

FiveWin = (Clipper)^{Windows}



Andrés Romero García

Edición 0.0 (Octubre-2000)

CONTENIDO

CONTENIDO	2
INDICE.....	5
1.- INTRODUCCION	7
Introducción al LIBRO	8
Requerimientos de hard. y soft.	10
Créditos.....	11
Introducción a Fivewin	12
¿ Que es Fivewin ?	12
Eventos	13
Programación Orientada a Objetos.....	14
Preprocesador	15
Programación con Fivewin	16
Requisitos para seguir el tutorial	16
Programa inicial	20
Barra de menú	23
Barra de botones	26
Cajas de diálogo	28
Texto fijo (SAY).....	29
Texto editable (GET).....	29
Push Button	31
Check Box.....	32
Radio Button	32
List Box	33
Combo Box	34
Scroll Bars.....	35
Icon	36
Bitmaps	37
Browse	38
Fólder.....	38
Colores.....	40
Conclusión	42
WORKSHOP	43
Tutorial de Workshop.....	45
Inclusión de recursos	58
Recursos en DLL	70
Programa WORKSHOP	71
Programación Orientada a Objetos.....	87
2.- CONTROLES.....	94
PROGRAMACIÓN DE CONTROLES	95
BIPMAPS	100
BROWSE.....	105

BRUSH	110
BTNBMP	117
BUTTON	121
BUTTONBAR	128
CHECKBOX	131
CLIPBOARD	135
COMBOBOX	136
CURSOR	140
DATABASE	142
DEFAULT	144
DIALOG	145
FOLDER	151
GET (no memo)	157
GET multilinea (memos)	164
GROUPS	169
ICONS	171
IDLEACTION	177
IMAGEN	178
LISTBOX	180
MENU	190
MESSAGE	197
METAFILE	199
METER	201
MSGITEM	205
PAGES	208
PEN	209
RADIO	210
RELEASE	214
SAY	215
SCROLL	220
TABS	225
TIMER	227
UNTIL	228
VBX	229
WINDOWS	232
PAGINA PATRON	248
3.- IMPRESION	249
CONTROL DE IMPRESORA	250
GENERADOR DE INFORMES	253
4.- OOP	273
PROGRAMACIÓN ORIENTADA A OBJETOS	275
Introducción	275
Estructura de un objeto	276
Encapsulamiento	279
Herencia	279
Polimorfismo	282
OBJECT: POO en Fivewin	283
Ejemplo de gerarquia	284

Consideraciones finales	290
5.- CLASES.....	293
BIPMAPS	294
BROWSE	296
BRUSH	298
BTNBMF	299
BUTTON	301
BUTTONBAR	302
CHECKBOX	304
CLIPBOARD	305
COMBOBOX	306
CURSOR	307
DATABASE	308
DIALOG	311
TFolder.....	322
GET (no memo)	324
GET multilinea (memos)	326
GROUPS	327
ICONS.....	328
IMAGEN.....	329
TListBox	330
TMenu.....	333
TMenuItem.....	335
TMsgBar	337
TMetaFile	339
TMeter.....	340
TMsgItem	341
TPages.....	342
TPen	343
TRadio	344
SAY	345
TPrinter	346
TScrollBar	360
Tabs.....	362
TIMER.....	363
UNTIL.....	364
TVbControl.....	365
TWindows	368
6.- FOROS DE NOTICIAS.....	405
FORO DE NOTICIAS.....	406
ENLACES EN INTERNET	411
7.- APENDICES	412
A – INDICES CDX.....	413
B - FICHEROS DE AYUDA	427

INDICE**A**

Acrobat, 9
Alfredo Sanz, 11
AYUDA, 427

B

Barra de botones, 26
Barra de menú, 23
BIPMAPS, 100, 294
Bitmaps, 37
Browse, 38
BROWSE, 105, 296
BRUSH, 110, 298
BTNBMP, 117, 299
BUTTON, 121, 301
BUTTONBAR, 128, 302

C

Cajas de diálogo, 28
CDX, 413

Ch

Check Box, 32
CHECKBOX, 131, 304

C

CLIPBOARD, 135, 305
Colores, 40
Combo Box, 34
COMBOBOX, 136, 306
CURSOR, 140, 307

D

DATABASE, 142, 308
DEFAULT, 144
DIALOG, 145, 311
DLL, 70

E

Encapsulamiento, 279
Eventos, 13

F

Fernando Ballesteros, 11
Fivewin, 12
FOLDER, 151
Fólder, 38
Francisco García, 11
Fwctrls, 11, 85

G

GENERADOR DE INFORMES, 253
GET, 29, 157
GET (no memo), 324
GET multilinea, 164
GET multilinea (memos), 326
GROUPS, 169, 327

H

Herencia, 279

I

Icon, 36
ICONS, 171, 328
IDLEACTION, 177
IMAGEN, 178, 329
IMPRESORA, 250

J

Jesús Morán, 11

L

List Box, 33
LISTBOX, 180
LNK, 18

M

MAK, 18
Manuel Expósito, 11
MENU, 190
MESSAGE, 197
METAFILE, 199
METER, 201
Método, 14
MSGITEM, 205

O

OBJECT, 283

P

PAGES, 208
PDF, 9
PEN, 209
Polimorfismo, 282
Preprocesador, 15
Programación Orientada a Objetos, 14, 87
PROGRAMACIÓN ORIENTADA A OBJETOS, 275
Push Button, 31

R

RADIO, 210
Radio Button, 32
recursos, 58
RELEASE, 214
Ricardo Ramírez, 11

S

SAY, 29, 215, 345

SCROLL, 220
Scroll Bars, 35

T

Tabs, 362
TABS, 225
TFolder, 322
TIMER, 363
TListBox, 330
TMenu, 333
TMenuItem, 335
TMetaFile, 339
TMeter, 340
TMsgBar, 337
TMsgItem, 341
TPages, 342
TPen, 343
TPrinter, 346
TRadio, 344
TScrollBar, 360
TVbControl, 365
TWindows, 368

U

UNTIL, 228, 364

V

Variable de instancia, 14
VBX, 229

W

WINDOWS, 232
WORKSHOP, 43, 71



1

INTRODUCCION

Este capítulo trata de dar una visión del conjunto del manual, y sobre todo de sintonizar entre el autor y el lector para comprender por que, para que y como fue hecho. De sus intenciones futuras y de la colaboración que se espera del amable lector.

Introducción al LIBRO

Existen varios y buenos “manuales” sobre Fivewin. No pretendo superarlos, sino más bien recopilarlos en uno solo. Labor ardua que ha de ir acompañada de ensayos, programación, etc. y que necesitaré de todos vosotros, los lectores, para corregirla al principio e intentar superarla en sucesivas ediciones.

Se han tomado como referencia los siguientes manuales y documentación:

- **Fivewin.**- Manual original de Fivetech.
- **Ayuda en línea.**- Ayuda que acompaña a la versión 2.0 de Fivewin.
- **Foro de noticias de Fivewin en español.**- Foro en internet
- **CA-Clipper + Fivewin.**- de Ortiz de Zuñiga
- **Introducción a la programación Windows Fivewin.**- de Francisco García, Jesús Morán y Fernando Ballesteros
- **Curso de Fivewin. Programación clipper para windows.**-
- **Otros artículos sueltos encontrados en el foro de noticias de Internet.**
(Ver el apartado de créditos más abajo para tener una relación detallada.)

De los documentos que indicaban explícitamente la prohibición de copia, no se ha transcrito NADA. No obstante en un tema técnico donde $2 + 2$ siempre son 4, es muy difícil que no aparezcan términos iguales, explicaciones que llevan a la misma respuesta etc.

El principal motivo es servir de ayuda a los que como yo empezamos en el entorno visual de Windows. Va dirigido a programadores de Clipper que YA conocen el lenguaje básico. Para seguir el tutorial primero y el manual después ha de tenerse buenos conocimientos de programación en clipper, y algunos conocimientos del preprocesador. No hace falta saber sobre Programación Orientada a Objetos (OOP) puesto que su enseñanza es parte de este libro. NO he querido hacer el libro cuando ya supiese sobre el tema, sino comenzarlo cuando todavía no conozco Fivewin. Al hacerlo así bajaré mas a

los detalles que otros autores dan por sabidos y que a mi me ha costado tanto aprender por mi cuenta leyendo entre líneas.

Una condición que ayuda bastante para comprender la programación en Windows, es **cambiar el Chip**. Es realmente difícil trabajar en DOS por las mañanas y en Windows por la tarde. Lo mejor (aunque no siempre posible) es que cuando comiences a programar en Windows, te dejes el sistema en DOS. Es similar al paso de la programación en línea a la programación estructurada. Una vez que se piensa y se vive en Windows, olvídate de DOS, es cosa del pasado.

Se ha preferido dividir el libro en dos partes: un **Tutorial** y un **Manual**

En el tutorial se expondrán los temas por orden de dificultad, empezando por lo más simple y terminando en lo más complejo. Va acompañado de muchos ejemplos e incluso para dar una idea de conjunto, se crea una pequeña aplicación sobre el control de una librería, similar a la que todos tenemos en casa o en la oficina.

El manual, al contrario que el tutorial, esta por orden alfabético dentro de las categorías. Su principal objeto es servir de consulta para los que ya saben desenvolverse con cierta soltura dentro de Fivewin y necesita un dato, una matización o un ejemplo que le sirva para su problema concreto.

Se ha procurado utilizar el inglés al mínimo, entre otros motivos porque lo desconozco. La traducción de palabras se hace de forma coloquial. No obstante hay algunas palabras o frases con las que los programadores estamos tan acostumbrados que no nos importa seguir utilizando el término inglés, o bien que es imprescindible puesto que son palabras de la propia programación. A nadie se le ocurre traducir Windows, ni Clipper, etc. Con respecto al idioma hay que hacer constar que vivo en una región española y por tanto se utilizan términos españoles. Por ejemplo, se dice ratón y no mouse como en Iberoamérica. De todas formas para poder “internacionalizar” el libro, no tengo inconveniente en incluir palabras y frases de otros países hispanoparlantes.

Si un artículo, fichero de ayuda u otro texto completo esta copiado de un original en inglés, puede que todavía no se haya traducido. Se supone que puedes leer un texto en inglés de modo que no puedo “perder” tiempo en la traducción (si alguien lo quiere traducir, me lo pasa, lo sustituyo y todos tan contentos)

La presentación del libro se hace en formato **PDF** (Acrobat) para que pueda leerse por pantalla y/o imprimir. No se da en el formato original del Word para que no pueda ser modificado fácilmente y proliferar copias distintas al original. No obstante, estoy dispuesto no solo a corregir aquello que este mal, sino también a mejorar o incluir nuevos conceptos, explicaciones, etc. La presentación original esta pensada para ser impresa en **DIN A5** a una sola cara en una impresora de chorro de tinta en color, pero también puede ser impresa en DIN A4 en blanco y negro (se han buscado colores claros en los fondos y oscuros en las letras para que se vean bien en blanco y negro). Si tienes una impresora láser podrás hacerlo fácilmente. Si es una de chorro de tinta, tendrás todos los colores de las pantallas de ejemplo así como las diversas matizaciones que en el texto se hace en color.

La edición del manual llevará un camino paralelo a mis necesidades de Fivewin. Sería demasiado pretencioso por mi parte hacer un manual de algo no conozco, de modo que a medida que vaya teniendo necesidades de programación, iré aprendiendo y pasándolo al manual. De esto se deduce que poco voy a enseñar a los que ya saben pero al menos, quizás aprendan los que ahora empiezan.

Requerimientos de hard. y soft.

Clipper y Fivewin tienen unas necesidades de hard y soft realmente escasas para los equipos y sistemas operativos actuales. Casi seguro que cumples con los requerimientos mínimos. Lo que ahora expongo son los requerimientos para seguir el tutorial y los ejemplos del manual.

Todos nosotros tenemos nuestro propio sistema de desarrollo en el que nos encontramos a gusto. Seguramente no hay dos entornos exactamente iguales. Como profesionales somos capaces de comprender el entorno de otro programador, pero a menos que nos demuestre que es mucho mejor, preferimos seguir con el nuestro. Con esto quiero decir que no pretendo que cambies tu sistema, pero sí explicarte como se hizo este libro

El equipo hard es un **Pentium III** a **500 Mhz**, con **64 Mb.** de RAM. El disco duro con capacidad suficiente. La pantalla una SuperVGA de **17"**. Un **CD-ROM** y un **CD-Writer**. Como impresora patrón se utiliza una **Epson Stylus Color 900**. El equipo está integrado dentro de una red local **100 BaseT**. El resto es irrelevante.

El soft consiste básicamente en el sistema operativo Windows-98 y en la red local un sistema Windows-NT. Lo importante son las herramientas de desarrollo tanto de compilación y enlazado (linker) como las librerías de terceros. Vamos a ello.

Como compilador **Clipper** se dispone de la versión **5.3c** en inglés. Se utiliza normalmente el sistema de índices **CDX** que viene con el propio clipper.

Como programa de enlace disponemos del **Blinker 5.1** en su versión original (no la versión que viene con clipper).

Como librería Fivewin, se dispone de la versión **2.0**

Como librerías externas, trabajamos fundamentalmente con **Funcky II** aunque procuraré no usarla.

También disponemos de una librería propia (**romewin.lib**) que procuraré no usarla en los ejemplos y en caso necesario indicaré el fuente de la función o funciones que se utilicen.

Como parte de los programas dispongo de un include (**romewin.inc**) que tiene definidos los datos que más se utilizan como por ejemplo **#define SI .T.** o **#define NO .F.** (El **SI** y el **NO** en lugar de **.T.** y **.F.**, es realmente cómodo)

Muchos controles de Fivewin se basan en dos **.DLL's** externas, **BWCC.DLL** y **CTL3D.DLL**.

Créditos

Este libro debería llamarse **Block de Notas**. Aquí se van poniendo el trabajo propio del autor, trabajos de colaboradores y trabajos de terceros aparecidos en Internet de una forma u otra, que si no indican expresamente la prohibición de su copia, se insertan previa adaptación y maquetación. Como sería muy difícil ir poniendo “este párrafo corresponde a Fulanito, este otro a Menganito” etc. he preferido hacer este apartado donde se reconocen los méritos de los distintos autores, su nombre y la o las partes en las que ha intervenido.

Alfredo Sanz	Artículos tomados de www.cincowin.com sobre las clases TFolder y Twindows.
Fernando Ballesteros	Coautor del artículo sobre ficheros de Ayuda (*.HLP)
Francisco García	Coautor del artículo sobre ficheros de Ayuda (*.HLP)
Jesús Morán	Coautor del artículo sobre ficheros de Ayuda (*.HLP)
Manuel Expósito	Notas parciales de un artículo sobre OOP.
Ricardo Ramírez	Creador de los recursos propios de Fivewin para añadir al workshop, Fwctrls Error! Marcador no definido.

Introducción a Fivewin

¿ Que es Fivewin ?

Parece la pregunta del millón. Hay muchas denominaciones según los diversos autores. Para mi, es simplemente una librería muy bien desarrollada. Actúa como una librería de terceros cuyas funciones son principalmente enlaces con objetos y estos a su vez son enlaces con el API de windows y funciones propias. Hace un extenso uso del preprocesador de forma que prácticamente no se utilizan funciones de tipo clásico a menos que se quieran expresamente. Al ser una librería con gran cantidad de objetos, se utiliza al máximo la herencia, encapsulación, polimorfismo y todas aquellas cualidades de la Programación Orientada a Objetos.

El usuario de Fivewin, (tú, el programador) no tiene porque saber de objetos ni el API de windows. Únicamente las instrucciones que llaman a esos objetos. De todas formas su conocimiento amplía las posibilidades de programación. Puede ser la diferencia entre una aplicación normal y otra con aspecto mas profesional. Dentro del tutorial y posteriormente en el manual, hay capítulos dedicados a esta forma de programar que te aconsejo dar un repaso antes de juzgar si es bueno o no el sistema OOP. Seguro que cuando lo utilices no podrás dejar de hacerlo como nos ha pasado a la mayoría de los programadores. Recuerdo cuando empecé con la programación estructurada. Venía del sistema clásico de Basic, Cóbol y Assambler. El no poder utilizar la instrucción **GOTO** me parecía imposible. Ahora aunque quiera utilizarla no se como hacerlo.

Lo mismo digo de la programación en lenguaje **C**, no es necesario, pero su conocimiento permitirá hacer funciones rápidas que en clipper serian lentas o imposibles. El lenguaje **C++** se sale de las intenciones de este libro y solo se verán algunos detalles de cómo interactuar entre clipper y C. De todas formas no te preocupes, no hay porque saber nada de **C**.

La programación en el entorno Fivewin (esta mal dicho programar en Fivewin) cambia su mecánica debido al sistema de eventos, es decir, ya no vale el sistema clásico de una instrucción, la siguiente, la siguiente. Ahora es **Windows** quien toma control de nuestro programa. Nosotros únicamente le pasamos los elementos con los que debe actuar, pero no la forma concreta de hacerlo.

