-RECORD-INSTANCE ya que si el usuario navega a otro registro con el ratón y pulsa en un campo de tipo check o botón de radio cambia su valor antes de saltar el disparador WHEN-NEW-RECORD-INSTANCE.

Toda activación y desactivación de menús se debe de realizar mediante el paquete FMENU que tiene los siguientes procedimientos:

- FMENU.IMPRIME(<parámetro>): Botón de imprimir.
- FMENU.CONSULTAR(<parámetro>): Botón de consultar (monitor).
- FMENU.GRAFICO(<parámetro>): Botón de gráficos.
- FMENU.LOV(<parámetro>): Botón de listas de valores.
- FMENU.BLOQUE_SIGUIENTE(<parámetro>): Botón de bloque siguiente.
- FMENU.BLOQUE_ANTERIOR(<parámetro>): Botón de bloque anterior.
- FMENU.GRABAR(<parámetro>): Botón de grabar.
- FMENU.DUPLICAR(<parámetro>): Botón de duplicar registro.
- FMENU.EDITAR(<parámetro>): Botón de editar campo.
- FMENU.BORRAR(<parámetro>): Botón de borrar registro.
- FMENU.PRIMER_REGISTRO(<parámetro>): Botón para navegar al primer registro del bloque.
- FMENU.ULTIMO_REGISTRO(<parámetro>): Botón para navegar al último registro del bloque.
- FMENU.SIGUIENTE_REGISTRO(<parámetro>): Botón para avanzar al siguiente registro.
- FMENU.ANTERIOR_REGISTRO(<parámetro>): Botón para retroceder al registro anterior.
- FMENU.LLAMADA(<parámetro>): Botón de hipervínculo a otro programa.
- FMENU.ENTER_QUERY(<parámetro>): Botón de entrada de consulta.
- FMENU.EXECUTE_QUERY(<parámetro>): Botón de ejecutar consulta.
- FMENU.SALIR(<parámetro>): Botón de salir del programa.
- FMENU.CREAR_REGISTRO(<parámetro>): Botón para crear registro nuevo.
- FMENU.EXCEL(<parámetro>): Botón de envío de contenido del bloque a Excel.

<parámetro>: Espera un valor booleano, es decir TRUE activa y FALSE desactiva.

Notas Importantes sobre el Menú y la Botonera

Salvo raras excepciones el menú que tiene el programa se debe dejar con el menu6 estándar.

Para los botones particulares de cada programa como por ejemplo lanzar proceso se ha habilitado una barra de botones vertical con 26 botones y 14 en la horizontal, que se pueden habilitar y usar con los siguientes procedimientos:

Procedimiento para hacer visible y activar a la vez el botón:

- **FMENU.BOTON_VERTICAL_VISIBLE**(<boton>,<etiqueta>,<icono>): Hace visible un botón. NOTA: debe de utilizarse siempre en el disparador INICIO
  - **<boton>**: Los botones van desde el B01 a B26 (para la botonera vertical) y del H01 al H14 (para la botonera horizontal).
  - **<etiqueta>**: Etiqueta que aparecerá cuando el usuario pasa el ratón por encima del botón.
  - **<icono>**: El icono será un fichero .ico que estará en el path de UI_ICON.
- **FMENU.BOTON_VERTICAL_ACTIVAR**(<boton>,<como>): Procedimiento para desactivar o activar el botón después de que ya sea visible:
  - **<como>**: Es un valor booleano, TRUE activa y FALSE desactiva.

Para gestionar el pulsado de cada uno de los botones se usará el disparador OPCION_MENU, se identificará el botón pulsado por el valor del parámetro OPCION_MENU.

```
OPCMENU.OPCION_MENU(:global.codigo_empresa, :parameter.opcion_menu);
IF :parameter.opcion_menu = 'B01' THEN
        --CODIGO PARTICULAR PARA ESE BOTON
END IF;
```

IMPORTANTE: Al nombre del fichero no se le pasará la extensión, ya que la asume automáticamente.

MUY IMPORTANTE:

- Los botones se deben declarar de forma correlativa comenzando en el B01 y sin dejar huecos, es decir, no se puede usar el B05 sin previamente haber hecho visibles con FMENU.BOTON_VERTICAL_VISIBLE los B01, B02, B03 y B04. Si no se cumple esto al añadir plug-ins al bloque se producirán funcionamientos totalmente inesperados.
- Solo se pueden usar los botones verticales B01 .. B26 dentro del fuente única y exclusivamente si han sido creados dentro del fuente, es decir, si no se ejecuta en el disparador INICIO un FMENU.BOTON_VERTICAL_VISIBLE('B01', ... dentro del fuente nunca se puede hacer referencia a 'B01'.

En el caso de que si un botón se inicializa con una determinada condición, por ejemplo el sector de la empresa, el código que ejecute en OPCION_MENU y en las llamadas a FMENU.BOTON_VERTICAL_ACTIVAR tiene que ir también la misma condición.

Por ejemplo si se inicializa un botón con esta condición:

```
IF pkpantallas.sector_empresa(:global.codigo_empresa) = 'XX' THEN
  FMENU.BOTON_VERTICAL_VISIBLE('B05', 'Borrar Todo', 'produccion');
END IF;
```

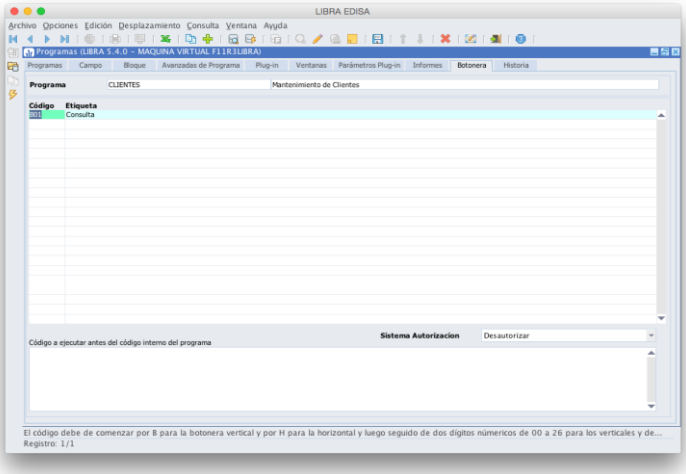
El código que va en el disparador OPCION_MENU tiene que ir dentro de esa misma condición, es decir:

```
IF :parameter.opcion_menu = 'B05' AND pkpantallas.sector_empresa(:global.codigo_empresa) = 'XX' THEN
-- codigo
END IF;
```

También en el caso de activar o desactivar por código el botón hay que meter esa condición, por ejemplo:

```
IF pkpantallas.sector_empresa(:global.codigo_empresa) = 'XX' THEN
  IF :b1.condicion = 'S' THEN
    FMENU.BOTON_VERTICAL_ACTIVAR('B05', TRUE);
  ELSE
    FMENU.BOTON_VERTICAL_ACTIVAR('B05', FALSE);
  END IF;
END IF;
```

Una buena práctica es añadir los botones definidos en el fuente con FMENU.BOTON_VERTICAL_VISIBLE en la pestaña “Botonera” del mantenimiento de programas.



- **Etiqueta**: Etiqueta que se mostrará cuando el usuario pase el ratón por encima del botón.
- **Código a ejecutar antes del código interno del programa**: Código que se ejecutará antes de que el programa ejecute el código asociado a ese botón en el disparador OPCION_MENU. Si este código ejecuta :p_parar_ejecucion := ‘S’; no se llegara a ejecutar el código que se encuentra en el código fuente del programa.

De esta forma es posible personalizar el programa para autorizar o desautorizar por perfiles el uso de esos botones, aparte de cambiar la etiqueta estándar o añadir un código pl/sql a ejecutar antes del código del programa.

Generar Logs de traza.

Si se quiere que un programa vaya generando una traza para su posterior comprobación se usará el paquete PANTLOG contenido en la librería PKLIBPNT. La traza consistirá en líneas de texto que se grabarán en un fichero plano.

NOTA: Antes era necesario utilizar PANTLOG.INICIALIZAR en el disparador INICIO. Ahora este paso ya no es necesario, el log se inicializa de forma automática.

IMPORTANTE: La variable DIRECTORIO_LOG se define en el archivo libra.env (o el indicado en formsweb.cfg) y el archivo de traza será generado en el servidor de aplicaciones. Si el usuario tiene autorizado (configuración por usuario / empresa) la visualización de registro de Log en “Acerca de…” podrá abrir directamente la traza haciendo doble click sobre el campo “Directorio LOG” del “Acerca de…”.

Para grabar una línea al final del archivo de log se usará la instrucción **PANTLOG.ESCRIBE**(<linea>). En el parámetro línea será el texto que se quiere que se grabe.

En PANTLOG.ESCRIBE se puede indicar un segundo parámetro “PANTLOG.ESCRIBE(<linea>, <nivel>)” para indicar en qué nivel de LOG debe ser escrita la línea en la traza. Los niveles de log posibles son (en negrita el código de <nivel>):

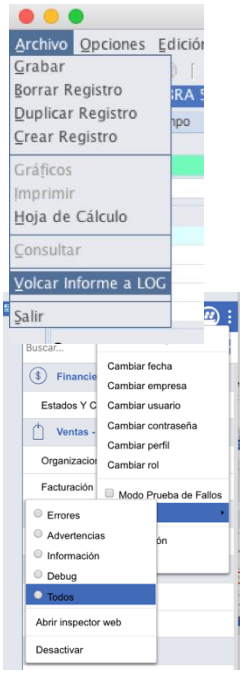
- **Errores**: Funcionamiento normal, sin indicar el parámetro de nivel de log. Siempre se escribe en el LOG.
- **Advertencias**: Registra los niveles anteriores + los marcados como **'WARN'**: Está pensado para registrar situaciones que puedan registrar algún tipo de problema en la ejecución del programa, pero que no impiden que el programa se ejecute
- **Información**: Niveles anteriores + **'INFO'**: Está pensado para registrar valores de parametrización.

- **Depuración**: Niveles anteriores + **'DEBUG'**: Está pensado para registrar valores de variable, funciones que se ejecutan, …, es decir, el nivel máximo de detalle. Este nivel no incluye ningún evento generado por el entorno.
- **Todos**: Niveles anteriores + eventos del entorno. No se puede usar en ningún programa, ya que está reservado de forma exclusiva para programas del entorno. Al activar este nivel de traza se registran multitud de eventos del entorno, WHEN-VALIDATE-ITEM, WHEN-NEW-ITEM-INSTANCE, ...

Para grabar un volcado de información del programa se llamará a PANTLOG.GRABA_DUMP('<bloque>', 'VALORES');

- **<bloque>**: Código del bloque del programa del que se quiere volcar información al archivo de log. Si se pasa en blanco, es decir, NULL, se generará de todos los bloques del programa.
- **'VALORES'**: Valor fijo.

Por seguridad, al hacer un volcado, sólo se mostrará el valor de los campos que el usuario esté visualizando en pantalla. Como excepción es que el usuario sea superusuario, o Libra se esté ejecutando en modo a prueba de fallos (modo que sólo un superusuario puede hacerlo).



Durante la ejecución del programa, se puede volcar información de los bloques al archivo de traza, se puede hacer con la opción de menú (Archivo -> Volcar informe a LOG).

El fichero solo se generará si está activada la traza, es decir, la variable global :global.traza contiene el valor SI. Desde el menú de Libra la puede activar el superusuario en Especial -> Activa Traza y se desactiva en Especial -> Desactiva Traza. Una vez se activa la traza, se puede indicar el nivel de log, desde sólo errores a Todos.

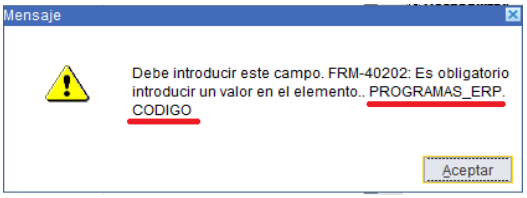
En el archivo de Log también se guardan los Logs de LIBRA_LOG registrados en base de datos.

El archivo de Log se puede abrir de forma rápida desde el "Acerca de…" de los programas haciendo doble click sobre el campo "Directorio Log".

Si se activa la traza a todos los mensajes que muestre Libra, se les añadirá el campo según el siguiente criterio (únicamente se concatenará uno, el primero de los 3 que tenga valor):

- Campo que produce el error.
- Campo que está lanzando el evento.
- Campo en que se encuentra el cursor.

Ejemplo:



Logs de incidencias ocurridas en la base de datos

En el paquete PKPANTALLAS hay un procedimiento llamado LOG destinado a registrar incidencias ocurridas en la base de datos. Se ejecuta mediante una transacción autónoma, por tanto, incluso si la transacción que lo llama hace un ROLLBACK queda registrada la incidencia. El registro queda en una tabla llamada LIBRA_LOG. Este procedimiento recibe 3 parámetros:

- Texto a registrar.
- Paquete desde el que lo estamos llamando.
- Punto dentro del paquete desde donde lo estamos llamando.

Esto es útil en paquetes de base de datos en donde capturamos el WHEN OTHERS (o se hace un RAISE Salir) y devolvemos un error genérico, normalmente cuando pasa esto encontrar el problema es cosa de locos, cuando la solución (por ejemplo) puede ser tan sencilla como ampliar un TABLESPACE.

Ejemplo: Si se llama desde el paquete SV en un WHEN OTHERS la forma de llamarlo sería:

- En el primer WHEN OTHERS: PKPANTALLAS.LOG (SQLERRM, 'SV', '1');
- En el segundo WHEN OTHERS: PKPANTALLAS.LOG (SQLERRM, 'SV', '2');

IMPORTANTE: Solo se debe usar donde realmente la incidencia es importante, no se debe usar para ir dejando trazas de por donde pasa un programa ya que sobrecarga bastante.

Para guardar trazas se puede usar PKPANTALLAS.TRAZA con los mismos parámetros. Se debe de usar únicamente dentro de procedimientos almacenados en la base de datos y se guarda también en la tabla LIBRA_LOG, pero únicamente cuando en el menú de Libra está activada la traza.

Disparadores estándar

Los disparadores estándar son aquellos que ejecutan el mismo código en todos los programas, este código está en la librería PKLIBPNT en el paquete DISPSTD.

Es recomendable que todos los disparadores ejecuten ese código, ya que si se hace una modificación en la PKLIBPNT para añadir una nueva funcionalidad esta ya estará incluida en todos los programas que la usen solo recompilando la librería.

Por ejemplo, si tenemos en un bloque el disparador WHEN-NEW-BLOCK-INSTANCE, a medida, es recomendable poner en el disparador, aparte del código personalizado, una llamada al código estándar DISPSTD.WHEN_NEW_BLOCK_INSTANCE.

Los disparadores estándar tienen el mismo nombre que los disparadores de Forms, pero con guiones bajos en vez de guiones altos.

Particularidades

El código estándar de DISPSTD.NEW_RECORD_INSTANCE, DISPSTD.WHEN_NEW_BLOCK_INSTANCE y DISPSTD.KEY_DELREC llevan implícito una llamada al procedimiento POSTEAR, entonces si algún dato ha sido modificado (variable :system.form_status = 'CHANGED') se ejecutará un POST forzando la validación de todos los campos y registros pendientes de validar.

Hay casos en el que no nos interesa que estos disparadores hagan un POST, pero si que ejecuten el resto del código estándar, para ello podemos desactivar el POST del disparador poniendo antes de la llamada al DISPSTD una llamada al procedimiento DISPSTD.DESACTIVAR_POSTEADO. El desactivado sólo tendrá efecto en el próximo disparador DISPSTD.WHEN_NEW_RECORD, DISPSTD.WHEN_BLOCK_INSTANCE, DISPSTD.KEY_DELREC.

Personalización Borrado y grabación.

Hay casos en los que hay que anular totalmente el disparador estándar, ya que la acción que hacen dependerá de una pregunta al usuario, en estos casos es recomendable ver el código que ejecuta el disparador estándar para no perder ninguna funcionalidad. Por ejemplo, si personalizamos el disparador KEY-EXIT hay que poner en el código para cancelar la lista de valores, calculadora, ...

Para minimizar el caso en los que hay que anular el disparador estándar, existen dos disparadores a nivel de formulario, para meter el código personalizado y no tener que cambiar nada de los disparadores estándar.

- **ANTES_BORRAR**: Se ejecuta antes de que el disparador estándar borre el registro y después de que el usuario confirmase la pregunta de borrado.
- **ANTES_GRABAR**: Se ejecuta antes de que el disparador estándar ejecute el COMIT_FORM y después de que el usuario que quiere grabar las modificaciones, tanto al pulsar el botón de grabar como al pulsar el botón de salir y había modificaciones pendientes de grabar.

Impresión.

Impresión por FAX.

Si en un programa se quiere dar la opción de que el usuario seleccione impresión por fax se ejecutará DISPSTD.ACTIVA_IMPRESION_FAX en el disparador INICIO. De esta forma en la lista de dispositivos de salida se añade la opción Fax a las opciones Pantalla, Impresora, ...

La llamada a esa función únicamente añade la opción en la lista de dispositivos de salida, la lógica del envío a fax tiene que estar en el programa en el botón BREPORT.IMPRIMIR.

Impresión multidestino.

En los programas siempre se va a lanzar en primer lugar la impresión que selecciona el usuario, pero por personalización en código pl/sql de antes de imprimir o dentro del fuente se puede forzar a que la impresión se haga en otros destinos de forma simultánea.

Para habilitar el multidestino hay que ejecutar una única vez el procedimiento pkpantallas.inicializa_multidestino_report y luego por cada destino al que se quiera enviar el report hay que ejecutar: **pkpantallas.add_multidestino_report(p_informe, p_dispositivo, p_tipo_fichero, p_imp_asincrona, p_imprimir_por, p_impresora, p_destino, p_email)**:

- p_informe: Se puede indicar otro informe a imprimir, si se pasa a NULL se imprimirá el informe seleccionado por el usuario.
- p_dispositivo: Puede recibir los siguientes valores: SCREEN, PRINTER, FILE, MAIL, FAX, GESTDOC.
- p_tipo_fichero: Se usa para cuando el dispositivo genera un archivo (FILE, MAIL, FAX y GESTDOC) e indica el tipo de archivo a generar, recibe los siguientes valores: PDF, XLS, HTML, RTF, XML
- p_imp_asincrona: Indica al servidor si la impresión se ha de ejecutar en modo síncrono (S) o asíncrono (N).
- p_imprimir_por: Se usa únicamente para indicar si se imprime por la impresora de WINDOWS, en ese caso hay que pasar el valor WINDOWS.
- p_impresora: En caso de que el destino sea PRINTER en este parámetro se pasa el código de la impresora lógica por la que se ha de imprimir. Si p_imprimir_por recibe el valor WINDOWS en este parámetro se pasará WINDOWS_V para indicar que imprima por la impresora asociada de Windows en vertical Y WINDOWS_H para horizontal.
- p_destino: Se usa para indicar la ruta y nombre de archivo a generar cuando el dispositivo está asociado a archivo (FILE, MAIL, FAX, GESTDOC).
- p_email: Dirección de correo electrónico que se usarán cuando el dispositivo sea MAIL.

Crear un formulario desde cero

Nos basaremos inicialmente en el formulario base "manbase6.fmb", este formulario ya incorpora el componente FORMULARIO_BASE de la librería de objetos objetospant.olb.

Antes de nada, para no sobrescribir el formulario base ya lo grabamos con el nombre definitivo que vaya a tener el programa.

Añadir un nuevo bloque de mantenimiento de una tabla:

- Incorporar un nuevo bloque usando el asistente.
- Seleccionamos tabla o vista.
- Indicamos la tabla y los campos que vamos a usar.
- En el asistente de diseño, añadimos los campos en el lienzo CANVAS_BASE y en el tab TAB0.
- Indicamos los campos que van a estar visibles en el mantenimiento.
- Poner la propiedad "*Clave Primaria*" a SI de todos los campos que forman la clave primaria de la tabla asociada al bloque. (*)
- Activar la propiedad del bloque "Forzar Clave Primaria" si la tabla asociada tiene clave primaria. (*)

(*) Solo si el programa no tiene numeración automática, como por ejemplo una entrada de albaranes, donde el número (parte de la clave primaria) se obtiene de un contador de forma automática.

El punto de asignar la clave primaria de muy importante, ya que la validación estándar si se encuentra con un campo que es clave primaria busca el resto de los campos que forman la clave primaria y si todos tiene valor comprueba si el registro ya existe, antes de que al usuario se le pidan más datos.

Para el punto de la validación de la clave primaria es importante que cuando se asigna algún valor por defecto de la clave primaria, por ejemplo, el valor para el código de la empresa no se realice en el disparador PRE-INSERT ya que si se hace en ese momento nunca todos los campos correspondientes a la clave primaria van a tener valor al ser validado alguno de ellos, por tanto, es recomendable asignarlos en el disparador WHEN-CREATE-RECORD, pero sin anular la funcionalidad estándar del mismo.

Ejemplo:

```
DISPSTD.WHEN_CREATE_RECORD;
:bloque.codigo_empresa := :global.codigo_empresa;
```

Poner el título del programa en la etiqueta de la pestaña TAB0 de CANVAS_BASE.

Poner a cada campo la etiqueta en castellano correspondiente en la propiedad Prompt. Nunca usar etiquetas de texto normales ya que no se pueden modificar de forma dinámica y por tanto no se pueden traducir.

Tampoco se debe de usar marcos con etiqueta, ya que al no poder cambiarse de forma dinámica tampoco se pueden traducir.

Borramos el marco que le pone el asistente de diseño al bloque.

Añadir al bloque la clase de propiedad correspondiente.

- CLASE_BLOQUE: Si solo se visualiza un solo registro.
- CLASE_BLOQUE_SCROLL: Si el bloque visualiza varios registros.

Añadir la clase correspondiente a los elementos visualizados:

- Elementos de texto
  - CLASE_TEXT_ITEM: El campo sólo se visualiza un registro.
  - CLASE_TEXT_ITEM_GRID: El campo está en un multiregistro.
- Elementos de fecha
  - CLASE_DATE_ITEM: El campo sólo se visualiza un registro.
  - CLASE_DATE_ITEM_GRID: El campo está en un multiregistro.
- Elementos numéricos

    - CLASE_TEXT_ITEM_NUMBER: El campo está en un bloque de 1 registro.
    - CLASE_TEXT_ITEM_NUMBER_GRID: Si el campo es multiregistro.
- Elemento de tipo casilla de verificación.: CLASE_CHECK_BOX.
- Botones de Radio: CLASE_RADIO_GROUP
- Elementos de lista de selección: CLASE_LIST_ITEM
- Botones: CLASE_BUTTON
- Lienzos y pestañas: CLASE_PAGE.
- Ventanas: CLASE_VENTANA

Modificar la propiedad "**Grupo de atributos visuales del Prompt**" dependiendo de las dos posibilidades siguientes:

- Campo Obligatorio: Atributo visual CAMPO_OBLIGATORIO_PROMPT.
- Campo Opcional: Atributo visual CAMPO_OPCIONAL_PROMPT.

Grupo de Atributos Visuales del Prompt	CAMPO_OPCIONAL_PROMPT

Si el campo se ha creado desde el asistente seguramente tenemos que volver a heredar las propiedades Alineamiento del prompt y Estilo de Visualización del Prompt, pulsando sobre la propiedad. Este paso se puede hacer con varios items seleccionados.

Prompt	
Prompt	*****
Estilo de Visualización del Prompt	Primer Registro
Justificación del Prompt	Principio
Borde del Anexo del Prompt	Superior
Alineamiento del Prompt	Centro
Desplazamiento del Anexo del Prompt	0
Desplazamiento del Alineamiento del Prompt	0
Orden de Lectura del Prompt	Por Defecto

Para el alineamiento del prompt en campos que no están en un grid de datos, es decir, no son multiregistro, el borde del anexo del prompt es fin, por tanto, inicialmente van a aparecer a la derecha del campo, lo que hay que hacer es poner el prompt a la izquierda y que se modifique la propiedad de desplazamiento del anexo del prompt y ***dejando la propiedad borde del anexo del prompt con el valor "fin"***. Esto es necesario para que las etiquetas sigan perfectamente alineadas después de ser modificadas en otro idioma.

MUY IMPORTANTE: Los campos que sean de descripción y que no pertenezcan a la tabla base del bloque les pondremos el mismo nombre que el campo, pero anteponiendo el prefijo 'D_'.

También hay que acordarse en las descripciones de poner el tamaño correcto para el campo descripción, acorde con el campo en la tabla en donde se almacena dicha descripción. Siempre es mejor pasarse de tamaño que quedarse corto.

A los campos de visualización les añadimos la clase correspondiente.

- CLASE_DISPLAY_ITEM: Si en el campo solo se visualiza un registro.
- CLASE_DISPLAY_ITEM_GRID: Si el campo es multiregistro.

En vez de usar el disparador POST-QUERY para cargar el valor, usaremos el campo "Nombre Columna Consulta" del mantenimiento de programas, usando el asistente (lista de valores) o introduciendo la SQL necesaria de la forma (SELECT <campo> FROM <tabla> WHERE <condición>), de esta forma se puede ordenar y filtrar por este campo.

Si no se utiliza el asistente para asociar la obtención de la descripción a una lista de valores y el campo puede ser NULL, podemos asignar a esta propiedad la siguiente sentencia SQL: *DECODE(<campo>, NULL, NULL, (SELECT <campo> FROM tabla WHERE <condición>))*. De esta forma si el campo es NULL directamente no hace nada, y si tiene valor va a buscar la descripción a la tabla correspondiente.

Esta forma de obtener la descripción da como principal ventaja que el usuario puede hacer filtros pulsando F7 y F8 y también se puede hacer una ordenación del bloque de datos por este campo.

Para más detalle ver el apartado: Campos de visualización de descripciones.

Ejemplo: DECODE(unidad_codigo1,NULL,NULL,(SELECT descripcion FROM unidades_almacen WHERE codigo= articulos.unidad_codigo1))

Colocar campos en la pantalla.

Para alinear los campos multiregistro los alinearemos al final y apilando horizontalmente. Una vez apilados horizontalmente los separaremos con el teclado un punto

Disparadores personalizados

Trataremos de poner todos los disparadores a nivel de bloque, controlando de que ítem viene el evento mediante la variable de sistema :system.trigger_item.

El orden de navegación de los bloques será el mismo que especificamos en el navegador de objetos del formulario, por tanto, el primer bloque navegable va a ser el primero que aparezca en el navegador de objetos.

IMPORTANTE: Siempre que usemos un disparador que tenga una funcionalidad estándar, deberemos dejar la lógica necesaria para que no se pierda esa funcionalidad, por ejemplo, si en el disparador WHEN-NEW-RECORD-INSTANCE, necesitamos desactivar un campo, deberemos dejar una llamada al procedimiento DISPSTD.WHEN_NEW_RECORD_INSTANCE.

Ejemplo:

```
DISPSTD.WHEN_NEW_RECORD_INSTANCE;
IF :bloque.campo = 1 THEN
  Set_Item_Property('BLOQUE.CAMPO2', ENABLED, PROPERTY_FALSE);
END IF;
```

Modificación de propiedades de campos (FITEM)

Normalmente las propiedades de los items se realizarán con la instrucción Set_Item_Property, excepto las siguientes (que a ser posible se usará el paquete FITEM).

- FITEM.VISIBLE(p_campo => <campo>, p_como => <parámetro>, p_activado => <parámetro>, p_navegable => <parámetro>, p_modificable => <parámetro>): Hace visible u oculta un Item. Cuando se hace visible además lo hace activo salvo que se indique p_activado => FALSE, navegable por teclado salvo que se indique p_navegable => FALSE y modificable salvo que ese indique p_modificable => FALSE.
- FITEM.ACTIVADO(p_campo => <campo>, p_como => <parámetro>, p_navegable => <parámetro>, p_modificable => <parámetro>): Activa o desactiva un Item. Cuando se activa además lo hace navegable por teclado salvo que se indique p_navegable => FALSE y modificable salvo que se indique p_modificable => FALSE.
- FITEM.MODIFICABLE(<campo>, <parámetro): Activa o desactiva la modificación de un campo, tanto en registros existentes como en los nuevos.
- FITEM.OBLIGATORIO(<campo>, <parámetro>): Activa o desactiva la obligatoriedad de que el usuario introduzca un valor para el campo, también cambia el atributo visual del prompt, por ejemplo, si se pone como obligatorio pone el Prompt en negrita.

<campo>: Cadena que contiene BLOQUE.CAMPO. Ejemplo: 'B8.ALMACEN'.

<parámetro>: Espera un booleano, es decir TRUE activa y FALSE desactiva.

NOTA: El uso de los parámetros p_activado, p_navegable, p_modificable hará que el fuente únicamente se pueda utilizar en versiones de entorno 6.4.8 o superior.

Campos de primary key de un bloque.

Si en un bloque a los campos que pertenecen a la clave primaria les activamos la propiedad de primary key a los campos que forman la clave primaria (en el bloque es recomendable activar la propiedad "Forzar Clave Primaria", pero no es obligatorio para esto) al estar introduciendo el registro cuando el usuario le da valor al último campo que forma la clave primaria se comprueba si ese registro ya existe en la base de datos, si ya existe se muestra un mensaje y no se espera a que termine de meter el registro completo para avisarle.

Para que funcione correctamente hay que tener muy en cuenta lo que se explicó en el apartado "Crear formulario desde cero" sobre los valores iniciales de los campos de la clave primaria asignarlos en el WHEN-CREATE-RECORD y no en el PRE-INSERT.

Campos Desde / Hasta

Normalmente en pantallas de filtro hay campos en los que se pide un valor desde y un valor hasta. Si es un campo que de forma normal el usuario va a especificar un mismo valor en el desde que en el hasta ya le proponemos el valor hasta igual que el desde en el momento de validar el campo desde.

Evitaremos mostrar mensajes de que el campo desde es mayor que el campo hasta, lo que haremos al validar el campo desde es comprobar si el hasta está vacío o es inferior que el desde, en ese caso asignamos al campo hasta el mismo valor que el desde. Si estamos validando el campo hasta y tiene el valor desde valor y este valor es mayor que el hasta le asignaremos al campo desde el valor del hasta.

Ejemplo de validación de un campo desde:

```
IF :b1.hasta IS NULL OR :b1.hasta < :b1.desde THEN
  :b1.hasta := :b1.desde;
  :b1.d_hasta := :b1.d_desde;
END IF;
```

Ejemplo de validación de un campo hasta:

```
IF :b1.desde IS NOT NULL AND :b1.desde > :b1.hasta THEN
  :b1.desde := :b1.hasta_cliente;
  :b1.d_desde := :b1.d_hasta;
END IF;
```

Máscaras

Una máscara será del tipo FM999G999G990D9990, en donde las G indican en donde irá el separador de millar, la D en donde irá el separador de decimales, el FM indica que si el importe es negativo que ponga el signo '-' pegado al primer número comenzando por la izquierda.

