TABLA DE CONTENIDO

CAPITULO VI PROGRAMACIÓN

DPXBASE es un ambiente de desarrollo para léxico de tipo Xbase y diseñado para crear aplicaciones de tipo comercial de manera rápida y sencilla. Además incorpora un conjunto de componentes tecnológicos que logran aportar robustez en las aplicaciones finales. Dentro de ellas tenemos:

1. Lenguaje de Programación xHarbour para 32 bits y C++ de Borland.
2. Acceso a Bases de Datos de tipo SQL a través de ODBC o Nativo MySQL.
3. Uso de Integridad Referencial en las bases de datos.
4. Reportes a Través de Crystal Report.
5. Generador de Reportes para la clase Treport.
6. Generador de Código Fuente para Formularios de múltiples propósitos: Mantenimiento de registros, Ventanas de Diálogos, Ventanas MDI.

DPxbase, posee un (DataBase IDE) que permite definir todas las tablas “SQL” que serán utilizadas por el sistema de manera jerárquica: Campos, Índices, Claves Primarias y referencias entre tablas. Además; importa estructuras desde formatos DBF.

Éste posee la siguiente barra de herramientas:

FICHEROS

Agrupa todas las tablas maestras de este módulo, en las que se especificarán las definiciones de aplicaciones o módulo del sistema, tablas de datos, menú de acceso, programas fuentes DPxbase, botones de la barra del menú principal y enlaces entre tablas.

APLICACIONES O MODULO DEL SISTEMA

 Permite modificar y eliminar los módulos existentes, así como también crear nuevos módulos al sistema. Los efectos se verán en “Aplicaciones”.

TABLA DE DATOS

El usuario selecciona la tabla de datos y por cada campo debe indicar el tipo de control “GET, COMBO, RADIO, CHECK, Etc.” y atributos para cada campo, que será utilizado para crear el programa fuente xBase permitiendo la ejecución de un formulario de carga de datos que incluye toda la funcionalidad necesaria para: Incluir o Modificar un registro de una tabla SQL. DPxbase provee la clase TDPEDIT que controla el formulario con la tabla de datos y el navegador de registros, posee métodos "methods" para: Iniciar, Grabar, Cancelar, Imprimir, Validar, enlaces, campos únicos y un sin número de utilidades. Cuando DPxbase detecta que un campo esta referenciado con otra tabla que incluye un formulario con toda la sintaxis en formato xBase, hace posible vincular a una o varias tablas con las propiedades de acceder, buscar y regresar con el campo deseado.

En cada campo, se puede personalizar el mensaje interactivo, el control requerido y las validaciones necesarias.

DPxbase, sugiere sintaxis para cada control, si un campo está referenciado con otra tabla, será sugerido el control BMPGET con toda la sintaxis que permita al usuario acceder a la tabla referenciada y relacionar ambos valores. La siguiente imagen, muestra los controles creados por DPxbase para campos vinculados con tablas referenciadas, para los primeros casos el programador seleccionó BMPGET, y el segundo seleccionó el control “COMBOBOX”, en ningún momento el programador ha necesitado escribir una sola línea del código. DPxbase, sugiere la sintaxis necesaria.

El objetivo de DPxbase, es reducir considerablemente el tiempo de desarrollo de aplicaciones, también brindar flexibilidad para acceder y adaptar los componentes creados, para los formularios crea un programa fuente escrito en formato PRG bajo las clases "classes" de Fivewin para el manejo de controles, cada control está definido y asociado al formulario contenedor y las coordenadas se

expresan en líneas y columnas, estas posiciones pueden ser reubicadas en tiempo de ejecución, el formulario dispone de esta opción. La siguiente imagen muestra un formulario definitivo compuesto por encabezado y Cuerpo “Renglones” y en ningún momento fue necesario escribir una línea de código, sólo fue necesario reubicar los controles.

La sintaxis generada para el control “Combobox” es la siguiente:

```
@ 6.1, 1.0 COMBOBOX oCOTIZA:oCTZ_CODMON;
        VAR COTIZA:CTZ_CODMON;
        ITEMS altems1;
        WHEN;
        AccessField("DPCOTIZA","CTZ_CODMON",oCOTIZA:nOption);
       .AND. oCOTIZA:nOption!=0);
        FONT oFontG
```

Cada valor está representado por:

oCotiza: Representa al Objeto de la clase TDPEDIT o formulario contenedor.
oCTZ_CODMON: Representa al control “ComboBox” y para su nombre se utiliza el mismo campo del código con el prefijo “o” que lo asocia con “Objeto”.

VAR COTIZA:CTZ_CODMON: Representa a la variable Virtual del formulario creada automáticamente desde los campos de la tabla de datos. Para mayor comodidad el formulario asume como variables todos los campos y valores de la tabla de datos.

AccessField("DPCOTIZA","CTZ_CODMON",oCOTIZA:nOption) Esta función asocia el acceso de la tabla y campos con el mecanismo de seguridad definible por el usuario donde puede restringir el acceso a los campos de una tabla.

GENERADOR DE BROWSES O LISTAS

Luego de Crear las tablas de Datos, DPxbase genera una lista de navegación utilizando la clase xBrowse, incluye botones sugeridos como: Grabar, Ejecutar, crear programa para formulario, entre otros. Además; el usuario puede seleccionar de manera ordenada los campos de las tablas con sus respectivos títulos que conformarán la lista de navegación de cursores de una o varias tablas SQL.

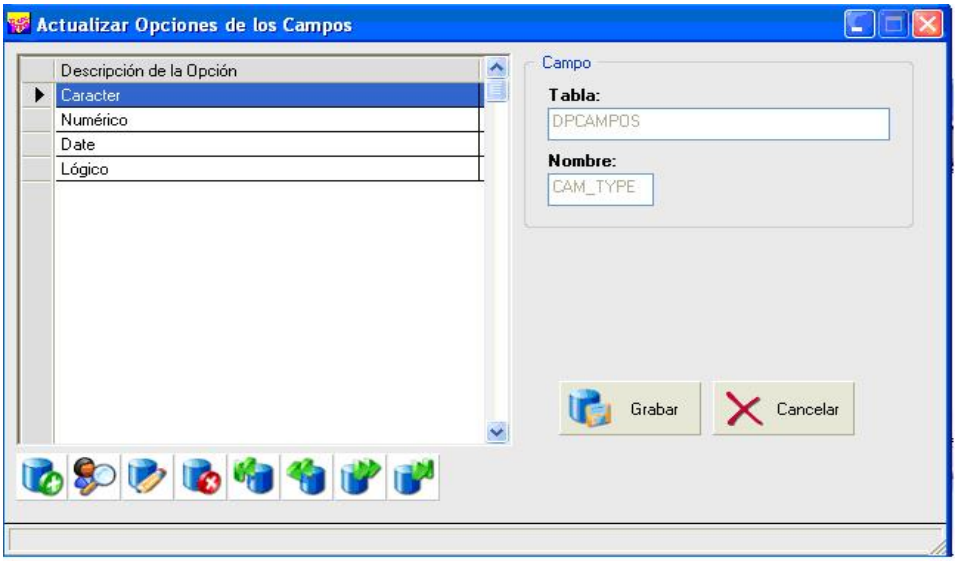
Al presionar la tecla “Ejecutar”, despliega el Browse definitivo en ventana MDI, con todas las opciones disponibles para: Incluir, Consultar, modificar, eliminar, buscar y funciones de navegación. Este nuevo Browse, puede ser modificado en tiempo de ejecución, todas las instrucciones están grabadas en un archivo de formato plano que permite ampliar sus alcances, como por ejemplo: Agregar nuevos botones.

OPCIONES PARA LOS CAMPOS

Forma parte del diccionario de datos y tiene como finalidad definir las diferentes opciones seleccionables a través del control “COMBOBOX” para asignar el valor de un campo durante la carga de datos desde formularios o emisiones de reportes.

También contribuye en la generación del código fuente para los formularios y reportes, el sistema detecta los campos que poseen opciones y construyen la sintaxis para el control COMBOBOX suministrando el contenido de opciones. Para emplear de manera dinámica se incluye en el código fuente la función: **GETOPTIONS("TABLA","CAMPO")**, facilitando emplear las nuevas opciones que pueden ser agregadas para el campo.