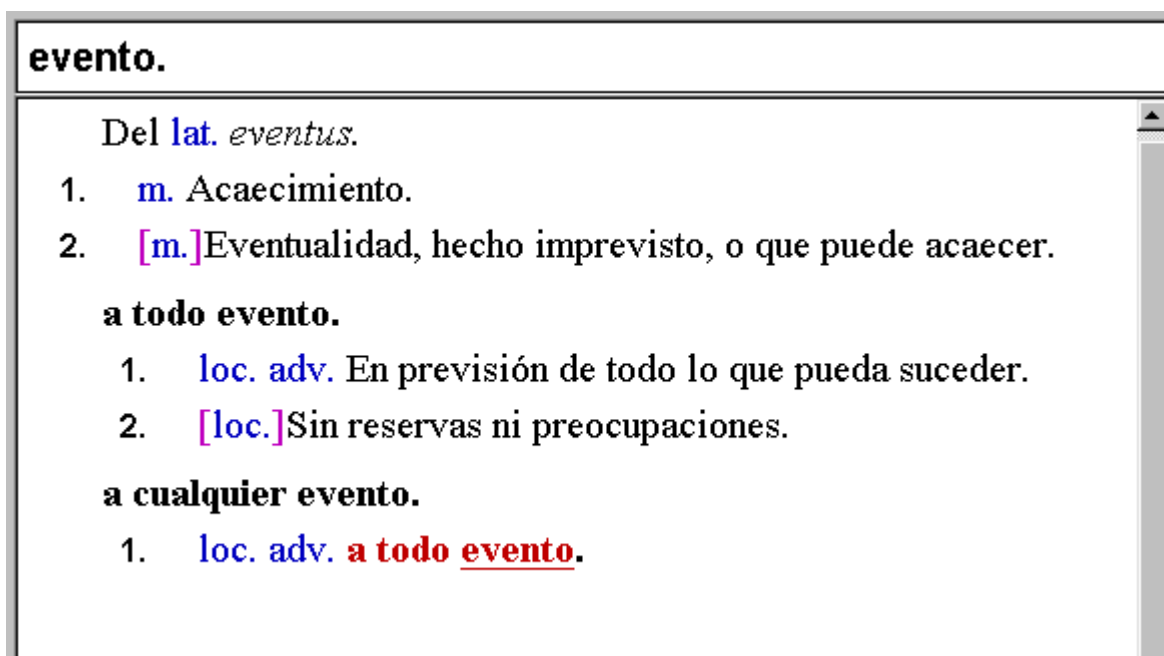
La gran ventaja de Fivewin para los que venimos del mundo de Clipper (el mejor sistema de programación de base de datos) es que no hay que renunciar a nuestro clipper de toda la vida en el 100% de las instrucciones **no visualizables ni imprimibles**. Solo hay que modificar nuestros hábitos en la programación que actúa sobre pantallas e impresoras. Después de una búsqueda por entornos Windows, como Visual Basic, FoxBase, Delphi y C++ Builder, el rendimiento optimo ha sido seguir con el sistema Clipper y "algo" que pudiese ser visualizado e impreso desde Windows en ventanas Windows con aspecto Windows. Naturalmente que Delphi o C++ Builder son más rápidos en tiempo de ejecución, pero para conseguir esas milésimas de segundo o quizás incluso segundos de rapidez, cuantas HORAS hay que tardar en hacer el programa. Cuando el

usuario final vaya a tener la aplicación, ya se ha cansado de esperar y se ha ido con la competencia. Os hablo por experiencia. Después de una año programando en Delphi tuvimos que volver a Clipper-Fivewin.

Otra ventaja de Fivewin es que nos introduce sutilmente en la Programación Orientada a Objetos. Es como si dijéramos “ya que esta aquí, vamos a verla” y una vez en ella quedaremos atrapados en sus ventajas. Ya no podremos programar sin ella, sobre todos los profesionales que hacen grandes aplicaciones.

Eventos

Según el CD-ROM de la Real Academia de la Lengua,



y tiene como palabras sinónimas entre otras, **acontecimiento** y **posibilidad**. Lo que veremos aquí, como es lógico, es su significado dentro de la programación en Windows.

En la programación estructurada estamos acostumbrados a que las instrucciones se realicen unas detrás de otras en el mismo orden que las hemos escrito en nuestro PRG. Puede haber bucles de tipo **do while**, ejecuciones selectivas de tipo **case** o **if** pero el flujo del programa es previsible en tiempo de programación. Ahora, al ir introduciéndonos en Windows tenemos que pensar en otra forma de programar. Naturalmente siguen existiendo las estructuras clásicas pero también entran en escena otras formas nuevas como son los **eventos**. Un evento tiene un sentido intuitivo solo con la palabra y puede ser consultado en cualquier diccionario. Según mi entender, para nosotros, **Evento es todo aquello que ocurre fuera de nuestro programa que puede influir sobre él.**

Supongamos nuestro sistema clipper. Si estamos en un **read**, un evento puede ser pulsar la tecla **AvPág**. No ha sido nuestro programa quien ha pulsado (no hemos puesto la instrucción **keyboard chr(K_ENTER)**). Ha sido el usuario. Por otro lado, el usuario no se ha levantado a tomar un café, cosa que no actuaría sobre nuestro programa. ha pulsado una tecla concreta que nuestro programa ha interpretado como “sal del read y continua con el proceso”. Esto quiere decir que las aplicaciones DOS también tienen eventos, la diferencia con Windows es que los eventos en este sistema controlan la realización del programa, y no como en DOS que simplemente hacen continuar el programa. Es como tener un **SET KEY** de todas las posibles actuaciones del usuario. Programar pensando en SET KEY para todo, los retornos de las funciones, nuevas llamadas a otras funciones, etc. en DOS es complicadísimo. En Windows es más fácil, pero tenemos que cambiar nuestra forma de pensar. Para que este cambio te sea cómodo se ha escrito este tutorial.

Realmente el sistema de eventos se realiza entre Windows y Fivewin. Al programador solo le llegan pequeños restos del diálogo entre ellos. La mayoría de los trabajos los realiza Fivewin de modo que no hemos de preocuparnos mucho.

Programación Orientada a Objetos

Como en el resto de este libro, las explicaciones se hacen en varios niveles. Ahora no vamos a ver el sistema OOP en profundidad. Ni siquiera superficialmente. Solo veremos de que se trata con el fin de que te suene el concepto. Si quieres ver una introducción a este tipo de programación, lee el capítulo correspondiente en la página **¡Error! Marcador no definido..**

Los objetos son como funciones de clipper donde las variables y funciones internas pueden modificarse o llamarse externamente. Naturalmente no son internas del todo, puesto que de serlo no podrían modificarse ni llamarse desde fuera. Por ese motivo no son funciones normales, son objetos 😊

La ventaja de los objetos es que su comportamiento puede modificarse desde fuera del programa objeto, es decir desde nuestro propio PRG. Voy a reproducir un cuadro del apartado sobre OOP. En este cuadro se muestra la terminología que necesitas para poder, al menos, “oir” el vocabulario.

Equivalente Clipper	Terminología OOP	Descripción
Variable	Variable de instancia	Lugar donde están los datos. Se puede modificar (no siempre) o leer.
Función	Método	Es la función que se puede ejecutar desde nuestro programa para que el objeto haga una determinada cosa. Por ejemplo oTbr:up() sube una línea en un objeto Browse. Los métodos pueden devolver datos a nuestro programa, o pueden modificar variables de instancia que después podemos analizar, o simplemente ejecutar algo sin modificar ningún dato.

Preprocesador

Fivewin no sería lo mismo si no existiese el preprocesador. Esta útil herramienta que viene con Clipper desde la versión 5.0 es fundamental para la sintaxis de Fivewin. Todos conocemos, al menos en parte, el preprocesador de clipper, no vamos a entrar en detalle. Lo que más se utiliza es la cláusula **#xcommand**. Veamos un trocito del fichero fivewin.ch que está en el directorio **\INCLUDE** y que es imprescindible insertar en nuestros fuentes:

```
#xcommand @ <nRow>, <nCol> SAY [ <oSay> <label: PROMPT,VAR > ] <cText> ;
    [ PICTURE <cPict> ] ;
    [ <dlg: OF,WINDOW,DIALOG > <oWnd> ] ;
    [ FONT <oFont> ] ;
    [ <lCenter: CENTERED, CENTER > ] ;
    [ <lRight: RIGHT > ] ;
    [ <lBorder: BORDER > ] ;
    [ <lPixel: PIXEL, PIXELS > ] ;
    [ <color: COLOR,COLORS > <nClrText> [,<nClrBack> ] ] ;
    [ SIZE <nWidth>, <nHeight> ] ;
    [ <design: DESIGN > ] ;
    [ <update: UPDATE > ] ;
    [ <lShaded: SHADED, SHADOW > ] ;
    [ <lBox: BOX > ] ;
    [ <lRaised: RAISED > ] ;
=> ;
    [ <oSay> := ] TSay():New( <nRow>, <nCol>, <{cText}>,;
    [<oWnd>], [<cPict>], <oFont>, <.lCenter.>, <.lRight.>, <.lBorder.>,;
    <.lPixel.>, <nClrText>, <nClrBack>, <nWidth>, <nHeight>,;
    <.design.>, <.update.>, <.lShaded.>, <.lBox.>, <.lRaised.> )
```

El preprocesador busca en nuestro programa fuente la estructura indicada después **#xcommand** (o mejor dicho, a medida que lee líneas, ve si corresponde a esta estructura). Si encuentra una línea con estas características, la convierte (en otro fichero) al formato que hay en la segunda parte, después de **=>**. Si encuentra una estructura parecida, pero con algún dato mal, da un error y no continua con la compilación. Si a nuestra línea le falta algún dato, podrá el valor por defecto o NIL y será la función `Tsay()new(...)` quien en tiempo de ejecución ponga los valores por defecto, de un error o cualquier otra cosa que haya decidido el programador del objeto. Cada uno de los valores indicados entre **<>** en la segunda parte, es sustituido por su equivalente de la primera.

En este ejemplo vemos que ha habido que reconstruir el comando SAY totalmente. Ahora hay conceptos como borde, píxel, ancho, alto, etc. que no había en DOS

Ya entraremos en detalle en la parte explicativa de los distintos elementos que componen la programación en windows. Ahora solo ver que prácticamente todos los comandos propios de Fivewin son convertidos gracias al preprocesador a llamadas a objetos que se encuentran en la librería.

Si quieres aprender sobre el preprocesador, además de leer la documentación que acompaña a Clipper, te recomiendo que compiles con el parámetro **-p** o **\p**. Después de compilar, mira el fichero con extensión **.PPO**. Este fichero contiene lo que realmente se compila. Todos los comandos han sido pasados a funciones. A mi me ayuda mucho cuando tengo que depurar un programa o comprender un determinado comando.

Programación con Fivewin

En este capítulo vamos a comenzar realmente con la programación. Vamos a ver únicamente los elementos de una aplicación Windows de una forma sencilla.

La intención ahora no es aprender a programar sino aprender sobre las cosas que vamos a programar. Es decir, no se trata de aprender como se hace y que hace un **Radio Button**, sino mas bien, aprender **que es** un **Radio Button**.

Como corresponde a un tutorial, vamos a ir aprendiendo con ejemplos sencillos pero operativos al ciento por ciento. El libro lo puedes leer como si fuese una novela (no te lo recomiendo) o poniéndolo al lado del ordenador e ir practicando con los ejemplos. Para poder hacer esto último es necesario que tu y yo hablemos el mismo idioma. Naturalmente que es el español, pero me refiero al mismo idioma en programación, ambiente, etc. Para ello lee el apartado siguiente:

Requisitos para seguir el tutorial

Por razones prácticas, el ordenador donde se desarrollan los ejemplos puede que se utilice también para otras cosas de forma que lo mejor es crear un ambiente propio de trabajo para Fivewin. Concretamente crearemos un directorio similar a de CLIPPER5\... pero para nuestros fines. Ya verás que ni los programas, ni los módulos OBJ, ni las librerías, son las mismas que para DOS. Muchas son iguales pero otras desaparecen y otros son nuevas. Te recomiendo que hagas un directorio del tipo **\CLIWIN** de donde colgarán los mismos subdirectorios que de CLIPPER5, es decir

- **\CLIWIN\BIN** Para los ejecutables, tipo Clipper, Blinker, etc.
- **\CLIWIN\INCLUDE** Los includes necesarios (lee más abajo)
- **\CLIWIN\OBJ** Los módulos OBJ clásicos
- **\CLIWIN\LIB** Las librerías

Puedes poner también si lo crees necesario:

- **\CLIWIN\SOURCE** Los Fuentes originales en ingles
- **\CLIWIN\FUENTES** Tus propios Fuentes de OBJ y LIB

Para decidir que ficheros incluir, una práctica que yo hago es la siguiente:

Primero pongo en cada directorio todo lo que estoy seguro que hará falta, por ejemplo CLIPPER.EXE, BLINKER.EXE FIVEWIN.CH, etc. Luego paso a compilar y enlazar los programas normalmente. Como es lógico me dice que falta algo, da un error y se para. Busco lo que falta y lo añado. Luego vuelvo a compilar. Ahora dirá que falta otra

cosa. Repito el ciclo hasta que el programa en cuestión esta realizado y funcionando. Si ese programa no utiliza una determinada cosa, no lo hecha en falta y yo me ahorro de llenar los directorios con cosas que no sirven. Cuando compile otro programa, ahora quizás haga falta algo más, pues lo pongo y ya esta. A la vuelta de varios programas, estarán todos los directorios al completo y sin nada superfluo. Naturalmente tu eres dueño de tu disco duro. También puedes poner todo y ahorrarte el ir añadiendo ficheros cada vez que te los pide.

Como directorio de trabajo crearemos **\FW** y dentro de éste, el subdirectorio **\TUTORIAL**. Todo esto es con el fin de sincronizar tu equipo con el libro. Si quieres hacerlo seguramente se te ocurrirán otras ideas perfectamente válidas.

Debes tener un fichero **BAT** que te cargue las variables de entorno, etc. Por ejemplo puede ser algo así como **FW.BAT** cuyo contenido sería:

```
SET PATH=%PATH%;\CLIWIN\BIN
SET INCLUDE=\CLIWIN\INCLUDE
SET OBJ=\CLIWIN\OBJ
SET LIB=\CLIWIN\LIB
SET TEMP=\TEMP
SET CLIPPER=\\SWAPPATH: '%TEMP%' \\TEMPPATH: '%TEMP%'
```

Después de muchos años programando en clipper he llegado a la conclusión de que es bueno cambiar “algunas” cosas en la programación. Son muy pocas, pero me he acostumbrado de tal forma que no puedo prescindir de ellas. Estos cambios los tengo en un fichero que denomino **romewin.inc** cuyo contenido básico es el siguiente:

```
#include 'inkey.ch'
#include 'set.ch'

#command si27ret => if lastkey()==27 ; return NIL ; endif
#command si27loo => if lastkey()==27 ; loop ; endif
#command si27exi => if lastkey()==27 ; exit ; endif

#define K_INTRO    13

#command BUSCA(<p>) => dbseek(<p>,.T.)

#define SI        .T.
#define NO        .F.

#define O2A oemtoansi
#define A20 ansitooem

memvar nNegro,nBlanco,nAzul,nVerde,nRojo
memvar cTemp

#include 'fivewin.ch'
```

Tu puedes incluir estas líneas en otro fichero include que utilices, o grabar este fichero y poner **#include 'romewin.inc'** en tus programas.

En cuanto a la compilación y el enlace, dispón de un fichero BAT en el directorio **\CLIWIN\BIN** que llama a **RMAKE.EXE** y a **BLINKER.EXE**. Mi fichero se llama **MAK.BAT** y el contenido es el siguiente:

```
@echo off
RMAKE %1.MAK %2
@echo.
```

```
if errorlevel = 1 goto fin
@echo.
blinker @@1
@echo -----
:fin
```

Como ves este fichero BAT es común para todos los programas. Para compilar y enlazar cada uno de ellos utilizo dos ficheros **<programa>.MAK** y **<programa>.LNK**.

<fichero>.MAK

La utilidad RMAKE.EXE permite que partiendo de una lista de ficheros PRG, solo se compilen aquellos que han tenido variación. El método consiste en comparar la fecha y hora de los PRG con sus respectivos OBJ. Si el PRG es más moderno que el OBJ, se compila. Si es igual o más antiguo no se compila.

```
.PRG.OBJ:
lh clipper $* -m -n -a -w -p
    Ejecutando clipper en memoria alta, tenemos mas sitio para los include.
```

```
<fichero1>.obj      : <fichero1>.prg
<fichero2>.obj      : <fichero2>.prg
<fichero3>.obj      : <fichero3>.prg
<fichero4>.obj      : <fichero4>.prg
.....
<ficheroN>.obj      : <ficheroN>.prg
```

Naturalmente el programa RMAKE hace mucho mas que esto, pero ahora no es el momento de analizar todas sus posibilidades.

<fichero>.LNK

El enlace de los módulos OBJ para convertirlos en EXE se realiza mediante BLINKER versión 5.1. (La ventaja de la versión 5.1 respecto a la 5.0 es un mayor espacio para el stack y las variables locales).

BLINKER también comprime el fichero EXE en el disco, con una autodescompresión en el momento de la carga en memoria, de forma que no se nota nada en tiempo de ejecución.

```
blinker incremental off
    Esto evita ampliar el ejecutable para el linkado incremental. Hoy día los ordenadores son muy rápidos y casi no hace falta. Si no lo pones, tardará menos tiempo en enlazar, pero el ejecutable será mas grande.
```

```

blinker clipper symbol on
    Aumenta el espacio de la table de símbolos. Si no hay muchos, puedes
    ponerlo a off.
blinker executable nodeleted
    En caso de quedarse un elace a medio hacer, con esta instrucción evitamos
    que se borre el anterior EXE.
blinker executable compress
    Esto comprime nuestro ejecutable. Al cargarse en memoria, lo descomprime
    automáticamente
blinker executable clipper f30
    Prepara el ejecutable para poder habrir hasta 30 ficheros. Ojo, esto es
    independiente del número máximo de ficheros que puede abrir windows. Esto
    último ha de controlarse desde el programa mediante la instrucción
    setHandleCount(<número_de_ficheros>) antes de abrir cualquier fichero. Si
    no se pone, por defecto Windows permite abrir 20. Si ponemos, por ejemplo
    30 y NO lo le indicamos a BLINKER que prepare el ejecutable, el máximo
    número de ficheros será el menor de los 2 (Windowso o Blinker)

packdata
packcode
noextdic
readonly

FILE    <fichero>
    Fichero PRG raiz.
OUTPUT <fichero>
    Nombre del fichero ejecutable.
DEFBEGIN
    name        <fichero>
        indica el nombre del programa que quieras se graba en el interior del
        ejecutable. Normalmente será el mismo del fichero EXE.
    description '<Lo que quieras>'
        Puedes poner la frase que quieras, pero ha de estar entre comillas. Esta
        frase se graba en el interior del ejecutable.
    exetype      Windows 3.1
    code         preload moveable discardable
    data         preload moveable
    stacksize    9500
    heapsize     2048
    segment      'PLANKTON_TEXT' nondiscardable
    segment      'EXTEND_TEXT'   nondiscardable
    segment      'OM_TEXT'       nondiscardable
    segment      'OSMEM_TEXT'    nondiscardable
    segment      'SORTOF_TEXT'   nondiscardable
    segment      'STACK_TEXT'    nondiscardable
DEFEND

#CLIWIN\OBJ
    file romewin

#CLIWIN\LIB
    lib romewin

SEARCH  Five, FiveC, Objects

LIB WinApi

```

Programa inicial

Vamos a hacer el programa que nos servirá como patrón para ir añadiendo los distintos elementos que se utilizan en Fivewin. No es el clásico “Hola mundo” sino más bien un esqueleto vacío donde iremos colgando las practicas. Le llamaremos **T01_00.*** y de él nacerán los programa hasta llegar al **T01_XX**. Por razones prácticas, en estos primeros ejemplos dejaremos a un lado los acentos o tildes de las letras. Esto está motivado a que una letra acentuada realizada con un editor DOS, no corresponde al mismo código que su equivalente en windows. Para convertir unos caracteres a otros, hay en Fivewin unas funciones especiales que por ahora no vamos a ver. Ya hablaremos de ellas mas adelante.