Las máscaras para asignar a campos numéricos que contengan información de un importe de una determinada divisa o un importe de una cantidad de almacén deben de usar el paquete PKMASCARAS. Este paquete tiene las siguientes funciones que devuelven la máscara adecuada para cada caso:

- **PRECIOS**(p_parte_entera, p_parte_decimal, p_divisa): Devuelve la máscara con los decimales adecuados al valor de DECIMALES_PRECIOS de la tabla DIVISAS para la divisa identificada por *p_divisa*. **Para precios en la divisa de la empresa en la que está validado el usuario lo más recomendable es poner en el mantenimiento de programas en la propiedad Máscara del campo el valor IMP.**
- **IMPORTES**(p_parte_entera, p_parte_decimal, p_divisa): Devuelve la máscara con los decimales adecuados al valor de DECIMALES_SIGNIFICATIVOS de la tabla DIVISAS para la divisa identificada por *p_divisa*. **Para importes en la divisa de la empresa en la que está validado el usuario lo más recomendable es poner en el mantenimiento de programas en la propiedad Máscara del campo el valor IMP.**
- **CANTIDADES**(p_parte_entera, p_parte_decimal, p_empresa): Devuelve la máscara con los decimales adecuados al valor de DEC_CANTIDAD de la tabla AL_PARAM01 para la empresa identificada por p_empresa. NOTA: Si no existe registro en AL_PARAM01 devolverá una

máscara con los decimales indicados en *p_parte_decimal*. **Para las cantidades lo más recomendable es poner en el mantenimiento de programas en la propiedad Máscara del campo el valor CTD.**

NOTA: Al estar la función de cálculo de máscaras en Base de Datos se puede usar tanto en Forms, Reports, etc. En Reports se puede sacar directamente como una columna más de la sentencia, en una columna de fórmula, etc.

Ejemplos de máscaras

Forms:

```
DECLARE
  v_mascara       VARCHAR2(50);
BEGIN
  v_mascara  := PKMASCARAS.PRECIOS(15, 4, :parameter.divisa_empresa);
  Set_Item_Property('BLOQUE.CAMPO', FORMAT_MASK, v_mascara);
END;
```

Reports:

```
FUNCTION <función> IS
  v_mascara                VARCHAR2(50);
BEGIN
  v_mascara := PKMASCARAS.PRECIOS(15, 4, :p_divisa);
  srw.attr.mask := srw.formatmask_attr;
  srw.attr.formatmask := v_mascara;
  srw.set_attr(0,srw.attr);
  RETURN (TRUE);
END;
```

Funciones Varias

- **BORDEN.MENU_ESTABLECER_ORDEN('REFRESCO')**: Vuelve a ejecutar consulta en el bloque en que se encuentra el cursor, respetando los filtros aplicados por el usuario y vuelve a posicionarse en el mismo registro y campo en el que se encontraba el cursor.

Control de Errores

Oracle para la gestión de errores tiene las excepciones y los errores ORA. En la aplicación se podrían codificar errores ORA entre el -20999 y el -20000 para gestión de errores personalizados lo que nos deja 1000 errores para ser codificados.

Tenemos que usar un sistema que permita estos puntos:

- Sea simple.
- Rápido de implementar.
- Que se genere justo en el punto en el que se detecta el error.
- Los errores le lleguen al usuario y traducidos si es necesario.
- Que los puedan capturar las alertas.
- Que sea gestionable

Para ello el entorno se ha reservado el código de error ORA-20100 para tratar cualquier error que pueda producirse **en procedimientos almacenados en base de datos**.

Cuando se produzca un error ORA-20100, el entorno lo gestionará de forma de que llegue al usuario o al motor de alertas de forma clara.

Cuando en el código se quiera lanzar un error hay que poner un RAISE_APPLICATION_ERROR(pkerr.c_ex_error_rae, 'Texto codificado del error');

En 'Texto codificado del error' tiene que ir con una codificación que pueda entender el entorno, por lo que **no se puede meter un texto directamente** y en vez de eso se llamará a la función PKERR.GENERA_MENSAJE_RAE. Esta función tiene los siguientes parámetros:

Obligatorios: Necesarios para mostrar el mensaje al usuario:

- **p_tipo_mensaje**: Tipo del mensaje codificado en Libra.
- **p_codigo_mensaje**: Código del mensaje dentro del tipo indicado en *p_tipo_mensaje*. Únicamente deberían de usarse mensajes que tengan un único botón, ya que desde base de datos no se puede interpretar si el usuario pulsa uno u otro botón.

Opcionales:

- **p_texto_ampliacion**: Texto que se concatenará al mensaje. Este texto se intentará traducir por lo que hay que evitar concatenar valores fijos variables, por ejemplo: 'Artículo: 1345451NL'. Lo mejor es meter códigos de sustitución, por ejemplo: 'Artículo: {art}', luego mediante las variables se podrá indicar que se sustituya {art} por un valor, pero esa sustitución la hace después de hacer la traducción.
- **p_codigo_excepcion**: Cuando se trata posteriormente la excepción se puede utilizar los valores indicados en p_tipo_mensaje y p_codigo_mensaje, pero esos mensajes pueden ser muy genéricos y utilizados en varios puntos. Si se indica p_codigo_excepcion, luego se podrá utilizar este código para tratar la excepción y determinar de forma más precisa el origen de esta, por lo que el valor indicado debería ser un valor único, como por ejemplo estos valores:
  - **ex_stock_negativo**
  - **ex_bloqueo_inventario**
  - **ex_cliente_bloqueado**
- **p_texto_info_adicional**: Cualquier información adicional que se quiera registrar. Esta información no se mostrará al usuario y simplemente se meterá en el texto del error, de forma que al tratar el error con pkpantallas.log se guardará en LIBRA_LOG o al ser capturado por la alerta en el LOG de la alerta. Es interesante guardar el valor de variables que puedan ayudar al equipo de soporte o a desarrolladores a poder determinar el motivo de que se hubiese producido el error que se está gestionando.
- **p_variableX** (dónde X es un valor entre 1 y 9): Permite indicar un código de variable que se encuentra en el texto del mensaje o en p_texto_ampliacion y que debe de ser reemplazado, por ejemplo, sin en p_texto_ampliacion se indica 'Artículo {art}', en p_variableX se asignará el valor '{art}'.

- **p_valor_variableX** (dónde X es un valor entre 1 y 9): Indica el valor por el que tiene que ser sustituida la variable en el texto del mensaje, por ejemplo, si en p_variable se indica {art}, se asignará el valor que contenga la variable o parámetro con el código de artículo. También se puede usar con p_etiqueta_variable, tal y como se explica a continuación.
- **p_etiqueta_variableX** (dónde X es un valor entre 1 y 9): Se pueden añadir valores al texto del mensaje de forma que se añaden únicamente cuando el p_valor_variableX es no nulo. El texto indicado se intentará traducir al idioma del usuario. Esto permite añadir trozos de texto al mensaje según una variable tenga o no valor, por ejemplo: **p_etiqueta_variable1 => 'Nº Lote Interno', p_valor_variable1 => p_numero_lote_int**. En el caso de que p_numero_lote_int sea NULL no se añadirá nada al mensaje, pero si p_numero_lote_int tiene el valor 'XXX123' se añadirá al mensaje que le llega al usuario: **, Nº Lote Interno: XXX123**

En PL/SQL Developer (configuración específica de Edisa) hay plantillas para meter este código de forma rápida:

RAE_

```
RAISE_APPLICATION_ERROR(pkerr.c_ex_error_rae, pkerr.genera_mensaje_rae(p_tipo_mensaje => '', p_codigo_mensaje
=> ''));
```

ARAE_

```
pkpantallas.assert(
  p_condicion => ,
  p_mensaje => pkerr.genera_mensaje_rae(p_tipo_mensaje => '', p_codigo_mensaje => ''),
  p_paquete => $$PLSQL_UNIT,
  p_punto => $$PLSQL_LINE,
  p_cod_excepcion => pkerr.c_ex_error_rae);
```

Ejemplo:

```
raise_application_error(pkerr.c_ex_error_rae,
  pkerr.genera_mensaje_rae(p_tipo_mensaje => 'A_STK',
                           p_codigo_mensaje => 'NO_HAY',
                           p_etiqueta_variable1 => 'Artículo',
                           p_valor_variable1 => p_articulo,
                           p_etiqueta_variable2 => 'Ubicación',
                           p_valor_variable2 => p_ubicacion,
                           p_etiqueta_variable3 => 'Palet',
                           p_valor_variable3 => p_palet,
                           p_etiqueta_variable4 => 'N° Serie Interno',
                           p_valor_variable5 => v_numero_serie_int,
                           p_etiqueta_variable6 => 'N° Lote Interno',
                           p_valor_variable6 => p_numero_lote_nt,
                           p_texto_info_adicional => 'lote_int: ' || v_lote_int));
```

Cuando se lanza ese RAISE_APPLICATION_ERROR se fuerza un ORA-20100 con un texto codificado en SQLERRM. La función o procedimiento que lanza ese RAISE_APPLICATION_ERROR deberá tener controlada la excepción WHEN OTHERS de la siguiente forma:

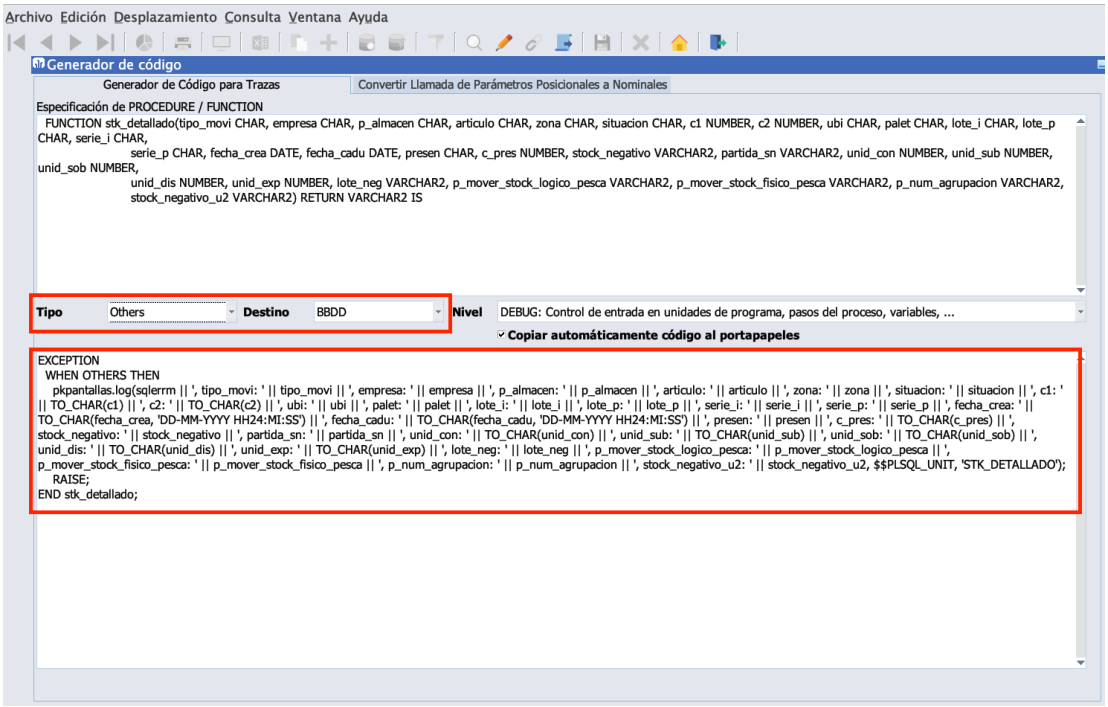
```
EXCEPTION
  WHEN OTHERS THEN
    pkpantallas.log(sqlerrm || ', lista de información a trazar', $$PLSQL_UNIT, 'NOMBRE');
    RAISE;
END;
```

Ejemplo:

```
  EXCEPTION
    WHEN OTHERS THEN
      pkpantallas.log(SQLERRM || ', tipo_movi: ' || tipo_movi || ', empresa: ' || empresa || ', p_almacen: ' || p_almacen || '
                      situacion || ', c1: ' || TO_CHAR(c1) || ', c2: ' || TO_CHAR(c2) || ', ubi: ' || ubi || ', palet: ' || pa
                      serie_i || ', serie_p: ' || serie_p || ', fecha_crea: ' || TO_CHAR(fecha_crea, 'DD-MM-YYYY HH24:MI:SS')
                      ', presen: ' || presen || ', c_pres: ' || TO_CHAR(c_pres) || ', stock_negativo: ' || stock_negativo || '
                      ', unid_sub: ' || TO_CHAR(unid_sub) || ', unid_sob: ' || TO_CHAR(unid_sob) || ', unid_dis: ' || TO_CHAR(
                      lote_neg || ', p_mover_stock_logico_pesca: ' || p_mover_stock_logico_pesca || ', p_mover_stock_fisico_pe
                      p_num_agrupacion || ', stock_negativo_u2: ' || stock_negativo_u2, $$PLSQL_UNIT, 'STK_DETALLADO');
      RAISE;
  END stk_detallado;
```

Ese pkpantallas.log y el RAISE; en el WHEN OTHERS **es vital** para que el entorno pueda decodificar el error para mostrárselo al usuario de forma clara.

En los WHEN OTHERS deben de meterse en el log todos los parámetros de la llamada a la función y cualquier otra variable que se considere importante. Para hacer esto se debe de utilizar el programa U_GENCODIGO de Libra en el qué indicando la especificación de la función nos dará el código a introducir para finalizar el procedimiento o la función.



En la unidad de programa llamadora, si fuese necesario se podría gestionar el error y tratarlo, por ejemplo, se está moviendo almacén de 30 artículos y uno no tiene stock, con lo que lanzará la excepción de que no hay stock, pero podría darse el caso de que el programa llamador sabe como gestionar eso y pueda mover el almacén de los 29 restantes y luego indicarle al usuario los que no ha podido realizar. Eso lo sabe la unidad de programa que llama al procedimiento de mover almacén, pero eso unidad de programa no tiene qué saber que quien le pide mover almacén sabe gestionar ese error, por lo que el tratamiento ahí es exactamente igual, si no hay stock se lanza una excepción con el mensaje A_STK – NO_HAY y el código de excepción “ex_stock_negativo”.

El programa llamador para gestionarlo tendrá que meter el control de la excepción y tratarla:

```
BEGIN
  llamada_a_mover_almacen();
EXCEPTION
  WHEN pkerr.ex_error_rae THEN
    IF pkpantallas.codigo_error_rae() = 'A_STK-NO_HAY' THEN
      pkpantallas.limpia_error_rae();
      -- Tratar el error como se considere oportuno.
      --...
      --...
    ELSE
      RAISE;
    END IF;
END;
```

- Con WHEN pkerr.ex_error_rae THEN se captura el error lanzado con cualquier RAISE_APPLICATION_ERROR(pkerr.c_ex_error_rae, ‘XXXX’);

- Con pkpantallas.codigo_error_rae() nos indica el tipo + mensaje asociado al error PKERR.EX_ERROR_RAE.
- Si es el error que se quiere gestionar hay que ejecutar **pkpantallas.limpia_error_rae()** para indicarle al entorno que vamos a tratar el error y que se olvide de él.
- Si no es un código de error conocido y que se pueda gestionar se propagará con **RAISE**;

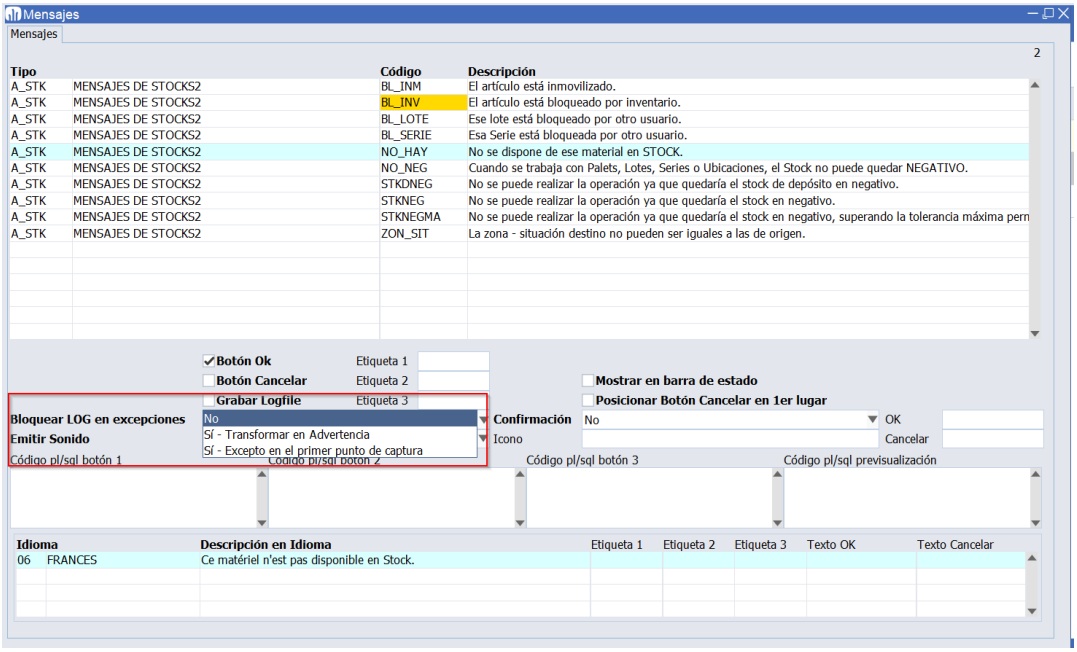
También se podría utilizar el código de excepción indicado (si no se indicó será igual que pkpantallas.codigo_error_rae).

```
BEGIN
  llamada_a_mover_almacen();
EXCEPTION
  WHEN pkerr.ex_error_rae THEN
    IF pkpantallas.codigo_excepcion_rae() = 'ex_stock_negativo' THEN
      pkpantallas.limpia_error_rae();
      -- Tratar el error como se considere oportuno.
      --...
      --...
    ELSE
      RAISE;
    END IF;
END;
```

El texto de un error SQERRM se puede traducir para que sea entendible para los usuarios con pkpantallas.texto_error_rae(p_mensaje => sqlerrm);

Al ser errores gestionados por el entorno, desde la configuración del mensaje se puede indicar como debe de comportarse sobre el registro en LIBRA_LOG de la excepción, ya que una excepción de que no hay stock podría ser que no interese que se registre en LIBRA_LOG.

En “Mensajes” y “Mensajes Personalizados” se puede indicar el comportamiento mediante el campo “Bloquear LOG en excepciones”



Los valores posibles son:

- **No**: Se graba LIBRA_LOG en todos los puntos en los que se capture la excepción.
- **Sí – Transformar en Advertencia**: No se graban en LIBRA_LOG, pero si está la traza activada se registran como WARNINGS.
- **Sí – Excepto en el primer punto de captura**: En la primera función o procedimiento que capture la excepción grabará en LIBRA_LOG, el resto son transformadas a WARNINGS que se registrarán si está la traza activa.

Nomenclatura de SQLS

Siempre que se haga un INSERT en una tabla se indicarán todos los campos que se están insertando, incluso si estamos metiendo valor a todos los campos, ya que en el momento de hacer la SQL vemos unos campos en la tabla, pero nadie nos asegura que más adelante (antes de sacar la versión) se añadan nuevos campos a esa tabla:

- Ejemplo Incorrecto: INSERT INTO tabla VALUES (v1, v2, ..., vn);
- Ejemplo Correcto: INSERT INTO tabla (campo1, campo2, ... campon) VALUES (v1, v2, ..., vn);

Comprobaremos que todas las líneas del sql están finalizadas con punto y coma (también sirve poner en la línea siguiente una barra /).

Toda secuencia de INSERT, UPDATE, DELETE llevará un COMMIT. A las sentencias DDL (es decir aquellas que cambian una estructura de la base de datos CREATE, DROP, ALTER, REPLACE) no hace falta hacer COMMIT ya llevan uno implícito.

Los nombres de las SQLS deben ser de la siguiente forma:

El nombre del objeto cuando la SQL sea de (En la versión solo saldrá la versión más reciente del objeto.

- Procedimiento.
- Función.
- Paquete
- Vista
- Trigger

Nombre del objeto más un sufijo (tal y como se genera desde libra) cuando la sql sea de (En la versión solo saldrá la versión más reciente del objeto, no hace falta mandar toda la historia de modificaciones)

- Programa: Nombre del programa + _PR
- Lista de valores: Nombre de la lista de valores + _LV
- Mensaje: Código del mensaje + _MSG

Resto de SQLS: **aa|mm|dd|nm|ce|md|fn.sql**

- **aa**: dos últimos dígitos del año.
- **mm**: mes (2dígitos).
- **dd**: día (2 dígitos).
- **nm**: numeración de las sqls del mismo día y módulo.
- **ce**: centro de Edisa:
    - **md**: Madrid.
    - **ov**: Oviedo.
    - **ou**: Ourense
    - **vg**: Vigo
    - **bc**: Barcelona.
- **md**: módulo de libra:
    - **us**: (Entorno)
    - **crm**: (CRM)
    - **fi**: (Financiero)
    - **cp**: (Compras - Aprovisionamiento)
    - **al**: (Logística - Almacén)
    - **fa**: (Ventas - Distribución)
    - **pr**: (Producción)
    - **ca**: (Calidad)
    - **vh**: (Mantenimiento SAT)

- **prc**: (Gestión de Proyectos)
- **no**: (Nóminas)
- **pre**: (Control de presencia)
- **rh**: (Recursos Humanos)
- **gd**: (Gestor Documental)
- **mof**: (Movilidad OFF-LINE)
- **mon**: (Movilidad ON-LINE)
- **web**: (Web)
- **gal**: (Servicios Galileo)
- **ron**: (Reporting ON-LINE)
- **bi**: (Business Inteligence)

- **fn**: funcionalidad de la SQL:
  - **mn** (Menús).
  - **tb** (Tablas). Modificación, creación, etc.
  - **pv** (Procesos Varios). Todo tipo de procesos de actualización de datos, etc., que no entren en los grupos anteriores.

Por ejemplo, si se cambia una tabla de facturación a fecha de hoy, y ya se han creado hoy 2 SQLS de facturación en Vigo, el nombre quedaría así: **01011003vgfatb.sql**

Notificación de errores en procesos desatendidos

La tendencia debería ser cada vez más a que las tareas se ejecuten sin ninguna intervención de usuarios, pero esto plantea el problema de cómo alertar a los administradores del sistema o a ciertos usuarios de que se están produciendo problemas y puedan solucionarlos.

Ahora mismo hay procesos de este tipo que en caso de fallo comienzan a grabar registros en LIBRA_LOG produciendo que se dispare su tamaño y nadie se entera de que algo está fallando ya que casi nadie está revisando de forma activa la tabla LIBRA_LOG en busca de problemas.

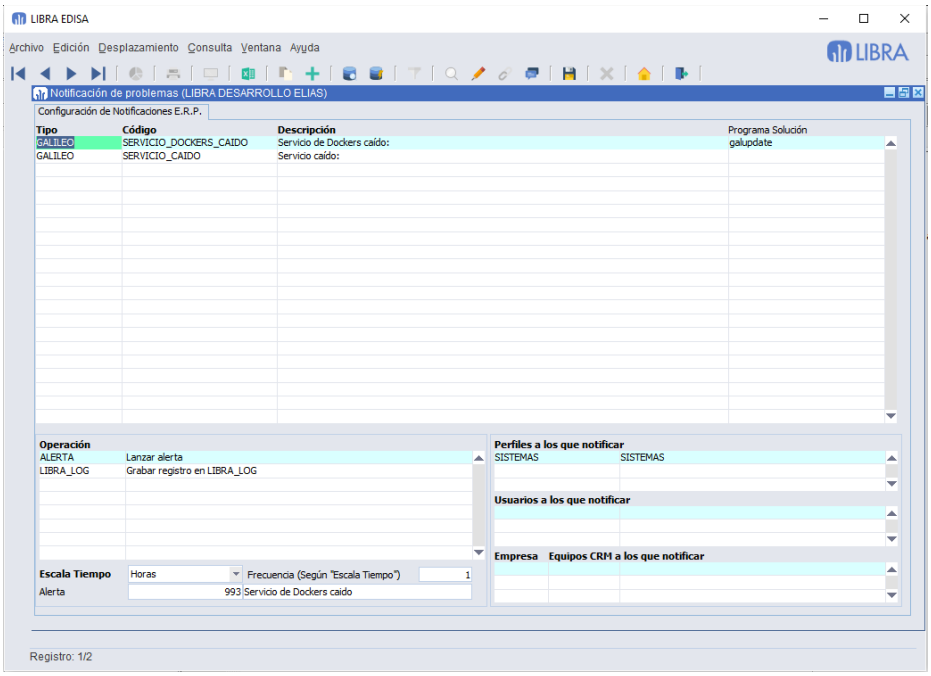
Los procesos desatendidos tienen que registrar un fallo deben de llamar a pk_notificaciones.notificar_error. Este procedimiento recibe 4 parámetros:

- **p_localizador**: Es el código por el cual se identifica el error, por tanto, hay que implementar algo que en el caso de que se produzca un mismo error se le asigne el mismo localizador para que la gestión de la notificación se haga correctamente. **NOTA**: En el caso de pasar este parámetro a NULL se usará como localizador la firma SHA1 del texto pasado en el parámetro **p_mensaje** y en la pantalla de visualización de notificaciones pendientes a la descripción no le será concatenado el localizador.
- **p_mensaje**: Texto que sea entendible para un humano que revise la notificación de error para que pueda localizar y arreglar el problema en el menor tiempo posible. NOTA: Para dar posibilidad de traducir algunas etiquetas dentro del mensaje dependiendo del idioma del usuario que lo esté visualizando se pueden introducir las etiquetas entre el prefijo {<etq> y el sufijo </etq>}, lo que esté entre esas etiquetas se intentará traducir al idioma del usuario. Por ejemplo: *{<etq>Este texto será traducido al idioma del usuario</etq>}*
- **p_tipo** y **p_codigo**: Codificación de la notificación. Esta codificación debe de estar dada de alta en el programa "u_param_notif" para que Libra sepa cómo tratarla, en caso de no estar dada de alta simplemente se grabará un registro en LIBRA_LOG de la misma manera que si se hubiese ejecutado pkpantallas.log.

Ejemplo:

```
pk_notificaciones_erp.notificar_error(v_servidor_dockers_error || '.' || p_peticion.id_galileo, v_texto_error, 'GALILEO', 'SERVICIO_DOCKERS_CAIDO');
```

Configuración de Notificaciones



En el programa u_param_notif se indicará por cada tipo + código de notificación lo siguiente:

- **Descripción**: Texto que se le añadirá al asunto de la notificación junto al código de localizador.
- **Programa Solución**: Cuando el usuario está visualizando la notificación por pantalla en Libra tendrá un botón que abrirá el programa que está configurado. Este campo es opcional.

Lista de Operaciones a realizar para realizar la notificación

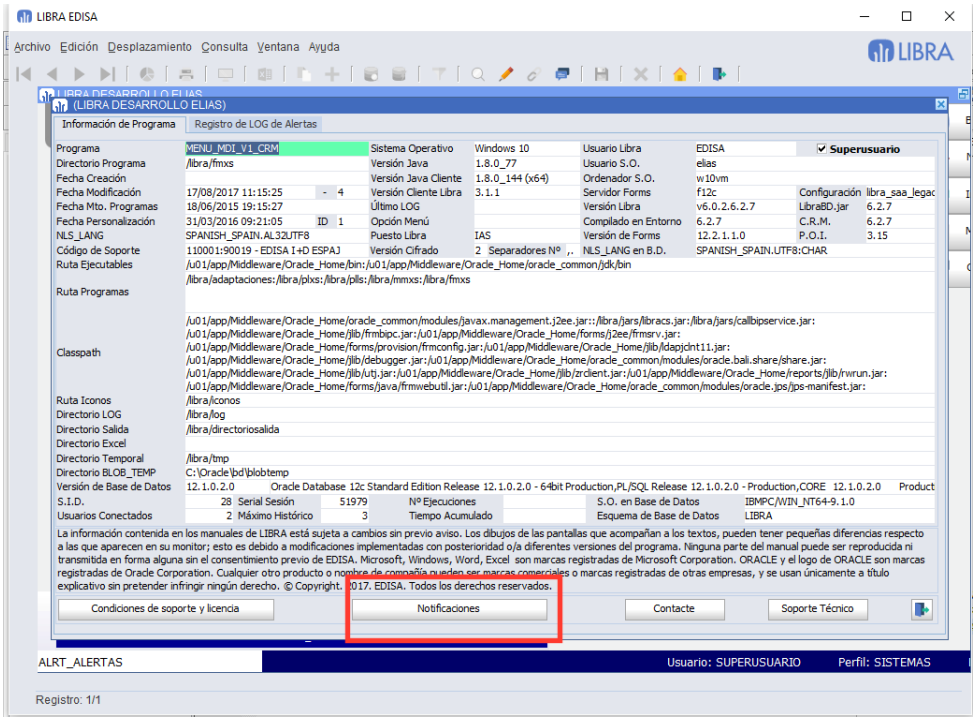
Están implementadas 2 operaciones posibles:

- **ALERTA**: Pondrá en cola una alerta para que realice la notificación por correo electrónico a los destinatarios parametrizados en esa alerta. Al indicar este tipo se mostrará el campo "Alerta" para poder indicar el número de alerta a ejecutar.
- **LIBRA_LOG**: Registra en la tabla LIBRA_LOG una entrada.

Estas operaciones no son excluyentes, es decir, si se configura ALERTA y LIBRA_LOG realizará las 2 y lo hará según lo que se configure en los campos ***Escala Tiempo + Frecuencia Según "Escala Tiempo"***. Si se ha lanzado la notificación y antes de que pase la frecuencia indicada de notificación, vuelve a llegar otra notificación no será lanzada, simplemente se registra aumentando el número de ocurrencias.

En la configuración de la notificación se puede indicar a qué **Perfiles / Usuarios / Equipos CRM** que se deben de notificar al entrar en Libra en el caso de que exista alguna notificación abierta de ese tipo + código. En el caso de que un usuario que se valide en Libra y tenga notificaciones que deben de serle notificadas se le abrirá el programa de notificaciones de forma automática al entrar, desde este programa ya podría marcarla como solucionada o ejecutar el programa que esté configurado como "Programa Solución".