Para asignar nuevas opciones para el campo, acceda al formulario de tablas, en la barra de botones se encuentra,  “Opciones de los campos”. Presenta en la tabla que está focalizada una nueva ventana contentiva de todos los campos que pueden aceptar opciones. Focalice con el mouse el campo deseado y haga dos clics o presione el botón  “modificar”, nuevamente se presentará una ventana de diálogo que permite: agregar, consultar, modificar o eliminar cada descripción que conforma las opciones del campo.



Esta ventana para el mantenimiento de opciones, indica el nombre de la tabla y nombre del campo, contiene las diversas opciones ya registradas y a través de: Incluir, Modificar y Eliminar, es posible actualizar cada línea de la lista.

Es importante tomar en cuenta la longitud del campo, si posee un dígito, será asumido el primer carácter de la opción. Ejemplo: La opción “Carácter” al ser seleccionada, solo podrá almacenar el primer carácter “C”, debido al espacio físico que soporta. En caso que requiera más opciones y dos o más de ellas coinciden en el primer carácter, debe ampliar la longitud del campo. Si la longitud del campo es de tres (3) dígitos, serán guardados los primeros tres valores de la opción especificada para el almacenamiento físico en la tabla

Consideraciones:

Los controles combobox mostrarán las opciones en el mismo orden como se registren.

Para utilizar los valores de las opciones en las columnas de los reportes puede emplear la función SayOptions() como expresión de la columna. Ejemplo:

SayOptions(“DPTABLAS”,”CAM_TYPE”,oCursor:CAM_TYPE)

MENÚ DE ACCESO

 Permite modificar y eliminar los menús de los módulos existentes, así como también crear nuevos menús para los módulos del sistema. Los efectos se verán en “Ficheros, Transacciones, Informes, Procesos y Otros”.

DPxbase, tiene un concepto de trabajo sistematizado donde el nombre de cada programa, reporte y formulario está directamente asociado a las tablas, si la tabla se llama DPMENU, el Browse se llamará DPMENU.LBX, el Programa DpXbase para el formulario se llamará DPMENU.SCR, el Reporte será: DPMENU.REP, logrando así ubicar cualquier componente para adaptarlo. Además; facilita personalizar el nombre de las tablas y formularios en tiempo de ejecución, logrando así que el usuario pueda indicar nombres específicos en cualquier tabla y directamente a las opciones de: Menú, Formularios y Reportes, ejemplo: Si la descripción de la tabla Menú de Acceso “DPMENU” es “Menú de Acceso”, el título del formulario de productos adoptará el nombre de “Menú de Acceso”, igualmente el campo “Singular” es utilizado para indicar el nombre de la tabla dentro de un tercer formulario que esté enlazado con la tabla. La sintaxis necesaria para asociar el nombre de las tablas dentro del menú de acceso debe ser: **{oDp:DPMENU}** en el campo “Título del Menú” después de hacer clic en el icono  “Menú de Acceso” de la barra de herramientas del sistema y luego en el botón  “Incluir”. {oDp:DPMENU} <Aplicaciones>.

DEFINICIÓN DEL MENÚ

Todas las opciones del menú principal del sistema, son definibles por el usuario, y está esquematizado por módulo, dentro de cada módulo están los menú: Ficheros, Transacciones, Informes, Procesos y Otros asociados exclusivamente a cada aplicación, las opciones: Macros y Barra de tareas también son definibles de la misma manera, solo difieren que estas opciones son generales para todas las aplicaciones, también podemos decir que una aplicación en un módulo y un sistema está compuesto por módulos.

Otras de las ventajas en la definición del menú es colocar el nombre de la tabla asociada como título de la opción en el menú, es decir; si la tabla de productos está definida como “Productos del Inventario”, este nombre puede figurar en el título de la opción del menú cuando se ejecuta, permitiendo adaptar las opciones de manera directa escrita “La opción” o a través del nombre de uso de una tabla, logrando personalizar las mismas opciones según el tipo de empresa o país. Como por ejemplo en Venezuela denominamos “Nota de Entrega” a un documento que indica los productos que se le entregan al cliente antes de facturar, en Colombia se llama “Nota de Despacho” y en España se denomina “Albarán”.

PROGRAMAS FUENTES DPXBASE

 DPxbase, genera y reconoce un código de léxico xBase basado en xHarbour y comandos para FiveWin. Este lenguaje es compilado e interpretado en tiempo de ejecución permitiéndole al usuario agregar o modificar programas sin necesidad de rehacer el programa ejecutable final. DpXbase acepta: IF, ELSE, ENDIF, WHILE, ENDDO, DOCASE, ENDCASE, y todos los comandos Clipper/xHarbour/FiveWin.

BUSCAR EN PROGRAMAS FUENTES

 Ésta opción permite buscar en todos los programas fuentes ya sea una “PALABRA” ó un campo determinado.

BOTONES DE LA BARRA DEL MENÚ PRINCIPAL

 Al igual que el menú principal, los botones son definidos por el usuario logrando indicar: El Archivo Bitmap, acción que ejecutará, aplicación asociada y condición de ejecución. Se pueden indicar las tablas más utilizadas para una aplicación específica o para todas las aplicaciones.

DPxbase ofrece a la aplicación final un conjunto de herramientas útiles para la operatividad del sistema, tales como: Usuarios, accesos para transacciones con tabla, restricciones para campos, menú de acceso, auditoria de actividades de usuario.

PERSONALIZAR MENSAJES EN LA BARRA DE BOTONES

El programa “DPBARMSG”, crea un objeto de tipo “TSAY” que presenta un recuadro en la barra de botones con la finalidad de indicar textos que puedan ayudar al usuario a identificar actividades como: Aplicación que se ejecuta, fecha del sistema, periodo contable o cualquier otra información que se desee indicar. Todos los parámetros del Objeto, tales como: Coordenadas, tamaño, fuente de letra, color del fondo, color del texto y mensajes pueden ser personalizados en este programa, quedando totalmente a criterio del usuario su utilización. Para mayor comodidad el programa “BOTBARCHANGE” que se ejecuta en cada cambio de aplicación elimina de la barra de botones todo su contenido para que luego el sistema coloque nuevamente los botones que cumplan con las condiciones de la aplicación, adicionalmente ejecuta “DPBARMSG” para mostrar los nuevos mensajes para la aplicación solicitada.

ENLACE ENTRE TABLAS

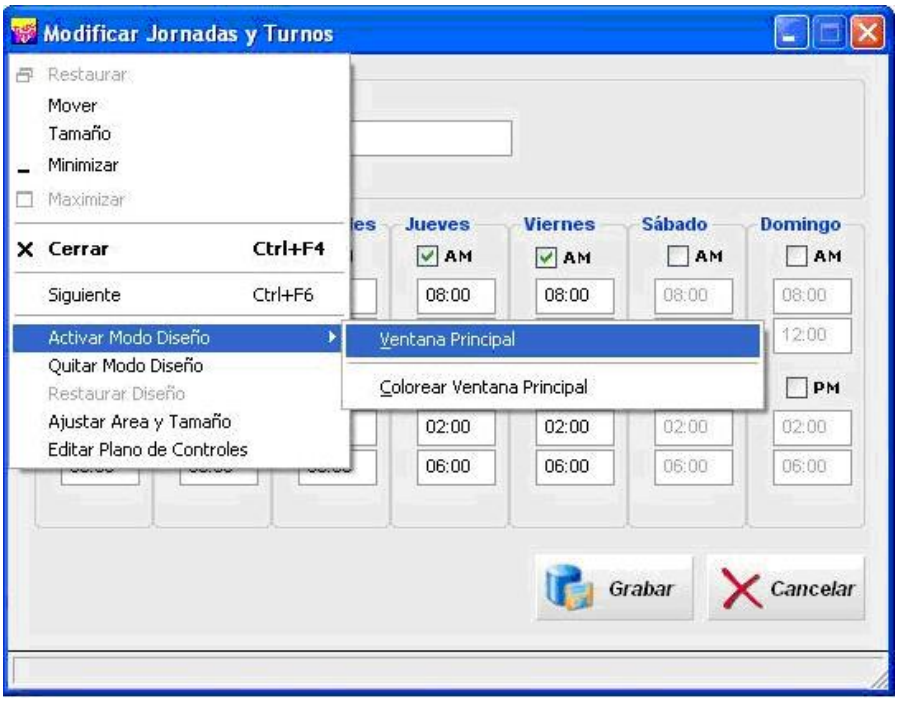
Como su nombre lo indica, permite hacer el enlace entre las tablas por medio de un campo común entre ambas tablas. Por ejemplo la tabla del trabajador “nmtrabajador” hay que enlazarla con la tabla de departamento, cargo, unidad funcional, ausencia, etc., para ello se utiliza el campo código del trabajador.