Escribe el siguiente fichero **T01_00.PRG**

```
* TUTOR 01

#include 'romewin.inc'
    Recuerda que romewin, tiene un #include "Fivewin.ch"

static oWnd

*****

function tutor

DEFINE WINDOW oWnd TITLE 'TUTOR 01_00'

ACTIVATE WINDOW oWnd MAXIMIZED

return NIL
```

Al compilar el programa tendrás una pantalla similar a esta:



Seguramente NO es tu primer programa en Windows pero disimulemos como si lo fuese. Ya ves que con dos simples líneas nuevas para ti, aparece una ventana que puedes minimizar y maximizar. Puedes moverla, etc. es decir, es una ventana windows vacía pero con todas las características propias de este tipo de ventanas.

Antes de seguir analicemos este pequeño programa

*TUTOR

Es una pequeña descripción del programa. Como es lógico tu podrás poner en tus programas reales todo lo que quieras.

```
#include 'romewin.inc'
```

Este include, si te fijas unas páginas atrás, tiene como includes encadenados a inkey.ch, set.ch y fivewin.ch. Yo le pongo extensión .INC para distinguir los míos de los normales, que suelen tener .CH. El include tiene también otras cosas pero ahora no es el momento de analizarlas.

```
static oWnd
```

Se declara como static la variable que contendrá el objeto ventana. Ahora no tiene mucho sentido que sea static, pero luego si.

```
*****
```

```
function tutor
```

```
DEFINE WINDOW oWnd TITLE 'TUTOR 01_00'
```

Esta es una nueva instrucción para ti. Indica un comando que el preprocesador convertirá en una llamada a un objeto Fivewin. Este objeto es el encargado de crear la ventana. Dicha ventana quedará creada, pero OJO, no queda activada. Si lees el ejemplo sobre matrices en el apartado de introducción a la Programación Orientada a Objetos, es como si hubiésemos creado una matriz con los datos de la ventana, pero todavía no lo hemos visualizado. Es importante que aprendas bien la idea que cuando se crea un objeto NO se activa. Necesita de una orden concreta de activación. De momento el objeto creado se ha puesto en la variable oWnd.

```
ACTIVATE WINDOW oWnd MAXIMIZED
```

He aquí el comando que activa el objeto oWnd. En este caso no solamente activa la ventana sino que también le indica que sea de forma MAXIMIZADA. Entre la definición y la activación podemos modificar el objeto con el fin de que en el momento de su activación empiece con los valores cambiados.

```
return NIL
```

Para no dejar ideas sueltas, antes de seguir vamos a ver como nuestro comando DEFINE y ACTIVATE se han convertido en ordenes Clipper dirigidas a objetos Fivewin. Si analizamos el fichero **T01_00.PPO** veremos las instrucciones convertidas por el preprocesador. Analizaremos solo algunas líneas. Estoy seguro de que tu inteligencia hará el resto.

```
function tutor
```

```
DEFINE WINDOW oWnd TITLE 'TUTOR 01_00' se ha convertido en:
```

```
oWnd := TWindow():New(,,, "TUTOR 01_00",,,,,,,,,, !.F., !.F., !.F.,  
!.F., .F. )
```

Aquí se llama al constructor(que crea) `Twindow():New()`, se le pasan los parámetros iniciales por defecto y él nos devuelve el objeto que depositamos en la variable `oWnd`.

ACTIVATE WINDOW `oWnd` MAXIMIZED se ha convertido en:

```
oWnd:Activate(    Upper("MAXIMIZED"),    oWnd:bLClicked,    oWnd:bRClicked,
                  oWnd:bMoved,            oWnd:bResized,    oWnd:bPainted,
                  oWnd:bKeyDown, oWnd:bInit,,,,,,,,, oWnd:bLButtonUp )
```

Aquí se invoca al método `Activate()` del objeto `oWnd`. También se pasan como parámetros para que sea de la forma MAXIMIZADA. El resto de los parámetros son las propias variables de instancia que no han sido modificadas

Espero que esto te haya aclarado un poco como funciona Fivewin. Profundizar mas en este pequeño programa sería entrar en los objetos y esto por ahora lo vamos a dejar. De ahora en adelante, en este capítulo, a menos que sea necesario no vamos a bajar a este nivel. Si tienes interés en saber como se convierte cada instrucción puedes leer la parte correspondiente en el capítulo de detalles, en el manual o en el propio fichero .PPO generado por el compilador. Una buena práctica que puedes hacer es tener cerca el fichero de cabecera **fivewin.ch** y consultar en él los detalles de la conversión. Dentro del capítulo sobre comandos, también ponemos en cada uno, la parte correspondiente al fichero **fivewin.ch**.

Vamos ahora a maquillar un poco el programa con dos o tres instrucciones y parámetros más con el fin de hacer mas bonito nuestro programa. (Los cambios con respecto al programa anterior se presentan en negrita y normalmente estarán comentados)

Modifica T01_00.PRG como sigue:

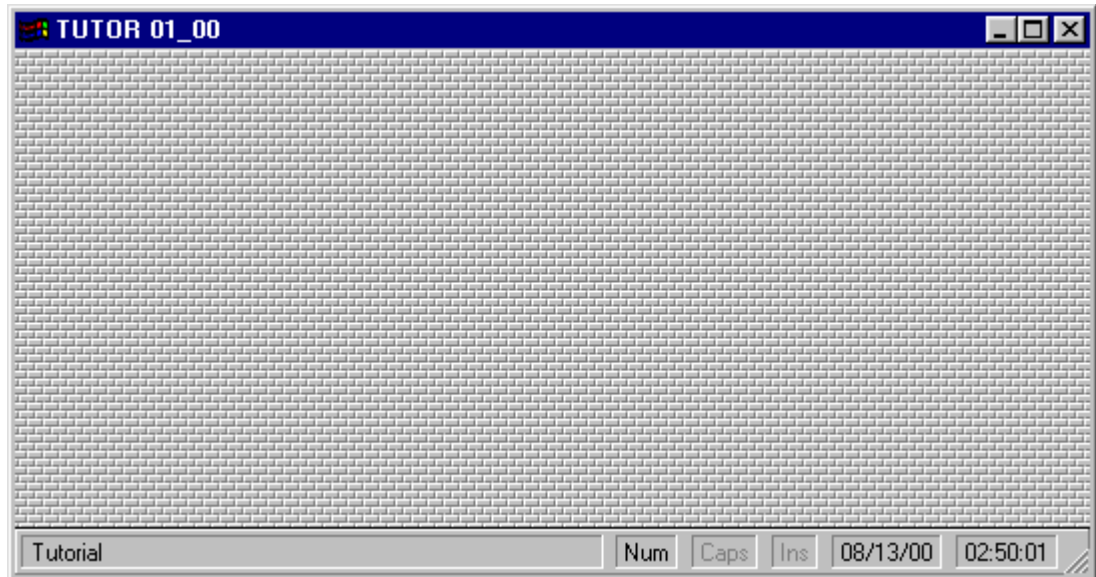
```
* TUTOR
#include 'romewin.inc'
static oWnd
*****

function tutor

local oBru
    Declaramos una variable que contendrá el objeto Brocha.
DEFINE BRUSH oBru STYLE BRICKS
    Definimos el objeto brocha y le indicamos que ha de ser stilo BRICKS.
DEFINE WINDOW oWnd TITLE 'TUTOR 01_00' BRUSH oBru
    Definimos la ventana indicando el titulo y la brocha.
SET MESSAGE OF oWnd TO 'Tutorial' CLOCK DATE KEYBOARD
    Indicamos que se cree una barra de mensajes con un mensaje predefinido, el
    reloj, la fecha y el estado del teclado.
ACTIVATE WINDOW oWnd MAXIMIZED ON INIT dialogo()
    Activamos la ventana e indicamos que cuando ocurra se llame
    automáticamente ala función dialogo() y que esta continúe con el proceso.
return NIL

/*****
DIALOGO Visualiza una caja de dialogo
*/
function dialogo
    De momento esta función no hace nada.
return NIL
```

Ahora la ventana que se nos presenta es la siguiente. Mas bonita ¿verdad?



Antes de seguir con otros elementos vamos a ver un poco en detalle el objeto ventana o mas propiamente dicho la clase **TWindows**.

Las partes de una ventana son:

- Barra del título. En nuestro caso TUTOR 01_00
- Botones de Minimizar, Maximizar y Salida
- Barra de mensajes, donde se presentan de forma dinámica mensajes de ayuda y otros textos que el programador haya decidido. También se puede presentar opcionalmente el estado del teclado, la fecha y la hora.

Una ventana por si sola no sirve para nada. Su función es contener otros elementos como botones, textos, respuestas, browses, etc. Todo esto lo iremos viendo mas adelante.

Para analizar el comando completo sobre la creación y activación de ventanas, pasa al capítulo que indica los detalles profundamente. Ahora solo quiero que sepas lo que es una ventana.

Veamos el resto de los elementos.

Barra de menú

Un menú en informática todos sabemos lo que es. Ahora lo vamos a ver desde una ventana Windows. Muchos de nosotros hemos hecho menús en DOS de una forma u otra. Windows nos da la oportunidad de standarizar el formato. Desde nuestro programa inicial

ahora debemos cambiar algunas líneas y añadir una nueva función. Esta opción se denomina **MENU Pop-up**. Consiste en una barra horizontal que se despliegan de forma emergente unos menús verticales. Cada una de las opciones pueden tener una acción en el programa o bien otro submenú con nuevas opciones, que a su vez pueden hacer lo mismo. Además de las opciones, el sistema de menús de windows nos da una serie de opciones para aumentar la estética. Estas opciones son líneas de separación, opciones visualizadas pero no activadas, posibilidad de modificación de las opciones en tiempo de ejecución, etc.

Ahora vamos a ver un menú simple. Si quieres saber más sobre menús mira el apartado correspondiente en la segunda parte del libro.

```
* TUTOR
#include 'romewin.inc'
static oWnd
*****

function tutor
local oBru
DEFINE BRUSH oBru STYLE BRICKS
DEFINE WINDOW oWnd TITLE 'TUTOR 01_01' BRUSH oBru MENU menu()
    Aquí hemos añadido la cláusula MENU. Esto activa una función al visualizar
    la ventana que devuelve el objeto menú para ser insertado en la ventana.
    Esta función que hemos llamado menu() (podría tener otro nombre
    cualquiera), tendrá los textos, acciones y mensajes de la barra de menús.
    Ver mas abajo la función menu().
SET MESSAGE OF oWnd TO 'Tutorial' CLOCK DATE KEYBOARD

ACTIVATE WINDOW oWnd MAXIMIZED ON INIT dialog()

return NIL

/*****
MENU
*/
static function Menu

local oMenu
    Declaramos la variable local que tendrá el objeto menu.

MENU oMenu
    La primera línea de un menú lo único que hace es declarar el objeto.
    MENUITEM '&Archivos' MESSAGE 'Utilidades sobre archivos'
    El primer nivel de opciones es el primer nivel de instrucciones MENUITEM.
    Aquí definimos el texto a visualizar, con indicación del acelerador por
    medio del signo & y un mensaje que se visualizará en la barra de mensajes.
    MENU
    Si una opción tiene un submenú, se pone la palabra MENU y a continuación
    las opciones del nuevo submenú.
        MENUITEM '&Empresa' ACTION msginfo('Empresa') MESSAGE 'Formulario
        de empresa, parametros, etc.'
    Lo novedoso en esta línea es la cláusula ACTION seguida de una función.
    Con ello indicamos que ha de hacer el programa si el usuario selecciona
    esta opción. En este ejemplo saldrá un mensaje en el centro de la pantalla
    con la palabra "Empresa".
        MENUITEM '&Ficheros' ACTION msginfo('Ficheros') MESSAGE 'Control
        interno de ficheros, indices, etc.'
    SEPARATOR
    Para visualizar una línea horizontal que separe las distintas opciones del
    submenú, solo hay que indicar la palabra SEPARATOR.
        MENUITEM '&Salida' ;
```

```

        ACTION If( MsgYesNo( oemtoansi(" Desea terminar ?")
                                ,oemtoansi('Elija opción SI/NO') ), oWnd:End(),) ;
La acción en esta opción es una ventana que pregunta si se desea terminar
o no. Si se elige SI, el programa termina.
        MESSAGE 'Salida del programa'
ENDMENU
    Termina el primer submenú.
MENUITEM '&Clientes'      MESSAGE 'Datos sobre clientes'
MENU
    MENUITEM '&Formulario' MESSAGE 'Formulario de clientes'
    MENUITEM '&Listado'   MESSAGE 'Listado de clientes en formato de anillas'
ENDMENU

MENUITEM '&Utilidades'  MESSAGE 'Diversas utilidades windows'
MENU
    MENUITEM '&Calculadora' ACTION WinExec('Calc')      MESSAGE 'Calculadora
                                de windows'
    Este es un ejemplo de como podemos llamar a un programa de windows desde
    nuestra aplicación. En este caso la calculadora.
    MENUITEM '&Write' ACTION WinExec('Write')      MESSAGE 'Editor Windows'
ENDMENU

MENUITEM '&Ayuda'      MESSAGE 'Funciones de ayuda'
MENU
    MENUITEM '&Ayuda' ACTION msginfo('Ayuda')MESSAGE 'Ayuda'
    MENUITEM '&Acerca de' ACTION msginfo('Acerca de') MESSAGE 'Ventana con
                                informacion de la aplicación'
ENDMENU

ENDMENU
    Fin del menú principal

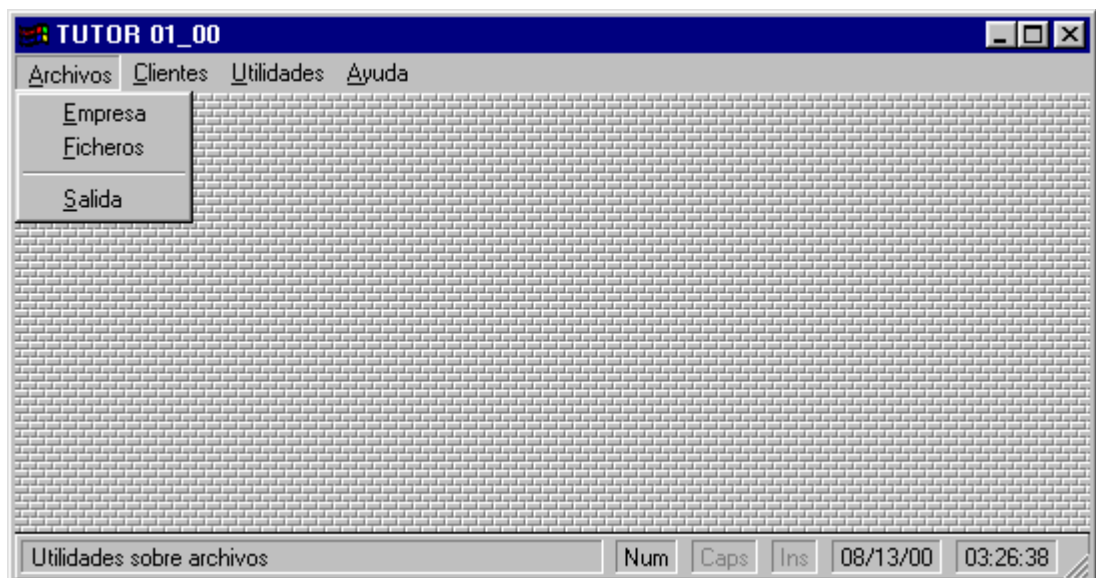
return oMenu

/*****
DIALOGO Visualiza una caja de dialogo
*/
function dialogo

return NIL

```

Si ahora realizas el programa verás algo similar a esto:



Naturalmente, si tu quieres practicar, puedes modificar lo que quieras. Incluso si quieres tener un menú rápido hacerlo con menos opciones para poder verlo rápidamente.

Barra de botones

El siguiente elemento que vamos a ver es la denominada Barra de botones. Consiste esta barra en la colección de botones que normalmente se pone debajo del menú que hemos visto en el apartado anterior. Como no vamos ahora a perder el tiempo en hacer iconos para los botones, utilizaremos algunos de los que se encuentran en los directorios que se crean al instalar Fivewin.