Una vez dentro de Libra, desde cualquier "Acerca de..." se puede ir a ese programa de notificaciones.



Generación de hojas de cálculo

Para generar hojas de cálculo desde Forms disponemos la librería PKLIBXLS.PLL. Su objetivo es generar de forma sencilla hojas de cálculo a partir de SQLS.

También se pueden generar hojas de cálculo desde Códigos PL/SQL, el funcionamiento es exactamente el mismo, pero en vez de hacer llamada al paquete PKXLS hay que hacer llamada a PKXLSBD, el resto de las funciones del API de generación de hoja de cálculo es idéntica, en la documentación se hace referencia a PKXLS, pero también es válido para PKXLSBD.

Pasos para la generación de una hoja de cálculo

Los pasos para generar una hoja de cálculo son los siguientes:

1. Llamada a: **pkxls.inicializa**: Simplemente inicializa estructuras internas del paquete pkxls, no recibe ningún parámetro, pero es obligatorio que sea la primera instrucción que se ejecute.
2. Dar las propiedades a la hoja de cálculo con: pkxls.set_propiedad_excel: Define propiedades a nivel del archivo de hoja de cálculo, como por ejemplo el directorio en donde se va a generar, nombre del archivo, etc.
3. Crear como mínimo una hoja de cálculo dentro del archivo, (se pueden generar tantas como sean necesarias) llamando a la función **pkxls.crea_hoja**: La llamada a esta función nos devuelve un NUMBER que identifica la hoja, ese dato será necesario almacenarlo para dar luego propiedades a la hoja, como por ejemplo las sqls que debe de ejecutar.
4. Dar propiedades a la hoja, con **pkxls.set_propiedad_hoja**: Define propiedades a nivel de hoja, como por ejemplo el nombre de la hoja. Este paso es opcional por defecto la crea con el nombre Datos.
5. Añadir a la hoja tantas sqls como sean necesarias con la función **pkxls.crea_sql**, hay que pasarle como parámetro el identificador de la hoja que ha devuelto la función pkxls.crea_hoja. Nos devuelve un NUMBER que identifica la SQL, ese dato será necesario almacenarlo para poder dar luego propiedades a la SQL. Las SQLS se ejecutarán de forma independiente y una al finalizar la otra y en el orden en que se llame a pkxls.crea_sql. **Este paso es opcional**, hay un método alternativo para asignar directamente valores a las celdas indicando las coordenadas de la celda y el valor a asignar con pkxls.excel_celda.
6. Dar propiedades a la SQL, con **pkxls.set_propiedad_sql**: Define propiedades a nivel de SQL, como por ejemplo la SELECT que ha de ejecutar. **Este paso es opcional**.
7. Dar propiedades a las columnas que se obtienen de la SQL con **pkxls.set_propiedad_columna**: Define propiedades a nivel de un determinado campo de la SQL, como por ejemplo la máscara de formato. **Este paso es opcional**.
8. Por defecto todos los campos numéricos se totalizan con suma, esa funcionalidad se puede modificar creando tantas fórmulas de totalización como sean necesarias con **pkxls.crea_formula_total_sql**, cada fórmula de totalización generará una fila de totales en el orden en que llamemos a la función. Esta función devuelve un NUMBER que identifica la fórmula, ese dato será necesario almacenarlo para poder dar luego propiedades a la fórmula. **Este paso es opcional**.
9. Lanzar el proceso de generación llamando a **pkxls.generar_xls**. **Este paso es obligatorio**, es el que compila toda la información que se le ha proporcionado al paquete pkxls y genera la hoja de cálculo.

Ejemplo:

```
DECLARE
  v_id_hoja NUMBER;
  v_id_sql  NUMBER;
BEGIN
  pkxls.inicializa;
  pkxls.set_propiedad_excel('DIRECTORIO', 'c:\temp');
  pkxls.set_propiedad_excel('ARCHIVO', 'prueba.xls');
  v_id_hoja := pkxls.crea_hoja();
  v_id_sql := pkxls.crea_sql(v_id_hoja);
  pkxls.set_propiedad_sql(v_id_sql, 'SQL', 'SELECT codigo_rapido "Código", nombre "Nombre", reservadon01 "Dato
Numérico" FROM clientes WHERE codigo_empresa = :global.codigo_empresa');
  pkxls.generar_xls();
END;
```

Propiedades

Archivo de hoja de cálculo

Se usará el procedimiento pkxls.set_propiedad_excel(<propiedad>, <valor>). Los valores que puede tomar <propiedad> son los siguientes:

- **DIRECTORIO**: Directorio en el que se va a generar el archivo.
- **ARCHIVO**: Nombre del archivo, si no se indica se generará un nombre de archivo concatenando el nombre del programa con el usuario de libra y la fecha y hora de la generación.
- **ABRIR_EXCEL**: Una vez generado el archivo lo abre con la aplicación predeterminada, sea Office o OpenOffice, por defecto está activado, para desactivarlo hay que pasar en <valor> 'N'.
- **USAR_CONEXION_DIRECTA**: Para la generación de la hoja de cálculo se establecerá una nueva conexión a la base de datos independiente.
- **NUMERO_FILAS_EN_MEMORIA**: Si se indica este parámetro en <valor> se indicará cada cuantas filas se debe de escribir en disco, de esta forma se libera memoria permitiendo superar el límite de filas al que se estaría limitado de la otra forma al quedarse sin memoria la máquina virtual de Java. Al indicar este parámetro obligatoriamente el formato de salida será XLSX en vez de XLS.
- **OCULTAR_COLUMNA_255**: La columna 255 se usa internamente para guardar un código de agrupación para luego poder hacer totalizaciones, por defecto esa columna se oculta ya que al usuario no le interesa para nada, pero mientras se está desarrollando posiblemente interese ver el código de agrupación que se asigna. Para desactivar la ocultación de la columna 255 hay que pasar en <valor> 'N'.
- **TEXTO_TOTAL**: Texto que se pone en las filas de totalización, como título de la totalización. Si es un total de una agrupación se concatena a este texto el título de la columna de la agrupación.
- **IMPRIMIR_CABECERA**: Si se pasa en <valor> 'N' se evita que se ponga el título del informe, nombre de la empresa y el usuario que genera la hoja de cálculo.
- **SQL_TITULOS_TAM_FUENTE**: Tamaño de la fuente de celdas de títulos. El valor se indica en unidades 1/20 de puntos, es decir, para un tamaño de 40 habría que pasar 800, es decir, 40 * 20.
- **SQL_TITULOS_NOMBRE_FUENTE**: Nombre de la fuente a utilizar en celdas de títulos.
- **SQL_TITULOS_COLOR_FUENTE**: Color de la fuente a utilizar en celdas de títulos. Consultar la tabla "Colores" en la sección "Constantes" para ver los colores disponibles.
- **SQL_TITULOS_COLOR_FONDO**: Color de fondo a utilizar en celdas de títulos. Consultar la tabla "Colores" en la sección "Constantes" para ver los colores disponibles.
- **SQL_TITULOS_NEGRITA**: Se indica si las filas de título van a ponerse en negrita, por defecto los títulos se ponen en negrita, para desactivarlo hay que pasar en <valor> la constante: **pkxls.boldweight_normal**. Para ver la lista de constantes que se puede usar consultar la sección "Constantes", tabla "Negrita"
- **SQL_TITULOS_SUBRAYADO**: Se indica si las filas de título van a llevar el texto subrayado, por defecto está desactivado, para activarlo hay que pasar en <valor> la constante: **pkxls.u_single**. Para ver la lista de constantes se puede consultar la sección "Constantes", tabla "Subrayado".

- **SQL_TITULOS_BORDE_SUP**: Se indica si a las celdas de título van a llevar marcado el borde superior, por defecto se le pone borde, para cambiarlo se puede pasar en <valor> una de las siguientes constantes descritas en la sección “Constantes”, tabla “Borde”.
- **SQL_TITULOS_BORDE_INF**: Igual que SQL_TITULOS_BORDE_SUP pero para el borde inferior de la celda, por defecto se le pone borde y puede llevar los mismos valores.
- **SQL_TITULOS_BORDE_DER**: Igual que SQL_TITULOS_BORDE_SUP pero para el borde derecho de la celda, por defecto se le pone borde y puede llevar los mismos valores.
- **SQL_TITULOS_BORDE_IZQ**: Igual que SQL_TITULOS_BORDE_SUP pero para el borde izquierdo de la celda, por defecto se le pone borde y puede llevar los mismos valores.
- **SQL_ROTACION_TITULO**: Ángulo de rotación del título, por ejemplo 90 para ponerlo en vertical.
- **SQL_TOTALES_TAM_FUENTE**: Tamaño de la fuente de celdas de totalización. El valor se indica en unidades 1/20 de puntos, es decir, para un tamaño de 40 habría que pasar 800, es decir, 40 * 20.
- **SQL_TOTALES_NOMBRE_FUENTE**: Nombre de la fuente a utilizar en celdas de totalización.
- **SQL_TOTALES_COLOR_FUENTE**: Color de la fuente a utilizar en celdas de totalización. Consultar la tabla “Colores” en la sección “Constantes” para ver los colores disponibles.
- **SQL_TOTALES_COLOR_FONDO**: Color de fondo a utilizar en celdas de totalización. Consultar la tabla “Colores” en la sección “Constantes” para ver los colores disponibles.
- **SQL_TOTALES_NEGRITA**: Igual que SQL_TITULOS_NEGRITA, pero para las filas de totalización, por defecto se ponen en negrita.
- **SQL_TOTALES_SUBRAYADO**: Igual que SQL_TITULOS_SUBRAYADO, pero para las filas de totalización, por defecto no se subrayan.
- **SQL_TOTALES_BORDE_SUP**: Igual que SQL_TITULOS_BORDE_SUP, pero para las filas de totalización, por defecto se marca el borde.
- **SQL_TOTALES_BORDE_INF**: Igual que SQL_TITULOS_ BORDE_INF, pero para filas de totalización, por defecto se marca el borde.
- **SQL_TOTALES_BORDE_DER**: Igual que SQL_TITULOS_BORDE_DER, pero para filas de totalización, por defecto se marca el borde.
- **SQL_TOTALES_BORDE_IZQ**: Igual que SQL_TITULOS_BORDE_IZQ, pero para filas de totalización, por defecto se marca el borde.
- **SQL_DATOS_TAM_FUENTE**: Tamaño de la fuente de celdas de datos. El valor se indica en unidades 1/20 de puntos, es decir, para un tamaño de 40 habría que pasar 800, es decir, 40 * 20.
- **SQL_DATOS_NOMBRE_FUENTE**: Nombre de la fuente a utilizar en celdas de datos.
- **SQL_DATOS_COLOR_FUENTE**: Color de la fuente a utilizar en celdas de datos. Consultar la tabla “Colores” en la sección “Constantes” para ver los colores disponibles.
- **SQL_DATOS_COLOR_FONDO**: Color de fondo a utilizar en celdas de datos. Consultar la tabla “Colores” en la sección “Constantes” para ver los colores disponibles.
- **SQL_DATOS_NEGRITA**: Igual que SQL_TITULOS_NEGRITA, pero para filas de datos, por defecto no se pone en negrita.
- **SQL_DATOS_SUBRAYADO**: Igual que SQL_TITULOS_SUBRAYADO, pero para filas de datos, por defecto no se pone en subrayado.
- **SQL_DATOS_BORDE_IZQ**: Igual que SQL_TITULOS_BORDE_IZQ, pero para las filas de datos, por defecto se marca el borde.
- **SQL_DATOS_BORDE_INF**: Igual que SQL_TITULOS_BORDE_INF, pero para las filas de datos, por defecto no se marca el borde
- **SQL_DATOS_BORDE_DER**: Igual que SQL_TITULOS_BORDE_INF, pero para filas de datos, por defecto se marca el borde.
- **SQL_DATOS_BORDE_SUP**: Igual que SQL_TITULOS_BORDE_SUP, pero para fila de datos, por defecto no se marca el borde.

- **SQL_DATOS_BORDE_ULT_FILA_SUP**: Igual que SQL_DATOS_BORDE_SUP, pero para la última fila de datos de la agrupación o del listado, por defecto no se marca el borde.
- **SQL_DATOS_BORDE_ULT_FILA_INF**: Igual que SQL_DATOS_BORDE_INF pero para la última fila de datos de la agrupación o del listado, por defecto se marca el borde.
- **SQL_DATOS_BORDE_ULT_FILA_DER**: Igual que SQL_DATOS_BORDE_DER pero para la última fila de datos de la agrupación o del listado, por defecto se marca el borde.
- **SQL_DATOS_BORDE_ULT_FILA_IZQ**: Igual que SQL_DATOS_BORDE_IZQ pero para la última fila de datos de la agrupación o del listado, por defecto se marca el borde.
- **SQL_PONER_FONDO_FILAS_IMPARES**: Sirve para resaltar las filas impares de la sql. Se puede usar cualquier color descrito en la sección "Constantes" en la "Colores", pero también se puede indicar a mayores uno de los siguientes:
    - S: Activa con el valor por defecto.
    - N: Desactiva el resaltado.

Fuentes y Estilos

De forma directa no se pueden asignar fuentes a celdas hay que hacerlos a través de un estilo. Si lo único que se busca es dar color al fondo y a la fuente de las celdas se pueden utilizar los estilos prefijados en la vista V_COLORES_ERP, indicando a las celdas como código de estilo el valor de V_COLORES_ERP.COLOR.

Para crear una fuente hay que usar la función pkxls.crea_fuente(), esta función devuelve un NUMBER que identifica de forma única a la fuente. Mediante el identificador se pueden asignar propiedades a la fuente usando el procedimiento pkxls.set_propiedad_fuente(<id_fuente>, <propiedad>, <valor>). El identificador será necesario también para asignar la fuente a un estilo. Los valores que puede tomar <propiedad> son los siguientes:

- **NEGRITA**: Permite activar o desactivar la negrita de la fuente, los valores posibles se detallan en la sección "Constantes" en la tabla "Negrita".
- **SUBRAYADO**: Permite activar o desactivar el subrayado de la fuente, los valores posibles se detallan en la sección "Constantes" en la tabla "Subrayado".
- **NOMBRE_FUENTE**: Nombre de la fuente a usar en las celdas. En <valor> hay que pasar directamente el nombre de la fuente, por ejemplo 'Courier New'.
- **TACHADO**: Si se pasa S en <valor> el texto aparecerá tachado.
- **CURSIVA**: Si se pasa S en <valor> el texto aparecerá tachado.
- **TAMANO_FUENTE**: Tamaño de la fuente en unidades 1/20 de puntos, es decir, para un tamaño de 40 habría que pasar 800, es decir, 40 * 20
- **COLOR**: Color con el que se pintará el texto de la fuente. Consultar la tabla "Colores" en la sección "Constantes" para ver los colores disponibles.

Para crear un estilo se usará la función pkxls.crea_estilo(), esta función devuelve un NUMBER que identifica de forma única al estilo. Mediante el identificador se pueden asignar propiedades al estilo usando el procedimiento pkxls.set_propiedad_estilo(<id_estilo>, <propiedad>, <valor>). Los valores que puede tomar <propiedad> son los siguientes:

- **CODIGO**: Código único que se le asigna al estilo para poder ser localizado de forma más sencilla que por ID.
- **FUENTE**: Identificador de la fuente creada con pkxls.crea_fuente().
- **MASCARA_FORMATO**: Máscara de formato para campos numéricos, la máscara hay que especificarla en formato Excel, por ejemplo: '#,##0.00'
- **AJUSTAR_TEXTO**: Si es una columna de texto se puede hacer que el texto se ajuste a la columna, para ello hay que pasar S en <valor>.
- **PROTEGER**: Si se pasa N y a la hoja se le ha establecido una contraseña con PASSWORD_HOJA, las celdas que tengan el estilo podrán ser modificadas. Por defecto esta propiedad es S, por lo que si no se indica las celdas estarán protegidas en hojas con contraseña.

- **BORDE_SUPERIOR**: Permite pintar el borde superior de la celda, para ver los valores posibles consultar la tabla “Borde” en la sección “Constantes”.
- **BORDE_INFERIOR**: Igual que BORDE_SUPERIOR pero para el borde inferior.
- **BORDE_IZQUIERDO**: Igual que BORDE_SUPERIOR pero para el borde izquierdo.
- **BORDE_DERECHO**: Igual que BORDE_SUPERIOR pero para el borde derecho.
- **COLOR_FONDO**: Color de fondo con el que se pintará la celda. Consultar la tabla “Colores” en la sección “Constantes” para ver los colores disponibles.
- **ROTACION**: Ángulo de rotación del texto, por ejemplo 90 para ponerlo en vertical.
- **ALINEACION_HORIZONTAL**: Alineado horizontal del texto. Consultar la tabla “Alineación Horizontal” en la sección “Constantes” para ver las posibles alineaciones disponibles.
- **ALINEACION_VERTICAL**: Alineado vertical del texto. Consultar la tabla “Alineación Vertical” en la sección “Constantes” para ver las posibles alineaciones disponibles.

Hoja.

Se usará el procedimiento pkxls.set_propiedad_hoja(<id_hoja>, <propiedad>, <valor>). Es necesario pasar en el parámetro <id_hoja> el identificador devuelto por la función pkxls.crea_hoja. Los valores que puede tomar <propiedad> son los siguientes:

- **NOMBRE_HOJA**: Nombre de la pestaña de la hoja. Si no se especifica se le pone el texto Datos.
- **PASSWORD_HOJA**: Permite proteger la hoja contra modificaciones con la contraseña indicada en <valor>. A nivel de estilo, se puede indicar que a un determinado estilo de celdas no se aplique la protección.
- **APAISADO**: Si se pasa el valor S se configuran las propiedades de la hoja para impresión en apaisado. El valor por defecto es N.
- **TAMANO_PAPEL**: Tamaño de la hoja para la configuración de la impresión. Se le puede pasar los siguientes valores: PKXLS.LETTER_PAPERSIZE, PKXLS.LEGAL_PAPERSIZE, PKXLS.EXECUTIVE_PAPERSIZE, PKXLS.A4_PAPERSIZE (Valor por defecto), PKXLS.A5_PAPERSIZE, PKXLS.ENVELOPE_10_PAPERSIZE, PKXLS.ENVELOPE_DL_PAPERSIZE, PKXLS.ENVELOPE_CS_PAPERSIZE, PKXLS.ENVELOPE_MONARCH_PAPERSIZE
- **FILA_FIN_REP_TITULOS**: Número de filas de la hoja que se repiten en cada página en la impresión.
- **COLUMNA_BLOQUEO**: Indica hasta que columna se debe de bloquear la hoja al hacer scroll.
- **FILA_BLOQUEO**: Indica hasta que fila se debe de bloquear la hoja al hacer scroll.
- **AUTOFILTRO**: Permite indicar a nivel de hoja una zona de autofiltro. Las coordenadas se indicarán en forma de cadena de texto separada por comas de la siguiente forma: ‘fila inicial,columna inicial,fila final,columna final’. Por ejemplo '4,1,34,5' indicará que se aplique autofiltro desde la fila 4 a la 34 y desde la columna 1 a la 5.
- **ID_FUENTE_COMENTARIOS:** Código de la fuente obtenido con pkxls.crea_fuente a aplicar a los comentarios de las celdas de la hoja.
- **ANCHO_COMENTARIOS**: Ancho de los comentarios de las celdas por defecto en la hoja. El ancho se indica según el ancho de la celda a la que se le añade el comentario y del ancho de las celdas que se encuentran a su derecha, por tanto, si se indica 3, se tomará como ancho la suma del ancho de la celda a la que se la asigna el comentario más el ancho de las 2 celdas contiguas a la derecha.
- **ALTO_COMENTARIOS**: Alto de los comentarios de las celdas por defecto en la hoja. El alto se indica según el alto de la fila de la celda a la que se le añade el comentario y del alto de las filas de las celdas que se encuentran abajo, por tanto, si se indica 3, se tomará como alto la suma del alto de la fila de la celda a la que se la asigna el comentario más el alto de las 2 filas contiguas hacia abajo.
- **CLONAR_DE_ID_HOJA**: Creará la hoja partiendo de una copia de la indicada en <valor>. El <valor> se corresponde con el ID_HOJA que se quiere duplicar. **NOTA**: Esta propiedad únicamente se utiliza al modificar una hoja de cálculo, no se utiliza cuando se crea una hoja de cero.

Columna de hoja

Se usará el procedimiento pkxls.set_propiedad_columna_hoja(<id_hoja>, <numero_columna>, <propidad>, <valor>). Es necesario pasar en el parámetro <id_hoja> el identificador devuelto por la función pkxls.crea_hoja. Los valores que puede tomar <propiedad> son los siguientes:

- **ANCHO_COLUMNA**: Si se especifica un ancho determinado para una columna lleva implícito que se desactiva el dimensionado automático de la columna, el parámetro <valor> será un dato numérico, y se especifica en 1/256 partes de carácter, por lo que si queremos dar un ancho de 1 carácter habría que indicar un valor de 256, aunque esa medida es un poco relativa a la fuente.
- **DESACTIVAR_AUTOSIZE**: Por defecto una vez terminado de procesar el sql se ajusta el tamaño de las columnas al valor más largo, este funcionamiento por defecto se puede deshabilitar para una determinada columna pasando en <valor> 'S'.
- **FACTOR_AJUSTE_ANCHO**: Hay casos en los que no se puede hacer un autosize de las columnas (Forms 12, y cuando se limita el número de filas en memoria), la librería lo que hace es intentar un ajuste aproximado calculando un ancho sobre la celda en la que puso el valor con más caracteres multiplicando por 256, con esta propiedad se puede cambiar ese valor de ajuste.
- **COLUMNA_FIN_GRUPO**: Crea una agrupación de columnas entre la columna <numero_columna> y la columna especificada en <valor>. Por lo general esta funcionalidad se utiliza cuando hay por ejemplo la columna 8 con base, la columna 9 con importe impuesto y la 10 con el total, para que agrupe la 8 y la 9 y sólo muestre la 10 hay que ejecutar lo siguiente: pkxls.set_propiedad_columna_hoja(v_id_sql1,8, 'COLUMNA_FIN_GRUPO', 9);

SQL

Se usará el procedimiento pkxls.set_propiedad_sql(<id_sql>, <propiedad>, <valor>). Es necesario pasar en el parámetro <id_sql> el identificador devuelto por la función pkxls.crea_sql. Los valores que puede tomar <propiedad> son los siguientes:

- **SQL**: SELECT que se ejecutará para extraer los datos de la base de datos para ponerlos en la hoja de cálculo.
- **SQL_BIG**: Es excluyente con SQL. La propiedad SQL tiene la limitación de un tamaño de 32767 bytes, si la SQL es de mayor tamaño se tiene que usar SQL_BIG, de forma que la SQL se divida en partes y por cada parte se le pasará a la librería usando SQL_BIG, por lo tanto, SQL sólo se puede pasar una vez por cada SQL, pero SQL_BIG se puede llamar tantas veces como partes en las que se divida la SQL.
- **LIMITAR_NUMERO_REGISTROS**: Permite indicar el número de registros máximos a procesar, si la SQL devuelve más de los registros indicados serán ignorados.
- **ALTO_FILA_TITULOS**: Permite indicar en <valor> el alto en pixeles que debe de tener la fila de títulos asociada a la consulta SQL.
- **ALTO_FILA_CABECERA_TITULOS**: En el caso de definirse agrupación de títulos en cabecera, se permite indicar en <valor> el alto en pixeles para la final de agrupación.
- **ALTO_FILA**: Permite indicar en <valor> el alto en pixeles que deben de tener todas las filas asociadas a la consulta SQL.
- **SQL_TITULOS_COLUMNAS**: Por defecto al procesar una SQL lo primero que hace es poner los títulos de las columnas, para que no los ponga hay que pasar en <valor> 'N'.
- **SQL_ESTILO_TITULOS**: Indica el estilo de las celdas de título para la SQL. Hay que pasar en <valor> el valor devuelto por pkxls.crea_estilo(). Si se indica esta propiedad se ignoran las siguientes propiedades a nivel de archivo: SQL_TITULOS_NEGRITA, SQL_TITULOS_SUBRAYADO, SQL_TITULOS_BORDE_SUP, SQL_TITULOS_BORDE_INF, SQL_TITULOS_BORDE_DER, SQL_TITULOS_BORDE_IZQ, SQL_ROTACION_TITULO.
- **SQL_AGRUPAR_EN_PRIMERA_COLUMNA**: Si se usan agrupaciones indica como se realiza la ruptura. Los posibles valores son:
    - **N**: Es el valor por defecto. Cuando se cambia de nivel de agrupación las columnas de los niveles superiores se dejan en blanco

  - **S**: Todas las agrupaciones las hace en la primera columna dejando espacios según el nivel de agrupación.
  - **T**: Se hace la agrupación para la totalización, pero se muestran todos los datos, es decir, lo mismo que si no hubiese agrupación, pero con totalizaciones parciales.
- **SQL_TOTALIZAR**: Por defecto se generan totales de todos los campos numéricos, para desactivar este funcionamiento hay que pasar en <valor> 'N', de esa forma la sql no genera ningún tipo de total.
- **SQL_DESP_Y_INICIAL**: Con esta propiedad se puede indicar cuantas filas en blanco se dejarán antes de poner la primera fila generada por la sql, el valor por defecto es 0.
- **SQL_COLUMNA_SELECTOR_ESTILO**: Si se pasa 'S' en <valor> quiere decir que la última columna de la SQL es la que indica el estilo de las hojas, por lo que no será trasladada a la Excel. La columna deberá de ser de tipo texto y tendrá el siguiente formato:
  - columnaX:código de estilo|columnaY:código de estilo|...|columnaZ:código de estilo
  - **Ejemplo**: 0:ROJO|2:AZUL (A la columna 2 se le aplica el estilo con código AZUL y al resto de las columnas se le aplica el estilo con código ROJO.
- **SQL_AUTOFILTRO**: Pasando el valor 'S', se activa la funcionalidad de Filtros de la hoja de cálculo para el rango de filas que utiliza la SQL. **NOTA**: Si a nivel de hoja se ha definido la propiedad "**PASSWORD_HOJA**", el autofiltro únicamente se tendrá en cuenta cuando a nivel de archivo se indica 'S' en "**USAR_CONEXION_DIRECTA**" y no se indica "**NUMERO_FILAS_EN_MEMORIA**".

Grupos de títulos de columnas

Los grupos de títulos permiten añadir una fila anterior a la fila de títulos en la que se agrupan las columnas de una SQL con un texto. Cada agrupación puede tener un estilo diferente, en el caso de no indicar un estilo se utilizará el genérico para títulos.

Si las columnas con el mismo tipo son contiguas las agrupa y si hay columnas intermedias que no pertenezcan al grupo se generarán grupos adicionales.

Se usará el procedimiento pkxls.set_propiedad_grupo_col_sql (<id_grupo>, <propiedad>, <valor>). Es necesario pasar en el parámetro <id_grupo> el identificador devuelto por la función pkxls.crea_grupo_columnas_sql, este indientifcador habrá que indicarlo a cada columna de la SQL que vaya a pertenecer al grupo. Los valores que puede tomar <propiedad> son los siguientes:

- **TITULO**: Texto que se va a visualizar al grupo.
- **ESTILO**: Identificador del estilo visual a aplicar al grupo. Ver "Fuentes y Estilos".

Columna de la SQL

Se usará el procedimiento pkxls.set_propiedad_columna(<id_sql>, <numero_columna>, <propiedad>, <valor>). Es necesario pasar en el parámetro <id_sql> el identificador devuelto por la función pkxls.crea_sql.

El parámetro <numero_columna> identifica el número del campo que saca la select comenzando a contar en 1.

Los valores que puede tomar <propiedad> son los siguientes:

- **SQL_TITULO_COLUMNA**: Por defecto se pone el título que se extrae de la sql, pero si se especifica esta propiedad prevalece sobre el alias que tenga el campo en la sql.
- **SQL_COMENTARIO_TITULO_COLUMNA**: Comentario a incluir en la celda que tenga el título de la columna en la hoja de cálculo.
- **SQL_ROTACION_TITULO**: Ángulo de rotación del título, por ejemplo 90 para ponerlo en vertical.
- **SQL_PONER_TITULO**: Si a nivel de SQL no se ha desactivado la impresión de los títulos se puede deshabilitar para un campo en concreto pasando en <valor> 'N'.