OTRAS FUNCIONES

DISEÑO DE FORMULARIOS

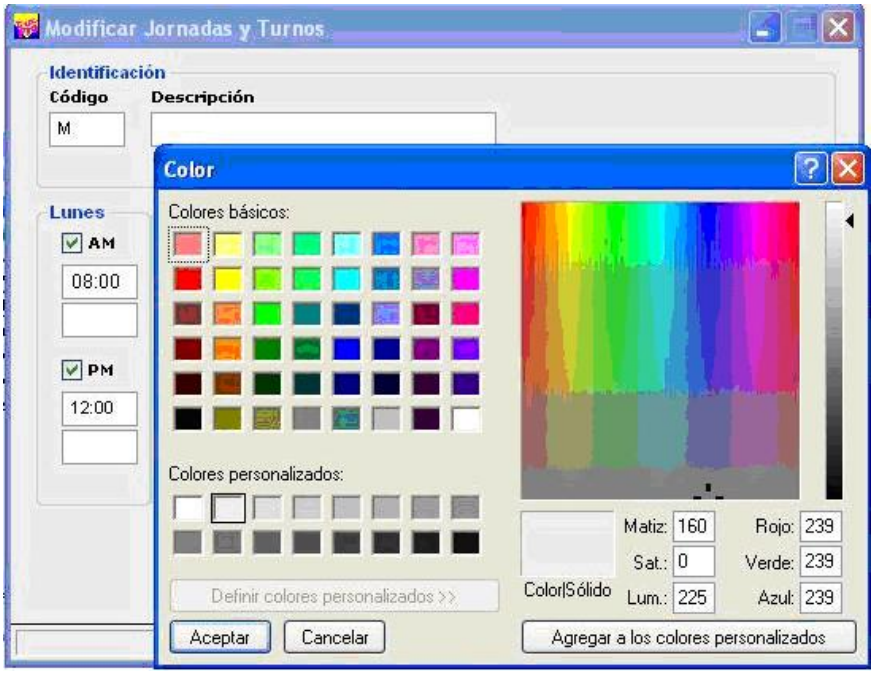
Partiendo del programa Fuente “xBase” creado por DPxbase, el usuario puede diseñar, modificar en tiempo de ejecución todos los controles del formulario utilizando el Mouse, para cada uno puede configurar: Línea, Columna, Alto, Ancho, Colores y Fuentes de Letra. Sin necesidad de utilizar recursos creados en otras aplicaciones.

En la parte superior de la ventana o comúnmente denominado “SysMenú”, el formulario dispone de varias opciones que permiten activar el modo diseño del formulario, permitiendo personalizarlo en tiempo de ejecución, esta labor puede ser efectuada por el programador o usuario final.

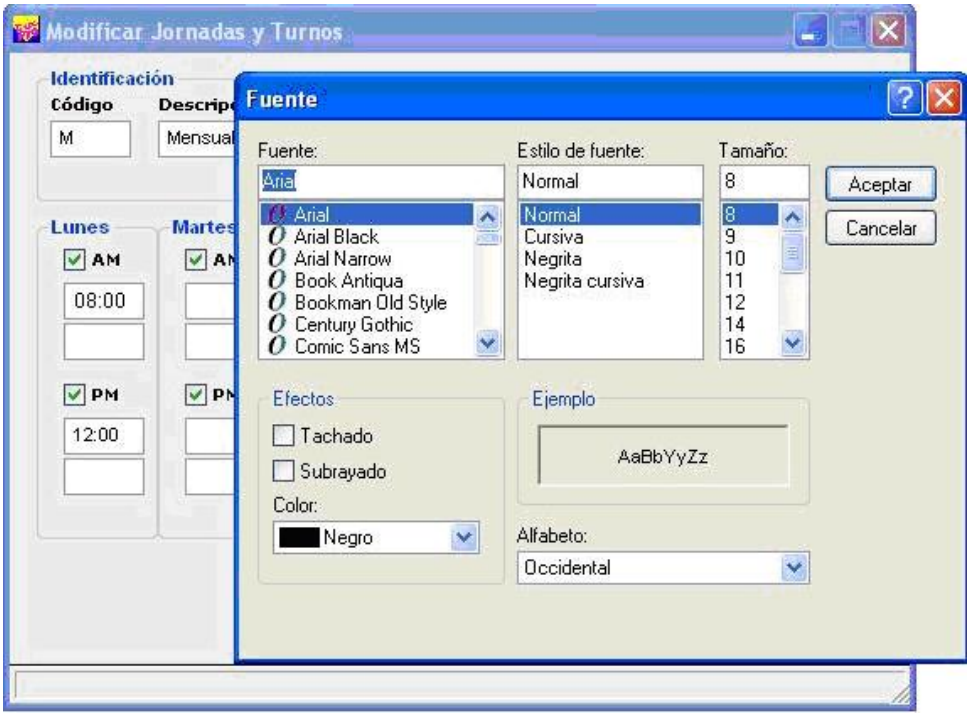


Luego de activar el modo diseño, es posible focalizar cualquier control a través del mouse, moverlo en cualquier posición, ampliar o reducir su tamaño y para mayor comodidad DPxbase ofrece alternativas para agilizar el diseño.

La siguiente imagen, ilustra el uso de la opción: "Color" seleccionado para el control focalizado, es decir; se puede seleccionar el color del fondo.



La siguiente imagen, muestra la posibilidad de cambiar el color del texto del control y también la fuente de las letras.



DPxbase integra bajo un solo concepto: La tabla de datos, el Navegador “Browse”, programas fuentes “DpXbase”, formulario para la carga de datos y Reportes bajo un conjunto de elementos identificados y asociados con la tabla de datos.

DPxbase ha sido creado exclusivamente para crear aplicaciones de tipo comercial y orientado al léxico xBase, generando productos finales en corto tiempo y bajo el concepto de arquitectura abierta facilitándole al integrador o usuario final adaptarlo a su criterio.

DPxbase mantiene con continuo desarrollo la búsqueda de la madurez de todas sus herramientas con el objetivo de darle valor agregado a los componentes de desarrollo y a los productos finales creados para el Cliente. DPxbase hereda toda la experiencia de DpXbase creado para Datapro durante 15 años bajo el popular y exitoso compilador "Clipper" que logró posicionarse en el mercado con miles de instalaciones satisfactorias, en parte gracias a la comodidad y facilidad de complacer los requerimientos del usuario. Ahora es DPxbase para 32 bits y base de datos de tipo SQL.

CONTINUIDAD EN DATAPRO PARA TABLAS DBF

Todas las tablas de datos en formato DBF, son totalmente portables en estructura y contenido hacia el nuevo formato de datos utilizado por DPxbase (Motor SQL). La opción Incluir Tablas, dispone de la opción “Importar desde DBF”, el usuario solo debe indicar el nombre de la tabla de Datos DBF y DPxbase se encarga de definirla en el diccionario de

Datos y construir su estructura. Además DPxbase provee un proceso de Migración de Registros desde las tablas DBF utilizadas por productos anteriores.

CONTINUIDAD EN EL LÉXICO XBASE DESDE LOS PRODUCTOS DOS

Aunque DPxbase esta compilado con C++, el leguaje Xbase incorporado soporta en gran parte los procedimientos desde las versión DOS, siempre y cuando no requieran interactuar con una interfaz de video o de impresión. Las tablas de datos en las versiones anteriores se identificaban a través de "ALIAS" ejemplo: "alias->campo", ahora en DPxbase será apuntando a un cursor, ejemplo "oCursor: Campo", los tipos de datos son 100% compatibles. Parte de los programas escritos en DpXbase para DPxbase han sido obtenidos de versiones anteriores con muy pocos cambios, donde se aprovecha la madurez del código ya escrito.