Con el fin de ir viendo la ventana que estamos creando lo mas parecida posible a una ventana real, deja la función menu() como esta y pon el programa como sigue:

```
* TUTOR
#include 'romewin.inc'
static oWnd

*****

function tutor
local oBru,oBar
    Crear la variable local que contendrá el objeto de la barra de botones.

DEFINE BRUSH oBru STYLE BRICKS
DEFINE WINDOW oWnd TITLE 'TUTOR 01_02' BRUSH oBru MENU menu()
SET MESSAGE OF oWnd TO 'Tutorial' CLOCK DATE KEYBOARD

DEFINE BUTTONBAR oBar SIZE 28,32 3D TOP OF oWnd
    Este comando crea el objeto. Con SIZE indicamos el tamaño. Con 3D
    indicamos que tenga aspecto de tres dimensiones. Con TOP indicamos que ha
    de ponerse en la parte superior de la ventana, debajo del menú.

DEFINE BUTTON of oBar ;
    FILENAME '\fw20\bitmaps\top.bmp' ;
    ACTION msginfo('Primer registro') ;
    NOBORDER ;
    TOOLTIP 'Primer registro' ;
    MESSAGE 'Primer registro'
    Aquí definimos el primer botón de la barra oBar. Con FILENAME indicamos el
    fichero BMP que contiene el dibujo (como verás corresponde a un directorio
    de Fivewin). Con ACTION indicamos lo que ha de hacerse si se pulsa el
    botón. En nuestro caso, sacar un mensaje, pero lo más lógico será para
    este icono hacer un skip en una base de datos. Con NOBORDER indicamos que
    no saque borde en el botón. Compara con los dos últimos botones de esta
    misma barra, que si tienen borde. Con TOOLTIP indicamos el pequeño texto
    que aparece cuando el cursor se posiciona durante unos segundos sobre el
    botón. Con MESSAGE se indica el texto que se visualizará en la barra de
    mensajes.
DEFINE BUTTON of oBar ;
    FILENAME '\fw20\bitmaps\prev.bmp' ;
    ACTION msginfo('Registro anterior') ;
    NOBORDER ;
    TOOLTIP 'Registro anterior' ;
    MESSAGE 'Registro anterior'
DEFINE BUTTON of oBar ;
```

```

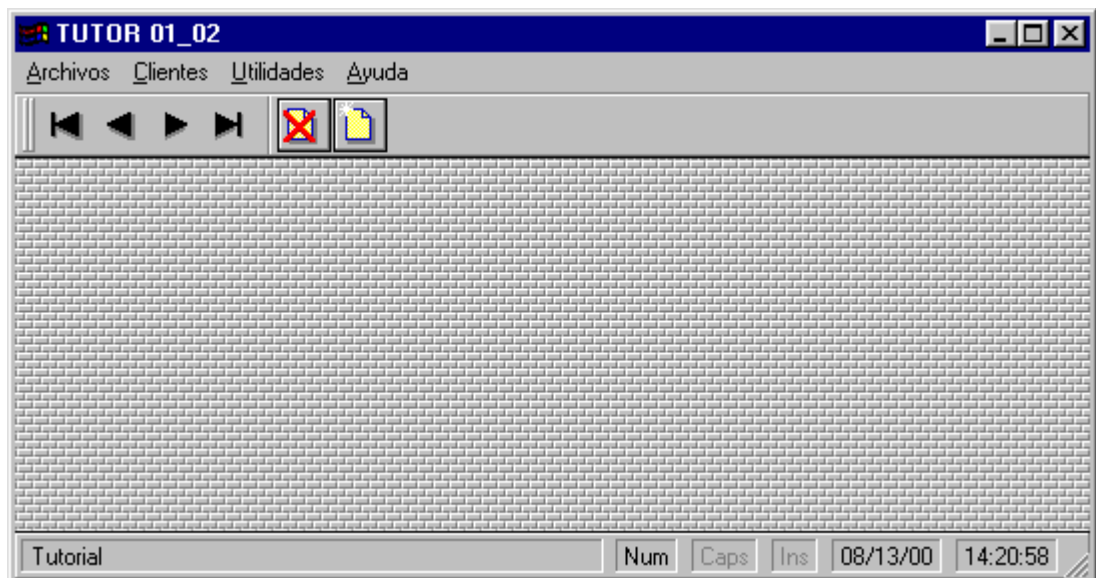
FILENAME '\fw20\bitmaps\next.bmp' ;
ACTION msginfo('Registro siguiente') ;
NOBORDER ;
TOOLTIP 'Registro siguiente' ;
MESSAGE 'Registro siguiente'
DEFINE BUTTON of oBar ;
FILENAME '\fw20\bitmaps\bottom.bmp' ;
ACTION msginfo('Ultimo registro') ;
NOBORDER ;
TOOLTIP 'Ultimo registro' ;
MESSAGE 'Ultimo registro'
DEFINE BUTTON of oBar ;
GROUP ;
    Fíjate en GROUP. Con esto se hace una pequeña barra vertical a la
    izquierda de este botón. Sirve para formar grupos de botones dentro de la
    misma barra.
FILENAME '\fw20\bitmaps\del.bmp' ;
ACTION msginfo('Borrar') ;
TOOLTIP 'Borrar' ;
MESSAGE 'Borrar'
    Al NO indicar NOBORDER, los botones TIENEN bordes.
DEFINE BUTTON of oBar ;
FILENAME '\fw20\bitmaps\new.bmp' ;
ACTION msginfo('Nuevo') ;
TOOLTIP 'Nuevo' ;
MESSAGE 'Nuevo'

ACTIVATE WINDOW oWnd MAXIMIZED ON INIT dialogo()

return NIL
    El resto sigue igual que en el programa anterior.

```

Al compilar y ejecutar el programa tiene el siguiente aspecto:



Si pulsas con el derecho del ratón en la barra, te saldrá un pequeño menú para que elijas si quieres ponerla a la derecha, a la izquierda o flotante.

Bien, ya hemos visto lo que se puede hacer con la ventana principal. Ahora veremos como se visualizan y rellenan con controles las llamadas cajas de dialogo.

Cajas de diálogo

Una caja de diálogo es como las ventanas que hacemos en DOS para incluir en ella los SAY's y GET's que utilizamos para las fichas de clientes, etc. Hasta ahora en nuestro programa hemos dejado a propósito una función que vamos a utilizar para crear la caja de diálogo. Deja todo el programa como está y modifica solamente la función **dialogo()**. Debes dejarla como sigue:

```

/*****
DIALOGO Visualiza una caja de dialogo
*/
static function dialogo

local oDlg
    Declara local la variable que contendrá el objeto.
DEFINE DIALOG oDlg ;
    OF oWnd ;
    TITLE 'CAJA DE DIALOGO' ;
    FROM 0,0 TO 20,80
    Con esto se define la caja de dialogo. Observa que con FROM a,b TO x,y
    definimos el tamaño de la caja. Lo importante es dejar FROM a 0,0 y poner
    en TO el tamaño, puesto que en el comando de la línea siguiente se declara
    CENTER. El sistema de medidas puede variar de un ordenador a otro, por el
    tipo de definición de pantalla. Si en los ejemplos siguientes ves que te
    falta caja para poner todos los elementos, modifica estos datos. De todas
    formas, en la práctica se utiliza el sistema de recursos, de manera que
    las coordenadas ahora no son importantes.
    ACTIVATE DIALOG oDlg CENTER
    Aquí se activa el diálogo de una forma centrada en la pantalla.
return NIL

```

Debe aparecerte una ventana similar a esta:



Como ves no están los botones de minimizar y maximizar. Esta ventana solo sirve como contenedor de otros elementos,

Texto fijo (SAY)

Este tipo de control es muy similar al SAY de toda la vida. Las diferencias son propias de Windows. Principalmente la posibilidad de cambiar de fuente y de color. La función quedará así:

```

/*****
DIALOGO Visualiza una caja de dialogo
*/
static function dialogo

local oDlg
local oSay1,oSay2
    Se declaran locales las variables que contendrán los objetos

DEFINE DIALOG oDlg ;
    OF oWnd ;
    TITLE 'CAJA DE DIALOGO' ;
    FROM 0,0 TO 20,80

@ 1,1 SAY oSay1 VAR 'Codigo' OF oDlg
    Posicionamos en la línea 1, columna 1 el primer texto: 'Codigo'
@ 2,1 SAY oSay2 VAR 'Nombre' OF oDlg
    Posicionamos en la línea 2, columna 1 el segundo texto: 'Nombre'

ACTIVATE DIALOG oDlg CENTER

return NIL

```

Como verás los valores de línea y columna se hace por el sistema clásico igual que en DOS. Normalmente para tener una mayor precisión se hace por medio de píxel, pero esta medida depende del font utilizado. Lo mejor es usar el Workshop que veremos en el capítulo siguiente, de forma que ahora dejamos el tema del posicionamiento aparcado.

El objeto no es necesario declararlo, pero si luego se quiere hacer algo con él habrá que declararlo.

Texto editable (GET)

Muy parecido a nuestro GET. Hagamos una pequeña modificación a la función anterior:

```

/*****
DIALOGO Visualiza una caja de dialogo
*/
static function dialogo

local oDlg
local oSay1,oSay2
local oGet1,oGet2

```

```
Se declaran locales las variables que contendrán los objetos
local nGet1 := 101
Se asigna un valor previo a la variable que contendrá el texto a editar
local cGet2 := pad('Andres Romero Garcia',40)
Idem a la variable anterior. En este caso, por ser de tipo carácter, ha de
ajustarse el tamaño, por ejemplo a 40.

DEFINE DIALOG oDlg ;
  OF oWnd ;
  TITLE 'CAJA DE DIALOGO' ;
  FROM 0,0 TO 20,80

@ 1,1 SAY oSay1 VAR 'Codigo' OF oDlg
@ 2,1 SAY oSay2 VAR 'Nombre' OF oDlg

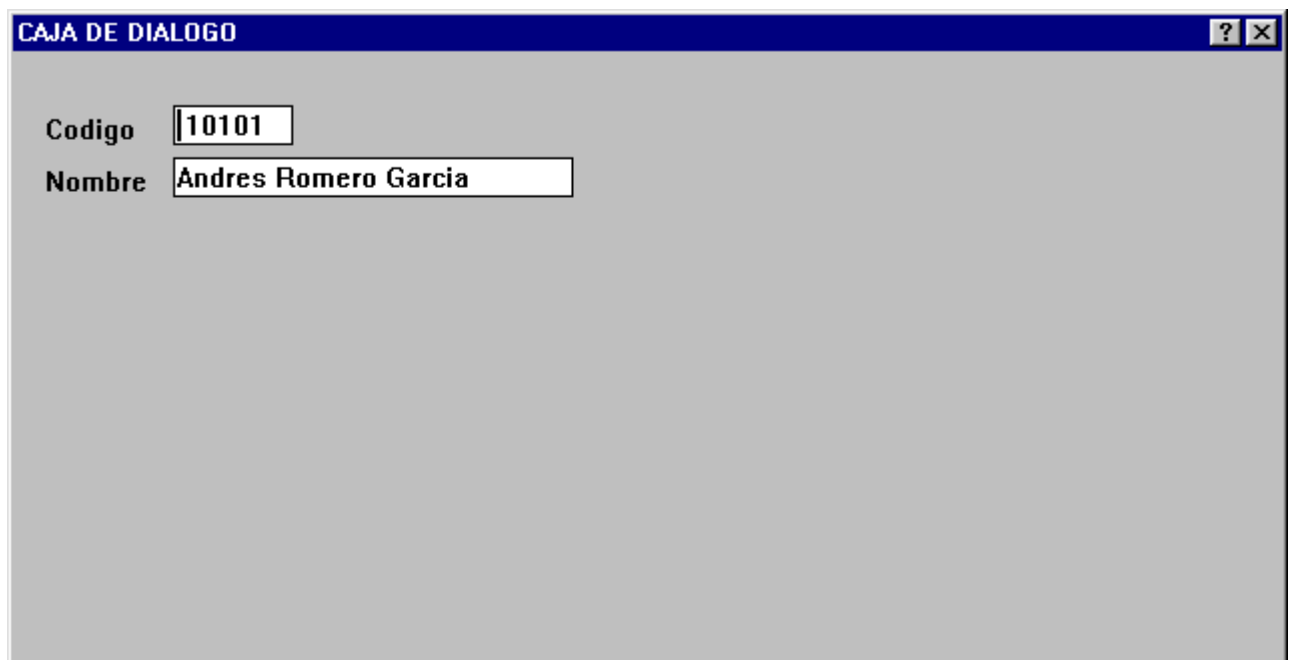
@ 1,5 GET oGet1 VAR nGet1 ;
  OF oDlg ;
  SIZE 30,10 ;
  PICTURE MK6
  Posicionamos en la línea 1, columna 5 el primer GET. Observa la cláusula
SIZE. En DOS con indicar PICTURE hubiese sido suficiente. Aquí es
necesario indicar el tamaño del recuadro donde saldrá el GET. Una
observación más, los valores son en píxel, mientras que el posicionamiento
es en líneas/columnas. Esto trae un poco de lío, pero se resuelve con el
workshop que veremos en otro capítulo, así que no te preocupes. Puedes
practicar un poco cambiando valores y te harás una idea.

@ 2,5 GET oGet2 VAR cGet2 ;
  OF oDlg ;
  SIZE 100,10
  Posicionamos en la línea 2, columna 5 el segundo GET.

ACTIVATE DIALOG oDlg CENTER

return NIL
```

Ahora el diálogo saldrá similar a esto:



Como curiosidad puedes tratar de editar el nombre y poner algo que ocupe 40 caracteres. Como no cabe, hará automáticamente un scroll horizontal similar al que sale con **"PICTURE @Sxx"** en el GET clásico de DOS.

Push Button

Vamos a poner dos botones de los que al pulsar se desencadena una acción. Estos botones pueden ser con texto o con un dibujo extraído de un BMP. Su funcionamiento es similar a los botones que pusimos en la barra de botones, debajo del menú. La diferencia es que estos se ponen en una cuadro de diálogo. Uno visualizará los datos que hayamos editado en los GET's y otro cerrará la ventana. Modifica la función y añade lo indicado en negrita.

```

/*****
DIALOGO Visualiza una caja de dialogo
*/
static function dialogo

local oDlg
local oSay1,oSay2
local oGet1,oGet2
local nGet1 := 101
local cGet2 := pad('Andres Romero Garcia',40)
local oBtn1,oBtn2
    Se asignan las variables locales para contener los objetos

DEFINE DIALOG oDlg ;
    OF oWnd ;
    TITLE 'CAJA DE DIALOGO' ;
    FROM 0,0 TO 20,80

@ 1,1 SAY oSay1 VAR 'Codigo' OF oDlg
@ 2,1 SAY oSay2 VAR 'Nombre' OF oDlg

@ 1,5 GET oGet1 VAR nGet1 ;
    OF oDlg ;
    SIZE 30,10 ;
    PICTURE MK6

@ 2,5 GET oGet2 VAR cGet2 ;
    OF oDlg ;
    SIZE 100,10

@ 1,30 BUTTON oBtn1 PROMPT 'Que' OF oDlg ;
    SIZE 32,16 ;
    ACTION msginfo(str(nGet1,6)+' '+cGet2)
    Aquí se define un botón con texto mediante la cláusula PROMPT. También se
    indica la posición en línea/columna y el tamaño en píxel. Como acción,
    saca un mensaje con el resultado de los GET's.

@ 13,260 BTNBMP oBtn2 OF oDlg ;
    FILENAME '\fw20\bitmaps\exit.bmp' ;
    SIZE 16,16 ;
    ACTION oDlg:end()
    Esta construcción es similar, pero en lugar de indicar BUTTON se ha puesto
    BTNBMP. Esto indica un botón con dibujo. El BMP está en el fichero

```

indicado en **FILENAME**. En cuanto a la situación, en este comando TODO es en píxels.

```
ACTIVATE DIALOG oDlg CENTER
```

```
return NIL
```

Check Box

El control **Check Box** es un pequeño recuadro donde solo se admiten dos estados: Activado o No activado. Puede haber muchos Check Box en una caja de diálogo, pero cada uno de ellos solo admite dos estados. Los check box, a diferencia de los Radio Button que veremos después no se relacionan unos con otros, aunque por estética pongamos varios Check Box juntos. Vamos a la práctica que es como mejor se ven las cosas:

Creemos un Check box con la indicación 'Activado'.

Para declarar la variable que contendrá el objeto, en la función dialogo(), añade a continuación de las declaraciones locales actuales la línea:

```
local oChk
```

```
local lChk := NO
```

Recuerda que el preprocesador convierte el "NO" de romewin.inc en .F.

También añade inmediatamente antes de activar la caja de diálogo, el siguiente control:

```
@ 1,20 CHEKBOX oChk var lChk OF oDlg ;  
PROMPT 'Activado'
```

Inicialmente presentará un cuadro desactivado. Si lo activamos, cambiará el contenido de la variable lChk de Falsa a Cierta.

Radio Button

Un Radio Button es un sistema que permite activar solo uno de los posibles "botones" que se compone. Su nombre viene de las antiguas radios de coche, que tenían una serie de botones para sintonizar las emisoras. Al pulsar uno de estos botones, los demás salían y quedaba uno solo metido hacia adentro. Algo parecido ocurre en los radiocaset domésticos. Si esta el botón de Play, al pulsar Rewin, Play se desactiva y se activa Rewin, etc. Veamos el ejemplo poniendo tres radiobutton que seleccionarán los colores básicos Rojo, Verde y Azul

Como siempre, declara el objeto:

```
Local oRbx
```

Esta variable contendrá el objeto.

Local nRbx := 0

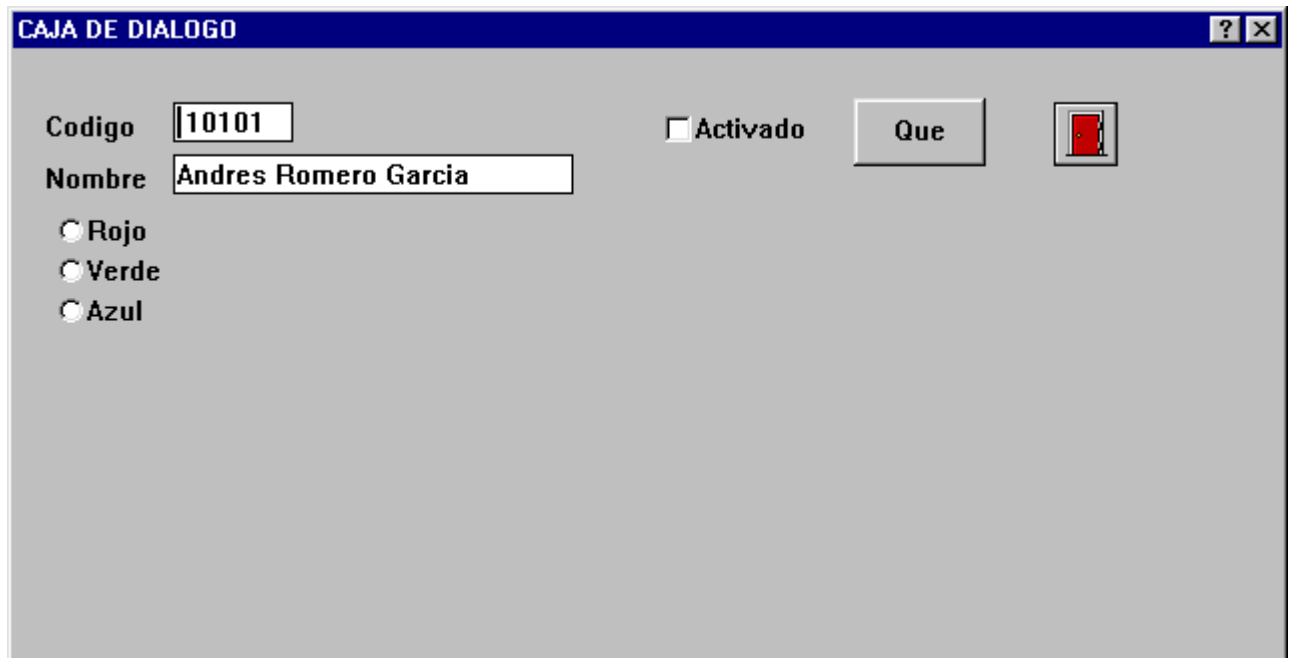
Esta variable contiene inicialmente el número de botón seleccionado, y contendrá el número de orden del botón que seleccione el usuario. Indicando 0, ningún botón estará seleccionado inicialmente.