- **SQL_AJUSTAR_TEXTO**: Si es una columna de texto se puede hacer que el texto se ajuste a la columna, para ello hay que pasar S en <valor>.
- **SQL_DESACTIVAR_AUTOSIZE (OBSOLETO: Se debería usar la propiedad ANCHO_COLUMNA de PKXLS.SET_PROPIEDAD_COLUMNA_HOJA)**: Por defecto una vez terminado de procesar el SQL se ajusta el tamaño de las columnas al valor más largo, este funcionamiento por defecto se puede deshabilitar para una determinada columna pasando en <valor> 'S'.
- **SQL_DESACTIVAR_TOTALIZACION**: Todas las columnas numéricas se totalizan, se puede deshabilitar para una determinada columna pasando en <valor> 'S'.
- **SQL_DESACTIVAR_SUBTOTAL**: Todas las columnas numéricas se subtotalizan. Se puede deshabilitar la subtotalización para una determinada columna pasando en <valor> 'S'.
- **SQL_ANCHO_COLUMNA (OBSOLETO: Se debería usar la propiedad ANCHO_COLUMNA de PKXLS.SET_PROPIEDAD_COLUMNA_HOJA)**: Si se especifica un ancho determinado para una columna lleva implícito que se desactiva el dimensionado automático de la columna, el parámetro <valor> será un dato numérico, y se especifica en 1/256 partes de carácter, por lo que si queremos dar un ancho de 1 carácter habría que indicar un valor de 256, aunque esa medida es un poco relativa a la fuente.
- **SQL_FIN_GRUPO**: Se pueden generar sqls con agrupaciones, en este caso hay que indicar qué campos son los que finalizan un grupo, es importante el orden de los campos de la SQL, debiendo mantener el orden de las agrupaciones, además es necesario que la SQL lleve un ORDER BY por todos los campos por los que se agrupe. A mayores se incluyen opciones para generar las agrupaciones con modalidad colapsada, mostrando inicialmente solamente las filas de totales/subtotales para dicha agrupación. Puede contener los siguientes valores:
    - **N**: Valor por defecto, el campo no es fin de agrupación.
    - **S**: El campo es fin de una agrupación y se generan totales.
    - **V**: El campo es fin de una agrupación y se generan totales y en el total se concatena el
    - valor de la agrupación que se está totalizando.
    - **NT**: El campo es fin de una agrupación y para esa agrupación no se generan totales.
    - **SC**: Mismo caso que “S” con modo colapsado.
    - **VC**: Mismo caso que “V” con modo colapsado.
- **SQL_MASCARA_FORMATO**: Máscara de formato de campos numéricos. Hay que indicarla la en formato Excel, ejemplo: '#,##0.00'
- **SQL_TIPO**: Tipo de la columna. Puede contener los siguientes valores:
    - **C**: Campo normal.
    - **F**: Campo con fórmula de totalización. En la sql podemos sacar una columna alfanumérica con una fórmula, al pasarle este valor se le indica a Excel que la trate como una fórmula y no como un texto normal, ver apartado: Variables disponibles en fórmulas.
- **SQL_FORMULA_TOTAL**: Formula a aplicar cuando el campo es numérico y se está totalizando, si no se especifica una fórmula usará la fórmula de totalización por defecto por defecto que es: SUBTOTAL(9;<columna><fila_inicial>:<columna><fila_final>)
- **SQL_FORMULA_TOTAL_ULT_NIVEL**: Formula a aplicar cuando el campo es numérico y se está totalizando y aparte en caso de haber agrupaciones esta será la fórmula aplicar en el nivel más bajo, es decir, se opera directamente con datos y no con resultado de otras fórmulas, si no se especifica una fórmula usará la fórmula de totalización por defecto: SUM(<columna><fila_inicial>:<columna><fila_final>)
- **SQL_FORZAR_SOLO_PRIMERA_RUPTURA**: Únicamente tiene sentido cuando a nivel de la SQL se ha indicado SQL_AGRUPAR_EN_PRIMERA_COLUMNA = T. De esta forma se puede indicar qué columnas únicamente se deben de imprimir una única vez en cada ruptura.
- **SQL_ESTILO_TITULO**: Indica el estilo de la celda de título. Hay que pasar en <valor> el valor devuelto por pkxls.crea_estilo(). Si se indica esta propiedad se ignoran las siguientes propiedades a nivel de archivo: SQL_TITULOS_NEGRITA, SQL_TITULOS_SUBRAYADO, SQL_TITULOS_BORDE_SUP,

SQL_TITULOS_BORDE_INF, SQL_TITULOS_BORDE_DER, SQL_TITULOS_BORDE_IZQ, SQL_ROTACION_TITULO y la propiedad SQL_ESTILO_TITULOS a nivel de SQL.

- **SQL_IMAGEN**: Indica si el campo es de tipo BLOB y contiene una imagen que debe de ser exportada a la hoja de cálculo. Hay que pasar en <valor> 'S' para activarlo. **NOTA**: Requiere que se establezca conexión directa.
- **SQL_FORZAR_ALTO_IMAGEN**: En el caso de haber indicado **SQL_IMAGEN** con el valor S, se puede forzar que se ajuste el alto de la imagen al valor indicado en Pixeles en <valor>.
- **SQL_TIPO_AJUSTE_IMAGEN**: Si se indicó **SQL_IMAGEN** con el valor S, se puede indicar como debe de comportarse la imagen al redimensionar o mover las celdas, en <valor> se puede indicar uno de estos tres valores:
  - **DONT_MOVE_AND_RESIZE**: No se mueve ni se redimensiona.
  - **MOVE_AND_RESIZE**: Se mueve y redimensiona con las celdas.
  - **MOVE_DONT_RESIZE**: Se mueve con las celdas, pero no se redimensiona la imagen en el caso de que cambie de tamaño la celda.
- **ID_GRUPO**: Identificador del grupo de la agrupación de títulos de columnas a la que pertenece la columna.

A nivel de fórmula.

Se usará el procedimiento pkxls.set_propiedad_formula_total(<id_formula>, <propiedad>, <valor>). Es necesario pasar en el parámetro <id_formula> el identificador devuelto por la función pkxls.crea_formula_total_sql(<id_sql>). Los valores que puede tomar <propiedad> son los siguientes:

- **FORMULA**: Texto de la fórmula. Ver apartado: Variables disponibles en fórmulas.
- **FORMULA_ULTIMO_NIVEL**: En caso de haber agrupaciones será la fórmula que se aplica en el nivel más bajo, cuando tiene que operar directamente con datos y no con el resultado de otras fórmulas. Ver apartado: Variables disponibles en fórmulas.
- **IMPRIMIR_EN_AGRUPACION**: Por defecto cuando se crea una formula se va a imprimir en los totales del informe y en las agrupaciones, para desactivar una fórmula en las agrupaciones hay que pasar 'N' en <valor>.
- **IMPRIMIR_EN_TOTAL**: Por defecto cuando se crea una formula se va a imprimir en los totales del informe y en las agrupaciones, para desactivar una fórmula en los totales del informe hay que pasar 'N' en <valor>.
- **MASCARA_FORMATO**: Por defecto cuando se totaliza una columna numérica se usa la misma máscara que la que hay especificada para la columna, para cambiar ese criterio para una determinada fórmula hay que pasar en <valor> la máscara de formato a aplicar.
- **TITULO**: Por defecto se le pone como título a la fila de totalización el título de la columna, para cambiar el título para una determinada fórmula hay que pasarlo en <valor>.

A nivel de columna de fórmula.

Se usará el procedimiento pkxls.set_propiedad_columna_total(<id_formula>, <numero_columna>, <propiedad>, <valor>). Es necesario pasar en el parámetro <id_formula> el identificador devuelto por la función pkxls.crea_formula_total_sql. El parámetro <numero_columna> identifica el número del campo que saca la SELECT comenzando a contar en 1.

Los valores que puede tomar <propiedad> son los siguientes:

- **FORMULA**: Fórmula a aplicar cuando el campo es numérico y se está totalizando, en caso de no especificarse se usará la fórmula indicada en pkxls.set_propiedad_formula_total. Ver apartado: Variables disponibles en fórmulas. Si se le pasa el valor STD se aplica la fórmula estándar.
- **FORMULA_ULTIMO_NIVEL**: En caso de haber agrupaciones será la fórmula que se aplica en el nivel más bajo, cuando tiene que operar directamente con datos y no con el resultado de otras fórmulas. En caso de no especificarse se usará la fórmula indicada en pkxls.set_propiedad_formula_total. Ver apartado: Variables disponibles en fórmulas. Si se pasa el valor STD se aplica la fórmula estándar de último nivel.

- **MASCARA_FORMATO**: Por defecto cuando se totaliza una columna numérica se usa la misma máscara que la que hay especificada para la columna, para cambiar ese criterio para una determinada fórmula hay que pasar en <valor> la máscara de formato a aplicar.

Variables disponibles en fórmulas.

Para construir las fórmulas de forma dinámica haciendo disponemos de las siguientes variables que serán sustituidas antes de ser asignadas como fórmula a la celda:

- **<fila_inicial>**: Número de fila en la que comienza el grupo. Variable alternativa: **<fini>**
- **<fila_final>:** Número de fila en la que termina el grupo. Variable alternativa: **<ffin>**
- **<fila>**: Número de fila en la que va a poner la fórmula. Variable alternativa: **<fl>**
- **<columna>**: Columna en que se va a poner la fórmula. Variable alternativa **<cl>**
- **<codigo_ruptura>**: Código de la ruptura que se está totalizando, el código de ruptura se almacena en la columna IV. Variable alternativa **<cr>**

Se pueden usar operaciones para desplazar el valor de una variable, por ejemplo, si queremos coger la columna anterior a la en que se va a poner la fórmula se pondría <columna-1>.

Fórmulas matriciales.

Se puede indicar que la fórmula es de tipo matricial (el equivalente en Excel a validar la fórmula con Control + Mayúsculas + INTRO) metiendo la fórmula entre llaves "{" y "}"., Por ejemplo: {SUM(<columna-2><fila-2>:<columna-2><fila-1>*<columna-1><fila-2>:<columna-1><fila-1>)}

Si la fórmula matricial tiene salida a un rango de celdas, este rango de celdas se puede indicar al principio metiendo el rango también entre llaves "{" y "}". Ejemplo: {**{D3:D5}**SUM(<columna-2><fila-2>:<columna-2><fila-1>*<columna-1><fila-2>:<columna-1><fila-1>)}

IMPORTANTE: Las fórmulas matriciales no se pueden utilizar cuando se indica USAR_CONEXION_DIRECTA junto a un número de registros en memoria.

Ejemplo 1:

```
DECLARE
  v_id_hoja       NUMBER;
BEGIN
  pkxls.inicializa;
  pkxls.set_propiedad_excel('DIRECTORIO', 'c:\temp');
  pkxls.set_propiedad_excel('ARCHIVO', 'prueba.xls');
  v_id_hoja := pkxls.crea_hoja();
  pkxls.excel_celda(v_id_hoja, 0, 0, 'N', 'P1', 10);
  pkxls.excel_celda(v_id_hoja, 1, 0, 'N', 26, 10);
  pkxls.excel_celda(v_id_hoja, 2, 0, 'N', 256, 10);
  pkxls.excel_celda(v_id_hoja, 0, 1, 'N', 'P2', 10);
  pkxls.excel_celda(v_id_hoja, 1, 1, 'N', 22, 10);
  pkxls.excel_celda(v_id_hoja, 2, 1, 'N', 233, 10);
  pkxls.excel_celda(v_id_hoja,  3,  2,  'F',  '{{D3:D5}SUM(<columna-2><fila-2>:<columna-2><fila-1>*<columna-
1><fila-2>:<columna-1><fila-1>)}', 10);
  pkxls.generar_xls();
END;
```

Ejemplo 2:

```
DECLARE
  v_id_hoja    PLS_INTEGER;
  v_id_sql     PLS_INTEGER;
  v_id_formula PLS_INTEGER;
BEGIN
  pkxls.inicializa;
  pkxls.set_propiedad_excel('DIRECTORIO','c:\temp');
  pkxls.set_propiedad_excel('ARCHIVO', 'prueba.xls');
  v_id_hoja := pkxls.crea_hoja();
  v_id_sql := pkxls.crea_sql(v_id_hoja);
  pkxls.set_propiedad_sql(v_id_sql,'SQL','SELECT articulo, uni_seralm, precio_presentacion
  FROM albaran_ventas_lin
 WHERE rownum <= 100');
  pkxls.set_propiedad_columna(v_id_sql, 2, 'SQL_DESACTIVAR_TOTALIZACION', 'S');
  v_id_formula := pkxls.crea_formula_total_sql(v_id_sql);
  pkxls.set_propiedad_columna_total(v_id_formula,       3,       'FORMULA_ULTIMO_NIVEL',       '{SUM(<columna-
1><fila_inicial>:<columna-1><fila_final>*<columna><fila_inicial>:<columna><fila_final>)}');
  pkxls.generar_xls();
END;
```

Asignar valores sin ser obtenidos de una SQL a determinadas celdas.

Hay un método complementario a la opción de cubrir la Excel usando datos extraídos de una SQL, este método permite ir indicando las posiciones x e y de la celda y asignarle un valor. Se puede dar valor mediante este método a tantas celdas como sea necesario.

Se usará el procedimiento: pkxls.excel_celda(<id_hoja>, <x>, <y>, <tipo>, <valor>, <posicion>, <estilo>);

- **<id_hoja>**: Identifica la hoja en que se va a crear la celda, será el identificador de la hoja que se ha obtenido con pkxls.crea_hoja.
- **<x>**: Coordenada x de la celda en la hoja, la primera celda es la 0.
- **<y>**: Coordenada y de la celda en la hoja, la primera celda es la 0.
- **<tipo>**: Indica el formato que va a tener la celda. Puede contener los siguientes valores:
  - **C**: Indica que es un campo de datos normal.
  - **T**: Indica que es un campo de título.
  - **F**: Indica que es un campo que debe de evaluarse como fórmula. Se pueden usar las etiquetas <fila> y <columna>.
  - **I**: Imagen. En <valor> se ha de indiciar una SQL que obtenga una única fila con un campo de tipo BLOB que contenga la imagen a mostrar. La SQL puede tener a mayores del campo BLOB campos que indiquen como se ha de ajustar la imagen en la hoja de cálculo, estos campos deben tener los siguientes alias:
    - **FORZAR_ALTO_IMAGEN**: Campo numérico que indique el alto que debe de tener la imagen. El ancho se ajustará proporcionalmente.
    - **TIPO_AJUSTE_IMAGEN**: Campo alfanumérico que indica como debe de moverse la imagen en el caso de que se redimensione o mueva la fila o columna a la que está asociada. Puede contener los siguientes valores:
      - **DONT_MOVE_AND_RESIZE**: No se mueve ni se redimensiona.
      - **MOVE_AND_RESIZE**: Se mueve y redimensiona con las celdas.
      - **MOVE_DONT_RESIZE**: Se mueve con las celdas, pero no se redimensiona la imagen en el caso de que cambie de tamaño la celda.

OBSERVACIONES:

La generación con imágenes únicamente está disponible cuando se utiliza la propiedad USAR_CONEXION_DIRECTA con el valor S.

Una imagen queda anclada a una serie de columnas por lo que puede que se distorsione en el momento de ajustar el ancho a las columnas. Para evitarlo hay que desactivar el autosize de las columnas de la hoja de cálculo o asignar un ancho fijo a las columnas para que se ajusten al principio de la generación y ya quede la imagen anclada con el ancho correcto. Ver propiedades a nivel de columna de hoja.

- **TIPO_AJUSTE_FILA_COLUMNA**: Campo alfanumérico que indica como debe de comportarse la fila y columna a la que se asocia la imagen:
  - **OVERLAY_ROW_AND_COLUMN**: La fila y columna no se adapta a la imagen, por lo que si la imagen es mayor no afecta al tamaño que tenga la fila y columna y simplemente ocupará las filas y columnas contiguas necesarias para mostrarse.
  - **EXPAND_ROW_AND_COLUMN**: Ampliar el tamaño de la fila y columna lo necesario para que entre la imagen.
  - **EXPAND_COLUMN**: Ampliar el tamaño de la columna lo necesario para que entre la imagen. Si el alto de la fila es menor que el alto de la imagen ocupará las filas contiguas necesarias para mostrarse.
  - **EXPAND_ROW**: Ampliar el tamaño de la fila lo necesario para que entre la imagen. Si el ancho de la columna es menor que el ancho de la imagen ocupará las columnas contiguas necesarias para mostrarse.

EJEMPLO: SELECT imagen_report, 50 forzar_alto_imagen, 'DONT_MOVE_AND_RESIZE' tipo_ajuste_imagen, 'OVERLAY_ROW_AND_COLUMN' tipo_ajuste_fila_columna FROM empresas_logo WHERE codigo_empresa = '013'

- **<valor>**: Contenido de la celda, puede ser numérico o alfanumérico.
- **<posicion>**: Indica en que momento se va a escribir en la celda, hay 5 posibilidades:
  - **10**: Escribe antes de procesar las sqls que tiene asignadas la hoja.
  - **20**: Escribe después de procesar las sqls que tiene asignadas la hoja, pero antes de hacer el autosize de las columnas, por lo que el tamaño de las celdas se verá afectado por el contenido asignado por este procedimiento.
  - **25**: Igual que el 20 pero el valor de la coordenada y es relativa a la última fila impresa en la hoja.
  - **30**: Se escribe después de procesar las sqls que tiene asignadas la hoja y después de haberse ejecutado el autosize, por lo que los valores asignados por este procedimiento no afectan al autosize.
  - **35**: Igual que el 30 pero el valor de la coordenada y es relativa a la última fila impresa en la hoja.
- **<estilo>**: Código de estilo a aplicar a la celda. (ver sección "Fuentes y Estilos").

También se pueden indicar valores a celdas siendo las coordenada <y> en relación a una determinada SQL (desde entorno 6.2.5), para ello se utilizará el procedimiento pkxls.excel_celda_sql(<id_sql>, <x>, <y>, <tipo>, <valor>, <posición>, <estilo>);

- **<id_sql>:** Identifica la SQL sobre la que debe ser relativa la coordenada <y>.
- **<x>**: Coordenada x de la celda de la hoja, la primera celda es la 0.
- **<y>**: Coordenada relativa al inicio o fin de los valores de la SQL.
- **<tipo>**: Igual que en pkxls.excel_celda.
- **<valor>**: Igual que en pkxls.excel_celda.
- **<posición>**: Indica en qué momento se va a escribir la celda.
  - **15**: Antes de comenzar a procesar la SQL, por tanto, se escribirá al principio de los valores de la SQL.
  - **25**: Después de procesar la SQL, por tanto, se escribirá al final de los valores de la SQL.
- **<estilo>:** Igual que en pkxls.excel_celda.

Ejemplo:

```
DECLARE
  v_id_hoja       NUMBER;
BEGIN
  pkxls.inicializa;
  pkxls.set_propiedad_excel('DIRECTORIO', 'c:\temp');
  pkxls.set_propiedad_excel('ARCHIVO', 'prueba.xls');
  v_id_hoja := pkxls.crea_hoja();
  pkxls.excel_celda(v_id_hoja, 0, 0, 'T', 'Valor 1', 10);
  pkxls.excel_celda(v_id_hoja, 1, 0, 'T', 'Valor 2', 10);
  pkxls.excel_celda(v_id_hoja, 2, 0, 'T', 'Total', 10);
  pkxls.excel_celda(v_id_hoja, 0, 1, 'N', 100, 10);
  pkxls.excel_celda(v_id_hoja, 1, 1, 'N', 200, 10);
  pkxls.excel_celda(v_id_hoja, 2, 1, 'F', 'SUM(<columna-2><fila>:<columna-1><fila>)', 10);
  pkxls.generar_xls();
END;
```

Ejemplo con pkxls.excel_celda_sql:

```
PROCEDURE test IS
  v_id_hoja PLS_INTEGER;
  v_id_sql1 PLS_INTEGER;
  v_id_sql2 PLS_INTEGER;
BEGIN
  pkxls.inicializa(TRUE);
  pkxls.set_propiedad_excel('DIRECTORIO','c:\temp');
  pkxls.set_propiedad_excel('ARCHIVO', 'prueba.xls');
  v_id_hoja := pkxls.crea_hoja();
  v_id_sql1 := pkxls.crea_sql(v_id_hoja);
  pkxls.set_propiedad_sql(v_id_sql1,'SQL','SELECT * FROM DIARIOS');
  v_id_sql2 := pkxls.crea_sql(v_id_hoja);
  pkxls.set_propiedad_sql(v_id_sql2,'SQL','SELECT * FROM DIARIOS');
  pkxls.excel_celda_sql(v_id_sql1,0,0,'C','TEXTO FIJO AL PRINCIPIO DE LA PRIMERA SQL', 15);
  pkxls.excel_celda_sql(v_id_sql2,0,0,'C','TEXTO_FIJO AL PRINCIPIO DE LA SEGUNDA SQL', 15);
  pkxls.excel_celda_sql(v_id_sql1,0,0,'C','TEXTO FIJO AL FINAL DE LA PRIMERA SQL', 25);
  pkxls.excel_celda_sql(v_id_sql2,0,0,'C','TEXTO FIJO AL FINAL DE LA SEGUNDA SQL', 25);
  pkxls.excel_celda(v_id_hoja,0,0,'C','TEXTO FIJO AL FINAL DE TODAS LAS SQLS',25);
  pkxls.generar_xls();
END;
```

Combinar celdas

Se pueden realizar agrupaciones de celdas usando la función (devuelve un número que identifica la agrupación) pkxls.pkxls.crea_region(<id_hoja>, <fila_inicial>, <columna_inicial>, <fila_final>, <columna_final>);

- **<id_hoja>**: Identifica la hoja en la que se van a combinar las celdas.
- **<fila_inicial>**: Número de fila en la que se encuentra la celda superior izquierda de la agrupación. La primera fila tiene el número 0.
- **<columna_inicial>**: Número de columna en la que se encuentra la celda superior izquierda de la agrupación. La primera columna tiene el número 0.
- **<fila_final>**: Número de fila en la que se encuentra la celda inferior derecha de la agrupación.
- **<columna_final>**: Número de columna en la que se encuentra la celda inferior derecha de la agrupación.

Ejemplo:

```
DECLARE
  v_id_hoja           PLS_INTEGER;
  v_id_region         PLS_INTEGER;
  v_id_fuente         NUMBER;
  v_id_estilo_sup     NUMBER;
  v_id_estilo_inf     NUMBER;
BEGIN
  pkxls.inicializa;
  pkxls.set_propiedad_excel('DIRECTORIO', 'c:\temp');
  pkxls.set_propiedad_excel('ARCHIVO', 'prueba.xls');
  v_id_hoja := pkxls.crea_hoja();
  v_id_fuente := pkxls.crea_fuente();
  pkxls.set_propiedad_fuente(v_id_fuente, 'NEGRITA', pkxls.boldweight_bold);
  v_id_estilo_sup := pkxls.crea_estilo();
  pkxls.set_propiedad_estilo(v_id_estilo_sup, 'FUENTE', v_id_fuente);
  pkxls.set_propiedad_estilo(v_id_estilo_sup, 'ALINEACION_HORIZONTAL', PKXLS.ALIGN_CENTER);
  pkxls.set_propiedad_estilo(v_id_estilo_sup, 'BORDE_SUPERIOR', PKXLS.BORDER_THIN);
  pkxls.set_propiedad_estilo(v_id_estilo_sup, 'BORDE_IZQUIERDO', PKXLS.BORDER_THIN);
  pkxls.set_propiedad_estilo(v_id_estilo_sup, 'BORDE_DERECHO', PKXLS.BORDER_THIN);
  v_id_estilo_inf := pkxls.crea_estilo();
  pkxls.set_propiedad_estilo(v_id_estilo_inf, 'FUENTE', v_id_fuente);
  pkxls.set_propiedad_estilo(v_id_estilo_inf, 'ALINEACION_HORIZONTAL', PKXLS.ALIGN_CENTER);
  pkxls.set_propiedad_estilo(v_id_estilo_inf, 'BORDE_INFERIOR', PKXLS.BORDER_THIN);
  pkxls.set_propiedad_estilo(v_id_estilo_inf, 'BORDE_IZQUIERDO', PKXLS.BORDER_THIN);
  pkxls.set_propiedad_estilo(v_id_estilo_inf, 'BORDE_DERECHO', PKXLS.BORDER_THIN);
  v_id_region := pkxls.crea_region(v_id_hoja, 1, 1, 1, 2);
  v_id_region := pkxls.crea_region(v_id_hoja, 2, 1, 2, 2);
  pkxls.excel_celda(v_id_hoja, 1, 1, 'C', 'FILA 1', 30, v_id_estilo_sup);
  pkxls.excel_celda(v_id_hoja, 2, 1, 'C', '', 30, v_id_estilo_sup);
  pkxls.excel_celda(v_id_hoja, 1, 2, 'C', 'FILA 2', 30, v_id_estilo_inf);
  pkxls.excel_celda(v_id_hoja, 2, 2, 'C', '', 30, v_id_estilo_inf);
  pkxls.generar_xls();
END;
```

Constantes

Colores

Los colores se pueden indicar de varias formas (**NOTA**: En formato XLSX se aplicará el color exacto indicado, pero en formato XLS se aplicará el que más se aproxime dentro de la paleta de colores disponible):

- Código de color definido en la vista V_COLORES_ERP. Ejemplo: RED_300
- Formato hexadecimal de la forma #RRGGBB, ejemplo: #E57373
- Formato RGB, rNNNgNNNbNNN, ejemplo: r229g115b115
- Número de color de la siguiente tabla de colores prefijados.

Código	*Color*	*Código*	*Color*	*Código*	*Color*
64	AUTOMATIC	49	AQUA	45	ROSE
8	BLACK	12	BLUE	57	SEA_GREEN
54	BLUE_GREY	11	BRIGHT_GREEN	47	TAN
60	BROWN	29	CORAL	15	TURQUOISE
24	CORNFLOWER_BLUE	18	DARK_BLUE	9	WHITE
58	DARK_GREEN	16	DARK_RED	30	ROYAL_BLUE
56	DARK_TEAL	19	DARK_YELLOW	40	SKY_BLUE
51	GOLD	17	GREEN	21	TEAL
22	GREY_25_PERCENT	55	GREY_40_PERCENT	20	VIOLET
23	GREY_50_PERCENT	63	GREY_80_PERCENT	13	YELLOW
62	INDIGO	46	LAVENDER	44	PALE_BLUE
26	LEMON_CHIFFON	48	LIGHT_BLUE	61	PLUM
31	LIGHT_CORNFLOWER_BLUE	42	LIGHT_GREEN	14	PINK
52	LIGHT_ORANGE	27	LIGHT_TURQUOISE	10	RED
43	LIGHT_YELLOW	50	LIME	53	ORANGE
25	MAROON	59	OLIVE_GREEN	28	ORCHID

Negrita

Código	Descripción
PKXLS.BOLDWEIGHT_BOLD	Activa Negrita
PKXLS.BOLDWEIGHT_NORMAL	Desactiva negrita

Subrayado

Código	Descripción
PKXLS.U_NONE	Sin subrayado
PKXLS.U_SINGLE	Subrayado simple
PKXLS.U_DOUBLE	Subrayado doble

Borde

Código	Descripción
PKXLS.BORDER_NONE	Sin borde
PKXLS.BORDER_THIN	
PKXLS.BORDER_MEDIUM	
PKXLS.BORDER_DASHED	
PKXLS.BORDER_HAIR	
PKXLS.BORDER_THICK	
PKXLS.BORDER_DOUBLE	
PKXLS.BORDER_DOTTED	
PKXLS.BORDER_MEDIUM_DASHED	
PKXLS.BORDER_DASH_DOT	
PKXLS.BORDER_MEDIUM_DASH_DOT	
PKXLS.BORDER_DASH_DOT_DOT	
PKXLS.BORDER_SLANTED_DASH_DOT	

Alineación Horizontal

Código	Descripción
PKXLS.ALIGN_GENERAL	Normal (horizontal)
PKXLS.ALIGN_LEFT	Izquierda
PKXLS.ALIGN_CENTER	Centrado
PKXLS.ALIGN_RIGHT	Derecho

Alineación Vertical

Código	Descripción
PKXLS.VERTICAL_TOP	Arriba
PKXLS.VERTICAL_CENTER	Centrado
PKXLS.VERTICAL_BOTTOM	Abajo

Ejemplo usando estilos y fuentes:

```
DECLARE
  v_id_hoja           NUMBER;
  v_id_sql            NUMBER;
  v_id_fuente         NUMBER;
  v_id_estilo2        NUMBER;
  v_id_estilo1        NUMBER;
BEGIN
  pkxls.inicializa;
  pkxls.set_propiedad_excel('DIRECTORIO', 'c:\temp');
  pkxls.set_propiedad_excel('ARCHIVO', 'prueba.xls');

  -- Se inicializan estilos
  v_id_estilo1 := pkxls.crea_estilo();
  pkxls.set_propiedad_estilo(v_id_estilo1, 'COLOR_FONDO', 10);
  pkxls.set_propiedad_estilo(v_id_estilo1, 'CODIGO', 'ROJO');
  v_id_fuente := pkxls.crea_fuente();
  pkxls.set_propiedad_fuente(v_id_fuente, 'NEGRITA', pkxls.boldweight_bold);
  pkxls.set_propiedad_fuente(v_id_fuente, 'SUBRAYADO', pkxls.u_single);

  v_id_estilo2 := pkxls.crea_estilo();
  pkxls.set_propiedad_estilo(v_id_estilo2, 'COLOR_FONDO', 13);
  pkxls.set_propiedad_estilo(v_id_estilo2, 'FUENTE', v_id_fuente);
  pkxls.set_propiedad_estilo(v_id_estilo2, 'CODIGO', 'AMARILLO');
  pkxls.set_propiedad_excel('ABRIR_EXCEL', 'S');
  v_id_hoja := pkxls.crea_hoja();

  -- Se asigna la sql a la hoja y se indica que la última columna tiene la información de estilos
  v_id_sql := pkxls.crea_sql(v_id_hoja);
  pkxls.set_propiedad_sql(v_id_sql,  'SQL',  'SELECT  codigo,  nombre,  apertura_cierre,  DECODE(apertura_cierre,  ''A'',
''0:ROJO'',''R'', ''0:AMARILLO|2:ROJO'') color FROM diarios');
  pkxls.set_propiedad_sql(v_id_sql, 'SQL_COLUMNA_SELECTOR_ESTILO', 'S');

  -- Se crean dos celdas con valores fijos con estilos
  pkxls.excel_celda(v_id_hoja, 0, 1, 'C', 'CELDA EN ROJO', 25, v_id_estilo1);
  pkxls.excel_celda(v_id_hoja, 1, 1, 'C', 'CELDA EN AMARILLO', 25, v_id_estilo2);
  pkxls.generar_xls();
END;
```

Resultado: (Si tiene A en la columna APERTURA_CIERRE pinta el registro de rojo y si tiene R pinta el registro de amarillo, pero la columna 2 de rojo).

	A	B	C
1	CODIGO	NOMBRE	APERTURA_CIERRE
2	APER	APERTURA	A
3	CIER	CIERRE	C
4	DETE	DETERMINACION DE RESULTADOS	R
5	VARI	OPERACIONES_VARIAS	N
6			
7	CELDA EN ROJO	CELDA EN AMARILLO	

Preparar en base de datos y ejecutar en Forms

Se puede hacer todo el proceso en código de base de datos llamando a pkxlsbd en vez de pkxls y luego hacer la ejecución en Forms, de esta forma el código queda compatible para ser ejecutado en base de datos con GAL_EXCEL o directamente en el programa de Forms. Una vez se han establecido las propiedades y antes de llamar al procedimiento generar_excel hay que recuperar la configuración de base de datos mediante pkxls.recupera_configuracion_pkxlsbd(); y luego ya se podría llamar a pkxls.generar_xls();

Hoja de Cálculo Simple

Se puede realizar una hoja de cálculo muy simple que únicamente tenga una SQL y no sea necesario especificar ningún tipo de propiedad a las columnas llamando directamente a PKXLS.SQL_SIMPLE('<sql>');

Ejemplo:

```
pkxls.sql_simple('SELECT codigo "Código", nombre "Nombre", apertura_cierre "Tipo" FROM diarios');
```

Lectura de hojas de cálculo

Para leer desde un programa de Forms un archivo de hoja de cálculo, habrá que incluir la librería "pklibxls.pll".