COMPONENTE DATASET

Es un gestor de datos controlado a través de una clase de objetos que interactúan con una tabla de datos y tiene como finalidad manipular cualquier tipo de datos compatible con el léxico xBase.

El componente DataSet ha sido incorporado en el sistema con el objetivo de almacenar múltiples tipos de datos seleccionados sin que sea necesario emplear tablas específicas para su almacenamiento.

El generador de reportes es una aplicación del sistema que emplea altamente el componente Dataset para almacenar valores como: Parámetros, Rango y Criterio, cada reporte conforma una sección del Dataset. Cuando el usuario desde la ventana de ejecución de un determinado reporte indica "Fijar Parámetros del reporte", está empleando el Dataset para almacenar todos los valores utilizados en esa ejecución del reporte, con el fin de retomarlos en futuras y suponiendo que esos parámetros sean comúnmente utilizados.

El sistema dispone de dos DataSet, cada uno almacena los valores en tablas distintas, "DPDATACNF" es utilizada para todo lo referido a la configuración general del sistema y "DPDATASET" almacena valores de manera independiente en cada empresa definida a través de un "DSN".

Para los parámetros del generador de informes (Fijar: Rango y Criterio) es empleado "DPDATACNF", debido a que todos los reportes pueden ser empleados en cualquier DSN y la configuración de cada empresa se almacena en "DPDATASET".

Para almacenar datos en el DataSet de configuración se emplea de la siguiente forma:

```
PROCE MAIN()
        LOCAL oData
        oData:=DataCnf("MISECCION")
        oData:cValor:="Prueba"
        oData:End()
RETURN NIL
```

Para leer los datos del DataSet, debe ser realizado de la siguiente forma:

```
PROCE MAIN()
        Local oData
        oData:=DataCnf("MISECCION")
        MsgAlert(oData:cValor)
        oData:End()
RETURN NIL
```

Puede observar que en ambos casos, se emplea un valor virtual **"oData:cValor"** de la clase TDataSet y de manera automática detecta la asignación de un nuevo valor, lo almacena y también lo restaura cuando se encuentra guardado en la tabla. El método "End()" almacena de manera automática los nuevos valores o los valores existentes que sufran algún cambio. Por ello es muy práctico emplear este componente en el almacenamiento de datos.

Para emplear el DataSet en la configuración de cada empresa es necesario utilizar el siguiente ejemplo escrito en DpXbase:

```
PROCE MAIN()

        Local oData,oDlg

        oData:=DataSet("DATEMPRESA")
        oData:cDir:=oData:Get("DIR",SPACE(20))
        oData:cTel:=oData:Get("TEL",SPACE(12))
        oData:cRep:=oData:Get("REP",SPACE(40))

        DEFINE DIALOG oDlg TITLE "CONFIGURAR"

        @ 1,1 GET oData:cDir
        @ 2,1 GET oData:cTel
        @ 3,1 GET oData:cRep

        ACTIVATE DIALOG oDlg

        oData:End()
```

```
RETURN
// EOF
```

CLASE TDATASET:

Se encarga de gestionar, leer, grabar todo lo referente al DataSet, se genera desde las funciones DataCnf() y DataSet(), ambas devuelven el objeto de la clase TDataSet, a continuación detallamos sus métodos.

- ***Get(cVarName,uValue):***
  Obtiene el valor referido de "cVarName" en caso de no existir el auto creado con el valor indicado en "uValue". También puede ser empleado de manera directa: oDataSet:cVarName.

- ***IsDef(cVarName):***
  Devuelve .T. o .F. si el valor de "cVarName" existe.
- ***Set(cVarname,uValue):***
  Asigna el valor "uValue" para "cVarname", también puede ser utilizado como: oDataSet:cVarname:=uValue.

- ***Save():***
  Almacena en la tabla todos los valores del DataSet.

- ***Del():***
  Elimina un valor del DataSet y de la tabla, ejemplo: oDataSet:Del(cVarName).

- ***DelGroup(cGroupName):***
  Elimina todos los componentes asociados a un grupo.

- ***Load():***
  Carga todos los valores desde la tabla y generan los valores virtuales del Dataset.

DataSet dispone tres modalidades para el almacenamiento y recuperación de datos, estos son:

"ALL": Almacena y recupera los datos para todos los usuarios y "PC" que accedan al sistema.

"PC": Almacena y recupera los datos sólo para el PC que lo solicita, el nombre del PC es obtenido del Sistema Operativo. Cabe destacar que en cada PC es posible generar diferentes configuraciones.

"USER": Almacena y recupera los datos sólo para el usuario que solicita, es decir; cada usuario puede disponer de configuraciones distintas.

Se denominan valores virtuales aquellos "ClassData" o variables que no están definidas en la clase, sino que se generan de manera dinámica bien sea por asignación directa "oDataSet:nValue:=10000" o a través de la lectura de la tabla donde se almacenan los valores del Dataset.

Para el sistema de Nómina DpNmWin, el DataSet ha sido altamente empleado en el almacenamiento de los tipos de nómina que se procesan con la finalidad de poder rescatar los tipos de nómina en las siguientes ejecuciones.

FUNCIONES PARA SENTENCIAS SQL

GetWhere(cOperador,uValor)

Construye una condición para la sentencia WHERE o SET para armar un comando SQL. Ejemplo

cNomre:="Datapro"
cWhere:="WHERE CAMPO"+GetWhere("=",cNombre)

Queda construida de la siguiente forma:

WHERE CAMPO='Datapro'

Recuerde utilizar comillas ['] o Chr(39) simples para indicar cadenas dentro de un comando SQL. Igualmente esta función está vinculada con la configuración de fechas empleadas por el motor de la base de datos, puede ser usada cómodamente sin tomar en cuenta las reglas para las fechas y la complejidad de construirla:

Ejemplo:

dFecha:=DATE()
cWhere:="WHERE FECHA"+GetWhere("=",dFecha)

La sentencia resulta así:
WHERE FECHA='20041231'

Tradicionalmente la condición WHERE empleando fechas debe construirse así:

WHERE
FECHA=[']+STRZERO(YEAR(dFecha),4)+STRZERO(MONTH(dFecha),2)+STRZERO(DAY(dFecha),2)+[']

Cómodamente **WHERE FECHA+GetWhere("=",dFecha)** lo simplifica.

FUNCIONES PARA EL MANEJO DE DATOS

Todo el manejo de datos en Datapro se realiza a través de comandos SQL, el acceso a los datos de una o varias tablas se solicita utilizando sentencias que retornan un cursor de datos en un plano bidimensional compuesto por líneas y columnas: Las líneas representan los registros y las columnas representan los campos.

OPENTABLE(CSQL,LDATA)

Solicita un cursor de datos SQL.
<cSql> Debe contener una sentencia de léxico SQL
<lData> Indica con .T. si carga los datos. Por defecto su valor es .T.

Ejemplo:

```
    oCursor:=OpenTable("SELECT TAB_NUMERO,TAB_TABLE"+;
                       " FROM DPTABLAS "+;
                       " ORDER BY TAB_NUMERO",.t.)
oCursor:Gotop()
WHILE !oCursor:Eof()
  ? oCursor:TAB_NUMERO,oCursor:TAB_TABLE
  oCursor:DbSkip()
ENDDO
oCursor:End()
```

Puede observar que la consulta puede ser manejada de manera similar a una tabla DBF abierta a través de funciones del RDD o mejor dicho abierta con: USE TABLA. Para mayor información consulte la clase TTABLE de este manual.

ASQL(CSQL)

Devuelve en tipo de datos Arreglo bidimensional el contenido del cursor generador por la consulta SQL. Ejemplo: aSql:=ASQL("SELECT CAMPO FROM XTABLA"). aSql Contiene la lista de datos. MsgAlert(aSql[1,1]), muestra la primera columna. Para evaluar la respuesta puede utilizar: ViewArray(aSql).