Antes de la activación de la caja añada el siguiente control:

```
@ 4,10 RADIO oRbx VAR nRbx OF oDlg ;
  PROMPT 'Rojo','Verde','Azul' ;
  SIZE 30,10
```

Las coordenadas son en línea/columna. El tamaño es en pixels. La cláusula **PROMPT** indica los textos que tendrán los distintos botones. Cuanto mas textos, mas botones. El tamaño ajustará la máxima longitud de texto y la distancia entre botones.

Nuestro cuadro de diálogo debe presentar un aspecto similar a este:



List Box

Como verás vamos complicando cada vez más los controles. Ahora le toca el turno a List Box que traducido sería Caja de Lista o simplemente, una lista. Realmente este control sirve tanto para matrices bidimensionales, como para ficheros. Si tuviésemos que buscar algo parecido en clipper de DOS, sería la primera, sería la función `achoice()` y para la segunda las instrucciones propias del `browse`. Ahora vamos a ver solo el List Box de matrices. El programa presentará una lista de textos que pueden ser seleccionados y depositados su contenido en una variable. Veamos como. Añade a las variables locales estas tres líneas:

```
local oLbx
  Variable para el objeto
```

```

local cLbx := ''
Variable que contendrá el texto seleccionado de la lista
local mLbx := {'Africa','America','Asia','Europa','Oceania'}
Matriz que contiene los elementos de la lista

```

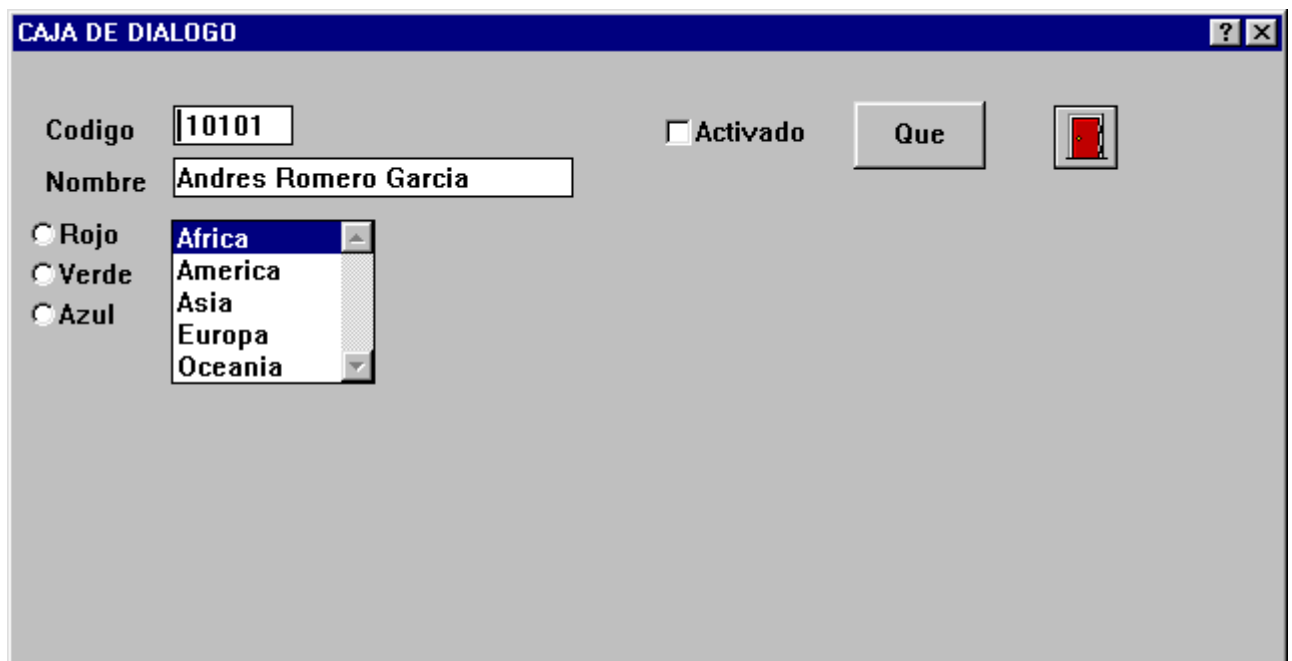
Ahora añade antes de activar el cuadro de diálogo lo siguiente:

```

@ 3,5 LISTBOX oLbx VAR cLbx OF oDlg ;
PROMPTS mLbx ;
SIZE 50,40
Aquí de nuevo la situación es línea/columna y el tamaño en pixels. La
cláusula PROMPTS (fíjate que es en plural) contiene la matriz con la lista
a visualizar.

```

El aspecto que presentará nuestra caja de diálogo será parecido a este:



Combo Box

Es una lista parecida a ListBox, pero aparece solo un texto con la posibilidad de ampliar la visualización y convertirlo en List Box. Una vez seleccionado vuelve a reducirse la lista y presentar una sola línea. Veamos la misma selección anterior pero en formato Combo box. Añade las siguientes variables locales:

```

local oCbx
Variable para el objeto
local cCbx := ''
Variable que contendrá el texto seleccionado de la lista
local mCbx := {'Africa','America','Asia','Europa','Oceania'}
Matriz que contiene los elementos de la lista

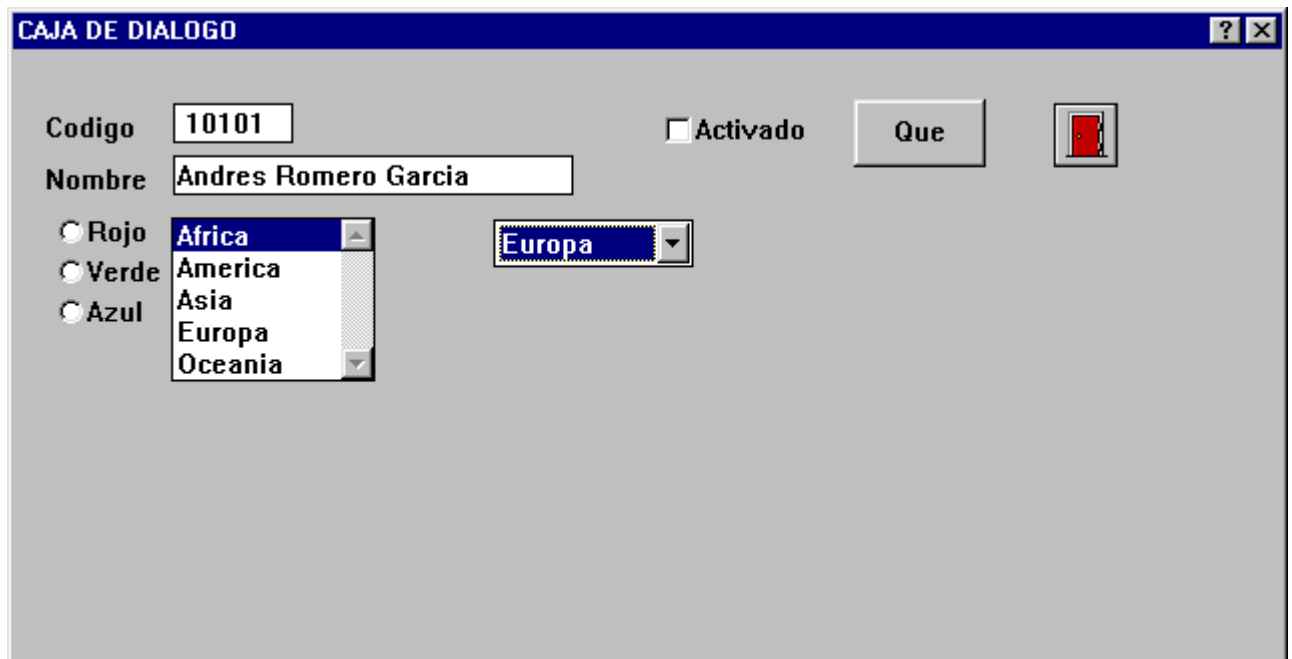
```

Ahora añade antes de activar el cuadro de diálogo lo siguiente:

```
@ 3,5 COMBOX oCbx VAR cCbx OF oDlg ;
  PROMPTS mCbx ;
  SIZE 50,40
```

Aquí de nuevo la situación es línea/columna y el tamaño en pixels. La cláusula **PROMPTS** (fíjate que es en plural) contiene la matriz con la lista a visualizar. El tamaño indica el rectángulo que se abrirá cuando el operador active la lista. Los elementos que no quepan se podrán ver mediante un scroll vertical.

Ahora el cuadro quedará así:



Scroll Bars

Windows a través de Fivewin nos da la posibilidad de hacer barras de scroll con muy poca programación. Estas barras pueden ser orientativas de la posición o devolver el control que ejerce el usuario sobre las barras. Es decir, puede servir como destino o como origen de información.

Añade estas variables locales

```
local oSbv,oSbh
```

Pon estas líneas antes de activar la caja de diálogo.

```
@ 1,36 SCROLLBAR oSbv VERTICAL OF oDlg ;
  SIZE 10,110
```

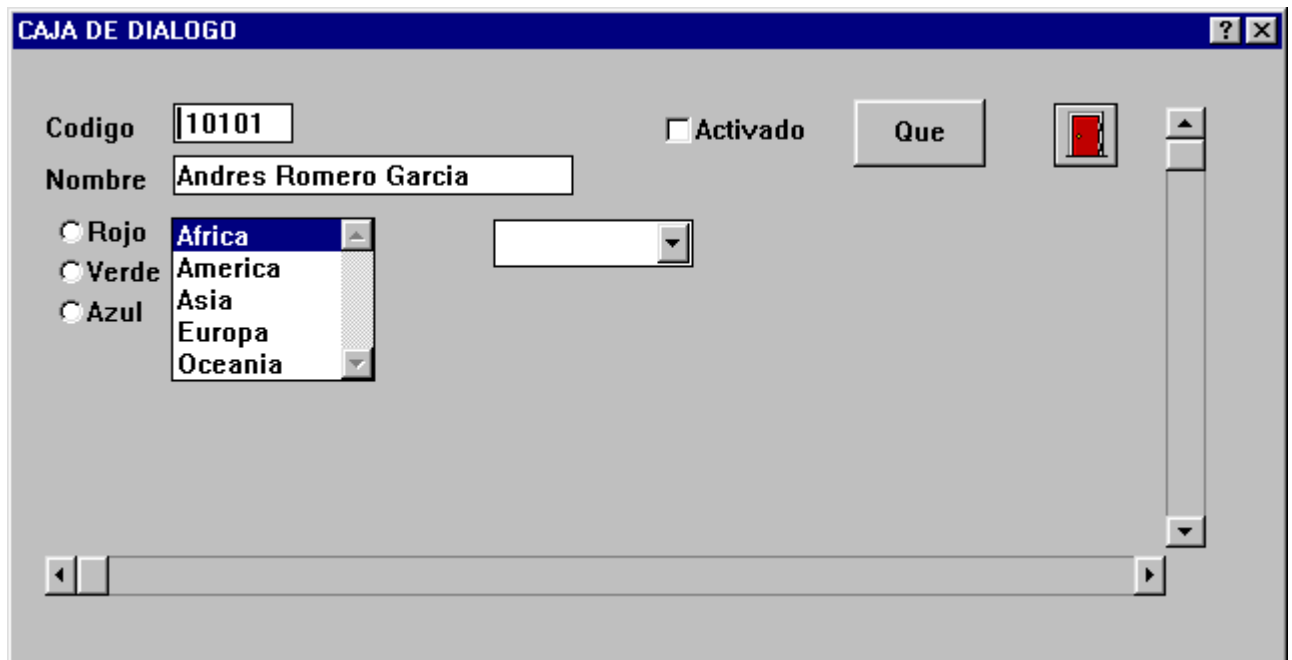
Crearé un objeto en oSbv correspondiente a la barra vertical de un tamaño en píxel de 10 de ancho por 110 de alto.

```
@ 9, 1 SCROLLBAR oSbh HORIZONTAL OF oDlg ;
```

SIZE 280,10

Crearé un objeto en oSbh correspondiente a la barra horizontal de un tamaño en píxel de 280 de ancho por 10 de alto.

La caja de diálogo quedará así:



Icon

Es parecido a los botones con BMP, pero la principal diferencia es que NO es un botón y que no es BMP. Los ficheros que contienen la información del dibujo tienen la extensión ICO, y como tales guardan las dimensiones comunes a todos los iconos de Windows. Añade la siguiente variable local:

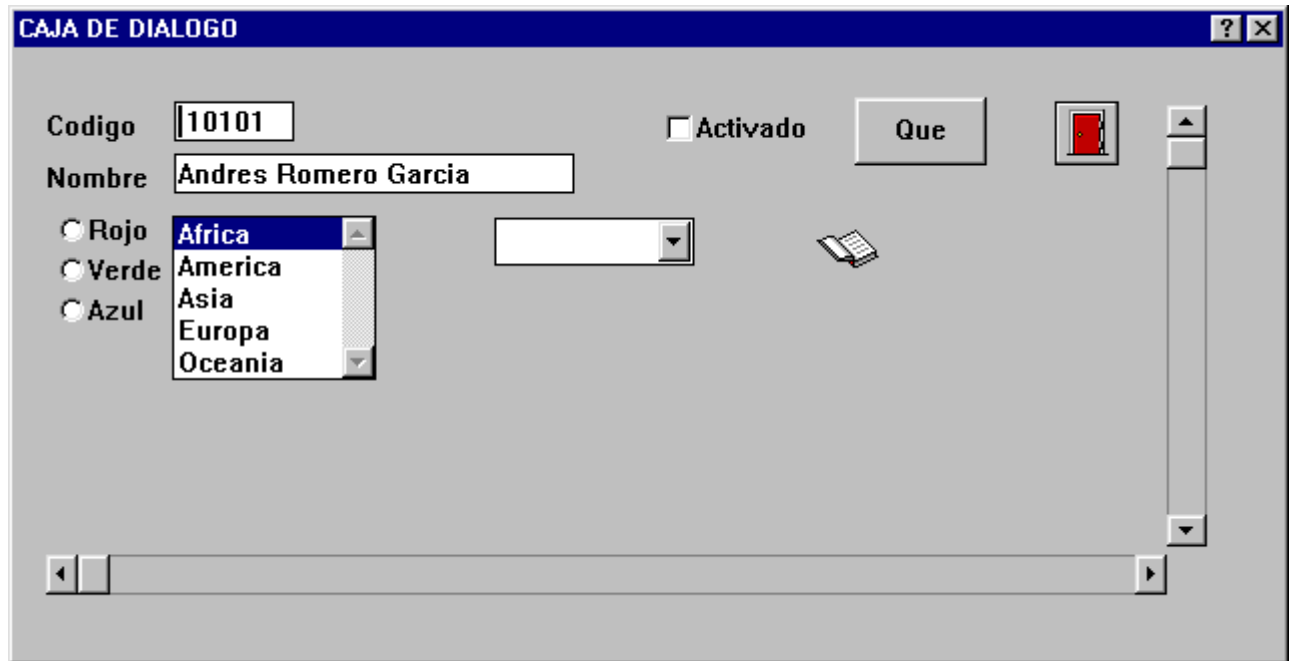
local oIco

Y pon las siguientes líneas antes de activar la caja de diálogo:

```
@ 3,25 ICON oIco OF oDlg ;
FILE '\fw20\icons\book.ico
```

Esto nos dibuja el icono **book.ico** en la línea 3, columna 25.

Te quedará una pantalla similar a esta:



Bitmaps

Si eres un poco artista en tus creaciones, con los bitmaps puede sacar mucho provecho a tus cualidades. Desde poner fondo a las cajas de diálogo, hasta hacer fichas de personas o artículos con sus fotos. Fíjate que sencillo. Como siempre declaramos la variable que contendrá el objeto Bitmaps:

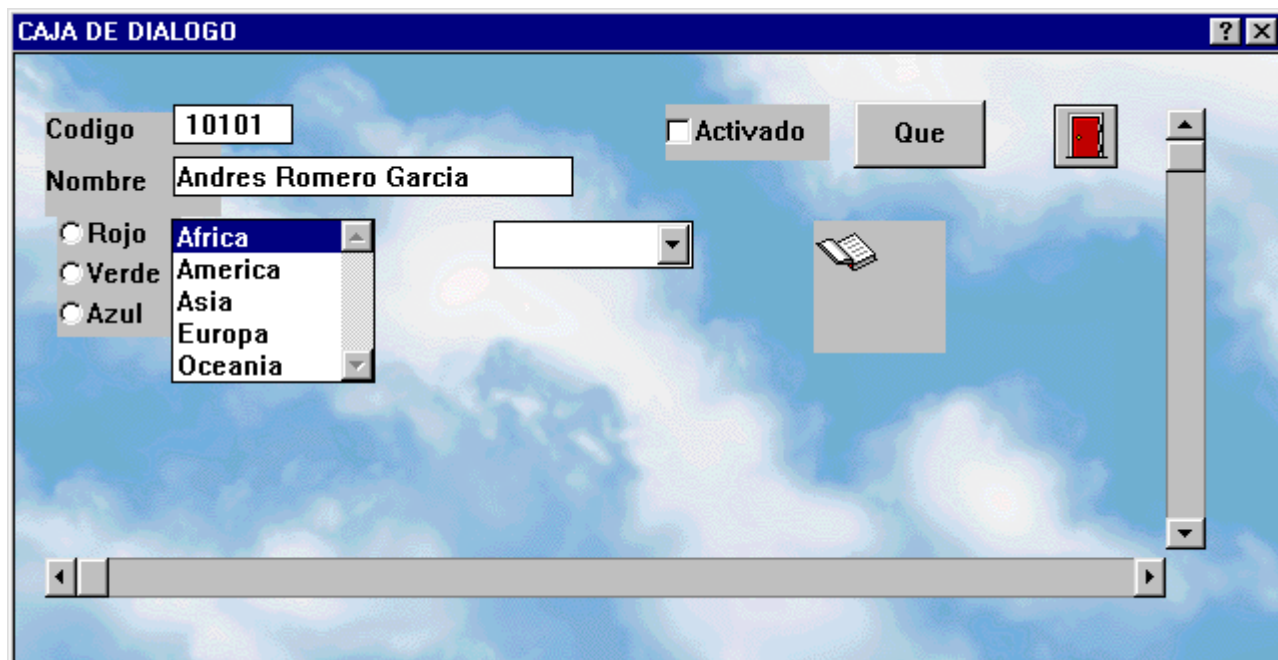
local oBmp

Luego añadimos, antes de activar la caja de diálogo, las siguientes líneas:

```
@ 0,0 BITMAP oBmp OF oDlg ;
FILE '\windows\nubes.bmp' ;
SIZE 400,300
```

He cogido para este ejemplo un BMP de windows. Por el tamaño no te preocupes. Si al presentar la imagen, se sale de la caja de diálogo, el exceso no se visualiza.