Para obtener los datos almacenados en las celdas se usará la función "pkxls.cargar_hoja_calculo (p_archivo, p_numero_hoja, p_ignorar_celdas_vacias, p_permitir_conexion_directa, p_en_ias, p_inicializar_hojas)".

- **p_archivo:** Ruta completa al archivo a cargar. Si la ejecución es en Forms 12c es importante el parámetro p_en_ias para indicar dónde se encuentra el archivo.
- **p_numero_hoja**: Número de hoja a cargar, si se pasa a NULL se cargarán todas.
- **p_ignorar_celdas_vacias**: Si se pasa 'S' las celdas vacías serán ignoradas, pero si se pasa 'N' se cargarán con valor NULL.
- **p_permitir_conexion_directa**: Puede recibir los siguientes valores:
    - **N:** No se utiliza ninguna tabla temporal en base de datos, el archivo se lee en memoria y se devuelven directamente los valores de las celdas. Este método es el ideal para hojas de cálculo pequeñas.
    - **S:** Se intentará hacer la carga a través de una conexión directa de Java a la base de datos, para archivos muy grandes es la opción más rápida.
    - **R:** Igual que "S" pero es exclusivo para archivos XLSX y la velocidad de carga es infinitamente más rápida con archivos muy grandes. Este método tiene las siguientes limitaciones:
        - No se evalúan las fórmulas antes de hacer la lectura.
        - Los campos numéricos se devuelven siempre como carácter y no recupera las fórmulas de las celdas.
        - Los valores no se pueden obtener con el método "pkxlsbd.get_tabla_excel()", hay que usar siempre la consulta SELECT con la tabla "TABLE(pkxlsbd.get_pipe_tabla_excel())"
- **p_en_ias**: Si se está ejecutando en Forms 12c indica en dónde hace referencia el parámetro "p_archivo", si en p_en_ias se pasa el valor de TRUE el archivo debe de encontrarse en el servidor de aplicaciones, si se pasa FALSE el archivo debe de encontrase en el equipo del usuario y para ser procesado tiene que internamente cargarse en el servidor de aplicaciones para poder ser leído.
- **p_inicializar_hojas**: Si se pasa TRUE en este parámetro, por cada hoja leída llamará a pkxls.crea_hoja y a pkxls.set_propiedad_hoja con la propiedad 'NOMBRE_HOJA'.

Esta función carga el archivo en la base de datos y devuelve un VARCHAR2 con el resultado de la lectura, siendo OK que la lectura ha sido correcta y ERROR en caso de no poderse realizar la carga.

Para acceder al resultado de la carga, se puede realizar de varias formas.

- A través de una consulta SELECT:

```
SELECT id_hoja hoja, x, y, valor_number, valor_char
  FROM TABLE(pkxlsbd.get_pipe_tabla_excel());
```

- Recoger un array con los valores de las celdas mediante la función "pkxlsbd.get_tabla_excel()". El resultado de la función es un array del tipo pkxlsbd.tabla_excel.

Para obtener las hojas que componen la hoja de cálculo se usará la función "pkxls.get_nombres_hojas_xls(p_archivo, p_en_ias)".

- **p_archivo:** Ruta completa al archivo a cargar. Si la ejecución es en Forms 12c es importante el parámetro p_en_ias para indicar dónde se encuentra el archivo.
- **p_en_ias**: Si se está ejecutando en Forms 12c indica en dónde hace referencia el parámetro "p_archivo", si en p_en_ias se pasa el valor de TRUE el archivo debe de encontrarse en el servidor de aplicaciones, si se pasa FALSE el archivo debe de encontrase en el equipo del.

Esta función devolverá un array del tipo PKPANTALLAS.VARCHAR2_TABLA con el ID de la hoja y su nombre. Ejemplo:

```
DECLARE
  t_hojaspkpantallas.varchar2_table;
  v_id            PLS_INTEGER;
BEGIN
  t_hojas := pkxls.get_nombres_hojas_xls('c:\temp\test.xls', FALSE);

  IF t_hojas.COUNT() != 0 THEN
    v_id := t_hojas.FIRST();

    WHILE v_id IS NOT NULL LOOP
      pkpantallas.log('id: ' || v_id || ', nombre: ' || t_hojas(i));
      v_id := t_hojas.NEXT(v_id);
    END LOOP;
  END IF;
END;
```

IMPORTANTE: Si ya se ha ejecutado PKXLS.CARGAR_HOJA_CALCULO indicando el parámetro P_INICIALIZAR_HOJAS con el valor TRUE ya no hace falta volver a acceder al archivo, se puede obtener el número total de hojas del archivo con **pkxls.get_propiedad_excel('NUMERO_HOJAS')** y al nombre de cada hoja con **pkxls.get_propiedad_hoja(id_hoja, 'NOMBRE_HOJA').**

NOTA: Los campos de tipo fecha vienen en cómo número y para convertirlos en fecha hay que sumar ese número a la fecha TO_DATE('30-12-1899', 'DD-MM-YYYY').

Modificación de hojas de cálculo

Archivo a modificar en el servidor de Forms o en el equipo del usuario

Para modificar desde un programa de Forms un archivo de hoja de cálculo (formato XLS), habrá que incluir la librería "pklibxls.pll", y ejecutar lo siguiente:

- `pkxls.inicializa();`
- `pkxls.excel_celda(<hoja>, <fila>, <columna>, 'C', <valor>, 30)`: Se puede ejecutar tantas veces como celdas se quieran modificar. Los valores comienzan en 1.
- `pkxls.modifica_archivo_excel(p_archivo, p_archivo_destino, p_en_ias)`: Devuelve OK si todo ha ido correcto y ERROR si ha fallado.
    - **p_archivo**: Ruta del archivo a modificar.
    - **p_archivo_destino**: Ruta del archivo en donde se grabará el archivo modificado. Si se pasa NULL se modificará directamente el archivo indicado en "p_archivo".
    - **p_en_ias**: Hay que indicar TRUE si el archivo está en el servidor de aplicaciones y FALSE si el archivo se encuentra en el equipo del usuario.

Ejemplo:

```
DECLARE
  v_resultado  VARCHAR2(30);
BEGIN
  pkxls.inicializa();
  pkxls.excel_celda(1, 1, 1, 'C', 'PRUEBA1', 30);
  pkxls.excel_celda(1, 1, 2, 'C', 'PRUEBA2', 30);
  pkxls.excel_celda(1, 1, 3, 'C', 1234, 30);
  pkxls.excel_celda(1, 2, 3, 'C', 7321, 30);
  v_resultado  :=  pkxls.modifica_archivo_excel(p_archivo  =>  'C:\Temp\formato.xls',  p_archivo_destino  =>
'C:\Temp\formato_modificado.xls', p_en_ias => FALSE);
END;
```

Archivo de plantilla almacenado en la base de datos

Si el archivo a del que se va a partir para la modificación se encuentra almacenado en la base de datos los la llamada a pkxls.modifica archivo_excel será la siguiente:

```
pkxls.modifica_archivo_excel(p_id_archivo,          p_archivo_destino,          p_procedimiento_modificacion,
p_trigger_modificacion, p_permitir_conexion_directa, p_visualizar_archivo)
```

Devuelve OK si todo ha ido correcto y ERROR si ha fallado. Parámetros que recibe:

- **p_id_archivo**: Identificador del archivo almacenado en base de datos.
- **p_archivo_destino**: Ruta del archivo en el equipo del usuario en en donde se grabará el archivo una vez modificado.
- **p_procedimiento_modificacion**: Este parámetro si no se pasa a la llamada se considerará el valor NULL. Procedimiento de base de datos que se ejecutará para que aplique los cambios sobre el archivo. En este procedimiento se puede utilizar pkxlsbd.get_tabla_excel() para recuperar las celdas y los procedimientos de pkxlsbd para modificar el archivo.
- **p_trigger_modificacion:** Este parámetro si no se pasa a la llamada se considerará el valor NULL. Trigger personalizado que será ejecutado en el programa que hace la llamada. En este procedimiento se puede utilizar pkxlsbd.get_tabla_excel() para recuperar las celdas y los procedimientos de pkxls para modificar el archivo.
- **p_permitir_conexion_directa:** Este parámetro si no se pasa a la llamada se considerará el valor 'S'. Si se utiliza p_procedimiento_modificacion o p_trigger_modificacion, se realiza una carga del archivo antes de aplicar la modificación. Para más información de este parámetro ver la información del parámetro con este mismo nombre de la función "pkxls.cargar_hoja_calculo".
- **p_visualizar_archivo**: Este parámetro si no se pasa a la llamada se considerará el valor TRUE. Si se pasa el valor TRUE una vez grabado el archivo en el equipo del usuario se abrirá con la aplicación asociada al tipo de archivo generado.

Ejemplo:

```
IF pkxls.modifica_archivo_excel(p_id_archivo => p_id_archivo_plantilla,
                                p_archivo_destino => p_cli_archivo_grabar,
                                p_procedimiento_modificacion => 'pkformulasstd.procesar_hoja_calculo()',
                                p_permitir_conexion_directa => 'N') != 'OK' THEN
  msg.mensaje('PROCE', 'GENERAL');
END IF;
```

Gestión de correos electrónicos

Envío

Para el envío es necesario que esté correctamente configurado el servidor SMTP, se puede configurar en varios niveles, en caso de no estar disponible un servidor SMTP se probará al siguiente nivel:

- **Usuario**: Mediante el programa MAIL_SENDERS se pueden indicar los parámetros para el envío de correo electrónicos:
  - **Servidor SMTP**: Dirección del servidor de SMTP asociado al usuario.
  - **Puerto**: Puerto TCP en el que escucha el servidor SMTP, por defecto el 25.
  - **Dirección de Correo**: Dirección de correo de origen del mensaje.
  - **Usuario**: En caso de que el servidor requiera validación se introducirá el nombre del usuario del servidor SMTP.
  - **Password**: En caso de que el servidor requiera validación se introducirá la password del usuario en el servidor SMTP.
  - **Requiere Validación**: Se activará la check cuando el servidor SMTP requiera de un usuario y contraseña para enviar el correo.
- **Alerta**: Se puede indicar una parametrización específica para una alerta (módulo de alertas). Esta configuración se realiza en el mantenimiento de parámetros generales del menú, programa

U_MPRMEN, en la pestaña “Notificaciones / Alertas”. Los campos necesarios son iguales que a nivel de usuario.

- **SMTP Genérico**: En el mantenimiento de parámetros generales de menú se puede indicar un servidor SMTP Genérico que se usará cuando el Usuario / Alerta no tienen un servidor específico.

La secuencia de envío es la siguiente:

- Inicializar.
- Opcionalmente cambiar el remitente del mensaje.
- Incorporar el asunto del mensaje.
- Incorporar el cuerpo del mensaje.
- Indicar los destinatarios.
- Adjuntar archivos.
- Procesar envío.

Inicializar

En primer lugar, se ejecutará la inicialización del envío del correo electrónico con la instrucción:

```
PK_EMAIL.INICIALIZAR('<código de usuario>');
```

Opcionalmente cambiar el remitente del mensaje

Para cambiar el remitente que se le asigna por defecto al usuario se puede ejecutar:

```
PK_EMAIL.SET_EMAIL_REMITENTE('<email>');
```

Incorporar el asunto del mensaje

Para añadir el asunto del mensaje se ejecutará:

```
PK_EMAIL.SET_ASUNTO('<texto_del_asunto>');
```

Incorporar el cuerpo del mensaje

Para añadir el texto al cuerpo del mensaje hay que tener en cuenta que puede ir en texto plano, en formato html o mixto, es decir, va tanto en texto plano como html y el dispositivo receptor mostrará según sus capacidades el que mejor se adapte.

Para añadir texto plano al cuerpo del mensaje se utiliza el procedimiento:

```
PK_EMAIL.SET_CUERPO('<texto plano del cuerpo del mensaje>');
```

Para añadir texto en formato html se utiliza el procedimiento:

```
PK_EMAIL.SET_CUERPO_HTML('<texto html del cuerpo del mensaje>');
```

Indicar los destinatarios

Para añadir destinatarios se puede ejecutar tantas veces como destinatarios del mensaje existan el procedimiento:

```
PK_EMAIL.ADD_DESTINATARIO('<tipo>', '<dirección>');
```

El parámetro <tipo> puede contener los siguientes valores:

- **TO**: Destinatarios principales del mensaje.
- **CC**: Destinatarios que irán en copia del mensaje.
- **BCC**: Destinatarios que irán con copia oculta del mensaje.

Adjuntar archivos

Hay varias formas de adjuntar archivos a un mensaje:

- El archivo se encuentra almacenado en Libra en la tabla ARCHIVOS_ERP. En este caso se ejecutará: PK_EMAIL.ADD_ADJUNTO_X_ID(<id_archivo>), en donde <id_archivo> es el identificador del archivo en la tabla ARCHIVOS_ERP.
- El archivo se tiene en una variable de tipo BLOB. En este caso se ejecutará: PK_EMAIL.ADD_ADJUNTO_BLOB(<variable_blob>, <nombre_archivo>);
- (**OBSOLETA**, debe de utilizarse: PK_EMAIL.ADD_ADJUNTO_BLOB). El archivo se encuentra en un directorio de la base de datos. En este caso se ejecutará: PK_EMAIL.ADD_ADJUNTO('<archivo>'), en donde <archivo> es la ruta completa al archivo a adjuntar. En el caso de no indicar la ruta completa se asumirá que se encuentra en el directorio parametrizado para adjuntos en parámetros generales de Libra o en su defecto en BLOB_TEMP.
- El archivo que se desea adjuntar es el resultado del proceso de informe del Generador de Informes. En este caso se ejecutará: PK_EMAIL.ADD_ADJUNTO_GI('<p_informe>', '<p_nombre_archivo>', '<p_idioma>', '<p_empresa>', '<p_usuario>', '<p_plantilla_valores>', '<p_tipo_archivo>', '<p_configuracion>');
    - **p_informe**: Código del informe a ejecutar.
    - **p_nombre_archivo**: Nombre del archivo a generar.
    - **p_idioma**: Código del idioma a utilizar para las etiquetas
    - **p_empresa**: Código de la empresa con la que se ejecutará el informe.
    - **p_usuario**: Código del usuario con el que se ejecutará el informe.
    - **p_plantilla_valores**: Sí el informe tiene guardada plantillas de valores, se puede indicar la plantilla a usar.
    - **p_tipo_archivo**: Valores posibles:
        - **EXCELXML**: Genera el archivo en formato hoja de cálculo. Si la extensión del archivo es XLS ó XLSX y está configurado GAL_EXCEL se generará en formato nativo, salvo que en el informe tenga informado que utiliza tablas GLOBAL TEMPORARY.
        - **HTML**
        - **CSV**: Separado por comas
        - **TXT**: Texto plano
    - **p_configuracion**: Si el informe tiene varias configuraciones de columnas, se puede indicar en este parámetro el código de configuración de columnas a utilizar.

Se pueden adjuntar tantos archivos al mensaje como sea necesario y también se pueden mezclar llamadas de ADD_ADJUNTO_X_ID con ADD_ADJUNTO y ADD_ADJUNTO_GI.

NOTA: El directorio en donde se encuentran los archivos deben tener permisos de lectura para el motor de Java de Oracle, para ello se debe de ejecutar (Si la base de datos está en Linux cambiar 'directorio*' por 'directorio/*'):

```
exec dbms_java.grant_permission('usuario LIBRA en mayúsculas', 'java.io.FilePermission', 'dir\*', 'read');
```

Ejemplo:

```
exec dbms_java.grant_permission('LIBRA','java.io.FilePermission', 'C:\Oracle\dir\blobtemp\*', 'read');
```

Procesar envío.

Para procesar el envío finalmente se ejecutará la función: PK_EMAIL.ENVIAR(). El resultado que devuelve es 'OK' en caso de que el envío se realiza correctamente o 'ERROR' en caso de producirse algún fallo. En este último caso quedará registrado en LIBRA_LOG el motivo del error, también se puede consultar por código el resultado del último envío llamando a la función PK_EMAIL.GET_ULTIMO_RDO y el detalle del error llamando a la función PK_EMAIL.GET_ULTIMO_TEXTO_ERROR.

Ejemplo:

```
DECLARE
  v_resultado    VARCHAR2(30);
BEGIN
  PK_EMAIL.INICIALIZAR('EDISA');
  PK_EMAIL.SET_ASUNTO('TEXTO ASUNTO');
  PK_EMAIL.SET_CUERPO('CUERPO TEXTO PLANO');
  PK_EMAIL.SET_CUERPO_HTML('<H1>CUERPO HTML</H1>');
  PK_EMAIL.ADD_DESTINATARIO('TO', 'correo@dominio.com');
  v_resultado := PK_EMAIL.ENVIAR();
END;
```

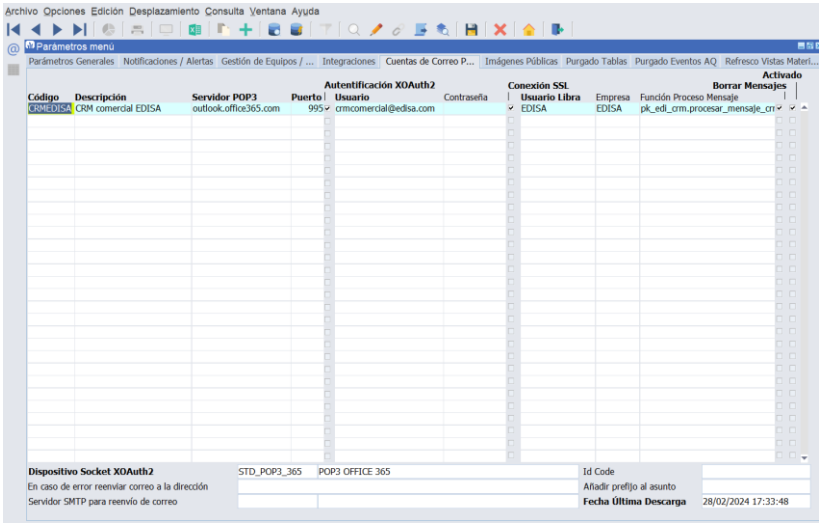
Funciones de control

Existen una serie de funciones de control que permitirán realizar verificaciones antes de enviar el correo,

- PK_EMAIL.HAY_SERVIDOR_SMTP: Se puede ejecutar después de llamar a PK_EMAIL.INICIALIZAR, y devolverá TRUE en caso de existir un servidor parametrizado para realizar el envío del correo y FALSE en caso contrario.
- PK_HAY_REMITENTE: Devuelve TRUE en caso de que exista una dirección de email de remite del mensaje, FALSE en caso contrario. Se puede ejecutar antes de PK_EMAIL.ENVIAR,
- PK_HAY_DESTINATARIOS: Devuelve TRUE si se ha especificado algún destinatario al mensaje y FALSE en caso contrario. Se puede ejecutar antes de PK_EMAIL.ENVIAR.

Descarga

Mediante el paquete PK_EMAIL también se puede realizar la descarga de correos de cuentas POP3. En primer lugar, se deben de configurar las cuentas POP3 en el mantenimiento de parámetros generales de menú, programa U_MPRMEN.



- **Código**: Identificador único que se le asigna la cuenta POP3, este código será necesario luego para indicar la cuenta de correo de la que se quiere descargar el correo.
- **Descripción**: Descripción breve de la cuenta de correo.
- **Servidor POP3**: Dirección IP o nombre del servidor de POP3 en donde se encuentra la cuenta.
- **Puerto**: Puerto TCP en el que escucha el servidor de POP3.
- **Usuario**: Usuario de la cuenta de correo.
- **Password**: Contraseña del usuario de la cuenta de correo.
- **Usuario Libra**: Código del usuario de Libra al que quedarán asociados los archivos adjuntos descargados.
- **Función Proceso Mensaje**: (Opcional). Función de base de datos que se invocará por cada mensaje una vez descargado para poder automatizar tareas. Esa función debe de devolver el resultado (si es correcto debe devolver OK) y recibe como parámetro el ID del mensaje.

Ejemplo:

```
CREATE OR REPLACE FUNCTION PRUEBA_EMAIL(p_id_mensaje NUMBER) RETURN VARCHAR2 IS
BEGIN
  RETURN('OK');
END;
```

Para lanzar la descarga de los correos de un buzón POP3 se llamará al procedimiento: PK_EMAIL.PROCESAR_SERVIDOR_POP3('<código cuenta pop3>'). En <código cuenta pop3> se pasará el código de la cuenta configurada en los parámetros generales de menú.

Los mensajes quedarán almacenados en las tablas:

- **EMAIL_GESTION_CORREOS**: Almacena los correos descargados.
  - ID: Identificador numérico único asignado al mensaje.
  - CODIGO_SERVIDOR_POP3: Código del servidor POP3 del qué procede el mensaje.
  - FECHA: Fecha del mensaje.
  - FECHA_DESCARGA: Fecha en la que se realizó la descarga del mensaje.
  - NUMERO_VECES_PROCESADO: Número de veces que se ha ejecutado la "Función Proceso Mensaje" asociada a la cuenta POP3.
  - FECHA_ULTIMO_PROCESO: Fecha en la que se realizó la última llamada a la "Función Proceso Mensaje".
  - RESULTADO_ULTIMO_PROCESO": Resultado que devolvió en la última llamada a la "Función Proceso Mensaje".
  - REMITENTE: Dirección de correo electrónico de la que procede el mensaje.
  - DESTINARIO: Lista de destinatarios del mensaje.
  - ASUNTO: Texto del asunto.
  - TEXTO_BODY: Texto del correo electrónico en formato de texto plano, si el mensaje viene en formato HTML exclusivamente este campo estará en blanco.
  - TEXTO_BODY_HTML: Texto del correo electrónico en formato HTML, si el mensaje viene en formato de texto plano exclusivamente este campo estará en blanco.
- **EMAIL_GESTION_CORREOS_ADJUNTOS**: Almacena los adjuntos de los correos:
  - ID: Identificador del mensaje, se utiliza para relacionar los adjuntos con EMAIL_GESTION_CORREOS.
  - ID_ARCHIVO: Identificador asignado al adjunto en la tabla ARCHIVOS_ERP, que es el almacenamiento real del archivo.
  - NOMBRE_ARCHIVO: Nombre del archivo tal y como venía en el mensaje.

Gestión de archivos XML.

Carga de archivo

El proceso de carga de un archivo se basa en su recorrido de forma secuencial, en cada nodo del XML se puede indicar la forma de procesarlo, insertar registros en una tabla, invocar la ejecución de una función de base de datos o realizar ambas a la vez.

Inicialización

En primer lugar, hay que parametrizar la lectura del XML, para ello en primer lugar es obligatorio ejecutar la instrucción: PKXML.XML_INICIALIZA_PARSER(p_namespace_automatico => TRUE).

Configuración de Nodo

A continuación, por cada nodo que se quiere contemplar en la lectura hay que parametrizarlo llamando a la función PK_XML.XML_CREA_NODO_PARSER, esta función devuelve un dato de tipo PLS_INTEGER que identifica el nodo y que es necesario identificar para luego poder configurar los items de ese nodo. La función recibe los siguientes parámetros:

- **p_etiqueta**: Este parámetro es obligatorio. Es la ruta XPATH completa del nodo a procesar, por ejemplo: /Clientes/Cliente/AlbaranesCliente/Albaran

- **p_funcion**: Este parámetro es opcional, en él se indica la función que se debe de ejecutar por cada nodo indicado en P_ETIQUETA. Los parámetros que recibe la función son variables y se identifican más adelante en la configuración de los items del nodo (Ver PK_XML.CREA_CAMPO_NODO_PARSER). La función debe devolver un VARCHAR2 con el texto OK en el caso de que el proceso sea correcto y otro valor en caso de que se produzca un error, en este último caso se cancelará el proceso de lectura del XML.
- **p_tabla**: Este parámetro es opcional. Nombre de la tabla en donde se debe de insertar el registro que contiene el nodo indicado en P_ETIQUETA. Los campos se identifican más adelante en la configuración de los items del nodo (Ver PK_XML.CREA_CAMPO_NODO_PARSER).
- **p_ignorar_errores**: Si se pasa S en este parámetro, un error en un nodo no finaliza el parseo de todo el XML, sino que se sigue con el siguiente nodo. (Por defecto tiene el valor N, de forma de que, si no se indica, un error para el parseo).
- **p_modo_sql**: Permite indicar la acción a realizar en la tabla de destino. Los valores posibles son INSERT (por defecto), UPDATE o MERGE.
- **p_funcion_exception**: Indica una función que se va a invocar en caso de que un nodo no se pueda parsear. Una utilidad clara para esta función es guardar los registros que no se puedan insertar. La firma de la función debe contener todos los campos que se parsean en el XML.

Configuración de Items del nodo

Por cada item del nodo que interese ser almacenado en la tabla o por la función parametrizada en PK_XML.XML_CREA_NODO_PARSER hay que parametrizarlo llamando a la función PK_XML_CREA_CAMPO_NODO_PARSER que devuelve un PLS_INTEGER que identifica el campo. La función recibe los siguientes parámetros:

- **p_id_nodo**: (Obligatorio). Identificador del nodo, devuelto por PK_XML.XML_CREA_NODO_PARSER.
- **p_campo**: (Opcional). Se utiliza cuando al crear el nodo se ha indicado una Tabla y en este parámetro se indica el campo de la tabla en donde se quiere guardar el valor que tiene el item.
- **p_parametro**: (Opcional). Se utiliza cuando al crear el nodo se ha indicado una función, se indica en qué parámetro hay que pasar el valor del item al hacer la llamada a la función.
- **p_valor_fijo**: (opcional). En vez de leer el dato de un item del XML se utiliza un valor fijo proporcionado en este parámetro, tanto para hacer el insert en la tabla parametrizada o en el parámetro de la función parametrizada.
- **p_etiqueta_valor**: (Opcional). Ruta XPATH completa al campo del que se quiere obtener el valor. No hay problema en usar un campo que se encuentra en un nodo de nivel anterior o posterior al nodo que se está procesando. Ejemplo: /Clientes/Cliente/Codigo
- **p_tipo**: Se usa para indicar el tipo de dato. Si no se indica nada se considera VARCHAR2. Los valores posibles son:
  - **VARCHAR2**
  - **NUMBER**
  - **DATE**
  - **CLOB**
- **p_mascara**: Se usa cuando se indica p_tipo => 'NUMBER', indica la máscara de formato con la que viene el número en el XML. Ejemplo: p_mascara => '999999D90'.
- **p_nls_numeric_characters**: Se usa cuando se indica p_tipo => 'NUMBER', para indicar cual es el carácter separador de millares y el de decimales, si el número viene con el formato 3443.23, hay que pasar p_nls_numeric_characters => '.,'.
- **p_nls_language**: Para el parseo de los campos de tipo DATE cuando no se especifica una máscara, indica la configuración regional que se utiliza para el parseo a campo date.
- **p_funcion_transformacion**: Función PL/SQL a utilizar para transformar una columna. Esta función debe tener un único parámetro de tipo varchar2 en la firma que recibirá el valor de la columna que queremos transformar.

- **p_es_pk**: Para las operaciones de MERGE y UPDATE es obligatorio conocer qué campos son clave primaria en la tabla destino. Este parámetro es de tipo booleano, por lo que hay que indicar TRUE para aquellos campos que sean clave primaria. Si no se indica el parámetro se asume que no es clave primaria.
- **p_solo_funcion**: Permite para una columna determinada que no se inserte en la tabla destino, y que su valor solo se pase en el caso de que el parseo del XML tenga como destino una función PL/SQL, y para la función de excepción. Un ejemplo claro de esto es si en el XML viaja el rowid de la tabla origen.

Ejecutar el proceso de lectura

El XML para procesar debe de estar almacenado en una variable XMLTYPE, en el paquete PK_XML se disponen de funciones para cargarlo desde un archivo (PK_XML.CARGA_XML_DESDE_FIC) o desde una dirección WEB (PK_XML.LEE_XML_DESDE_URL).

Para invocar la lectura del XML hay que llamar a la función PK_XML.PARSEAR_XML, la función devolverá OK si la lectura se ha realizado correctamente o el código de error devuelto por la función que cancelase el proceso. Ejemplo: v_resultado := PK_XML.PARSEAR_XML(v_xml);

```
DECLARE
  v_xml        XMLTYPE;
  v_empresa    VARCHAR2(5):= '013';
  v_id_nodo    PLS_INTEGER;
  v_id_campo   PLS_INTEGER;
  v_resultado  VARCHAR2(30);
BEGIN
  v_xml := pk_xml.carga_xml_desde_fic('BLOB_TEMP', 'elXMl.xml');
  pk_xml.xml_inicializa_parser(p_namespace_automatico => TRUE);
  v_id_nodo := pk_xml.xml_crea_nodo_parser(p_etiqueta => '/Cli/Cl/Albaranes/Albaran',
                                           p_funcion => 'F_PRUEBA_ELIAS_LECTURA_XML',
                                           p_tabla => 'PRUEBA_ELIAS_LECTURA_XML');
  v_id_campo := pk_xml.xml_crea_campo_nodo_parser(p_id_nodo => v_id_nodo,
                                                  p_campo => 'EMPRESA',
                                                  p_parametro => 'P_EMPRESA',
                                                  p_valor_fijo => v_empresa);
  v_id_campo := pk_xml.xml_crea_campo_nodo_parser(p_id_nodo => v_id_nodo,
                                                  p_campo => 'CODIGO_CLIENTE',
                                                  p_parametro => 'P_CODIGO_CLIENTE',
                                                  p_etiqueta_valor => '/Cli/Cl/Codigo');
  v_id_campo := pk_xml.xml_crea_campo_nodo_parser(p_id_nodo => v_id_nodo,
                                                  p_campo => 'NOMBRE_CLIENTE',
                                                  p_parametro => 'P_NOMBRE_CLIENTE',
                                                  p_etiqueta_valor => '/Cli/Cl/Nombre');
  v_id_campo := pk_xml.xml_crea_campo_nodo_parser(p_id_nodo => v_id_nodo,
                                                  p_campo => 'NUMERO_ALBARAN',
                                                  p_parametro => 'P_NUMERO_ALBARAN',
                                                  p_etiqueta_valor => '/Cli/Cl/Albaranes/Alb/No');
  v_id_campo := pk_xml.xml_crea_campo_nodo_parser(p_id_nodo => v_id_nodo,
                                                  p_campo => 'IMPORTE_ALBARAN',
                                                  p_parametro => 'P_IMPORTE_ALBARAN',
                                                  p_etiqueta_valor => '/Cli/Cl/Albaranes/Alb/Im');
  v_resultado := pk_xml.parsear_xml(v_xml);
  pkpantallas.log('RESULTADO PRUEBA: ' || v_resultado);
END;
```

Generación de archivos XML

Para generar un archivo XML, hay que parametrizar los nodos, los atributos y los campos que van a tener los nodos.