SQLDELETE(CTABLA,CWHERE)

Borra de la tabla <CTABLA> la cantidad de registros indicados en <CWHERE>

SQLUPDATE(CTABLA,CCAMPO,XVALOR,,CWHERE)

Permite Actualizar el valor de un campo de la cantidad de registros indicados en la cláusula cWhere. Ejemplo:
SqlUpdate("NMTRABAJADOR","TIP_NOM","Q","TIPO_NOM='S'") Cambia el Tipo de Nómina para Quincenal de todos los trabajadores de Nómina Semanal.

SQLGET(CTABLA,CCAMPO,CWHERE)

Obtiene el valor de un campo desde una tabla según la condición indicada en Where. Where puede incluir la clausula JOIN.
SQLGET("NMTRABAJADOR","CONCAT(APELLIDO,",",NOMBRE)","CODIGO='00001'").
Obtiene el nombre y apellido del Trabajador.
SqlGet(), puede fácilmente reemplazar el uso de las funciones DBLocate() y DbSeek() utilizadas en las tablas DBF según RDD.

DTOSQL(DFECHA)

Convierte el formato de fecha xBase en formato de fecha para el Motor SQL, según el formato definido en "oDp:cSqlDate" previamente configurado en ODBC.INI [SQLDATE].

[UPDATE FROM TABLA BCAMPO=']+DTOSQL(<bFecha>)+[']

Queda Así:

UPDATE FROM TABLA BAMPO='2005-04-02'

SQLTODATE(CFECHA)

Convierte las fechas SQL en fecha xBase, según el formato definido en "oDp:cSqlDate" previamente definido en ODBC.INI [SQLDATE]. Algunas funciones MAX o MIX de SQL aplicadas sobre campos de tipo de fecha, generan expresiones de tipo carácter el cual requiere ser convertida en forma de fecha real.

bDate:=SQLGETMAX("DATABLA","CAMPO",<WHERE>) // 2005-05-02
bData:=SQLTODATE(cFecha) // 02/05/2005

<cTable> Nombre de la tabla
<aFields> Lista de campos, puede ser un arreglo un una lista de campos entre comillas.

BDLIST(CTABLE,AFIELDS,LGROUP,CWHERE)

Genera un cursor de datos según:

<cTable> Nombre de la tabla
<aFields> Lista de campos, puede ser un arreglo en una lista de campos entre comillas separadas con coma, ejemplo: “Campo1,Campo2,Campo3”
<lGroup> si es verdadero .T. genera el cursor empleando GROUP BY en todos los campos. Su valor por defecto es .f.
<cWhere> Valor opcional, puede conformar la cláusula WHERE de la consulta y permite filtrar las líneas de la consulta.

Presenta la consulta a través de un editor de registros que permite seleccionar cualquier registro devolviendo el valor de la primera fila. Tiene alta utilización en las casillas de rango y criterio en la ejecución de un reporte, permitiendo importar los valores de un campo hacia las casillas.

Ejemplo:
uValue:=BDLIST(“NMTRABAJADOR”,”CODIGO,APELLIDO,NOMBRE,SALARIO”)

BDSELECT(CTABLE,AFIELDS,LGROUP,CWHERE))

Genera un cursor de datos según:

<cTable> Nombre de la tabla
<aFields> Lista de campos, puede ser un arreglo en una lista de campos entre comillas separadas con comas, ejemplo: “Campo1,Campo2,Campo3”
<lGroup> si es verdadero .T. genera el cursor empleando GROUP BY en todos los campos. Su valor por defecto es .f.
<cWhere> Valor opcional, puede conformar la cláusula WHERE de la consulta y permite filtrar las líneas de la consulta.

Presenta la consulta “Cursor” a través del editor de registros, con la finalidad de realizar selección múltiple de registros, es decir; marcar los registros necesarios, al finalizar devuelve una cadena con todos los valores seleccionados y representados por la primera columna.

cValue:=BDSELECT(“NMTRABAJADOR”,”CODIGO,APELLIDO,NOMBRE,SALARIO”)

Si el usuario seleccionó cuatro trabajadores, el valor de cValue será: “001”,”002”,”003”,”004”

Si desea convertirlo en arreglo puede emplear:

aList:=MacroEje(“{“+cValue+”}”)

Tambien, recomendada:
aList:=_Vector(cValue)

GETOPTIONS("TABLA","CAMPO")
Devuelve en un arreglo todas las opciones registradas para un campo de determinada tabla. Ejemplo:

aOpcion:=GETOPTIONS(“DPTABLAS”,”CAM_TYPE”)

Devuelve una lista o arreglo con los siguientes valores: “Numerico”, ”Date(Fecha)”, ”Carácter”, ”Logica” y ”Memo”.

SAYOPTIONS(cTable,cField,uValue)

Busca y devuelve la opción encontrada según el valor de <uValue> según la lista de opciones registradas en campo <cField> de la tabla <cTabla>.
Ejemplo:
cOpcion:=SAYOPTIONS(“DPTABLAS”,”CAM_TYPE”,”C”)

El valor de <cOpcion> es “Carácter”. En caso de no encontrar la opción buscada, devolverá el valor vacío.

INNERJOIN(cTabla1,cTabla2,cInner)

Busca en el Diccionario de Datos la relación existente entre dos tablas y construye la cláusula INNER JOIN de las sentencias SQL necesarias para el enlace entre las tablas solicitadas.

Ejemplo: INNERJOIN(“DPTABLAS”,”DPCAMPOS”)

Devuelve: INNER JOIN DPCAMPOS ON TAB_NUMERO=CAM_NUMTAB

OBJETOS

Clase TTable
Clase DpLbxRun
Clase DpEdit

ARCHIVOS DE AYUDA

Están ubicados en la carpeta “HELP\” de la aplicación, acepta archivos de ayuda HLP y CHM.

Está compuesto por dos tipos de archivos “Tradicionales HLP” que componen la ayuda ofrecida por las librerías FiveWin y estas no forman parte de la licencia.

FIVEWIN.HLP	Conceptos de FiveWin
FWCMD.HLP	Classes, métodos y data.”
FWCMD.HLP	Comandos
FWFUN.HLP	Funciones

El segundo conjunto de archivos de extensión CHM, contienen documentación referida a la aplicación datapro y se compone de los siguientes archivos.

GeneradorRPT.CHM	Generador de Reportes.
Programación.CHM	Programación
Capítulo[n].CHM	[n] se refiere al número del capítulo y cada uno se refiere a cada aplicación del sistema. El “Capitulo1.chm” se refiere al proceso de instalación y configuración del sistema. Para facilitar los tópicos en cada capítulo su nombre debe ser asociado en cada tabla y en caso de referirse a algún proceso se identificará según la posición del menú: Ejemplo: Aplicación+Vertical+Horizontal. Cada aplicación está asociada a un “capítulo[n].chm” y cada tabla también está asociada a una aplicación por ello es posible asociarlas y activar la ejecución del archivo de ayuda correspondiente al editor de registros que muestra el contenido de una tabla o un formulario para la carga de datos.

CLASES PROPIETARIAS DATAPRO

Conforman un conjunto de clases que generan objetos para funciones específicas para Datapro.

TPUBLIC

Esta clase genera un objeto en público que está disponible en todo el sistema y tiene como finalidad almacenar todos los valores que el sistema necesita para su funcionamiento técnico y conceptual. Este novedoso modelo de datos es altamente productivo al evitar el uso excesivo de memoria para declarar variables públicas.

DpXbase provee el objeto **<oDp>** desde que arranca el sistema y contiene toda la información referida a la configuración del sistema, por ejemplo:

oDp:cEmpresa: Nombre de la Empresa.
oDp:cDsnData : Nombre del DSN de Datos de la empresa Activa.
oDp:cDsnConfig: Nombre del DSN de Datos de la base de datos de configuración.
oDp:cUsuario: Código del usuario que accede al sistema.
oDp:dFecha: Fecha del Sistema.

La asignación de valores se realiza en forma directa oDp:xValor:="Nuevo Valor", los valores de <oDp> son Virtuales, es decir; no están definidos en la Clase sino; se asigna en forma dinámica. Ejemplo:

oDp:cMiDato:="Cualquier Dato"
MsgAlert(oDp:MiDato)
Mostrará la ventana de dialogo Alert indicando "Cualquier Dato"
El programa DPINI es la primera ejecución DpXbase que se realiza al iniciar Datapro. Aquí se asignan la mayoría de valores que definen la ejecución y/o presentaciones del sistema. Como por ejemplo el tamaño de los botones

Métodos:

GET(cVarName). Obtiene el Valor de una DATA del Objeto, según la identificación solicitada en <cVarName>, ejemplo: oDp:Get("cEmpresa"), igualmente puede ser oDp:cEmpresa.