Ahora la caja se ha transformado visualmente de forma significativa.



Todos los controles quedan como estaban con dos particularidades:

- El último control, en este caso el Bitmap, no ha tapado al resto. Se pone como fondo.
- Los controles que tienen su propio fondo, tapan el BMP. Fijaté por ejemplo en el icono del libro. Para evitar esto hay técnicas y trucos que veremos más adelante.

Browse

Los browses que todos conocemos desde clipper-DOS, se hacen mediante la instrucción LISBOX pero en lugar de poner la cláusula PROMPT (o ITEMS) poniendo FIELD. Esto genera un objeto que no tiene nada que ver con los generados por el LISTBOX con PROMPT.

Existen magníficas clases hechas por terceros para simular los browses.

Fólder

Los fólder son esas carpetas que tienen varias solapas en la parte superior. Según la solapa que se active, la superficie de la carpeta presentará unos controles u otros. De esta forma es fácil poner en un mismo espacio muchos controles, con la única particularidad de que no se pueden visualizar todos al mismo tiempo.

En DOS también se pueden hacer folder's pero su control es tedioso. En Windows, esto es relativamente fácil. Lo primero que hay que tener en cuenta desde el punto de vista de la programación, es que un fólder es un espacio reservado en una caja de diálogo

para tener superpuestas en el mismo sitio varias cajas hijas del diálogo contenedor. Cada una de estas cajas hijas, tendrá sus propios controles, incluso más fólder. Cuando se programa directamente, sin recursos, no te has de preocupar de las cajas hijas, pero con recursos si. Por ahora veremos la programación directa. Escribe en la función diálogo las siguientes líneas

```

/*****
DIALOGO Visualiza una caja de dialogo
*/
static function dialogo

local oDlg
local oFol,oBut1,oBut2

DEFINE DIALOG oDlg OF oWnd TITLE 'Tutorial' FROM 0,0 TO 20,80
    Definimos una caja de diálogo. Hasta aquí normal.
@ 1,1 FOLDER oFol OF oDlg ;
    PROMPT 'Uno','Dos' ;
    SIZE 300,100
    Definimos el commando FOLDER. La cáusula PROMPT indica una secuencias de
    textos que serán puestos como etiquetas en las distintas solapas. Habrá
    una solapa por cada texto. El resto es el desarrollo normal en los
    controles por programa.

@ 1,1 BUTTON oBut1 PROMPT 'Boton 1' OF oFol:aDialogs[1]
    Se define un botón. Todo es normal excepto la cláusula OF. Fijate, el
    contenedor no es la caja de diálogo oDlg. Ahora el contenedor es el primer
    elemento de una matriz que se encuentra en la variable de instancia
    aDialogs del objeto fólder, es decir, oFol:aDialogs[n], donde n
    representará el orden dentro de las pestañas.

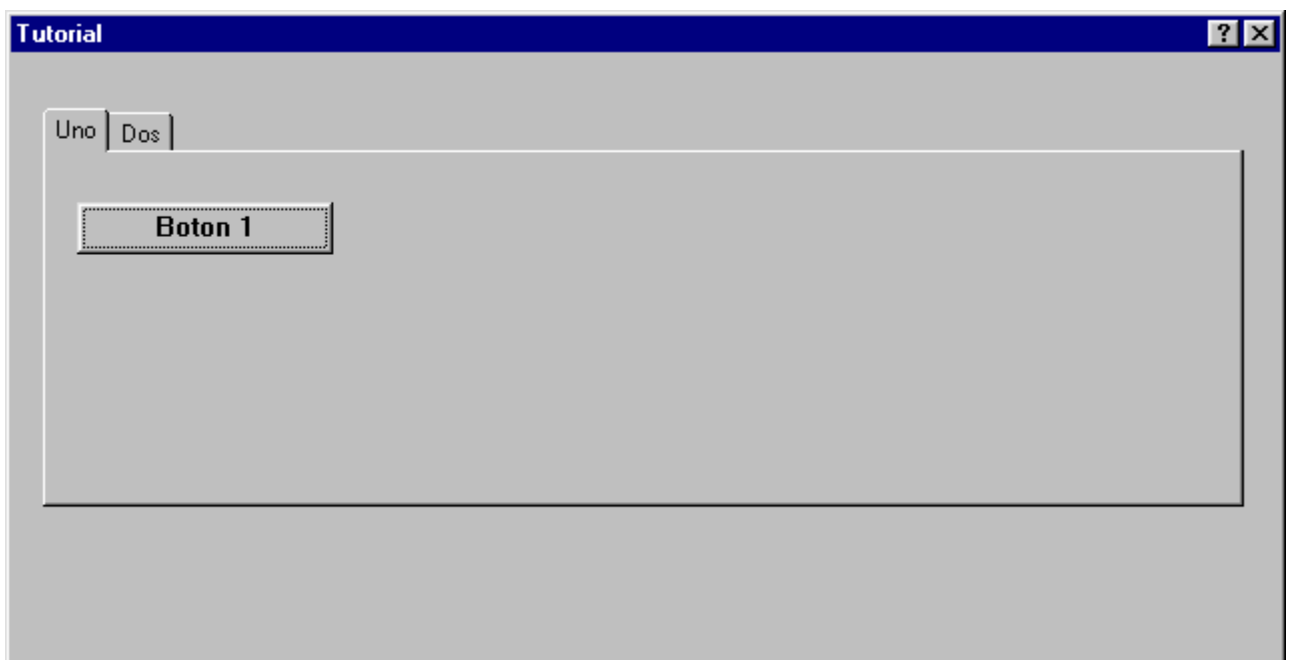
@ 1,1 BUTTON oBut2 PROMPT 'Boton 2' OF oFol:aDialogs[2]
    Aquí es igual, pero se ha variado el puntero de la matriz. Ahora es [2]

ACTIVATE DIALOG oDlg CENTER

return NIL

```

La imagen será parecida a esta:



Colores

Aunque no es un control propiamente dicho, su importancia en un entorno gráfico es tal que merece la pena hacer un apartado específico del color.

En DOS todos sabemos (mas o menos) como funciona el color. Una letra indica uno de los colores básicos. Si hay dos letras separados por una barra (N/W) indicará el color del texto y el color del fondo. En este sistema, la posibilidad de colores solo es de 7 y sus correspondientes brillantes (o intermitentes), pero en Windows, de una forma natural podemos llegar a 2^{24} ó 255^3 colores, es decir una posible mezcla de 255 tonos de rojo, con 255 tonos de verde y 255 tonos de azul. Para controlar todo esto, Fivewin nos facilita la seudofunción del preprocesador **RGB(<rojo>,<verde>,<azul>)**. Así los colores básicos serán:

Negro	RGB(0,0,0)
Blanco	RGB(255,255,255)
Rojo	RGB(255,0,0)
Verde	RGB(0,255,0)
Azul	RGB(0,0,255)

Haz un pequeño programa con la siguiente función:

```

/*****
DIALOGO Visualiza una caja de dialogo
*/
static function dialogo

local oDlg,oFon

local color := {0,50,100,150,200,255}
  Matrices con 6 valores de tonos para cada color básico (puedes variar el
  número de tonos para cada color, pero tendrás que cambiar también el bucle
  for y los calculos de línea y columna).
local x,y,z
  Punteros de los tonos

DEFINE FONT oFon NAME 'Arial' WEIGHT -8  // pone el font y sus puntos

DEFINE DIALOG oDlg OF oWnd TITLE 'Tutorial' FROM 0,0 TO 37,70 FONT oFon

for x = 1 to 6
  for y = 1 to 6
    for z = 1 to 6
      @ ((x-1)*7+(y-1))/2 ,(z-1)*6 ;
      say str0(color[x],3)+'-' +str0(color[y],3)+'-' +str0(color[z],3);
      COLOR RGB(color[x],color[y],color[z])
    next
  next
next

```

```
next
```

```
ACTIVATE DIALOG odlg CENTER
```

```
return NIL
```

```
function str0(nNumero,nLen)
return leadchar(str(nNumero,nLen),'0')
```

Tendrá una pantalla similar a la siguiente.

Tutorial					
000-000-000	000-000-050	000-000-100	000-000-150	000-000-200	000-000-255
000-050-000	000-050-050	000-050-100	000-050-150	000-050-200	000-050-255
000-100-000	000-100-050	000-100-100	000-100-150	000-100-200	000-100-255
000-150-000	000-150-050	000-150-100	000-150-150	000-150-200	000-150-255
000-200-000	000-200-050	000-200-100	000-200-150	000-200-200	000-200-255
000-255-000	000-255-050	000-255-100	000-255-150	000-255-200	000-255-255
050-000-000	050-000-050	050-000-100	050-000-150	050-000-200	050-000-255
050-050-000	050-050-050	050-050-100	050-050-150	050-050-200	050-050-255
050-100-000	050-100-050	050-100-100	050-100-150	050-100-200	050-100-255
050-150-000	050-150-050	050-150-100	050-150-150	050-150-200	050-150-255
050-200-000	050-200-050	050-200-100	050-200-150	050-200-200	050-200-255
050-255-000	050-255-050	050-255-100	050-255-150	050-255-200	050-255-255
100-000-000	100-000-050	100-000-100	100-000-150	100-000-200	100-000-255
100-050-000	100-050-050	100-050-100	100-050-150	100-050-200	100-050-255
100-100-000	100-100-050	100-100-100	100-100-150	100-100-200	100-100-255
100-150-000	100-150-050	100-150-100	100-150-150	100-150-200	100-150-255
100-200-000	100-200-050	100-200-100	100-200-150	100-200-200	100-200-255
100-255-000	100-255-050	100-255-100	100-255-150	100-255-200	100-255-255
150-000-000	150-000-050	150-000-100	150-000-150	150-000-200	150-000-255
150-050-000	150-050-050	150-050-100	150-050-150	150-050-200	150-050-255
150-100-000	150-100-050	150-100-100	150-100-150	150-100-200	150-100-255
150-150-000	150-150-050	150-150-100	150-150-150	150-150-200	150-150-255
150-200-000	150-200-050	150-200-100	150-200-150	150-200-200	150-200-255
150-255-000	150-255-050	150-255-100	150-255-150	150-255-200	150-255-255
200-000-000	200-000-050	200-000-100	200-000-150	200-000-200	200-000-255
200-050-000	200-050-050	200-050-100	200-050-150	200-050-200	200-050-255
200-100-000	200-100-050	200-100-100	200-100-150	200-100-200	200-100-255
200-150-000	200-150-050	200-150-100	200-150-150	200-150-200	200-150-255
200-200-000	200-200-050	200-200-100	200-200-150	200-200-200	200-200-255
200-255-000	200-255-050	200-255-100	200-255-150	200-255-200	200-255-255
255-000-000	255-000-050	255-000-100	255-000-150	255-000-200	255-000-255
255-050-000	255-050-050	255-050-100	255-050-150	255-050-200	255-050-255
255-100-000	255-100-050	255-100-100	255-100-150	255-100-200	255-100-255
255-150-000	255-150-050	255-150-100	255-150-150	255-150-200	255-150-255
255-200-000	255-200-050	255-200-100	255-200-150	255-200-200	255-200-255
255-255-000	255-255-050	255-255-100	255-255-150	255-255-200	255-255-255

Observa como el negro es 0-0-0; el verde es 0-255-0; el azul 0-0-255; el blanco es el 255-255-255. Hay colores como el amarillo que ha de formarse por combinación de verde y la suma de rojo y azul, por ejemplo 100-255-200.

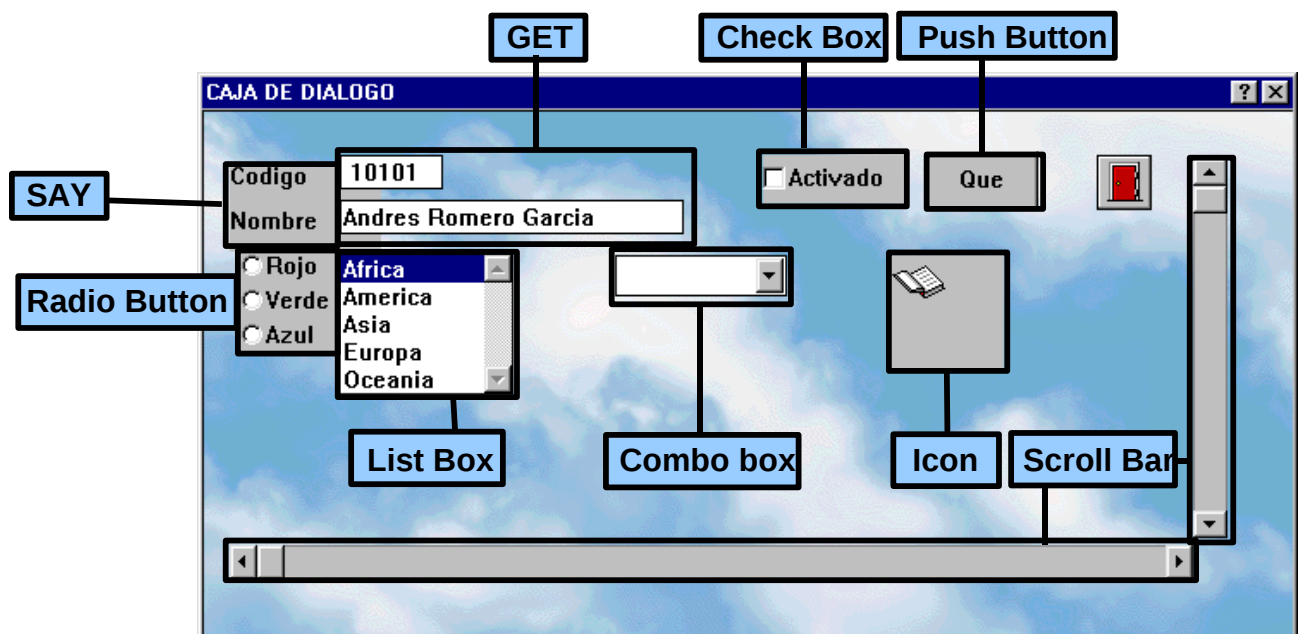
El sistema de color es igual para pantalla que para impresora, pero naturalmente los resultados dependerán de las características físicas de estos equipos. Lo mejor es tener la función de pantalla asequible en cualquier momento. En caso de necesitarlo, se llama a la función, se ve lo que se quiere y se quita. Para la impresora, igual, pero esta

tiene la ventaja de que se puede guardar el papel impreso. Ya se hablará de esta hoja de muestra en el apartado de la clase **TPrinter**.

Con el color de fondo ocurre algo parecido, pero es imposible hacer una pantalla resumen para todos los colores. Lo mejor es que practiques un poco con valores normales. Puedes hacer un programa con 3 RadioButton de 6 opciones y un botón. Un Radio para cada color y una opción para 6 tonalidades. Cuando has seleccionado las tres tonalidades, pulsando el botón aparecerá la imagen de mas arriba pero con el fondo cambiado.

Conclusión

Hemos visto la mayoría de los controles que se pueden hacer programando directamente, sin utilizar recursos. Estos controles tienen numerosos parámetros que los modifican, en algunas ocasiones, bastantes. No obstante, la idea principal de cada control se mantiene. En el capítulo correspondiente a ver con detalle cada uno de ellos profundizaremos hasta el nivel previo al propio objeto. Con lo que sabes puedes hacer tus pinitos, pero como te falta bastante no pretendas hacer una aplicación completa. Lo único que se pretende hasta ahora es conocer el significado de cada control, no su programación. Resumiendo podemos dar un repaso a nuestra última pantalla:



WORKSHOP

El programa, o mejor habría que decir, la filosofía de trabajo con **Workshop**, consiste básicamente en una herramienta de ayuda al programador que debe hacer su trabajo en entorno gráfico. La mecánica de programación por medio de un editor de textos donde escribimos los programas (*.PRG's) estaba bien para los sistemas basados exclusivamente en DOS. Al llegar windows y su entorno gráfico, si bien los editores de texto siguen siendo necesarios, no son la mejor forma de programar la parte de la pantalla. Ya no basta con señalar la clásica instrucción @ **nl,nc SAY...** Ahora es mejor referir las coordenadas en píxel. Coordenadas que dependen de muchos factores, como definición de la pantalla, tamaño de los font, etc. Mentalmente era muy fácil ubicar un SAY o un GET cuando la pantalla tenía 25 líneas por 80 columnas. Con el sistema de píxel la ubicación subconsciente es mucho mas difícil. Para ayudarnos en este trabajo viene a nosotros el sistema **Workshop**.