Inicialización

Todo documento XML debe de tener un nodo raíz que engloba la totalidad del resto de los nodos, en el proceso de inicialización se llama a la función a PK_XML.XML_INICIALIZA, esta función devuelve un PLS_INTEGER que identifica al nodo raíz y recibe por parámetro la etiqueta del nodo Raíz. Ejemplo:

v_id_nodo_raiz := pk_xml.xml_inicializa('Clientes');

Incluir nodos al documento

Se pueden añadir tantos nodos como sea necesario, un nodo puede estar si es necesario a una tabla o puede tomar valores fijos. Para crear un nodo se llamará a la función PK_XML.XML_CREA_NODO, esta función devolverá un PLS_INTEGER que identifica al nodo y que será necesario para luego añadirle campos. La función recibe los siguientes parámetros:

- **p_id_nodo_padre**: Obligatorio, se indica el nodo del que va a colgar. Todo nodo va a tener un nodo padre, ya que como mínimo hay un nodo raíz que engloba a todos.
- **p_etiqueta_registro**: Obligatorio. Etiqueta XML que va agrupar los campos de cada registro.
- **p_etiqueta_grupacion**: Opcional. Etiqueta que agrupa a todos los registros.
- **p_tabla**: Opcional. Nombre de la tabla que debe de recorrerse para obtener los registros a incluir en el XML.
- **p_where**: Opcional. Condición a aplicar a los registros de la tabla.
- **p_order_by**: Opcional. Ordenación de los registros a obtener.

Nota sobre variables en p_where: en la where se pueden usar variables de tipo :xxxx, esas variables serán enlazadas de forma dinámica. Para indicar los valores a esas variables hay que llamar al procedimiento PK_XML.SET_VARIABLE('<variable>', <valor>);

Por ejemplo, si en la where se utiliza “codigo_empresa = :p_empresa” habrá que enlazar “:p_empresa” con el valor correspondiente con PK_XML.SET_VARIABLE('P_EMPRESA', '013');

Incluir campos a un nodo

Para incluir campos a un nodo hay que llamar a la función PK_XML.XML_CREA_CAMPO_NODO, esta función devuelve un PLS_INTEGER que identifica al campo y recibe los siguientes parámetros:

- **p_id_nodo**: Obligatorio, se indica el nodo del que va a colgar el campo.
- **p_etiqueta**: Obligatorio, etiqueta que va a tener el valor del campo en el XML.
- **p_campo_tabla**: Opcional, si el nodo está asociado a tabla, se indica de que campo debe de obtenerse al valor a incluir en la etiqueta.
- **p_valor_fijo**: Opcional, valor que llevará la etiqueta, sin necesidad de ser recuperado de la tabla.
- **p_obligatorio**: Se utiliza exclusivamente cuando se indica “p_campo_tabla” y los valores que recibe son ‘S’ y ‘N’. Si se pasa el valor ‘N’ y el campo indicado en p_campo_tabla es NULL ya no se incluye la etiqueta en el XML. Si no se indica, el valor por defecto es ‘S’.

Incluir atributos a un nodo.

En XML los nodos pueden contener información en forma de atributos, para ello se incorpora la función PK_XML.XML_CREA_ATRIBUTO_NODO que devuelve un PLS_INTEGER que identifica el atributo y recibe los siguientes parámetros:

- **p_id_nodo**: Obligatorio, se indica el nodo del que va a colgar el atributo.
- **p_etiqueta**: Obligatorio, etiqueta que va a tener el atributo dentro del nodo.
- **p_campo_tabla**: Opcional, si el nodo está asociado a tabla, se indica de que campo debe de obtenerse al valor a incluir en el atributo.
- **p_valor_fijo**: Opcional, valor que llevará el atributo, sin necesidad de ser recuperado de la tabla.

Ejecutar el proceso de generación

Una vez parametrizada la estructura del XML se dispone de la función PK_XML.CALCULA_SQL_XML() que devuelve el XMLTYPE con el contenido del XML.

En el paquete PK_XML hay funciones para gestionar el XMLTYPE, por ejemplo, se podría guardar en archivo mediante PK_XML.GRABA_XML_EN_FICHERO.

Ejemplo:

```
DECLARE
  v_id_nodo_raiz       PLS_INTEGER;
  v_id_nodo_generico   PLS_INTEGER;
  v_id_nodo_clientes   PLS_INTEGER;
  v_id_nodo_albaranes  PLS_INTEGER;
  v_id_campo           PLS_INTEGER;
  v_id_atributo        PLS_INTEGER;
  v_xml                XMLTYPE;
BEGIN
  v_id_nodo_raiz := pk_xml.xml_inicializa('Clientes');
  v_id_nodo_generico := pk_xml.xml_crea_nodo(p_id_nodo_padre => v_id_nodo_raiz,
                                             p_etiqueta_registro => 'DatosExportacion');
  v_id_campo := pk_xml.xml_crea_campo_nodo(p_id_nodo => v_id_nodo_generico,
                                           p_etiqueta => 'UsuarioExportacion',
                                           p_valor_fijo => 'ELIASF');
  v_id_campo := pk_xml.xml_crea_campo_nodo(p_id_nodo => v_id_nodo_generico,
                                           p_etiqueta => 'Fecha',
                                           p_campo_tabla => 'TO_CHAR(SYSDATE, ''DD/MM/YYYY'')');
  v_id_nodo_clientes := pk_xml.xml_crea_nodo(p_id_nodo_padre => v_id_nodo_raiz,
                                             p_etiqueta_registro => 'Cliente',
                                             p_tabla => 'clientes c',
                                             p_where => 'c.codigo_empresa = :p_empresa',
                                             p_order_by => 'c.codigo_rapido');
  v_id_atributo := pk_xml.xml_crea_atributo_nodo(p_id_nodo => v_id_nodo_clientes,
                                                 p_etiqueta => 'CodigoEnAtributo',
                                                 p_campo_tabla => 'c.codigo_rapido');
  v_id_campo := pk_xml.xml_crea_campo_nodo(p_id_nodo => v_id_nodo_clientes,
                                           p_etiqueta => 'Codigo',
                                           p_campo_tabla => 'c.codigo_rapido');
  v_id_campo := pk_xml.xml_crea_campo_nodo(p_id_nodo => v_id_nodo_clientes,
                                           p_etiqueta => 'Direccion',
                                           p_campo_tabla => 'c.direccion',
                                           p_obligatorio => 'N');
  v_id_campo := pk_xml.xml_crea_campo_nodo(p_id_nodo => v_id_nodo_clientes,
                                           p_etiqueta => 'Nombre',
                                           p_campo_tabla => 'c.nombre');
  v_id_nodo_albaranes := pk_xml.xml_crea_nodo(p_id_nodo_padre => v_id_nodo_clientes,
                                              p_etiqueta_registro => 'Albaran',
                                              p_etiqueta_agrupacion => 'Albaranes',
                                              p_tabla => 'albaran_ventas av',
                                              p_where  =>  'av.cliente  =  c.codigo_rapido  AND  av.empresa  =
:p_empresa');
  v_id_campo := pk_xml.xml_crea_campo_nodo(p_id_nodo => v_id_nodo_albaranes,
                                           p_etiqueta => 'Numero',
                                           p_campo_tabla => 'av.numero_albaran');
  v_id_campo := pk_xml.xml_crea_campo_nodo(p_id_nodo => v_id_nodo_albaranes,
                                           p_etiqueta => 'Importe',
                                           p_campo_tabla => 'ROUND(av.importe_bruto)');
  pk_xml.set_variable('p_empresa', '013');
  v_xml := pk_xml.calcula_sql_xml();
  pk_xml.graba_xml_en_fichero(v_xml, 'BLOB_TEMP', 'prueba.xml', NULL);
END;
```

Recursos HTML en programas de Forms.

Con el fin de extender las funcionalidades de Oracle Forms, se ha implementado a través de los objetos BEAN de clases Java, cargar el navegador web y en el cargar recursos HTML que previamente se descargan en el equipo local del usuario.

Programa Archivos de Recursos [U_RESOURCES]

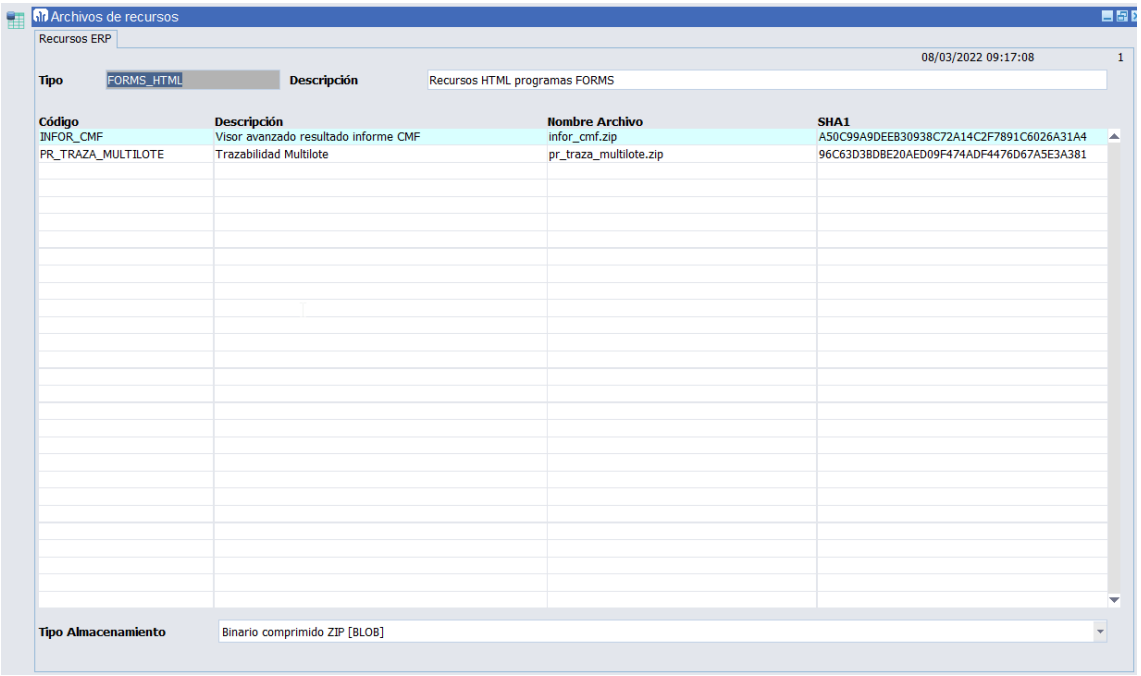
Este programa sirve de repositorio de los recursos disponible, los cuales mediante la librería "pklibrsc.pll" se descargarán en el equipo local del usuario, controlando su versionado a partir del SHA1 de los mismos.

Cabe destacar que este programa no solo sirve para recursos HTML, sino cualquier tipo de archivo que se quiera enviar al equipo local del usuario.

Otra característica, es que el concepto de TIPO_RECURSO el cual a través del prefijo "GLOBAL_" permite registrar aquellos recursos que sean comunes a un tipo, realizando su descarga previa.

La nomenclatura de recursos HTML a seguir es la siguiente

TIPO	FORMS_HTML
NOMBRE	<NOMBRE_FMB> al que pertenece



PKLIBRSC.PLL

Esta PLL tiene registrado el paquete "PKRESOURCES" que contiene métodos para la carga de los recursos en el equipo local.

- *FUNCTION habilitado RETURN BOOLEAN;* Retorna si está disponible su uso. Esto será cierto en un entorno 6.4.2 o superior.
- *FUNCTION carga_recurso(p_tipo VARCHAR2, p_codigo VARCHAR2) RETURN VARCHAR2;* Cargar el recurso solicitado. Retorna la ruta donde fue descargado.
- FUNCTION carga_archivo_bd_temp(p_nombre_archivo VARCHAR2) RETURN VARCHAR2; Cargar el fichero almacenado PK_BLOB2BD.GET_FICHERO en el directorio temporal de recursos, retornando la ruta al mismo.

- *FUNCTION carga_archivo_erp_temp(p_id_archivo NUMBER) RETURN VARCHAR2;* Cargar desde ARCHIVOS_ERP el archivo identificado por ID_ARCHIVO en el directorio temporal de recursos, retornando la ruta al mismo.
- *PROCEDURE borra_archivo_temp(p_nombre_archivo VARCHAR2);* Borra el archivo del almacenamiento temporal resultado de una carga anterior.
- *PROCEDURE borra_archivo_erp_temp(p_id_archivo NUMBER);* Borra el archivo del almacenamiento temporal resultado de una carga anterior.
- *PROCEDURE limpia_archivos_temp;* Eliminar todo el contenido del directorio de almacenamiento temporal.
- *PROCEDURE limpia_directorio_temp(p_solo_antiguos BOOLEAN DEFAULT FALSE);* En el caso de enviar "p_solo_antiguos" a FALSE se comporta del mismo modo que el anterior método. En el caso de enviar con valor TRUE, se borran solo aquellos que han sido cargados mediante llamadas a "*carga_archivo_erp_temp*", los cuales registran en una variable global todos los archivos generados.

PKLIBWEBBROWSER.PLL

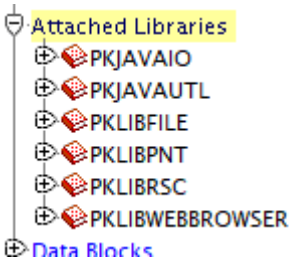
Esta librería se basa en el código de "PKLIBMENUADF" el cual incluye un paquete PKWEBBROWSER con una serie de métodos que permiten gestionar las pestañas con navegador integrado en el programa de inicio (BPM, Widgets, Comunidades…)

Para esta librería, se ha limpiado todo lo referente a las pestañas, registrando un único BEAN Java en el cual residirá el navegador web y sobre el que propagar los eventos de sus métodos.

- *PROCEDURE inicio(p_ventana…);* Método en el que se registra el BEAN indicando una serie de propiedades respecto a su tamaño, url de inicio…
- PROCEDURE when_new_item_instance; Agregar este código al disparador estándar del ITEM del BEAN.
- *PROCEDURE when_window_resized(p_ventana VARCHAR2);* Agregar este código al disparador estándar para que el navegador sea responsive.
- *PROCEDURE when_window_activated(p_ventana VARCHAR2);* Agregar este código al disparador estándar para que el navegador sea responsive.
- *PROCEDURE ejecutar_javascript(p_javascript VARCHAR2);* Permite enviar un JavaScript al navegador. Evento asíncrono.
- *PROCEDURE cambiar_url(p_url VARCHAR2);* Permite cambiar la URL del navegador.
- *FUNCTION ejecutar_javascript(p_javascript VARCHAR2) RETURN VARCHAR2;* Permite enviar un JavaScript al navegador del cual se espera respuesta para ser tratada. Es un evento síncrono.
- *PROCEDURE ejecutar_javascript_gzip_bd;* Permite enviar como JavaScript el contenido de PK_BLOB2BD, el cual se espera que esté comprimido en GZIP. Método Asíncrono.
- *FUNCTION ejecutar_javascript_gzip_bd RETURN VARCHAR2;* Permite enviar como JavaScript el contenido de PK_BLOB2BD, el cual se espera que esté comprimido en GZIP y recuperar su respuesta. Método síncrono.
- *PROCEDURE ejecutar_javascript_bd;* Permite enviar como JavaScript el contenido de PK_BLOB2BD el cual previamente comprimirá como GZIP. Método asíncrono.
- *FUNCTION ejecutar_javascript_bd RETURN VARCHAR2;* Permite enviar como JavaScript el contenido de PK_BLOB2BD el cual previamente comprimirá como GZIP y recuperar su respuesta. Método síncrono.
- *PROCEDURE cerrar_navegador;* Cerrar el navegador, limpiando sus recursos. No es necesario aunque si recomendable.

Manual de uso en programa

Agregar las librerías “pklibrsc.pll” y “pklibwebbrowser.pll” al programa.



Crear parámetro “recurso_inicializado” con valor por defecto a N

Esto nos permite evitar volver a cargar el recurso en el método dónde se realice la carga del recurso, asociar el BEAN al navegador y por último ejecutar el JavaScript para mostrar los datos en el recurso HTML.

Registrar los siguientes disparadores

WHEN-WINDOW-RESIZED

```
pkwebbrowser.when_window_resized(:system.event_window);
```

WHEN-WINDOW-ACTIVATED

```
pkwebbrowser.when_window_activated(:system.event_window);
```

BLOQUE-ITEM - WHEN-NEW-ITEM-INSTANCE

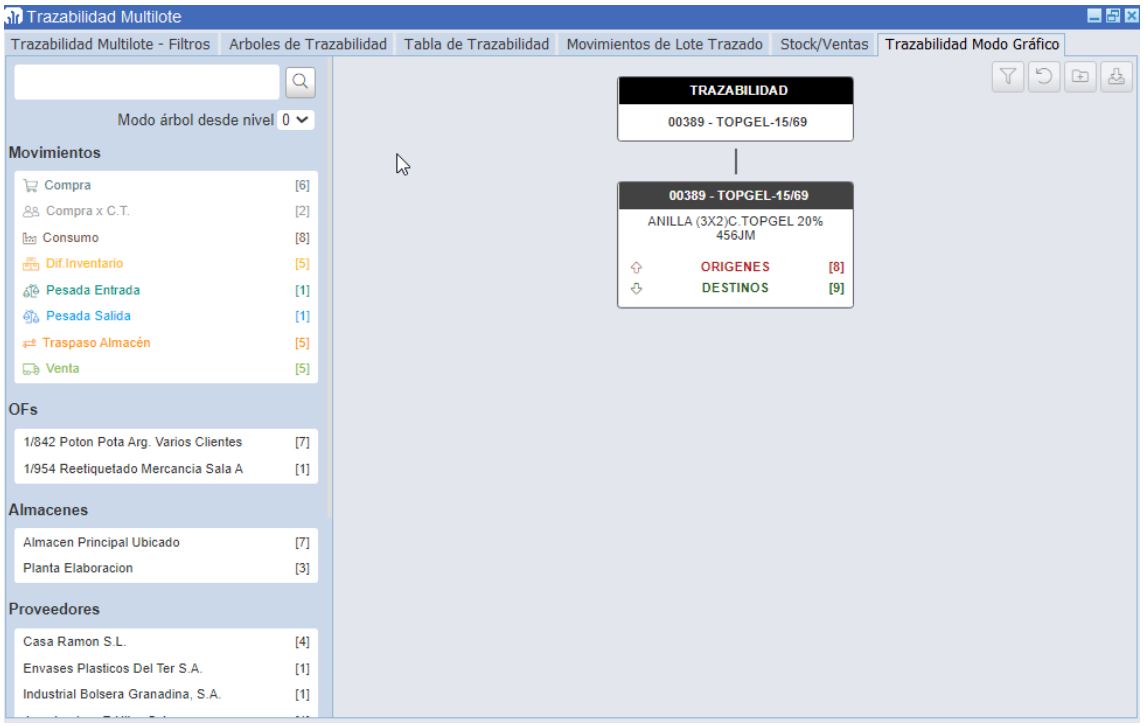
```
pkwebbrowser.when_new_item_instance;
```

KEY-EXIT

```
IF NVL(:parameter.recurso_inicializado, 'N') = 'S' THEN
  pkwebbrowser.cerrar_navegador;
  pkresources.limpia_archivos_temp;
END IF;
DISPSTD.KEY_EXIT;
```

Ejemplo de carga de recurso HTML y envío evento JavaScript.

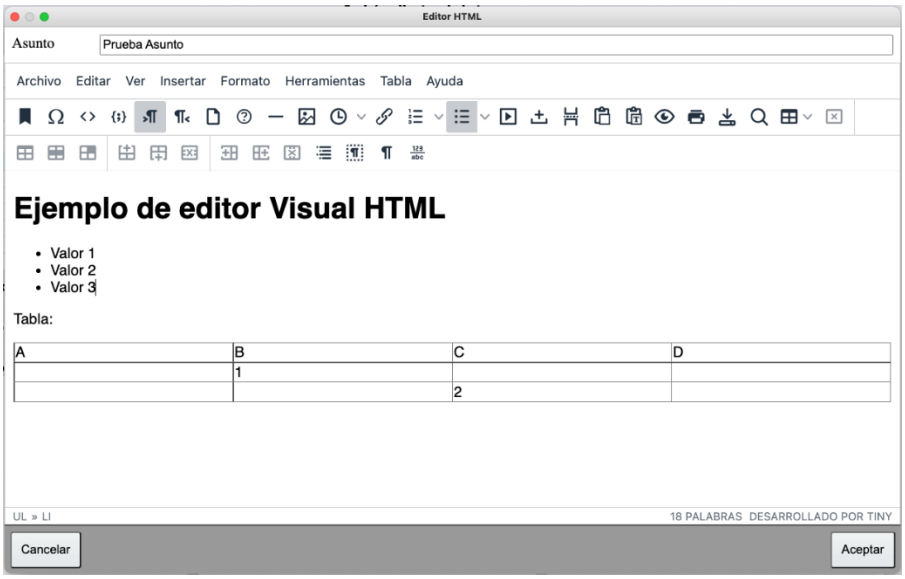
```
--Cargar recurso HTML
IF NVL(:parameter.recurso_inicializado,'N') = 'N' THEN
  v_directorio_recurso := pkresources.carga_recurso(p_tipo => :parameter.resource_tipo, p_codigo =>
:parameter.resource_codigo);
  pkwebbrowser.inicio(p_bean_area => 'B6.BEAN_WWW', p_url => v_directorio_recurso || 'index.html',
p_ajuste_horizontal_bean => 0.05, p_ajuste_vertical_bean => 0.25);
  :parameter.recurso_inicializado := 'S';
END IF;
--Lanzar generación CLOB
f_generar_javascript_bd(p_empresa => :global.codigo_empresa);
pkwebbrowser.ejecutar_javascript_bd;
```



Salida gráfica del programa "Trazabilidad Multilote"

Editor Visual HTML

El editor visual HTML se abre en una ventana flotante similar a la siguiente:



Para que un programa pueda hacer uso del editor HTML debe de incorporar la librería: pkeditorhtml.pll

Inicializar

El primer paso que hay que ejecutar (y únicamente una vez) es procedimiento PKEDITORHTML.INICIALIZAR();, de esta forma se indica que se desea utilizar el editor HTML.

Propiedades.

A continuación, hay que indicar el comportamiento que se desea que tenga el editor, para ello se indican propiedades mediante PKEDITORHTML.SET_PROPIEDAD('<codigo_propiedad>', '<valor_propiedad');

Las propiedades disponibles son las siguientes:

- **ACTIVA_CAMPO_ASUNTO**: Si se pasa el valor S en <valor_propiedad>, mostrará un campo a mayores donde se le solicita al usuario un Asunto, si no se pasa ese campo estará oculto al usuario. El texto del asunto puede ser recuperado de forma independiente del texto HTML una vez el usuario cierra el editor.
- **ACTIVA_CAMPO_PARA**: Si se pasa el valor S en <valor_propiedad>, mostrará un campo a mayores donde se le solicita al usuario direcciones de correo electrónico. Estas direcciones pueden ser recuperadas de forma independiente del texto en HTML una vez el usuario cierra el editor.
- **ACTIVA_CAMPO_CC**: Si se pasa el valor S en <valor_propiedad>, mostrará un campo a mayores donde se le solicita al usuario direcciones de correo electrónico "Con copia". Estas direcciones pueden ser recuperadas de forma independiente del texto en HTML una vez el usuario cierra el editor.
- **ACTIVA_CAMPO_CCO**: Si se pasa el valor S en <valor_propiedad>, mostrará un campo a mayores donde se le solicita al usuario direcciones de correo electrónico "Con copia oculta". Estas direcciones pueden ser recuperadas de forma independiente del texto en HTML una vez el usuario cierra el editor.
- **ETIQUETA_PARA**: Permite personalizar en <valor_propiedad> un texto diferente a la etiqueta que va a tener el campo "Para".
- **ETIQUETA_CC**: Permite personalizar en <valor_propiedad> un texto diferente a la etiqueta que va a tener el campo "Con copia".
- **ETIQUETA_CCO**: Permite personalizar en <valor_propiedad> un texto diferente a la etiqueta que va a tener el campo "Con copia oculta".

- **ETIQUETA_LISTA_ASUNTO**: Permite personalizar en <valor_propiedad> un texto diferente a los textos fijos que puede seleccionar el usuario a incluir en el asunto. Ver apartado: Incluir etiquetas fijas.
- **ETIQUETA_LISTA_CUERPO**: Permite personalizar en <valor_propiedad> un texto diferente a la etiqueta del campo de textos fijos que puede seleccionar el usuario a incluir en el cuerpo. Ver apartado: Incluir etiquetas fijas.
- **ETIQUETA_LISTA_IMAGEN**: Permite personalizar en <valor_propiedad> un texto diferente a la etiqueta del campo de imágenes prefijadas que pueden ser incluidas en el cuerpo. Ver apartado: Incluir imágenes.
- **VALOR_INICIAL_PARA**: Contenido que va tener el campo "Para" cuando se inicie el editor. Lleva implícito un ACTIVA_CAMPO_PARA.
- **VALOR_INICIAL_CC**: Contenido que va tener el campo "Con copia" cuando se inicie el editor. Lleva implícito un ACTIVA_CAMPO_CC.
- **VALOR_INICIAL_CCO**: Contenido que va tener el campo "Con copia oculta" cuando se inicie el editor. Lleva implícito un ACTIVA_CAMPO_CCO.
- **TEXTO_INICIAL_ASUNTO**: Texto que va a contener el campo Asunto de forma inicial, si se indica, automáticamente se activa el campo asunto, sin necesidad de indicar la propiedad ACTIVA_CAMPO_ASUNTO.
- **TEXTO_INICIAL_CUERPO**: Texto HTML que va mostrarse en el editor HTML al iniciarse.
- **PIXELS_ANCHO**: Permite indicar el ancho inicial de la ventana del editor, por defecto es 1024.
- **PIXELS_ALTO**: Permite indicar el alto inicial de la ventana del editor, por defecto es 768.

Incluir etiquetas fijas

Se permite añadir campos de tipo LIST-ITEM en donde se muestren constantes que el usuario podrá incorporar al los campos "Asunto" y "Cuerpo", estos campos pueden ser valores para ayudar al usuario a configurar el Parser, es decir, que el usuario seleccione "Nombre Cliente" y se incorpore el texto {clientes.nombre} en el punto en donde se encuentra el cursor .

Para incluir etiquetas hay que llamar al procedimiento PKEDITORHTML.ADD_ETIQUETA('<Texto de la etiqueta>', '<valor de la etiqueta>'); por cada etiqueta que se quiera incluir. Ejemplo:

```
PKEDITORHTML.ADD_ETIQUETA('Nombre Cliente', '{clientes.nombre}');
```

Incluir imágenes

Se pueden incluir imágenes en el cuerpo del mensaje, pero estas imágenes tienen que estar prefijadas en el mantenimiento de parámetros generales de menú (U_MPRMEN), en la pestaña "Imágenes Públicas".

En esa pantalla se introduce una descripción para la imagen, la URL de la imagen pública en Internet y opcional el ancho y alto en pixels al que debe de ajustarse la imagen.

Si se utiliza Forms 12c y está configurado un directorio público de imágenes, se pueden subir imágenes al servidor de aplicaciones mediante un plug-in en la botonera vertical.

Ejecutar y recuperar los datos introducidos por el usuario.

Para visualizar el editor hay que ejecutar el procedimiento PKEDITORHTML.MOSTRAR().

Una vez se cierra se pueden recuperar los datos del editor mediante la función PKEDITORHTML.GET_PROPIEDAD('<código_propiedad>);. Las propiedades disponibles son las siguientes:

- **ASUNTO**: Texto del campo Asunto.
- **CUERPO**: Texto en HTML del cuerpo.
- **PARA**: Valor del campo "Para".
- **CC**: Valor del campo "Con copia".
- **CCO**: Valor del campo "Con copia oculta".

Gestión de Archivos

Para gestionar archivos en una tabla únicamente es necesario que la tabla tenga los siguientes campos:

- **NOMBRE_ARCHIVO** de tipo VARCHAR2(500): El nombre del campo puede variar, no tiene que ser necesariamente NOMBRE_ARCHIVO, pero lo recomendable sería utilizar ese nombre. Este campo será el que será visible por el usuario y le hay que asignar la clase CLASE_ARCHIVO o CLASE_ARCHIVO_GRID dependiendo si el campo está o no en un multiregistro.
- **ID_ARCHIVO** de tipo NUMBER: Este campo no será visible al usuario, por lo que no tendrá especificado lienzo. En este campo se va almacenar el puntero al archivo que realmente almacenado en la tabla ARCHIVOS_ERP.

La subida y descarga de archivos los gestiona el entorno, lo único que hay que gestionar dentro del fuente es el borrado del registro. Al borrar un registro que tenga un campo con valor en ID_ARCHIVO hay que ejecutar el procedimiento pk_blob2bd.borra_archivo(<empresa>, <id_archivo>, <tabla>) en el disparador PRE-DELETE del bloque.

Este procedimiento recibe 3 parámetros:

- **<p_empresa>**. Código de la empresa, normalmente se pasará :global.codigo_empresa
- **<p_id_archivo>**. Valor del campo ID_ARCHIVO del bloque.
- **<p_tabla>**. Nombre de la tabla que tiene el nombre del archivo.
- **<p_usuario>**. Código del usuario, normalmente se pasará :global.usuario.

Ejemplo:

```
IF :b2.id_archivo IS NOT NULL THEN
  pk_blob2bd.borra_archivo(p_empresa => :global.codigo_empresa,
                           p_id_archivo => :b2.id_archivo,
                           p_tabla => 'CRMEXPEDIENTES_LIN_NOTAS',
                           p_usuario => :global.usuario);
END IF;
```

Borrar un archivo en la base de datos

El archivo debe de encontrarse en un directorio definido en ORACLE mediante CREATE DIRECTORY <NOMBRE_DIRECTORIO> AS '<ruta>';

Por ejemplo, CREATE DIRECTORY BLOB_TEMP AS '/u01/bte';

El directorio es necesario que tenga permisos de Oracle y de Java, para ello se le asignarán de la siguiente forma:

```
GRANT READ, WRITE ON DIRECTORY BLOB_TEMP TO <USUARIO_LIBRA>;
exec                        dbms_java.grant_permission('<USUARIO_LIBRA>','java.io.FilePermission','/u01/bt/*',
'read,write,execute,delete');
```

Por ejemplo:

```
GRANT READ, WRITE ON DIRECTORY BLOB_TEMP TO LIBRA;
exec dbms_java.grant_permission('LIBRA','java.io.FilePermission', '/u01/bt/*', 'read,write,execute,delete');
```

Si se han concedido correctamente los permisos para borrar un archivo simplemente hay que ejecutar la función:

```
PK_BLOB2BD.BORRAR_ARCHIVO_EN_DIRECTORIO('<DIRECTORIO>', '<NOMBRE_ARCHIVO');
```

Esta función devolverá OK si ha podido borrar el archivo y ERROR en caso contrario.