ADD(cVarname,uValue). Genera una nueva DATA Virtual dinámicamente y asigna el valor, si cVarName ya existe omite la solicitud. Ejemplo: oDp:Add("cEmpresa","Nuevo Valor"), también puede ser: oDp:cEmpresa:="Nuevo Valor"

SET(cVarName,uValue): Asigna un valor a una Data. Si no existe es agregada automáticamente. También puede ser oDp:cEmpresa:="Nuevo Valor".

TTABLE

Es una clase propietaria de DpXbase que accede a consultas SQL empleando la función OpenTable(cSql,lRead) que devuelve un Objeto contentivo de la consulta SQL al estilo de datos RDD (xBase). Ttabla hereda todos los campos de la estructura permitiendo navegar en cada registro y conocer el contenido de cada campo, ejemplo: oCursor:CAMPO.

Ttable es utilizada en toda la aplicación para acceder a las consultas de la Base de Datos necesarias en los Browser, Formularios, Consultas e Informes.

oTable:=OpenTable("SELECT CAMPO FROM XTABLA",.t.)
MsgAlert(oTable:CAMPO) // Muestra el contenido del CAMPO

oTable:End()

También puede manejar campos virtuales con la finalidad de rediseñar la consulta, ejemplo: oTable:Replace("CAMPONUEVO",oCursor:Monto*2), genera un nuevo campo llamado CAMPONUEVO que se derivó de la expresión: oCursor:Monto*2

Sintaxis: Ttable():New(cSql)

Datos de la Clase TTABLA

aDataFill: Es un arreglo multidimensional que contiene en forma bidimensional el contenido de la consulta SQL. Cada línea representa un registro y cada columna representa un Campo. Para copiar aDataFill debe utilizar aData:=ACLONE(oTable:aDataFill). Si desea visualizar su contenido puede utilizar ViewArray(oCursor:aDataFill)

aFields: Contiene información de la estructura de la Tabla, está compuesta por líneas y columnas, cada línea representa un registro y cada columna representa la estructura de los campos: Nombre, tipo, longitud y Decimales.

oOdbc: Contiene el Objeto de la conexión DSN donde pertenece la tabla. También es posible utilizar los métodos de esta clase: oCursor:oOdbc:Execute(<cSQL>)

Métodos de la Clase TTABLA

Eof(): Determina si la lectura a llegado al último registro de la consulta.

Goto(<nRecord>): Apunta al registro <nRecord> de la Consulta.

GoTop(): Posiciona el primer registro de la consulta.

GoBottom(): Posiciona el último registro de la consulta.

FieldGet(<nField>): Obtiene el valor del campo según su posición <nField>, puede ser el número del campo o nombre del campo.

FieldName(<nField>): Devuelve el nombre del campo según el número o posición del campo.

RecCount(): Indica la cantidad de líneas o registros del cursor o consulta.

RecNo(): Indica la posición del registro del cursor.

FCount(): Determina la cantidad de campos utilizados por la consulta.

DbSkip(<nSkip>): Permite navegar (Ascendente/Descendente) sobre el cursor.

Append(): Prepara el cursor para realizar la sentencia INSERT INTO.

Replace(cField,uValue): Reemplaza el valor de la columna (Campo) identificada con <cField> según el valor de <uValue>.

Commit(): Ejecuta el comando INSERT INTO para agregar un nuevo registro en la tabla, previa preparación con los métodos append() y replace().

Commit(<cWhere>): Ejecuta el comando UPDATE para actualizar uno o varios registros según la cláusula WHERE, previa preparación del método Replace().

CtoDbf(cFile): Crea una tabla en formato DBF/FPT con el contenido del Cursor. Es necesario que las consultas que utilicen funciones, representen los resultados a través de etiquetas. Ejemplo SELECT MAX(XCAMPO) AS CAMPO , la columna MAX(XCAMPO) será representada por CAMPO.

Browse(): Muestra el Contenido de la Consulta mediante una ventana MDI.

Execute(<cSQL>): Ejecuta un comando SQL directamente con la conexión del Servidor de la Base de Datos.

Ejemplo para agregar un nuevo registro:

```
#INCLUDE "DPXBASE.CH"

PROCE MAIN()

  LOCAL oTable

  oTable:=OpenTable("SELECT * FROM XTABLA",.F.)
  oTable:Append()
  oTable:Replace("CAMPO1","Valor 1")
  oTable:Replace("FECHA" ,DATE())
  oTable:Commit()
  oTable:End()

RETURN
```

En este ejemplo se puede observar que el cursor <oTable> se requiere sin datos, por esta razón la función **`OpenTable("SELECT * FROM XTABLA",.F.)`**. Solicitó a través de **.F.** el cursor en forma vacía, debido a que solo es necesaria la estructura de la tabla y como será realizado el proceso de inclusión no es necesario leer datos de la tabla.

Si se requiere buscar información en la misma tabla para agregar un nuevo registro como sucede en los casos de numeración de documentos, podemos usar la siguiente sintaxis:

```
#INCLUDE "DPXBASE.CH"
PROCE MAIN()

  LOCAL oTable,xValor,nLen

  xValor:=SQLGET("XTABLA","MAX(XCAMPO)")
  xValor:=VAL(xValor)

  oTable:=OpenTable("SELECT * FROM XTABLA",.F.)
  nLen  :=oTable:FieldLen("XCAMPO")
  xValor:=STRZERO(xValor+1,nLen)
  oTable:Append()
  oTable:Replace("CAMPO1",xValor)
  oTable:Replace("FECHA" ,DATE())
  oTable:Commit()
  oTable:End()

RETURN
```

El método commit() construye la real sentencia SQL para la inserción del registro de la siguiente forma:

```
INSERT INTO XTABLA (CAMPO1,FECHA) VALUES ('0001','3005-05-2005').
```

TDPEDIT

Es una clase DpXbase que permite diseñar formularios de tipo (MDI) en forma declarativa, es decir a través de comandos xbase para la declaración de Controles que en tiempo de ejecución puedan ser rediseñados en forma visual. El formulario acepta como métodos virtuales todas las funciones definidas con "FUNCTION" en el mismo programa DpXbase, igualmente permite la asignación de Data en forma dinámica con tan solo ejecutar la declaración "oForm:nVariable:=100"

Ejemplo

```
#INCLUDE "DPXBASE.CH"

PROCE MAIN()

  LOCAL oFont

  DEFINE FONT oFont NAME "Arial"   SIZE 0, -12 BOLD ITALIC
```

```
   oFrm:=DPEDIT():New("Titulo","Fichero.edt","oFrm" , .T. )
   oFrm:cVarName:=SPACE(20)
   oFrm:nValor  :=100
   oFrm:dFecha  :=oDp:dFecha

   @ 0,0  SAY " Prueba de Texto ";
          SIZE 120,10 RIGHT;
          COLOR CLR_HBLUE,CLR_YELLOW

   @ 2,03 GET oFrm:oNombre VAR oFrm:cVarName;
          OF oFrm:oDlg;
          SIZE 50,10;
          FONT oFont;
          VALID oFrm:MyValid()

    XSAY("Nombre")

   @ 4,03 GET oFrm:oValor VAR oFrm:nValor;
          PICTURE "999999.99";
          SIZE NIL,10 RIGHT;
          FONT oFont;
          WHEN !Empty(oFrm:cVarName)

   XSAY("Valor")

   @ 6,03 BMPGET oFrm:oFecha VAR oFrm:dFecha;
          PICTURE "99/99/9999";
          NAME "BITMAPS\Calendar.bmp";
          ACTION LbxDate(oFrm:oFecha,oFrm:dFecha);
          SIZE 50,10;
          FONT oFont;
          WHEN !Empty(oFrm:cVarName)

   XSAY("Fecha")

   @ 6,04 BUTTON " Aceptar ";
          SIZE 36,12;
          ACTION oFrm:Aceptar()

   @ 6,15 BUTTON " Cerrar  ";
         SIZE 36,12;
          ACTION oFrm:Close()

   oFrm:Activate()

RETURN

FUNCTION MyValid()

   IF Empty(oFrm:cVarName)
      MensajeErr("Nombre no Puede Estar Vació")
      RETURN .F.
```

```
  ENDIF

RETURN .T.

FUNCTION XSAY(cText)
  LOCAL oObj
  oObj:=ATAIL(oFrm:oDlg:aControls)

  @ oObj:nTop-9,oObj:nLeft SAY cText PIXEL;
    SIZE 40,8

RETURN .T.

FUNCTION ACEPTAR()
   MsgAlert("Aquí se ejecuta el proceso")
RETURN .T.
```

Métodos de la clase

New(<cTitulo>,<cFichero>,<coFrm>,<lWindows>): Inicia la construcción del formulario.