- <DIRECTORIO>: Código del directorio creado con CREATE DIRECTORY.
- <NOMBRE_ARCHIVO>: Nombre del archivo que se encuentra en la ruta a la que apunta el directorio <DIRECTORIO>

Obtener el listado de archivos de un directorio de la BD.

Mediante la función PK_BLOB2BD.GET_LISTA_ARCHIVOS('<directorio>'); devolverá una tabla de tipo PKPANTALLAS.VARCHAR2_TABLE con la lista de archivos que contiene el directorio. **Ejemplo**:

```
DECLARE
  t_archivos  PKPANTALLAS.VARCHAR2_TABLE;
BEGIN
  t_archivos := PK_BLOB2BD.GET_LISTA_ARCHIVOS('BLOB_TEMP');

  FOR i IN 1..NVL(t_archivos.LAST, 0) LOOP
    PKPANTALLAS.LOG('ARCHIVO: ' || t_archivos(i));
  END LOOP;
END;
```

Comprimir un archivo en la base de datos

El archivo debe de encontrarse en una carpeta con permisos (Ver apartado: Borrar un archivo en la base de datos, ya que los permisos son exactamente los mismos)

Para comprimir un archivo se ejecutará la función:

```
PK_BLOB2BD.COMPRIMIR_ARCHIVO('<DIRECTORIO>', '<ARCHIVO>', <P_BORRAR_ORIGINAL>);
```

Esta función devolverá OK si ha podido comprimir el archivo y ERROR en caso contrario. El archivo comprimido tendrá el mismo nombre, pero se le añade la extensión ".zip", por ejemplo, si se comprime "prueba.pdf" el archivo resultante será "prueba.pdf.zip".

- <DIRECTORIO>: Código del directorio creado con CREATE DIRECTORY.
- <ARCHIVO>: Nombre del archivo a comprimir y que se encuentra en la ruta a la que apunta el directorio <DIRECTORIO>
- <P_BORRAR_ORIGINAL>: Si se pasa TRUE y la compresión es correcta el archivo original se borra, en caso de pasar FALSE se mantiene.

Comprimir varios archivos en un único ZIP en base de datos

Todos los archivos deben de encontrarse en una carpeta con permisos (ver apartado: Borrar un archivo en la base de datos, ya que los permisos son exactamente los mismos).

En primer lugar, hay que indicar el directorio en donde se va a realizar la operación compresión, llamando al: pk_blob2bd.inicializar_compresion('<directorio>');

Ejemplo:

```
pk_blob2bd.inicializar_compresion('BLOB_TEMP');
```

Por cada archivo que se desea meter en el ZIP hay que llamar al procedimiento pk_blob2bd.agregar_archivo_compresion:

```
pk_blob2bd.agregar_archivo_compresion(<nombre_archivo>, <nombre_archivo_en_zip>);
```

- **<nombre_archivo>**: Nombre del archivo que se encuentra en el directorio indicado en "pk_blob2bd.inicializar_compresion".
- **<nombre_archivo_en_zip>**: Ruta y nombre que se le dará al archivo dentro del ZIP. En el caso de pasar NULL o no indicar este parámetro el archivo se meterá en el raíz del ZIP y con el mismo nombre que el indicado en el parámetro <nombre_archivo>.

Para ejecutar el proceso de compresión se llamará al procedimiento:

```
pk_blob2bd.comprimir(<nombre_archivo_destino>, <borrar_originales>);
```

- **<borrar_originales>**: Valores posibles:
  - **S**: En el caso de que el proceso de compresión sea realizado satisfactoriamente, los archivos originales serán borrados.
  - **F**: Se borran los archivos independientemente del resultado de la compresión.
  - **N**: Los archivos no serán borrados, aunque el resultado del proceso sea correcto.

- **<nombre_archivo_destino>**: Nombre del archivo que se generará. Al nombre indicado se le añadirá la extensión .ZIP de forma automática.

Descomprimir un archivo en la base de datos

- El archivo debe de encontrarse en una carpeta con permisos (ver apartado: Borrar un archivo en la base de datos, ya que los permisos son exactamente los mismos).
- Si se descomprime un archivo con directorios, el nombre del archivo será la concatenación del directorio (cambiando las barras de separación de directorio por guiones bajos) y el nombre del archivo.

Para descomprimir un archivo se ejecutará la función:

```
PK_BLOB2BD.DESCOMPRIMIR_ARCHIVO('<DIRECTORIO>', '<ARCHIVO>', <P_BORRAR_ORIGINAL>);
```

Esta función devolverá OK si ha podido descomprimir el archivo y ERROR en caso contrario. El archivo comprimido puede tener varios archivos en su interior, llamando a PK_BLOB2BD.GET_LISTA_ARCHIVOS(); devolverá una lista de tipo PKPANTALLAS.VARCHAR2_TABLE con los archivos que contenía el zip.

- <DIRECTORIO>: Código del directorio creado con CREATE DIRECTORY.
- <ARCHIVO>: Nombre del archivo a descomprimir y que se encuentra en la ruta a la que apunta el directorio <DIRECTORIO>
- <P_BORRAR_ORIGINAL>: Si se pasa TRUE y la descompresión es correcta el archivo comprimido original se borra, en caso de pasar FALSE se mantiene.

Ejemplo:

```
DECLARE
  t_archivos        pkpantallas.varchar2_table;
  t_archivos_zip    pkpantallas.varchar2_table;
  v_directorio      VARCHAR2(100) := 'BLOB_TEMP';
BEGIN
  t_archivos := pk_blob2bd.get_lista_archivos(v_directorio);

  FOR i IN 1..t_archivos.COUNT LOOP
    IF UPPER(SUBSTR(t_archivos(i), -4)) = '.ZIP' THEN
      --Descomprimimos el .ZIP y lo borramos
      IF pk_blob2bd.descomprimir_archivo(v_directorio, t_archivos(i), TRUE) = 'OK' THEN
        t_archivos_zip := pk_blob2bd.get_lista_archivos();

        FOR z IN 1..t_archivos_zip.COUNT LOOP
          pkpantallas.log('ARCHIVO: ' || t_archivos(i) || ' -> ' || t_archivos_zip(z));
        END LOOP;
      END IF;
    ELSE
      pkpantallas.log('ARCHIVO: ' || t_archivos(i));
    END IF;
  END LOOP;
END;
```

Impresión de archivos PDF

Para lanzar la impresión de un archivo PDF hay que añadir la librería PKLIBFILE al programa.

El archivo a imprimir puede encontrarse en un directorio físico del ordenador del usuario o en una dirección WEB.

Para realizar la impresión se ejecutará el procedimiento, STDFILE.IMPRIME_DOCUMENTO, este procedimiento recibe los siguientes parámetros:

- **p_archivo_o_url**: Ruta al archivo PDF a imprimir o a la URL donde se encuentra el documento.
- **p_tipo**: Si en p_archivo_o_url se indicó una ruta a un archivo local se deberá de pasar 'PDF', si se indicó una URL, hay que pasar 'URL_PDF'.
- **p_impresora**: Nombre de la impresora del sistema operativo por la que se quiere realizar la impresión.

Para imprimir un archivo almacenado en ARCHIVOS_ERP simplemente hay que usar la función STDFILE.IMPRIME_DOCUMENTO_ARCHIVOS_ERP(<empresa>, <id_archivo>, <tabla>, <impresion_por_windows>, <impresora>):

- <empresa>: Código de la empresa en la que está validado el usuario.
- <id_archivo>: Identificador del archivo de ARCHIVOS_ERP a imprimir.
- <tabla>: Tabla a la que está asociado el archivo.
- <impresion_por_windows>: Si se pasa TRUE, quiere decir que lo que se va a indicar en <impresora> es la cola del ordenador del usuario en la que se debe de realizar la impresión, si se pasa FALSE en <impresora> hay que pasar el código de una impresora lógica de Libra.
- <impresora>: Código de la impresora lógica de Libra o de la cola de impresión del ordenador donde se va a realizar la impresión, depende del parámetro <impresion_por_windows>.

Cambiar codificación de archivos de texto

Hay casos en donde es necesario generar archivos con codificación ANSI o UTF-8. En la librería PKLIBFILE existe la función para realizar la conversión de archivos usando la función:

```
STDFILE.CONVIERTE_CODIFICACION(p_codificacion_origen, p_codificacion_destino, p_nombre_archivo).
```

- **p_codificacion_origen**: Hay que indicar la codificación en la que se encuentra el archivo, valores posibles (si se pasa a NULL se intentará detectar de forma automática la codificación del archivo):
  - ISO-8859-1
  - UTF-8
  - UTF-8+BOM
- **p_codificacion_destino**: Código de la codificación a la que se quiere llevar el archivo, las codificaciones son iguales que las de origen (si se pasa a NULL se asumirá la codificación de la variable NLS_LANG del servidor de Forms.
- **p_nombre_archivo**: Ruta completa al archivo que se quiere convertir de codificación.

La función devuelve OK si la conversión se ha realizado correctamente, si se ha producido un error devolverá el motivo de este.

Consultar la codificación de un archivo de texto

Se dispone de la función STDFILE.GET_TIPO_CODIFICACION(archivo) que devolverá UTF8 o WE8ISO8859P1 según esté codificado el archivo, (En Forms 12c el archivo tiene que estar en el servidor de aplicaciones).

Obtener lista de archivos de un directorio

En base de datos

Para obtener el listado de archivos que se encuentran en un directorio de la base de datos se utiliza la función PKBLOB2BD.GET_LISTA_ARCHIVOS(<p_directorio>). Esta función devuelve un array de VARCHAR2 de tipo PKPANTALLAS.VARCHAR2_TABLE.

- **<p_directorio>**: Código del directorio creado con CREATE DIRECTORY

Ejemplo:

```
DECLARE
  t_archivos pkpantallas.varchar2_table;
BEGIN
  t_archivos := pkblob2bd.get_lista_archivos(p_directorio => 'BLOB_TEMP');

  FOR z IN 1..t_archivos_zip.COUNT LOOP
    pkpantallas.log('ARCHIVO: ' || t_archivos(i) || ' -> ' || t_archivos_zip(z));
  END LOOP;
END;
```

En equipo del usuario o en el servidor de aplicaciones

El programa ha de tener incorporada la librería pklibfile.pll. Para obtener el listado de archivos que se encuentran un directorio del equipo del usuario o del servidor de aplicaciones se utiliza la función: STDFILE.F_LISTA_ARCHIVOS_DIRECTORIO(<p_directorio>, <p_en_ias>, <p_desde_fecha>, <p_hasta_fecha>, <p_patron_nombre_archivo>). Esta función devuelve un array de VARCHAR2 de tipo PKPANTALLAS.VARCHAR2_TABLE.

- **<p_directorio>:** Directorio en donde se encuentran los archivos.
- **<p_en_ias>**: Si se pasa TRUE <p_directorio> hará referencia a una carpeta en el servidor de aplicaciones, si se pasa FALSE hará referencia a una carpeta en el equipo del usuario.
- **<p_desde_fecha>**: Filtro desde fecha de última modificación.
- **<p_hasta_fecha>**: Filtro hasta fecha de última modificación.
- **<p_patron_nombre_archivo>**: Filtro a aplicar sobre el nombre de archivos.

Ejemplo:

```
DECLARE
  t_archivos pkpantallas.varchar2_table;
BEGIN
  t_archivos := stdfile.f_lista_archivos_directorio(p_directorio => '/Users/usuario/Temp',
                                                    p_en_ias => FALSE,
                                                    p_desde_fecha => NULL,
                                                    p_hasta_fecha => TRUNC(SYSDATE -1),
                                                    p_patron_nombre_archivo => '*.png');
  FOR i IN 1..t_archivos.count LOOP
    pkpantallas.log(t_archivos(i));
  END LOOP;
END;
```

Gestión de fecha de última modificación de un archivo

Obtener fecha de un archivo en base de datos

Para obtener la fecha de última modificación de un archivo que se encuentra en la base de datos se utilizará la función PK_BLOB2BD.FECHA_MODIFICACION_ARCHIVO(<p_directorio>, <p_archivo>).

- **p_directorio**: Código del directorio creado con CREATE DIRECTORY
- **p_archivo**: Nombre del archivo del que se quiere obtener la fecha de última modificación.

Ejemplo:

```
DECLARE
   v_fecha   DATE;
BEGIN
  v_fecha    :=    pk_blob2bd.fecha_modificacion_archivo(p_directorio    =>    'BLOB_TEMP',    p_archivo    =>
'nombre_archivo.xml');
  pkpantallas.log('archivo con fecha: ' || TO_CHAR(v_fecha, 'DD/MM/YYYY'));
END;
```

Obtener fecha de un archivo en servidor de aplicaciones o en el equipo del usuario

Para obtener la fecha de un archivo que se encuentra en el servidor de aplicaciones o en el equipo del usuario se utilizará la función stdfile. fecha_ult_modificacion(<p_archivo>, <p_en_ias>) que se encuentra en la librería pklibfile.pll.

- **p_ archivo**: Ruta completa al archivo del que se quiere obtener la fecha de última modificación.
- **p_en_ias**: Si se pasa TRUE <p_archivo> hará referencia a un archivo en el servidor de aplicaciones, si se pasa FALSE hará referencia a un archivo en el equipo del usuario.

Ejemplo:

```
DECLARE
   v_fecha   DATE;
BEGIN
  v_fecha := stdfile.fecha_ult_modificacion(p_directorio => 'c:\temp\archivo.xml', p_en_ias => FALSE);
  pkpantallas.log('archivo con fecha: ' || TO_CHAR(v_fecha, 'DD/MM/YYYY'));
END;
```

Cambiar la fecha de última modificación de un archivo en servidor de aplicaciones o equipo del usuario.

Para cambiar la fecha de última modificación de un archivo que se encuentra en el servidor de aplicaciones o en el equipo del usuario se utilizará el procedimiento stdfile.set_fecha_ult_modificacion(<p_archivo>, <p_fecha>, <p_en_ias>) que se encuentra en la librería pklibfile.pll.

- **p_ archivo**: Ruta completa al archivo del que se quiere cambiar la fecha de última modificación.
- **p_fecha**: Fecha a asignar como fecha de última modificación del archivo indicado en p_archivo.
- **p_en_ias**: Si se pasa TRUE <p_archivo> hará referencia a un archivo en el servidor de aplicaciones, si se pasa FALSE hará referencia a un archivo en el equipo del usuario.

Ejemplo:

```
BEGIN
  stdfile.set_fecha_ult_modificacion(p_archivo => 'c:\temp\archivo.xml', p_fecha => SYSDATE, p_en_ias =>
FALSE);
END;
```

Parser de textos para reemplazar etiquetas

Se dispone del paquete PKBDPARSER que generar un texto partiendo de una plantilla que contiene etiquetas, esas etiquetas serán reemplazadas por los valores de campos de tablas de la base de datos

Por ejemplo, partiendo de esta plantilla:

Estimado Sr. {clientes.

} le informamos que dispone de la factura {facturas_ventas.numero_serie}/{facturas_ventas.numero_factura} de fecha {facturas_ventas.fecha_factura} disponible para descarga.

Una vez aplicado el paquete PKBDPARSER sobre la plantilla de ejemplo se obtendrá algo similar a esto:

ARIDOS LOPEZ E HIJOS le informamos que dispone de la factura FV/3433 de fecha 08/06/2011 disponible para descarga.

Para realizar el proceso del ejemplo hay que indicar qué factura es la que tiene usar para reemplazar el texto. El ejemplo para ejecutar el parseador para obtener el resultado del ejemplo sería el siguiente:

```
DECLARE
  v_resultado           VARCHAR2(30);
  v_cadena_parseada     CLOB;
BEGIN
  pkbdparser.inicializar();
  pkbdparser.set_variable('numero_factura', 3433);
  pkbdparser.set_variable('numero_serie', 'FV');
  pkbdparser.set_variable('ejercicio', '2011');
  pkbdparser.set_variable('empresa', '013');
  pkbdparser.set_propiedad_tabla('FACTURAS_VENTAS', 'WHERE_DEFECTO', 'numero_factura = {numero_factura} AND
numero_serie = {numero_serie} AND ejercicio = {ejercicio} AND empresa = {empresa}');
  pkbdparser.set_propiedad_tabla('CLIENTES', 'WHERE_DEFECTO', 'codigo_rapido = {facturas_ventas.cliente} AND
codigo_empresa = {empresa}');
  pkbdparser.set_propiedad('PLANTILLA', 'Estimado Sr. {clientes.nombre} le informamos que dispone de la factura
{facturas_ventas.numero_serie}/{facturas_ventas.numero_factura} de fecha {facturas_ventas.fecha_factura}
disponible para descarga.');
  v_resultado := pkbdparser.parsear_plantilla();
  v_cadena_parseada := pkbdparser.get_resultado_parseado();
END;
```

Tipos de etiquetas

Se contemplan los siguientes tipos de etiquetas

- **Variable**: Valor indicado previamente en una variable, tendrá el formato {variable}. Ejemplo: {numero_factura}
- **Valor de tabla**: Indica que debe sustituirse esa etiqueta por el valor del campo de una determinada tabla. Esta etiqueta tendrá el siguiente formato: {tabla.campo}. Ejemplo: {facturas_ventas.cliente}. En caso de estar en una zona de repetición, es decir, entre etiquetas de inicio de repetición y de fin de repetición se usará {alias.campo} en vez de {tabla.campo}

  Se pueden usar modificadores para alterar el resultado, en el caso de usar modificadores el formato será {alias.campo:modificador1|valor modificador1|modificador2|valor_modificador2|..|modificador n| valor modificador n}, por ejemplo: {crmexpedientes_cab.fecha_alta|FM|DD/MM/HH24:MI:SS}. Modificadores posibles:
  - FM: Máscara de formato.
  - TYPE: Tipo de campo, Valores posibles:
    - HTML, Se le aplica al resultado la función pk_xml.codifica_texto_to_html, para reemplazar los caracteres especiales del HTML por la codificación correcta. También se reemplazan los retornos de carro por
    - ESHTML: Se supone que se el resultado ya viene codificado en HTML y por tanto no se le debe de alterar.
- **De repetición**, se dividen en otras dos etiquetas:
  - **Inicio de repetición**: Tiene el siguiente formato. {*R:tabla:alias:condición:orden:R}*, dentro de condición se podrán usar etiquetas de *valor de tabla* o de *variable* (las del punto anterior). Ejemplo: {R:albaran_ventas_lin:AVL:articulo = {articulos.codigo_articulo} AND empresa = {empresa}:numero_albaran DESC:R}. En caso de no indicar una condición o una ordenación se dejará en blanco, es decir: {R:albaran_ventas_lin:AVL:::R}, en ese caso se usará la condición y la ordenación que tenga asignada la tabla por defecto.
  - **Fin de repetición**: Indica que todo lo que se encuentra entre "Inicio de repetición" y "Fin de repetición" se procesará tantas veces como filas devuelva la consulta sobre la tabla indicada en la etiqueta de inicio. Tendrá el siguiente formato: *{E:alias:E}*, ejemplo: {E:AVL:E}
- **Generador de informes**: Ejecuta un generador de informes y el resultado del informe lo incluye en el texto. El formato de la etiqueta es: {GI:<informe>:<idioma>:<empresa>:<usuario>:<plantilla de valores de filtro>:<tipo>:<código de configuración>}.
  - <informe>: Código del informe del generador de informes a ejecutar.
  - <idioma>: Idioma en el que se generarán las etiquetas de los campos.
  - <empresa>: Código de la empresa sobre la que se ejecutará el informe.
  - <usuario>: Usuario de Libra con el que se ejecutará el informe, el usuario determinará los permisos de ejecución.
  - <plantilla de valores de filtro>: Valores de filtro a aplicar al informe.
  - <tipo>: Si se indica HTML se incluirán al resultado etiquetas para maquetar el resultado en formato HTML. Si lo que se está formateando es un campo de tipo HTML lo detectará y no es necesario incluirlo. Si no se está formateando un campo de tipo HTML y no se indica nada en este campo el resultado será formateado en texto plano.
  - <código de configuración>: Si el informe tiene varias configuraciones de columnas, se puede indicar la que se debe de utilizar.

  Ejemplo: {GI:DIARIOS:01:013:EDISA:245:HTML:45} (Ejecuta el informe DIARIOS, con las etiquetas en idioma 01, sobre la empresa 013, con los permisos del usuario EDISA, aplicando la

plantilla de valores por defecto 245, el resultado se genera con etiquetas HTML y utilizando la configuración de columnas con código 45.

- **Función de base de datos:** Ejecuta la función de base de datos indicada que devuelva un resultado en VARCHAR2 o en CLOB, con el formato {SF:<función a ejecutar>:EF}. A la función se le pueden pasar los parámetros que sean necesarios obtenidos de variables, con {variable} o de un campo con {tabla.campo}. **Ejemplo**: {SF:PRUEBA_FUNCION(p_numero_serie => {facturas_ventas.numero_serie}, p_numero_factura => {facturas_ventas.numero_factura}):EF}
- **Imagen:** En parámetros generales de menú, en la pestaña "Imágenes Públicas" se pueden configurar enlaces a URLs que apunten a imágenes. Con el parseador se puede generar la etiqueta HTML que haga referencia a esa imagen con: {IMGHTML:CODIGO}, por ejemplo: {IMGHTML:GE5MOLW084CE8YLNPQFTI4DV5TN8F4}
- **Modificación de propiedades**, se dividen en:
    - **Variables**: Permite modificar en un momento dado una variable, tiene el siguiente formato: {SVA:<código de la variable>:<valor de la variable>:SVA}, Ejemplo: {SVA:codigo_consgen:CONSGEN:SVA}
    - **Propiedades generales**: Permite cambiar en un momento dado propiedades generales del parseador, tiene el siguiente formato: {SPG:<código de la propiedad>:<valor de la propiedad>:SPG}, Ejemplo: {SP:MASCARA_FECHAS:YYYY/MM/DD:SP}
    - **Propiedades de tablas**: Permite cambiar en un momento dado propiedades generales de una tabla, tiene el siguiente formato: {SPT:<código de tabla / alias>:<código de la propiedad>:<valor de la propiedad>:SPT}, Ejemplo: {SPT:PROGRAMAS_ERP:WHERE_DEFECTO:codigo={codigo_consgen}:SPT}
    - **Añadir columnas calculadas**: Permite añadir columnas calculadas a una tabla y ser usadas dentro del parseador como si fuese una columna más de la tabla, tiene el siguiente formato: {ACT:<nombre de tabla / alias>:<nombre de columna>:<sentencia SQL para obtener el resultado:ACT}. Ejemplo: {ACT:CRMEXPEDIENTES_CAB:NUMERO_LINEAS:(SELECT COUNT(*) FROM crmexpedientes_lin l WHERE l.numero_expediente = crmexpedientes_cab.numero_expediente AND l.empresa = crmexpedientes_cab.empresa):ACT}

Inicializar

Es obligatorio en primer lugar ejecutar la instrucción **pkbdparser.inicializar();.** Esta instrucción inicializa estructuras internas del paquete para realizar el proceso.

Propiedades generales del proceso

Para establecer las propiedades generales del proceso se usará el procedimiento **pkbdparser.set_propiedad('<codigo_propiedad>', '<valor>');**

Las propiedades que se pueden establecer son las siguientes:

- **MASCARA_FECHAS**: Máscara a aplicar a los campos de tipo fecha, en el caso de no especificar esta propiedad se usará DD/MM/YYYY
- **MASCARA_NUMEROS**: Máscara a aplicar a los campos de tipo numérico, en el caso de no especificar esta propiedad se realizará un TO_CHAR sin indicar ninguna máscara.
- **PLANTILLA**: Texto que contiene la plantilla a usar.
- **TRAZA**: Si se pasa el valor 'S' se guardará en LIBRA_LOG una traza interna para depurar del proceso.

Variables

Las variables son valores que no se obtienen de ninguna tabla, por lo que hay que suministrárselos al proceso, luego serán usadas en las etiquetas de tipo Variable.

Para declarar una variable hay que llamar al procedimiento **pkbdparser.set_variable('<codigo de la variable>', <valor>);**. Por ejemplo: **pkbdparser.set_variable('numero_factura', 3433);**

Propiedades de tabla

Para ejecutar el proceso hay que limitar los registros de las tablas que se usan y en el caso de bloques de repetición el orden en el que se procesarán, para ello se dispone del procedimiento **pkbdparser.set_propiedad_tabla('<alias>', '<propiedad>', '<valor>')**;. Por ejemplo: **pkbdparser.set_propiedad_tabla('CLIENTES', 'WHERE_DEFECTO', 'codigo_rapido = {facturas_ventas.cliente} AND codigo_empresa = {empresa}');**

- **<alias>**: Si es una tabla que se usa en un grupo repetitivo se indicará el alias del grupo repetitivo en vez del nombre de la tabla.
- **<propiedad>**: Existen las siguientes propiedades:
    - **WHERE_DEFECTO**: Condición que se aplicará al hacer la consulta de la tabla. Se pueden usar etiquetas de tipo variable o de valor de tabla. Si en <alias> se indicó el alias de un grupo repetitivo y en este grupo repetitivo se indicó una condición, esa condición prevalecerá sobre el valor pasado en esta propiedad.
    - **ORDER_BY_DEFECTO**: Condición de ordenación que se aplicará al consultar la tabla. Al igual que en WHERE_DEFECTO, si en el repetitivo se indica una condición de ordenación, esa condición de ordenación prevalecerá sobre la que se pasa en esta propiedad.

Propiedades de columna

Permite añadir columnas calculadas a una tabla y ser usadas dentro del parseador como si fuese una columna más de la tabla.

Para añadir una columna calculada hay que llamar al procedimiento pkbdparser.add_columna_tabla('<alias>', '<nombre de columna>', '<sentencia SQL para obtener el resultado>');

Ejemplo: pkbdparser.add_columna_tabla('CRMEXPEDIENTES_CAB', 'NUMERO_LINEAS', '(SELECT COUNT(*) FROM crmexpedientes_lin l WHERE l.numero_expediente = crmexpedientes_cab.numero_expediente AND l.empresa = crmexpedientes_cab.empresa)');

- **<alias>**: Si es una tabla que se usa en un grupo repetitivo se indicará el alias del grupo repetitivo en vez del nombre de la tabla.
- **<nombre de columna>**: Nombre de la columna, que será luego usada en el texto como alias.columna.
- **<sentencia SQL para obtener el resultado>**: Sentencia SQL que se debe de añadir a la SELECT para obtener el resultado de la columna. Se puede indicar que la sentencia se obtenga de la configuración de campos de tablas del generador de informes, para ello en este parámetro hay que indicar GI:<tabla>.<campo>, de esta forma se buscará la parametrización del campo en la configuración de tablas del generador de informes. Ejemplo: GI:CRMEMPRESAS.D_PAIS

Obtener el resultado

Para obtener el resultado en primer lugar hay que llamar a la función **pkbdparser.parsear_plantilla();**. Está función devolverá OK en caso de tener éxito y ERROR en caso de producirse algún fallo. Si se produce un fallo en LIBRA_LOG quedará el motivo del error.

Una vez se ejecutó el parser, si el resultado que se espera puede ser almacenado en un VARCHAR2, se llamará a la función: **pkbdparser.get_propiedad('RESULTADO_PARSEADO');** Para obtenerlo en una variable CLOB se llamará a la función: **pkbdparser.get_resultado_parseado();**

Variables y parámetros globales

Variables globales

Las variables globales que están definidas en todos los programas de libra son:

- **IDIOMA_USUARIO**: Código del idioma que tiene el usuario en su ficha. No es el idioma del mantenimiento de Idiomas, es el idioma para el que se buscan las traducciones de las etiquetas de los programas. Si en la ficha del usuario no tiene definido contendrá el valor '01'.
- **IDIOMA_EMPRESA**: De momento no tiene uso, siempre tiene le valor 01.
- **CODIGO_EMPRESA**: Código de la empresa que está validada.
- **NOMBRE_EMPRESA**: Nombre de la empresa que está validada.
- **FECHA_CONEXION**: Fecha y hora en formato CHAR con formato DD/MM/YYYY HH24:MI:SS en la que entró el usuario en Libra.
- **MENUS_PERFILES**: Código de menú del último programa en que entró el usuario. Es muy importante recalcar que es el último programa en que entró el usuario, ya que si el usuario entra en el programa A y por ventanas entra también en el programa B y vuelve al programa A el valor de esta variable contendrá el código del programa B.
- **FECHA_TRABAJO**: Fecha de trabajo en formato CHAR con el formato definido por :global.nls_date_format. Esta fecha se propone automáticamente en campos fecha que son obligatorios y que el usuario intenta dejar en blanco.
- **DESDE_PUESTO**: Contiene el valor de la variable LIBRA_ID de libra6.ini con el que inició sesión el usuario. Si no se especifica esa variable el menú al entrar pone el ella el valor de los primeros 30 caracteres del nombre del ordenador de Windows.
- **PERFIL**: Perfil principal del usuario, el que tiene asociado en la ficha.
- **SUPERUSUARIO**: Si el usuario tiene la marca de superusuario contendrá el código del usuario validado, si el usuario validado no es superusuario contendrá 'EDISA'.
- **USUARIO**: Código del usuario de Libra validado.
- **ROL_ACTIVO**: Si el usuario tiene activada la selección manual de Rol al iniciar sesión, esta variable contendrá el código de ROL con el que se ha validado el usuario.
- **USUARIO_SO**: Usuario del sistema operativo del equipo en el que se esta ejecutando Libra.
- **EQUIPO_SO**: Nombre del equipo en el que se está ejecutando Libra.
- **IMPRESORA_WINDOWS**: Cuando entra en el menú se inicializa con el nombre de la impresora predeterminada que tiene el usuario en su ordenador. El usuario la puede cambiar para esa sesión de libra (es decir, mientras no salga de libra) por otra y se cambia el valor de esta variable.
- **PRUEBA_FALLOS**: Contendrá el valor S si libra se está ejecutando en modo a prueba de fallos y N en caso contrario. El modo a prueba de fallos consiste en que no se ejecutará ninguna personalización de los programas, únicamente lo estándar.
- **TRAZA**: Contendrá el valor SI en el caso de que se esté ejecutando en modo de traza y NO en caso contrario. Hay algunos programas que si activas en el menú en modo traza generan un archivo .log del proceso que realizan.
- **REG_FALLOS_TRADUCCION**: Si tiene el valor S durante la traducción de los campos a otros idiomas distintos de castellano la traducción no existe en etiquetas_erp_traducidas se guarda el fallo en etiquetas_erp_fallo_traduccion.
- **SEP_DIR**: Separador de directorios en servidor de aplicaciones.
- **SID_ESCRITORIO**: Sid de Oracle con el que está conectado el Menú.
- **IAS_CLIENTOSNAME**: Contiene el nombre del sistema operativo del equipo cliente que ejecuta Libra.

En programas dinámicos hay también un campo muy útil para personalizaciones **:PLANTILLA.CODIGO_PLANTILLA**, contiene el código de la plantilla que tiene seleccionada el usuario.

Parámetros disponibles para personalizaciones

A partir de la versión 5.3.2 de Libra, los programas disponen de 10 parámetros alfanuméricos PARAMETER.PAxx (por ejemplo, PARAMETER.PA01) y 5 numéricos PARAMETER.PNxxx (por ejemplo, PARAMETER.PN01) que pueden ser usados en cualquier personalización, tanto en códigos pl/sql como para el paso de valores a estos parámetros en plug-ins.

Variables globales accesibles mediante pkpantallas

Estas variables están accesibles tanto desde los programas, desde los procedimientos almacenados en la base de datos, vistas, etc. Su utilidad es múltiple, como hacer vistas parametrizadas por el usuario validado, hacer sqls con autorización de permisos en el generador de informes, etc

- **Pkpantallas.usuario_validado**: Devuelve el usuario que ha iniciado sesión, es decir, el equivalente a la variable :global.usuario.
- **Pkpantallas.superusuario**: Si el usuario validado es superusuario lo devuelve, si no es superusuario devuelve EDISA, es el equivalente a la variable :global.superusuario.
- **Pkpantallas.perfil_usuario_validado**: Devuelve el perfil principal del usuario validado, es el equivalente a la variable :global.perfil.
- **Pkpantallas.programa_validado**: Devuelve el nombre del programa que se está ejecutando, es equivalente a la variable :parameter.id_programa.
- **Pkpantallas.idioma_usuario_validado**: Devuelve el idioma del usuario que está validado, es equivalente a la variable :global.idioma_usuario.
- **Pkpantallas.id_personalizacion**: Devuelve el identificador de la personalización de PROGRAMAS_ERP_PRES que se está ejecutando. No hay equivalente en variable global.
- **Pkpantallas.get_valor_ultima_ejecucion_lov**: Ver sección código PL/SQL.
- **Pkpantallas.sector_empresa**(<empresa>): Devuelve el sector de la empresa que se pasa por parámetro, es necesario pasar la empresa ya que se puede usar tanto en programas de nueva como de vieja estética, internamente la consulta a la base de datos solo la hace la primera vez, luego ya deja cargado el valor. No tiene equivalente en variable global.
- **Pkpantallas.get_usuario_so**: Usuario del sistema operativo que está ejecutando Libra.
- **Pkpantallas.get_equipo_so**: Nombre del equipo que esta ejecutando Libra.

Definibles dinámicamente

Se pueden definir variables de forma dinámica, desde el fuente de un programa o desde código pl-sql del mantenimiento de programas, para ello tenemos las siguientes funciones y procedimientos:

- **Pkpantallas.set_variable_env**(<variable>, <valor>): Asignamos en la variable '<variable>' el valor de '<valor>'. Ejemplo: PKPANTALLAS.SET_VARIABLE_ENV('PRUEBA','VALOR_PARA_PRUEBA');
- **Pkpantallas.get_variable_env_varchar2**(<variable>): Devuelve el valor de la variable '<variable>', siempre y cuando se le hubiese asignado un VARCHAR2, si se le pasó un DATE o un NUMBER hará la conversión a VARCHAR2. Ejemplo: PKPANTALLAS.GET_VARIABLE_ENV_VARCHAR2('PRUEBA'), devolvería 'VALOR_PARA_PRUEBA' si ejecutamos antes el ejemplo del punto anterior.
- **Pkpantallas.get_variable_env_number**(<variable>): Devuelve el valor de la variable '<variable>', siempre y cuando se le hubiese asignado un NUMBER.
- **Pkpantallas.get_variable_env_date**(<variable>): Devuelve el valor de la variable '<variable>', siempre y cuando se le hubiese asignado un DATE.
- **Pkpantallas.inicializar_variables_env**: Borra todas las variables definidas.

Variables de inicio de libra.env

- FORMS_PATH: Camino que usa libra para buscar los programas y las librerías. Se especificarán los directorios separados por dos puntos ':' y se buscaran los programas comenzando por el primer directorio especificado, sino se encuentran por el siguiente, ...
- REPORTS_PATH: Únicamente se utiliza para localizar los informes de Oracle Reports con traducciones a otros idiomas. En el caso de no ser necesarios informes en idiomas esta variable no es necesaria, ya que el propio servidor de Reports tiene las rutas a los informes de forma independiente a Oracle Forms.
- PATH: Camino en los que se buscan los ejecutables del sistema operativo para ejecuciones en el servidor de aplicaciones.
- NLS_LANG: Idioma:
- NLS_DATE_FORMAT: Formato de la fecha.
- NLS_NUMERIC_CHARACTERS: Si se especifica ,. se supone que , es el separador de decimales y . el de millares.
- NLS_SORT: Tipo de ordenación que aplicará Oracle, normalmente se especificará BINARY.
- DIRECTORIO_SALIDA: Directorio que usan múltiples programas de Libra para generar salidas a archivos. Su uso está considera obsoleto.
- LIBRA_ID: Puesto desde el que se ejecuta Libra, debe de existir en la tabla de puestos para que funcionen correctamente las listas de valores de impresoras.
- IDIOMA_USUARIO: Idioma en que presentará la pantalla de login.
- REG_FALLOS_TRADUCCION: Si se pone S se activa por defecto la opción de menú de registrar fallos de traducción.
- REP2EXCEL_PARAM: Parámetros con los que se llamará a rep2excel para convertir el archivo html en xls.
- PROGRAMA_INICIO: Nombre del fmx que se ejecutará según se valide el usuario, si el usuario tiene especificado un programa de inicio prevalecerá sobre el que se especifique en esta variable.
- PROGRAMA_FIN: Programa que se ejecutará al salir de Libra, si el usuario tiene especificado un programa de fin prevalecerá sobre el que se especifica en esta variable.
- DIRECTORIO_SALIDA_REP_FILE: Directorio que se propondrá por defecto cuando se seleccione el envío de informes a fichero.
- DIRECTORIO_SALIDA_REP_FAX: Directorio que se propondrá por defecto cuando se seleccione el envío de informes a fax.
- DIRECTORIO_SALIDA_REP_GESTDOC: Directorio que se propondrá por defecto cuando se seleccione el envío de informes a gestor documental.
- DESACTIVAR_RF: Si se asigna el valor S, no hace falta tener compilado ni que exista nada relacionado con la radiofrecuencia para que funcione el menú.
- COMANDO_REP2EXCEL: Indicar el comando completo para convertir un html en un xls, si no se especifica nada se usa los valores por defectos de rep2excel. Del comando de sustituyen <archivo_html> por el nombre del archivo con su ruta generado por el report, y <archivo_xls> lo sustituirá por la ruta y nombre del archivo XLS que debe generar. Ejemplo: COMANDO_REP2EXCEL=start http://your_server_name:7777/cgi-bin/rep2excel.exe?baseidr=<archivo_html>
- REP2EXCEL_PARAM: Esta variable es incompatible con COMANDO_REP2EXCEL, si se especifican ambas se usarán únicamente COMANDO_REP2EXCEL, siendo REP2EXCEL_PARAM ignorada. En esta variable podremos añadir parámetros al comando rep2excel.
- REP2EXCEL_DESACTIVAR_BORRADO: Para generar excel, rep2exel se basa en el archivo html generado por reports, si a esta variable se asigna el valor S el archivo html no se borra una vez hecha la llamada al rep2excel, además al archivo html y xls se añade el nombre del usuario y la fecha para evitar que dos usuarios concurrentes generen el mismo archivo en un momento dado.

- PAGESIZE_EXCEL: Se puede usar para indicar a reports el tamaño de la página cuando se envía a excel, es útil para evitar el paginado en el informe, por ejemplo: PAGESIZE_EXCEL=39 X 1300
- DIRECTORIO_LOG: Directorio en donde se guardarán los archivos de traza.
- COMANDO_FAX: Comando de sistema operativo que se ejecutará cuando el usuario selecciona fax y reports ya ha generado el archivo se pueden usar las siguientes variables:
    - <fax>: Se sustituye por el número de fax al que se va a enviar el informe.
    - <archivo_pdf>: Nombre del archivo y ruta del fichero .pdf generado.
    - <archivo_ps>: Nombre del archivo y ruta del fichero .ps generado.
- COMANDO_PS2PS2: Se usa cuando se especificó el COMANDO_FAX, el comando que se parametriza en esta variable se usa para convertir el archivo .pdf generado por forms a .ps necesario en muchos programas de envío por fax. Se pueden usar las siguientes variables:
    - <archivo_pdf>: Nombre del archivo y ruta del fichero .pdf generado.
    - <archivo_ps>: Nombre del archivo y ruta del fichero .ps que se va a generar.
- CENTRAR_LOV: Si se asigna el valor N las listas de valores no se centran en pantalla y se muestran en la posición 0,0.
- PROGRAMA_DESTINO_MENSAJE: Cada vez que se muestre un mensaje, en vez de hacerlo por el método estándar de forms se llama al programa que se indica en esta variable.
- TIPO_PUESTO: Posibles valores:
    - N: Normal, es el valor por defecto si no se especifica nada.
    - T: Optimizado para el funcionamiento en terminal server con tamaño de menú normal.
    - P: Optimizado para el funcionamiento en terminal server con tamaño de ventana optimizado para pocket.
- DESACTIVAR_CIERRE_CON_ESC: Si se asigna el valor N se fuerza a que con la tecla ESC se salga de Libra cuando se está en el menú y no hay ningún programa abierto.
- VENTANA: Indica como se quiere que abra la ventana de Libra:
    - MAX: Maximizada, este el el valor por defecto en caso de no indicar esta variable.
    - MIN: Minimizada.
    - NOR: Normal, en este caso se puede indicar el tamaño en pulgadas que se quiere que tenga la ventana y la posición X, Y en donde debe de abrirse, para ello hay que especificar: VENTANA=NOR:X:Y:ANCHO:ALTO, por ejemplo: VENTANA=NOR:0:0:8,3:5,4 (Se abrirá Libra en la posicion X = 0 e Y = 0, ANCHO = 8,3 y ALTO = 5,4)
- EXTENSIONES_VISUALIZACION: Lista de **extensiones** de archivo separadas por comas que se pueden visualizar en el equipo en donde se está ejecutando Libra. Si en parámetros generales de menú está cubierto esta variable es ignorada.
- ACTIVA_TRAZA: Si se indica el valor S, se activa la traza desde un principio, esto es especialmente útil para trazar la pantalla de Login.

Desarrollo de aplicaciones para pocket - Terminal Server

Configuración del entorno.

El principal problema de la ejecución en Terminal Server o Citrix en un dispositivo Pocket es el tamaño de la pantalla física y que suele haber una pantalla lógica más grande y para recorrerla hay que usar las barras de scroll, algo muy engorroso para los usuarios.

Al ser la pantalla lógica más grande que la física, todo lo que salga centrado posiblemente se vea en una zona de la pantalla que no se está visualizando en ese momento y para verlo el usuario tendrá que hacer scroll.

El objetivo que hay que plantearse, aparte de un menú más pequeño, que todo salga en la posición 0,0 de la pantalla para que lo pueda visualizar bien el usuario. Para ello se debe modificar en el mantenimiento de puestos desplegable “Tipo Puesto” e indicar “Pocket”.

Desarrollo o adaptación de un programa a pantalla pequeña.

El desarrollo hay que tener en cuenta los siguientes puntos:

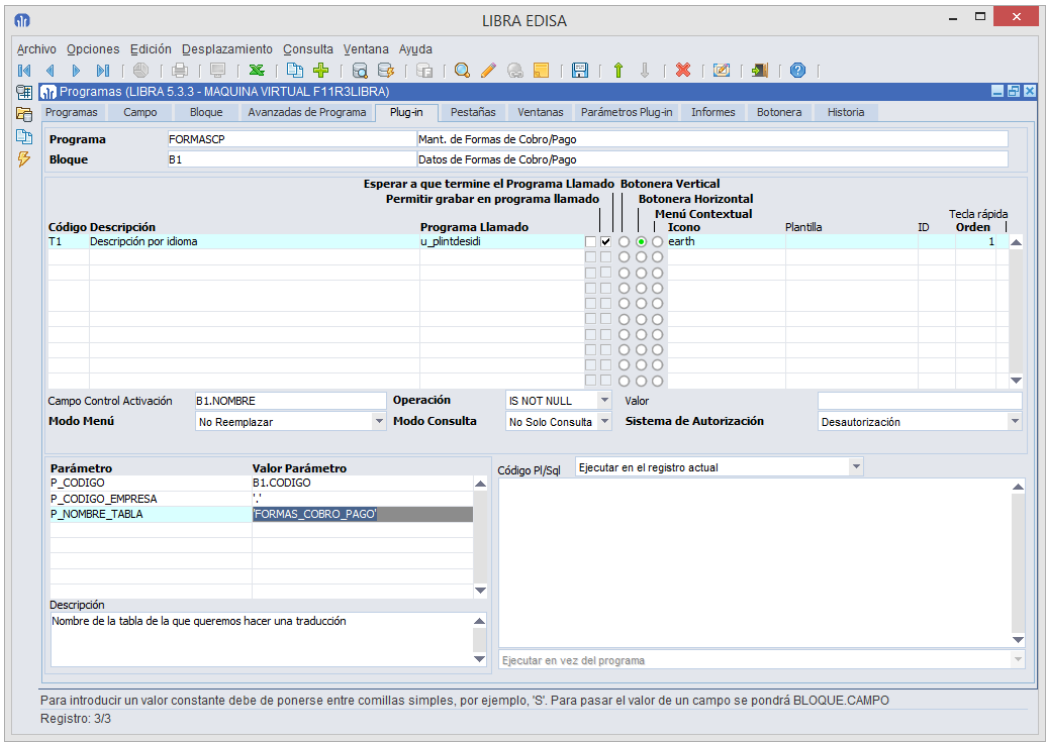
- Cambiar en las propiedades del formulario el “menu6” por “menu6_pocket” o uno adaptado a las necesidades con las opciones que se quieran dar por menú a los usuarios.
- Se ajustará el tamaño de la ventana “VENTANA” y del lienzo “CANVAS_BASE” al tamaño de la pantalla de los pocket.
- Añadir a los programas el grupo de objetos LISTA_VALORES_GRUPO y cuando se asigna en el mantenimiento de programas la lista de valores en la pestaña “Campo” en “Lista de valores por grupos” seleccionar Si – 9 registros o Si – 5 registros dependiendo del tamaño de la pantalla. Se pueden usar listas de valores normales, pero las de grupo son mucho más cómodas para usar con el lápiz del pocket.
- En las listas de valores activamos las listas de valores por grupo. Seguramente se tengan que personalizar las listas de valores para optimizar el tamaño de las columnas.

Localización de descripciones de tablas de parametrización

De la traducción de las pantallas al idioma del usuario se encarga totalmente el entorno de Libra, pero para traducir tablas específicas de otros procesos simplemente proporciona ayuda para hacer dicha tarea.

Para ello se dispone del plug-in "u_plintdesidi" que se puede aplicar a cualquier programa para guardar las traducciones por idioma.

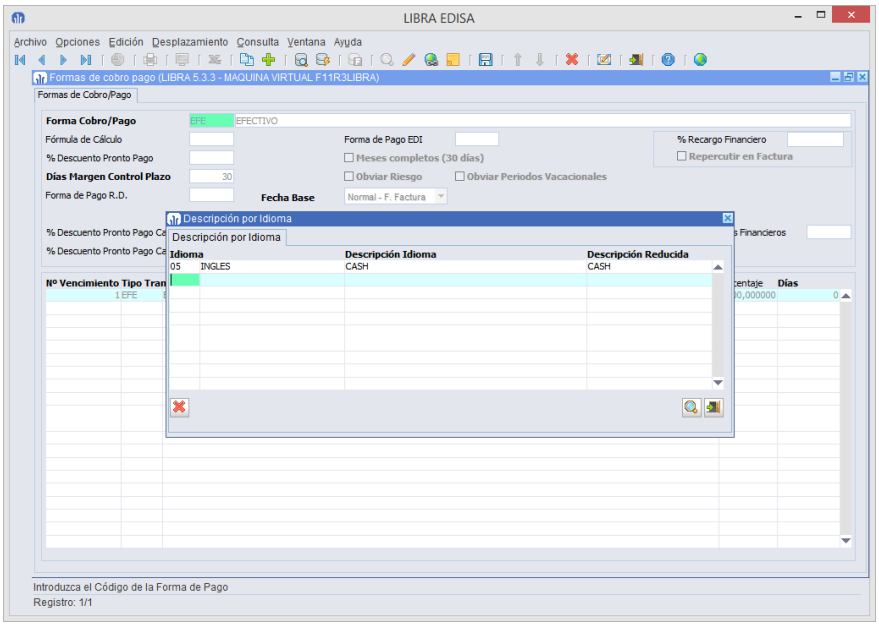
Por ejemplo, para las formas de cobro / pago se añadiría el plug-in de la siguiente forma:



Particularidades de la configuración del plug-in:

- **Permitir grabar en programa llamado**: Se desactiva esta check, ya que en el plug-in donde se gestionan las traducciones el usuario no puede grabar, para grabar los datos introducidos lo hará desde el programa llamador, en este caso, desde el mantenimiento de formas de cobro pago.
- **Botonera Horizontal**: Este plug-in se define en la botonera horizontal para tenerlo bien diferenciado del resto de plug-ins.
- **Icono**: Ya propone "world", es recomendable dejar este para que en todos los procesos tenga el mismo icono.
- **Modo Menú**: Al ser un programa con una ventana flotante, para conseguir que una mejor integración con el programa llamador le indicamos "No Reemplazar".
- **Parámetros**:
    - **P_CODIGO**: Se indica el campo en el que se encuentra el código del registro del que se quiere traducir la descripción.
    - **P_CODIGO_EMPRESA**: En este caso al ser una tabla que los registros son comunes a todas las empresas de Libra se le pasa '.', en el caso de ser una tabla que la codificación fuese independiente de por empresa se pasaría GLOBAL.CODIGO_EMPRESA
    - **P_NOMBRE_TABLA**: Nombre de la tabla de la que se quiere gestionar las traducciones, en este caso es FORMAS_COBRO_PAGO, por eso se pasa 'FORMAS_COBRO_PAGO'.

El resultado será el siguiente:



Para gestionar el borrado, de forma de que al borrar en la tabla padre borre las traducciones asociadas, hay que meter en el PL/SQL de “Post Borrado” lo siguiente:

```
PKPANTALLAS.BORRAR_TRADUCCIONES_TABLA('.', :b1.codigo, 'FORMAS_COBRO_PAGO');
```

El primer parámetro recibe el código de la empresa, al ser una tabla que no se define por empresa se le pasa '.', en caso de ser una tabla que se definiese por empresa se pasaría :global.codigo_empresa;

Para recuperar la traducción se llamará la función: PKPANTALLAS.BUSCAR_TRADUCCION_TABLA ('<empresa>', '<código>', '<tabla>', '<código de idioma>', '<descripción sin traducir>', '<tipo');

- <empresa>: Si los registros de la tabla no van por empresa se pasará '.' (punto), en otro caso el código de la empresa.
- <código>: Código del registro del que se busca la traducción.
- <tabla>: Tabla donde está almacenada la descripción a traducir.
- <código de idioma>: Código de idioma definido en la tabla IDIOMAS, en el que se quiere obtener la descripción en idioma.
- <descripción sin traducir>: Texto de la descripción sin traducir, se devolverá en el caso de que no se encuentre una traducción específica para el idioma indicado en <código de idioma>.
- <tipo>: 'R' para obtener la traducción reducida y 'D' para la completa.

Ejemplo:

```
pkpantallas.buscar_traduccion_tabla('.', 'EFE', 'FORMAS_COBRO_PAGO', '02', 'EFECTIVO', 'D')
```

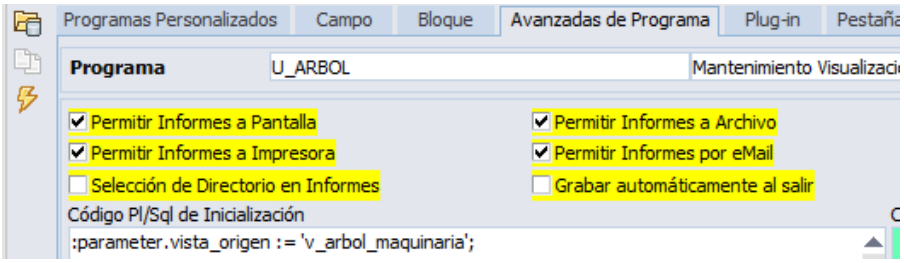
Consulta de datos jerárquicos

Mediante el programa U_ARBOL se pueden realizar consultas jerárquicas de tipo árbol. Por cada consulta se debe de generar una nueva personalización del programa U_ARBOL.

El origen de los datos tiene que estar normalizado, para ello se puede usar una vista con una estructura fija:

```
CREATE OR REPLACE FORCE VIEW v_...(empresa, codigo, descripcion, codigo_padre, reservadoa01, d_reservadoa01, reservadoa02,
                                   d_reservadoa02, reservadoa03, d_reservadoa03, reservadoa04, d_reservadoa04, reservadoa05,
                                   d_reservadoa05, reservadoa06, d_reservadoa06, reservadoa07, d_reservadoa07, reservadoa08,
                                   d_reservadoa08, reservadoa09, d_reservadoa09, reservadoa10, d_reservadoa10, reservadon01,
                                   d_reservadon01, reservadon02, d_reservadon02, reservadon03, d_reservadon03, reservadon04,
                                   d_reservadon04, reservadon05, d_reservadon05, reservadon06, d_reservadon06, reservadon07,
                                   d_reservadon07, reservadon08, d_reservadon08, reservadon09, d_reservadon09, reservadon10,
                                   d_reservadon10, icono, icono02, reservado_check01, reservado_check02, reservado_check03,
                                   reservado_check04, reservado_check05, reservado_check06, reservado_check07,
                                   reservado_check08, reservado_check09, reservado_check10, reservado_fecha01,
                                   reservado_fecha02) AS
```

Esa vista se indicar en el código pl/sql de inicialización en la personalización a nivel de programa:



Otra forma es modificar la consulta del bloque B2, en la personalización y meter ahí la SELECT equivalente.

En la personalización se pueden añadir hasta 10 filtros alfanuméricos, 10 filtros numéricos, 2 de fecha y dos filtros códigos fijos (numérico o carácter según si la tabla que estemos utilizando el código es numérico o carácter).

Estos filtros se encuentran en el bloque B1 y por defecto están ocultos, en la personalización se cambiará la check “Ocultar”, la etiqueta y se añadirá si es necesario alguna lista de valores.

Los datos adicionales que se quieran visualizar, hay que personalizarlos en los campos del bloque B2, cambiando la check “Ocultar” y la etiqueta.

En el bloque B2 se pueden añadir plug-ins para ampliar la información mostrada, según la rama del árbol seleccionada.

El programa admite parámetro el parámetro codigoA (código carácter) o codigoN (código numérico) para ser llamado como plugin y que salga el bloque del árbol directamente filtrado por el código.

Gráficos integrados en Programas

Esta funcionalidad sólo funcionará cuando el programa se ejecute en Forms 12c.

En el programa que utilice esta funcionalidad hay que incluir la librería: pklibgraf.pll, y en los bloques hay que añadir campos con la clase de propiedad CLASE_GRAFICO

Para representar el gráfico es necesario partir de una SQL que obtenga los datos a mostrar. La sql deberá de devolver los registros lo más agrupados posibles, es decir, si queremos las ventas por clientes, será: SELECT cliente, SUM(importe) importe FROM tabla GROUP BY cliente. En vez de SELECT cliente, importe FROM tabla.

Inicializar Gráfico.

El primer paso para utilizar un campo de gráfico es inicializarlo. Esta inicialización se hará llamando a la función: pkgraficos.inicializar('BLOQUE.CAMPO_GRAFICO'). Esta función devolverá un identificador numérico que deberá ser almacenado en una variable, ya que ese valor identificará el gráfico para poder aplicarle propiedades. Ejemplo: :parameter.id_grafico := pkgraficos.inicializar('B1.GRAFICO');

Propiedades a nivel Gráfico.

Para modificar propiedades a nivel de gráfico se utilizará el procedimiento pkgraficos.set_propiedad(<id_grafico>, <propiedad>, <valor>).

- **<id_grafico>**: Identificador del gráfico obtenido por pkgraficos.inicializar.
- **<propiedad>**: Código de la propiedad a modificar del gráfico.
  - **DEVOLVER_VALOR_SELECCIONADO**: Cuando el usuario hace click sobre algún elemento del gráfico, podemos indicar que es lo que queremos que nos devuelva (Valores posibles para <valor>):
    - 'ALL': Todo
    - 'ROWLABEL': Etiqueta de fila.
    - 'COLUMNLABEL': Etiqueta de columna.
    - 'CELLVALUE': Valor representado.
    - 'PRIMARY_KEY': Código asignado.

REGISTRAR_TRIGGER: Cuando interesa recoger el valor seleccionado por el usuario hay que registrar un TRIGGER que deberá existir en el programa (trigger de usuario), cuando el usuario pulsa sobre algún componente del gráfico se disparará ese trigger, en el cual se deberá recoger el valor con la función pkgraficos.get_valor_devuelto(). En <valor> de la propiedad se indicará el nombre del trigger. El valor devuelve depende de lo que se indique en DEVOLVER_VALOR_SELECCIONADO. Si no se ha indicado la propiedad DEVOLVER_VALOR_SELECCIONADO el registro del trigger será ignorado. También hay que añadir en el programa en el disparador WHEN-CUSTOM-ITEM-EVENT la siguiente línea: PKGRAFICOS.WHEN_CUSTOM_ITEM_EVENT;

- **TIPO_GRAFICO**: Tipo de gráfico a representar, en <valor> se puede indicar uno de los siguientes tipos: HORIZONTAL_BAR, HORIZONTAL_BAR_2Y, VERTICAL_BAR, VERTICAL_BAR_2Y, VERTICAL_STACKED_BAR, HORIZONTAL_STACKED_BAR, VERTICAL_PERCENT_BAR, HORIZONTAL_PERCENT_BAR, VERTICAL_LINE_GRAPH, HORIZONTAL_LINE_GRAPH, RING_BAR, VERTICAL_STACKED_LINE_GRAPH, HORIZONTAL_STACKED_LINE_GRAPH, VERTICAL_AREA_GRAPH, VERTICAL_PERCENT_AREA_GRAPH, VERTICAL_STACKED_AREA_GRAPH, PIE_GRAPH, PIE_BAR_GRAPH, MULTI_PIE_GRAPH' COMBINATION_GRAPH, 3D_BAR_GRAPH, 3D_AREA_GRAPH
- **TEXTO_TITULO**: Título que se mostrará en la cabecera del gráfico. En <valor> se indicará el texto.
- **TOOLTIPS**: Se indicará el comportamiento deseado cuando el usuario pase con el ratón sobre alguna parte del gráfico. En <valor> se pueden indicar los siguientes valores:
  - 'NONE': No se muestra nada.
  - 'ALL': Mostrará toda la información disponible.
  - 'LABELS': Muestra las etiquetas de la columna.
  - 'VALUES': Se muestra el valor del dato representado.

- **SHOW_GRID**: Permite activar o desactivar la rejilla de fondo del gráfico. Valores posibles: TRUE, FALSE
- **MOSTRAR_COLUMNAS_EN_FILAS**: Permite permutar las filas por columnas. Valores posibles: TRUE, FALSE
- **TRAZA**: Activa la salida de información de traza para el gráfico. Valores posibles: TRUE, FALSE

Añadir SQL a Gráficos

Un gráfico por lo general contendrá una única SQL. La SQL será la que determine los valores a representar y debe tener las siguientes características:

- Tantos campos numéricos como valores a representar en el gráfico.
- Un campo para obtener el valor del título de la serie a representar.

Para añadir una SQL al gráfico se utilizará la función: pkgraficos.add_sql(<id_grafico>, <sql>'); Esta función devolverá un identificador numérico que será necesario guardar en una variable para luego asignar propiedades a la SQL.

Propiedades de SQL.

Para modificar propiedades a nivel de SQL se utilizará el procedimiento pkgraficos.set_propiedad_sql(<id_grafico>, <id_sql>, <propiedad>, <valor>).

- **<id_grafico>**: Identificador del gráfico obtenido al ser inicializado por pkgraficos.inicializar.
- **<id_sql>**: Identificador de la sql, obtenido en pkgraficos.add_sql.
- **<propiedad>**: Código de la propiedad a modificar de la sql.
    - **COLUMNA_TITULO**: Identifica la columna que contendrá el título de las series a representar. En <valor> se indicará el número de columna de la SQL que contiene este dato.
    - **COLUMNA_PRIMARY_KEY**: Esta propiedad es opcional, y se utiliza cuando se registra un trigger en el gráfico y se indica que se debe de devolver la clave primaria del valor seleccionado por el usuario. En <valor> se indicará el número de columna de la SQL que contiene la clave primaria.
    - **MASCARA_FORMATO_COLUMNA_TITULO**: Si en COLUMNA_TITULO se indicó una columna de tipo DATE, se puede modificar la máscara de formato. En <valor> se indicará una máscara de formato válida.

Mostrar el gráfico

Una vez asignadas las propiedades necesarias para representar el gráfico, hay que invocar al procedimiento: **pkgraficos.mostrar_grafico(<id_grafico>);**

EDISA

LATINOAMÉRICA	ESPAÑA
COLOMBIA	MADRID
ECUADOR	BARCELONA
MÉXICO	VALENCIA
REP. DOMINICANA	VIGO
	OVIEDO
	LAS PALMAS
	OURENSE (CENTRO I+D)