> cTitulo: Nombre del título de la ventana.
> cFichero: Nombre del fichero con extension (.edt) y ubicación "forms\" contentiva de los parámetros (línea, columna, alto, ancho, color fondo y color texto) de cada control del formulario.
> coForm: Nombre de la variable que contiene e identifica el formulario.
> lWindows: Si es verdadero <.T.> genera la ventana del formulario, necesario para declarar los controles "@ Lin,Col <CONTROL>"

CreateWindows(): Dibuja la ventana MDI (oWnd) y el Dialogo (oDlg) para controles incrustados y dependientes de MDI. Windows() no es necesario si el parámetro <lWindows> es .T. en la construcción del formulario con NEW(). Todos los controles deben ser definidos después de ejecutar el método Windows debido a que oDlg es el contenedor por defecto de todos los controles.

Activate(bInit,bEnd,bResize): Presenta en el video y activa la ejecución del formulario.

bInit: Bloque de código de iniciación y previa ejecución de la ventana, puede ser utilizado para llamar a un método virtual (oFrm:xUDFunction) para crear barra de botones o ejecuciones previas a la presentación del formulario.

bEnd: Bloque de Código que debe ejecutarse al cierre del formulario, si éste devuelve .f., el formulario no puede ser cerrado. Esta utilidad es de gran ayuda para realizar validaciones de cierre.

bResize: Bloque de Código que se ejecuta cuando el tamaño de la ventana es reajustado con el Mouse.
Data de la Clase:

oDlg: Nombre del Objeto Dialogo contenedor de Controles “oFrm:oDlg:aControls”.
oWnd: Nombre de la Ventana de estilo MDI (CHILD) hija de la ventana principal y se encarga de ejecutar todos los eventos de la venta: minimizar, restaurar, ajuste de tamaño y permitir el uso en forma MDI.

oCursor: Contiene el objeto del cursor SQL de la consulta asociada con el formulario.

bValid: Bloque de código que permite validar el cierre del formulario, puede evitar salirse con [X] de la ventana MDI.

lActivate: Indica si el formulario ha sido activado o ejecutado el método Activate().

lEscClose: Indica con .T. si el formulario puede ser cerrado con la tecla ESC.

CONTROLES DE LA LIBRERÍA FIVEWIN PARA FORMULARIOS.

Constituye un conjunto de controles de gran uso para la definición del formulario. La sintaxis clásica de la librería fivewin (www.fivetechsoft.com) es utilizada en DpXbase logrando la mayor compatibilidad posible y aprovechamiento de toda la documentación existente **c:\dpnmwin\help\fwcmd.hlp** debido a que DpXbase utiliza al máximo el potencial de esta fabulosa librería.

Todos los controles definidos deben ser asociados sólo con el formulario.

Check Box (Casilla de Selección)

```
OFrm:lVar:=.T.

@ 02,02 CHECKBOX oFrm:oChk VAR oFrm:lVar;
        PROMPT " CheckBox " ;
        OF oFrm:oDlg;
        SIZE 90,10;
        ON CHANGE {||.T.};
        VALID   .T.;
        COLOR NIL,oDp:nGris;
        MESSAGE "Indique el valor del CheckBox";
        WHEN .t.
```

Button (Botones Simples)

```
@ 01,01 BUTTON oFrm:oBtn;
        PROMPT "Cerrar ";
        SIZE 36,12;
        ACTION oFrm:Close();
        OF oFrm:oDlg ;
        MESSAGE "Cerrar Formulario";
        WHEN .t.;
       CANCEL
```

Get (Introducir Datos)

```
oFrm:cVar:=SPACE(40)

@ 1, 1 GET oFrm:oGet VAR oFrm:cVar;
       OF oFrm:oDlg;
       SIZE NIL,10;
       MESSAGE "Indique el valor del Campo";
       WHEN .t.;
       VALID;
       MensajeErr("No puede estar Vacio”,;
       vacio",nil,{||!Empty(oFrm:cVar)});
       ON CHANGE .t.

// Get para Campos Numéricos

oFrm:nValor:=1000

@ 4, 10 GET oFrm:oGet VAR oFrm:nValor;
       PICTURE "99,999,999.99";
       RIGHT;
       OF oFrm:oDlg;
       SIZE NIL,10;
       MESSAGE "Indique el Monto";
       WHEN .t.;
       VALID .t.;
       ON CHANGE .t.
```

Get (Introducir Datos en Campo MEMO)

```
oFrm:cMemo:= "Campo Memo, Texto"
oFrm:cMemo:=Alltrim(oFrm:cMemo)

@ 5, 5 GET oFrm:oGet VAR oFrm:cMemo;
       MEMO;
       HSCROLL;
       OF oFrm:oDlg;
       SIZE 100,100;
       MESSAGE "Introduzca los textos";
       WHEN .t.;
       ON CHANGE .t.
```

ComboBox (Cuadro Combinado)

```
 oFrm:aLista:={"Lúnes","Martes","Miércoles","Jueves","Viernes"}
 oFrm:cLista:=oFrm:aLista[1]
@ 5,15 COMBOBOX oFrm:oCbx VAR oFrm:cLista;
       ITEMS oFrm:aLista;
       OF oFrm:oDlg;
       ON CHANGE .T.;
       VALID  .T.;
       FONT oFont;
       MESSAGE "Seleccione una Opción";
       WHEN .T.
```

Browse (Visualización y Edición de Datos).

El tamaño del browse se define mediante el método Move(nlin,nCol,nAlto,nAncho,.t.) en la cláusula ON INIT del formulario (oFrm:Activate({||oFrm:oBrw:Move(1,1,200,200,.t.)})

```
  oFrm:aData:=ASQL("SELECT CODIGO,APELLIDO,NOMBRE FROM NMTRABAJADOR")

  oFrm:oBrw  := TXBrowse():New( oFrm:oDlg )
  oFrm:oBrw:oFont := oFont
  oFrm:oBrw:SetArray(oFrm:aData,.T.)

  oFrm:oCol:=oFrm:oBrw:aCols[1]
  oFrm:oCol:cHeader:="Código"
  oFrm:oCol:nWidth :=140

  oFrm:oCol:=oFrm:oBrw:aCols[2]
  oFrm:oCol:cHeader:="Apellido"
  oFrm:oCol:nWidth :=200

  oFrm:oCol:=oFrm:oBrw:aCols[3]
  oFrm:oCol:cHeader:="Nombre"
  oFrm:oCol:nWidth :=200

  oFrm:oBrw:bClrStd := {|oBrw|oBrw:=oFrm:oBrw,;
 {0, iif( oBrw:nArrayAt%2=0, 15790320, 16382457 ) } }

  oFrm:oBrw:CreateFromCode()
```

BAR (Barra de Botones).

Debido a que el formulario provee el control Dialogo como contenedor de controles es necesario definir la barra de botones en la cláusula (ON INIT) del dialogo, para lograrlo la clase TDPEDIT del formulario permite definir en el primer parámetro del método Activate(bInit) la llamada al método virtual o función donde se ejecuta la iniciación del formulario.

```
#INCLUDE "DPXBASE.CH"

PROCE MAIN()
```

```
  oFrm:=DPEDIT():New("Titulo","Fichero.edt","oFrm" , .F. )
  oFrm:CreateWindow()

  oFrm:Activate({||oFrm:HacerBarra()})

RETURN

FUNCTION HacerBarra()

   LOCAL oCursor,oBtn

   DEFINE CURSOR oCursor HAND

   DEFINE BUTTONBAR oFrm:oBar SIZE 38,38 OF oFrm:oDlg 3D CURSOR oCursor

   DEFINE BUTTON oBtn;
          OF oFrm:oBar;
          NOBORDER;
          FILENAME "BITMAPS\XSALIR.BMP";
          ACTION MensajeErr("Salir"),oFrm:Close()

  oBtn:cToolTip:="Salir del Formulario"

RETURN .T.
```

MÓDULOS HRB

Son programas generados por el compilador harbour (www.harbour-project.org) y se ejecutan en forma similar a los programas en DLLS, están compuestos por funciones y/o procedimientos que se cargan al inicio del programa y se puede hacer llamadas a las funciones definidas en el programa HRB.

En la carpeta hrb ubicada dentro de la carpeta principal del sistema se encuentran los siguientes ficheros.
Harbour.exe (Compilador 32 Bits)
Build.bat (Fichero de lotes para compilar)
Testhrb.prg (programa prg ejemplo)

Programa test.prg
// Ubicado c:\dpnmwin\hrb\test.prg
#INCLUDE "DPXBASE.CH"

MEMVAR oDp

FUNCTION HRBtest(cText)

```
        MensajeErr(oDp:cEmpresa,cText)

    RETURN NIL
    // eof
```

Pueden notar que el test.prg posee la función HrbTest() que puede ser ejecutada desde cualquier parte de DpXbase. La función HrbTest(), puede hacer llamadas a funciones propias de DpXbase "MensajeErr" y llamadas a procedimientos Dpxbase con tan solo indicar: EJECUTAR("XPROGRAMA",xValor).

Para compilarlo, es necesario acceder a la carpeta c:\dpnmwin\hrb\
Ejecutar:

```
BUILD TEST
```

Build.bat se encarga de compilar el programa test.prg y generar c:\dpnmwin\test.hrb listo para ser cargado y utilizado desde DpXbase.

Ejecución de Funciones del HRB

Es necesario cargar en memoria las funciones definidas en test.hrb, para ello utilizamos el siguiente ejemplo escrito en DpXbase.

```
#INCLUDE "DPXBASE.CH"

PROCE MAIN()

   HRBLOAD("test.hrb")

   HRBTEST("Desde DpXbase")

RETURN
```

Reglas generales: Las funciones no deben tener nombres similares a las funciones existentes en DpXbase. No deben declarar variables estáticas ni definir clases para objetos. La principal función de los módulos HRB es la definición de funciones. Si al cargar el módulo HRB desde DpXbase se presenta un mensaje de error se debe a que el programa HRB está haciendo llamado a funciones no existentes en el ejecutable de DpXbase.

Los módulos HRB sólo se generan desde la compilación de un programa PRG y no como los DLLs de los productos Datapro DOS que requerían compilación y enlazamiento (Linkado) donde se podían involucrar varios programas *.PRG, archivos objetos *.OBJ y archivos de librerías *.LIB

Que es mejor entre DpXbase y HRB?

Ambos modos de trabajo tienen sus ventajas. La principal ventaja de los módulos HRB está en la velocidad de ejecución que es mucho mayor que DpXbase, además la declaración de variables locales son reales permitiendo crear diversas instancias de controles como diálogos (no modales) o ventanas MDI. El programa fuente no es necesario para la ejecución, lo cual permite proteger la propiedad intelectual del código.

DpXbase, posee su editor propietario y es más cómodo, escribir, compilar y ejecutar. Siempre estará disponible el programa fuente. No permite crear instancias para diálogos no modales ni ventanas MDI, por ello siempre debe apoyarse en clases como TDPEDIT o TMDI.

INFORMES

CAMPOS DE LAS TABLAS

Este reporte muestra todos los campos de las tablas a través de los campos; descripción, tipo, longitud y decimales. Está ordenado por tabla.

RELACIONES ENTRE TABLAS

Muestra los reportes de las tablas a través de; tablas enlazadas, campos de la tabla enlazada, campos de la tabla solicitante, modifica y elimina. Está ordenado por tabla.

TABLAS

Muestra todas las tablas a través de los campos número, nombre de la tabla, utilización, aplicación y nombre del destino. Está ordenado por número.

PROCESOS

EXPORTAR TABLAS DE CONFIGURACIÓN

Este Proceso permite exportar todas aquellas tablas pertenecientes a la base de datos principal ADMCONFIG hacia tablas DBF para luego ser utilizadas para importar la información hacia otras tablas con su respectiva parametrización.

RECOMPILAR PROGRAMAS

Proceso mediante el cual se puede volver a precompilar todos los programas fuentes en caso que uno de ellos se haya modificado y se requiera volver a la configuración inicial del mismo.

GENERAR FICHEROS DE CONFIGURACIÓN

Proceso mediante el cual permite al usuario realizar una revisión de todos los formularios con su respectiva configuración inicial.

EXPORTAR ESTRUCTURA DE DATOS

Permite exportar una o todas las tablas bien sea nueva o ya existente, la cual podrá ser adaptada a cualquier tipo de base de datos que previamente tenga su respectiva configuración.

IMPORTAR ESTRUCTURA DE DATOS

Permite Importar una tabla al sistema ya sea nueva o existente con otra configuración distinta a la original.

GLOSARIO DE TÉRMINOS

UDF: User Define Function. Definición otorgada a las definiciones creadas por el usuario.

CURSOR: Respuesta recibida en un plano bidimensional (Línea y Columnas) desde una consulta SQL.**DSN:** Data Source Name. Nombre de las conexiones ODBC con servidores de bases de datos, previamente definidas en orígenes de Datos ODBC de Herramientas Administrativas del Panel de Control de Windows.

CLASE: Código fuente o partitura de las acciones de un objeto, contiene: clase padre (opcional si hereda desde otro objeto), métodos que indica lo que hace el objeto y Data que indica lo que tiene el objeto. Obligatoriamente debe poseer un método constructor denominado NEW() y un Destructor denominado END().

MÉTODO. Method de la clase. Conforman las funciones o actividades que realiza el objeto, en forma analógica, puede ser comparada con las funciones estáticas de un programa que solo se pueden usar desde el mismo programa donde fue declarada. Para utilizar el método de una clase debe ser llamada indicando el nombre del objeto, seguido dos puntos y finalmente el nombre del método con sus respectivos parámetros. Ejemplo: oObjeto:Metodo(uPar1,uPar2,uParn). Método Virtual, ha sido una definición adicional implantada en DpXbase.

DATA: Valores o contenidos del Objeto. Se representan a través de nombres válidos como variables. También puede ser definido como “Lo que tiene el objeto”. Ejemplo: Objeto:xData.

OBJETO: Componente abstracto que realiza actividades según las tareas definidas en los métodos y valores almacenados en memoria (data). El objeto permite trabajar múltiples veces en forma

simultánea (Instancias). Una clase puede ser definida a partir de otra clase aprovechando toda su partitura (herencia).

FUNCTION: Definición de un programa compuesto de parámetros, declaración de variables, instrucciones, llamadas a otras funciones y que devuelven un valor. La función debe ser identificada con un nombre y debe recibir parámetros desde la instrucción que la ejecuta: Ejemplo: Sumar(10,20).

```
FUNCTION Sumar(nValor1,nValor2)
RETURN nValor1+nValor2
```

ENLAZAMIENTO: Definición del proceso final para la generación de un programa ejecutable o DLL donde se utilizan programas objetos producidos por la compilación y librerías contentivas de las funciones invocadas por los programas fuentes.

COMPILACION: Proceso de conversión en archivo objeto (OBJ) a partir de las instrucciones escritas en forma (humana) por un programador.

DLLS: Dynamic Link Library. Programas compilados y enlazadas de la misma forma que un programa ejecutable con la diferencia que se carga una sola vez en memoria y sus funciones pueden ser llamados o utilizados desde varios programas ejecutables. Otro uso de ficheros DLLs pueden estar compuestos de recursos compuestos por diseños de controles: Combobox, text, checkbox, etc.