Igual que nos pasa a nosotros como programadores de clipper en DOS que estamos emigrando a windows, les paso hace años a los programadores de C++ y Pascal que de DOS emigraban a Windows. Para solucionarles este problema a sus clientes, tanto Microsoft como Borlan crearon una herramienta de trabajo. Microsoft creo AppStudio de la que no hablaremos aquí prácticamente nada. Por su parte Borlan creo **Workshop**. Puesto de que de hecho este es el entorno que se ha ido imponiendo entre la mayoría de nosotros, solo hablaremos de este último. Borlan lo suministra como una herramienta de ayuda en muchos de sus programas, como es el caso de Turbo Pascal, el C++ y ahora en Delphi y C++ Builder. (Nota: No todas las versiones lo tienen. Lee en la introducción de este tutorial para ver como conseguirlo.)

A medida que iban apareciendo versiones de Pascal y C++, también iban apareciendo versiones de Workshop. La que veremos aquí es la versión 4.5 que parece

 Workhelp	hlp	643.393	22/05/95 04:50
 Workshop	exe	42.448	01/01/94 00:04
 Workres	dll	407.616	01/01/94 00:04
 Worklib2	dll	120.656	01/01/94 00:04
 Worklib1	dll	731.328	14/02/99 19:38
 Worked5	dll	34.128	01/01/94 00:04
 Worked4	dll	25.456	01/01/94 00:04
 Worked3	dll	250.016	01/01/94 00:04
 Worked2	dll	220.672	01/01/94 00:04
 Worked1	dll	73.040	14/02/99 19:38
 Bwcc	dll	164.928	28/02/95 11:14
 Bivbx30	dll	102.400	01/01/94 00:04

ser la última y sin continuidad, puesto que los nuevos sistemas Delphi y C++ Builder no necesitan de esta herramienta y Borlan ha dejado de desarrollar y suministrar nuevas versiones.

El sistema necesita de los programas indicados en la tabla adjunta.

Este esquema esta tomado de un directorio real. Si conoces otro más actual ayúdame a mejorar.

Lo más práctico es copiarlo en un directorio de trabajo que se tenga acceso mediante la instrucción SET PATH, por ejemplo en \CLIPPER5\BIN o como haremos a lo largo del tutorial en \CLIWIN\BIN. Veamos ahora un poco sobre la forma de trabajar y algunos conceptos necesarios antes de empezar.

El sistema divide los resultados que graba en los siguientes conceptos:

El primer nivel es el denominado **Proyecto**. Este nivel es el conjunto contenedor de todos los demás. Podemos decir que este nivel sería el equivalente al concepto de **Aplicación**.

Dentro de un proyecto existen los denominados **Recursos**. Estos recursos, que luego hablaremos muy ampliamente es lo que podemos equiparar a las piezas que forman el proyecto. Los recursos pueden ser ventanas de windows con el equivalente a say's, get's, carpetas, colores, etc. etc. o pueden ser iconos, BMP's, etc. Las distintas piezas o recursos que se componen un proyecto pueden estar físicamente dentro del fichero proyecto o pueden estar externamente y solo haber una referencia a su situación.

Podemos hacer una aplicación (PRG's) haciendo uso de un proyecto con todos los recursos necesarios o solo algunos y el resto hacerlo por el sistema clásico de @ nl,nc SAY. Como veremos posteriormente, en el sistema de recursos no todo es lo mejor al cien por cien, de forma que lo usaremos únicamente cuando sea mejor. Por ejemplo un sistema de menús y submenús es mejor hacerlo con nuestro editor de texto de toda la vida.

Los recursos y el proyecto contenedor puede actuar en nuestro programa de dos formas distintas:

- Insertando los elementos en formato binario dentro de nuestro programa *.EXE.- Es como si tuviésemos una librería externa antes de enlazar que por medio del proceso de enlazado se incorpora al ejecutable. Desde fuera solo se ve un fichero con extensión EXE donde están nuestros módulos OBJ y los módulos de la librería externa.
- Formando un fichero independiente con extensión *.DLL que nuestro programa ha de llamar cada vez que se inicia, dejarlo abierto, utilizar los recursos y por último cerrar cuando termina.

Cada uno de estos sistemas tiene sus ventajas e inconvenientes. Como casi siempre las ventajas de uno son los inconvenientes del otro.

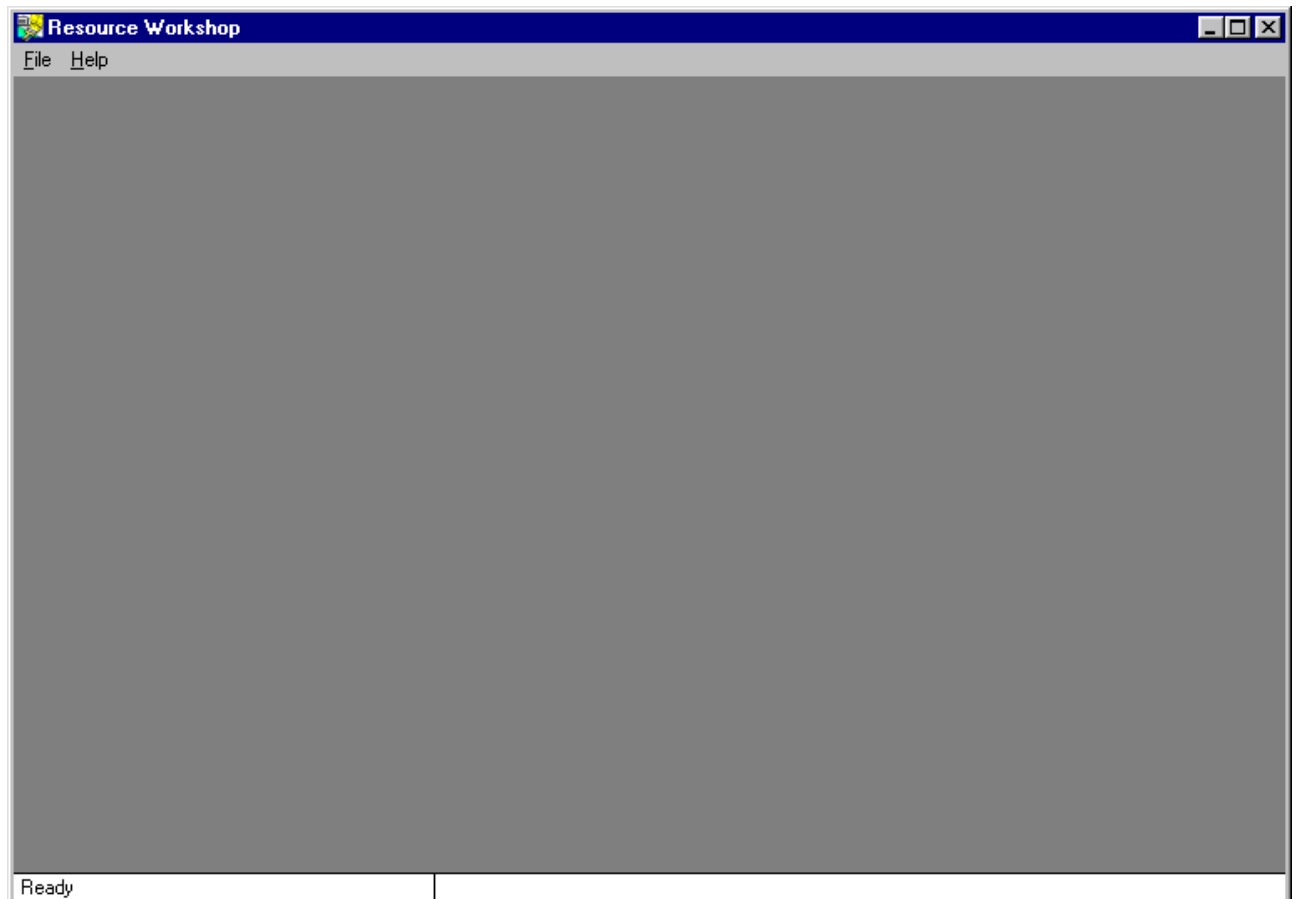
Concepto	RC	DLL
Compilar cada vez que se enlaza el ejecutable.	SI	NO
Recursos en el mismo fichero .EXE.	SI	NO
Se compila solo cuando se modifica algún recurso.	NO	SI
Forma un fichero aparte que se puede suministrar independientemente del ejecutable.	NO	SI

Nosotros utilizaremos ahora solo la forma de fichero **RC** (Recursos compilados) e inclusión de los recursos en el ejecutable. El sistema de DLL lo veremos en otro capítulo.

Tutorial de Workshop

Vamos a ver ahora como se trabaja con el programa Workshop, que de ahora en adelante llamaremos simplemente **WS**. (personalmente tengo un BAT que poniendo WS llama al workshop. De esta manera, cada vez que veo el directorio \BIN, veo el fichero tal como debe ser y no renombrado.)

Lo primero y fundamental es agenciarnos el programa. (si necesitas saber como, lee la parte de introducción del tutorial). Una vez que tengamos al menos los ficheros básicos, los copiaremos en **\CLIWIN\BIN** y simplemente para saber que funciona llamar al programa. Debe aparecer una pantalla como esta:



Sal por ahora pulsando **[Alt-F4]** y luego volveremos al programa.

Como programa clipper/Fivewin tendremos una muestra patrón muy sencillita a la que iremos añadiendo los distintos recursos para ver como funcionan. Si quieres seguir el tutorial lo mas exacto posible crea un directorio **\FWTUTORIAL**. El él iremos haciendo las distintas prácticas.

El programa **PRG**, el fichero **MAK** y el fichero **LNK** pueden ser algo parecido a esto:

```
* TUTOR
#include 'romewin.inc'
static oWnd
*****

function tutor
local oBru
LoadLibrary('\cliwin\bin\bwcc.dll')           // carga la librería de Borlan
DEFINE BRUSH oBru STYLE BRICKS                // pone el fondo de la ventana
DEFINE WINDOW oWnd TITLE 'TUTOR 0' BRUSH oBru // define la ventana principal
SET MESSAGE OF oWnd TO 'Tutorial' CLOCK DATE KEYBOARD // pone los mensajes
ACTIVATE WINDOW oWnd MAXIMIZED ON INIT dialogo() // activa la ventana...
                                                // ...llamando a la función
return NIL                                    // termina el programa

/*****
DIALOGO Visualiza una caja de dialogo
*/
function dialogo
return NIL
```

El fichero MAK puede ser este

```
.PRG.OBJ:
  clipper $* -m -n -a -w -p
tutor.obj      : tutor.prg
```

El fichero LNK puede ser este

```
blinker incremental off
blinker executable clipper f30;v8;r0;e0
blinker overlay pageframe on

FILE  tutor, getsys
OUTPUT tutor

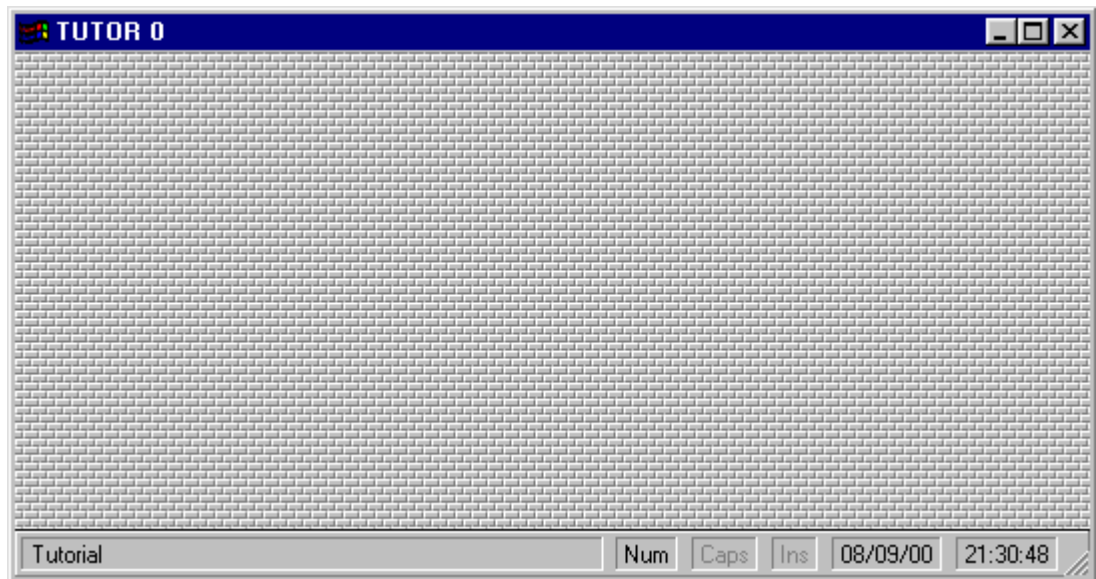
DEFBEGIN
  name      FiveWin
  description 'Clipper for Windows library'
  exetype   Windows 3.1
  code      moveable discardable preload
  data      preload moveable
  stacksize 9500
  heapsize  2048
  segment   'PLANKTON_TEXT' nondiscardable
  segment   'EXTEND_TEXT'   nondiscardable
  segment   'OM_TEXT'       nondiscardable
  segment   'OSMEM_TEXT'    nondiscardable
  segment   'SORTOF_TEXT'   nondiscardable
  segment   'STACK_TEXT'    nondiscardable
DEFEND

LIB rootvm

#VARIOS
```

```
#CLIPPER5\OBJ
file rddsys
#CLIPPER5\LIB
lib romewin
lib dbfcdx
lib _dbfcdx
SEARCH Five, FiveC, Objects
LIB WinApi
```

Con este programa tan simple solo aparece una ventana que llena la pantalla y que si la reducimos tiene el siguiente aspecto:

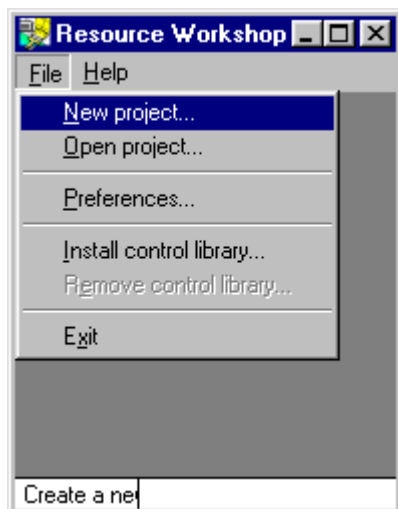


Ahora vamos a ir añadiendo los famosos recursos. Lo primero será crear una caja de diálogo donde iremos poniendo los otros. Vallamos por partes.

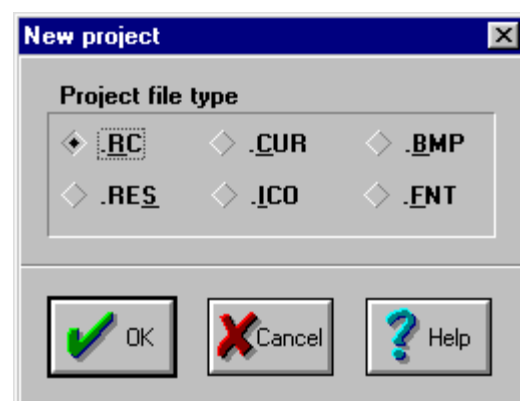
Primero llamamos al programa WS.



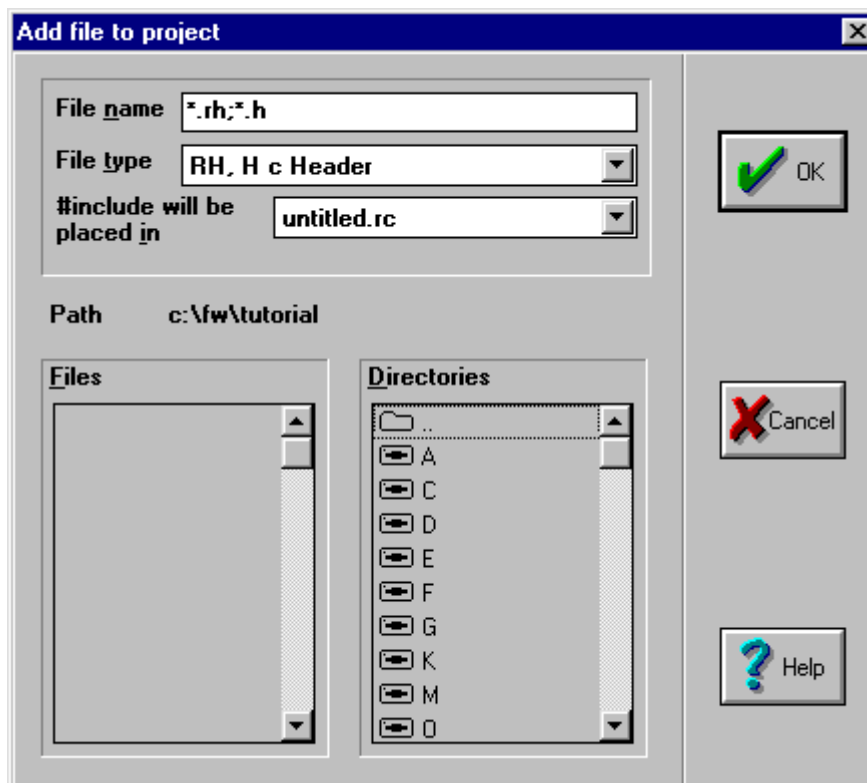
Presenta dos opciones **File** y **Help**. Como en muchas otras aplicaciones entramos por el apartado **File** [**Alt-F**]. El apartado **Help** (ayuda) lo dejaremos para otra ocasión) Aparecerá el siguiente submenú:



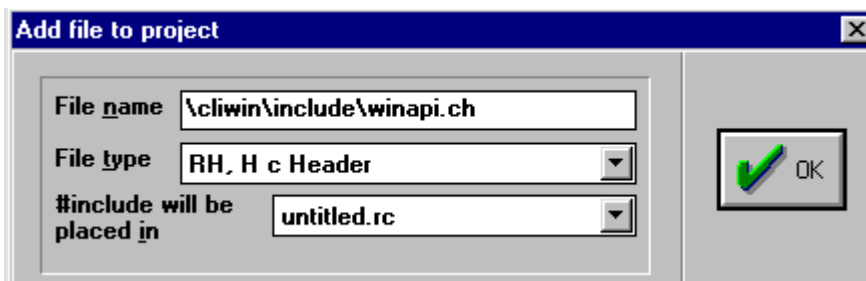
Dentro de este submenú elegiremos ahora **New Project...** y aparecerá la nueva ventana:



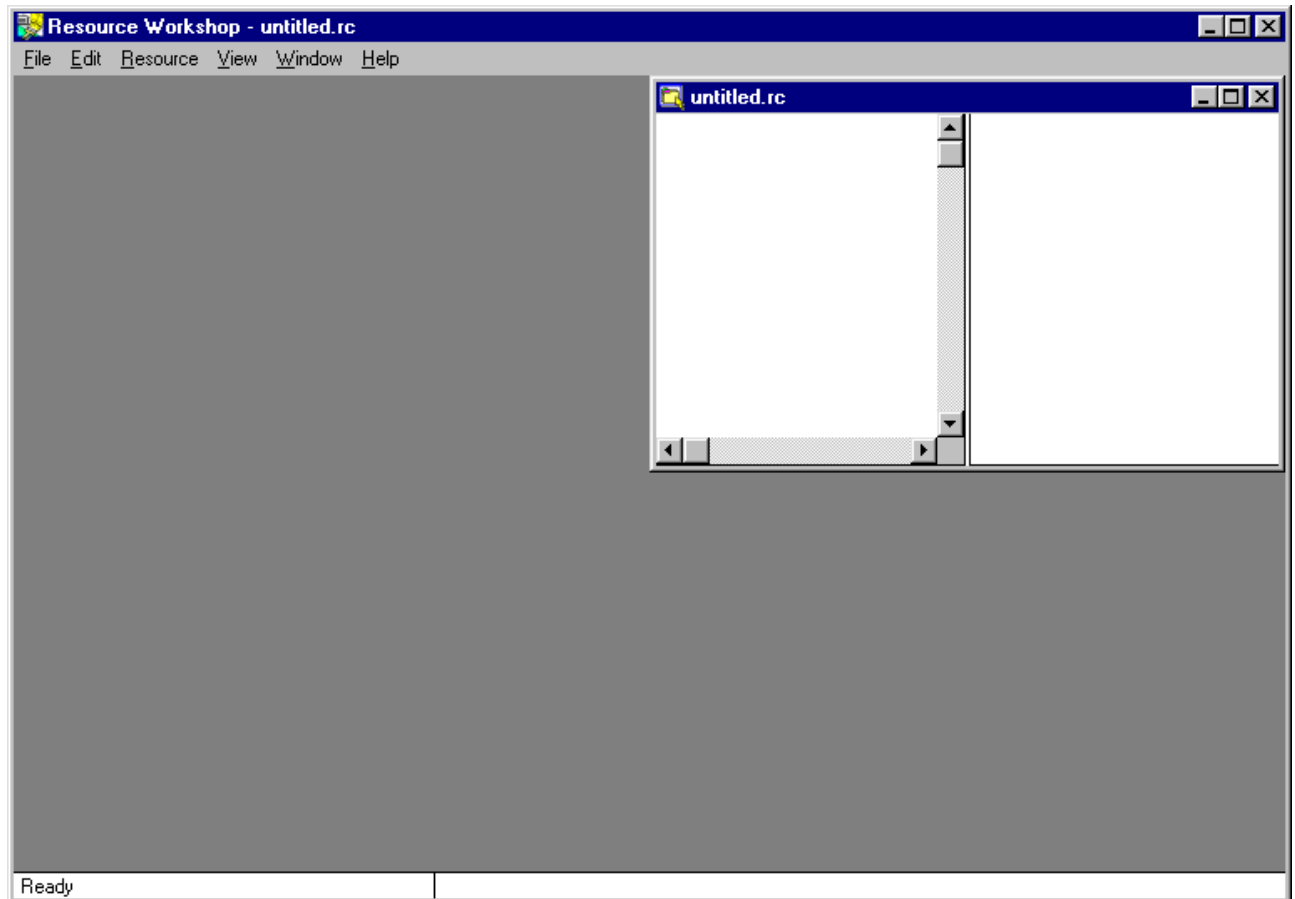
De momento pulsaremos en **OK**. El resto de las opciones lo veremos mas adelante.



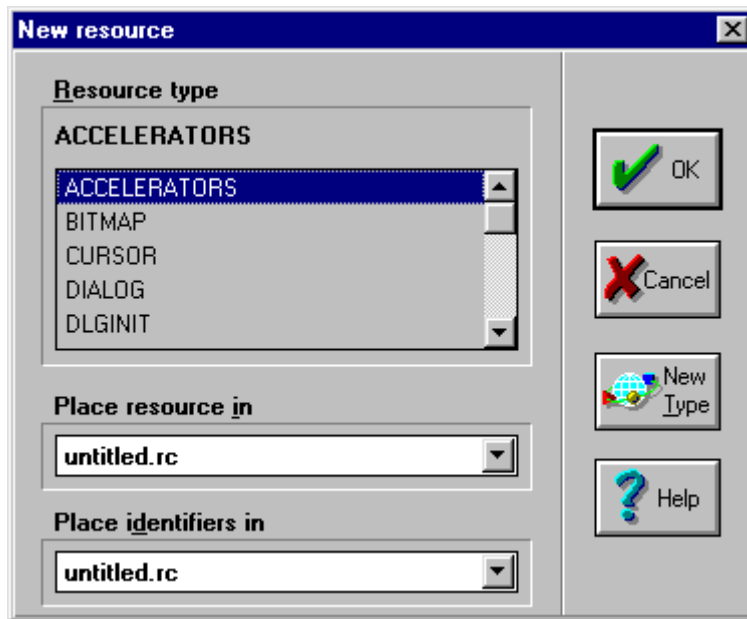
El programa ya sabe que queremos hacer un proyecto de tipo RC y ahora nos pregunta si queremos hacer un **#include** antes de continuar. Es bueno (y para ciertos casos imprescindible) incluir **winapi.ch**. Lo haremos poniéndolo en **File name** de la siguiente forma:



Pulsamos **OK** y aparecen dos cambios significativos. El primero en notarse es la pequeña ventana a la derecha. De momento esta vacía, luego irán apareciendo los distintos recursos (y otros datos) de nuestro proyecto. Otro cambio importante es el menú. Ahora no solo aparece **File** y **Help**, también aparecen las opciones **Edit**, **Resource**, **View** y **Window**. Por ahora lo que nos interesa es hacer nuestro primer recurso, de modo que dejaremos todo lo demás para luego. Pinchemos en **Resource** o pulsemos **[Alt-R]**. Aparecerá el submenú que se indica a continuación.

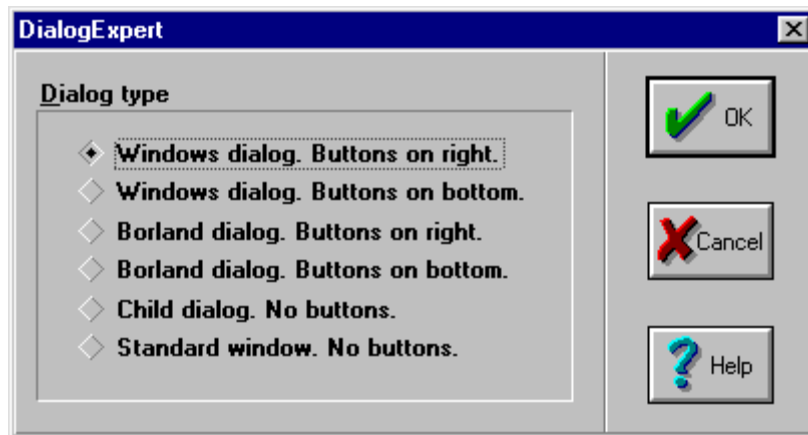


La opción de este submenú que ahora nos interesa es como se puede imaginar **New...**. Pulsemos en ella y aparece una nueva ventana de diálogo.



En esta ventana nos pide el programa, el tipo de recurso que se quiere hacer. Para nuestro ejemplo debemos elegir **DIALOG** que está en la cuarta posición. Una vez pinchado **DIALOG** debemos pulsar **OK** como es natural. De momento dejamos todo lo demás igual. Luego volveremos sobre ello.

Vaya, otra ventana más. Ahora aparece la siguiente:

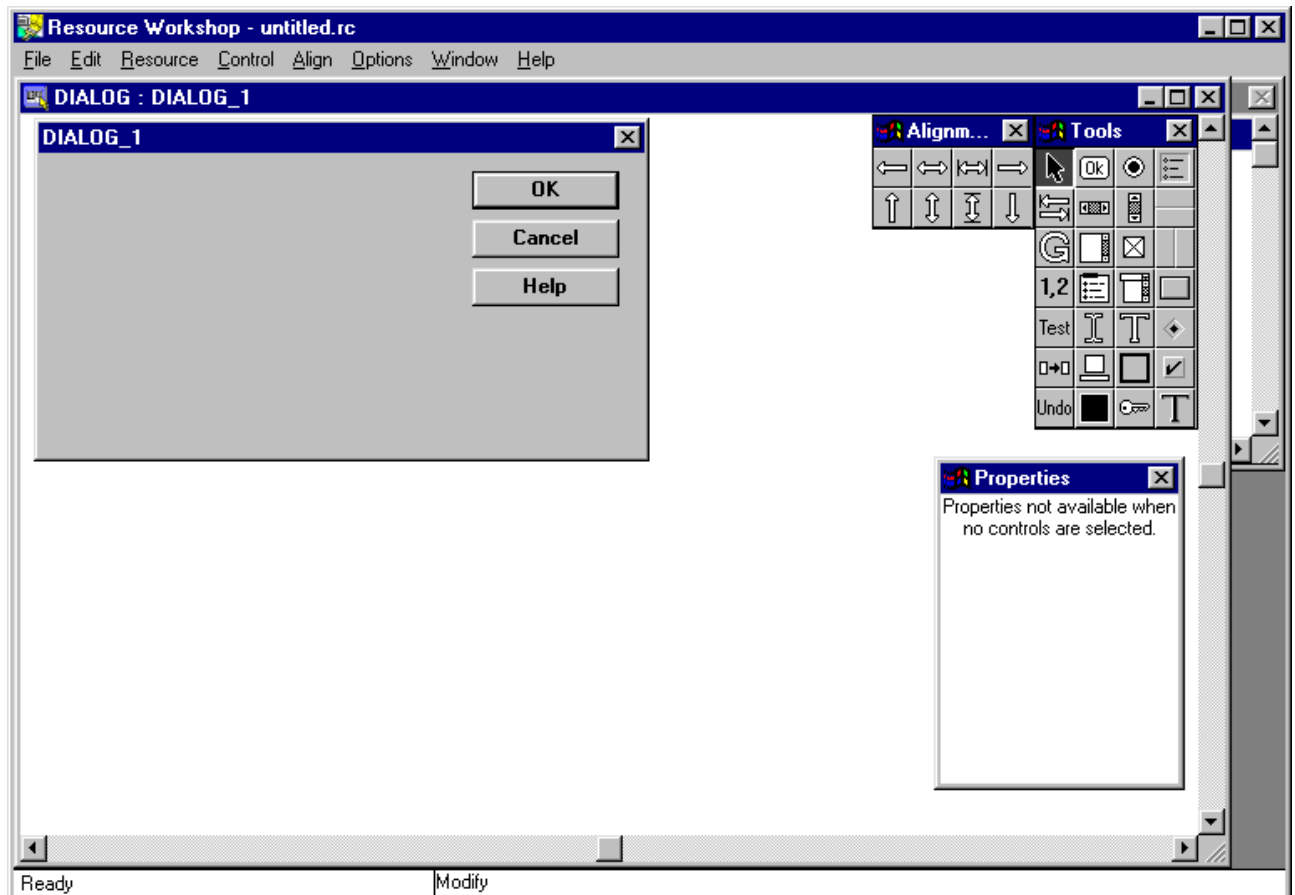


Sequimos queriendo hacer nuestra primera ventana. No nos debe preocupar todo lo demás. En la primera opción nos presenta una ventana de tipo diálogo con los botones a la derecha. Esto de los botones, que nosotros no hemos pedido, es porque el programa con el interés de ayudarnos cree que toda ventana ha de tener al menos

tres botones: OK, Cancelar y Ayuda. Por este motivo crea una ventana con estos tres botones. Como para nuestro propósito no estorban, daremos el **OK** y continuamos.

Fíjate en la pantalla de la página siguiente. Aparecen los siguientes cambios:

- Una nueva ventana que a su vez tiene otras como ahora veremos.
- En el menú han cambiado las opciones.
- Tres ventanas flotantes y la esperada ventana de DIALOGO que vamos a utilizar como recurso.
- Según tu configuración puede que no veas todas las ventanas. Para verlas, entra en la opción del menú **Option**. Según se vean o no las ventanas, las tres primeras opciones empezarán por Hice (si se ve, para que no se vea) o Show (si no se ve, para que se vea). Tu debes tener todo a Hide, indicando que ya se ve.
 - o Hide tools (herramientas)
 - o Hide Alignment (alineación)
 - o Hide propiedades (propiedades)

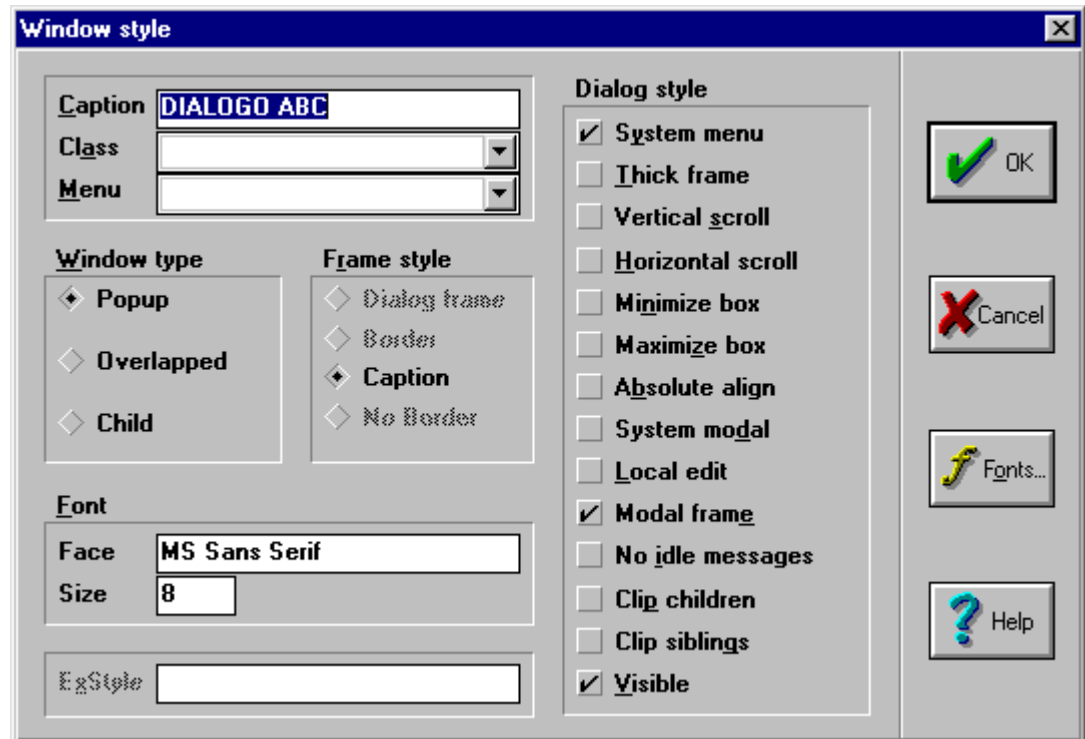
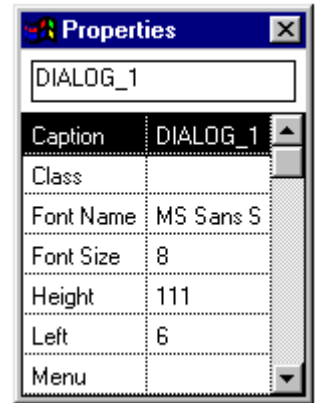


Veamos por encima lo que son cada ventana:

- La ventana DIALOG_1. Es nuestro recurso DIALOGO que el programa, por su cuenta le da el título de DIALOG_1. Ahora lo cambiaremos. En ella podemos ver también los tres botones que el programa ha puesto por su cuenta. Recuerda que le dimos el consentimiento en el paso anterior.
- La ventana Alignm... Se utiliza para alinear los distintos elementos que pueden haber dentro de nuestro DIALOGO. Es una cuestión de estética y también de ayuda para ajustar cada recurso en su sitio.
- La ventana Tools... Es quizás la que más se utiliza. Sirve para ir eligiendo los distintos elementos que iremos poniendo en nuestra ventana. Por ahora la dejamos así, luego tendremos ocasión de volver a ella efusivamente.
- La ventana Properties. Indica las propiedades del elemento seleccionado. (Digo elemento y no recurso puesto que teóricamente todavía no sabes lo que es un recurso.) Puesto que ahora no hay señalado ninguno, sale un mensaje que no es posible indicar las propiedades.

Se que quieres toquetear con el ratón todas las ventanas y botones, pero por bien del tutorial sigue las instrucciones. Ya tendrá tiempo. Lo primero que vamos a hacer es cambiar ese feo nombre de DIALOG_1, por uno mas normal, por ejemplo DIALOGO. En una aplicación pondríamos FICHA DE CLIENTES o algo parecido. Veamos como. Hay dos formas. Las veremos someramente con la única intención de cambiar el nombre de la ventana y pasar de todo lo demás.

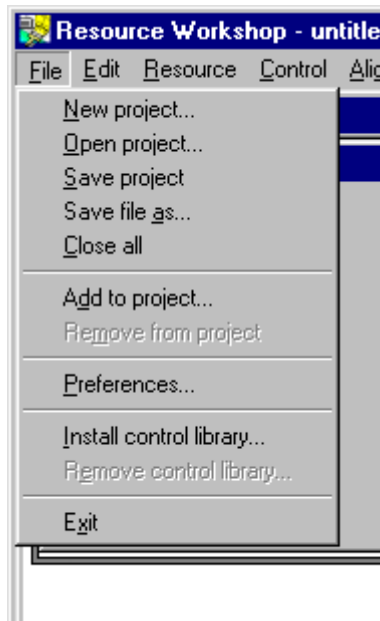
- Pincha una sola vez en la parte azul de la ventana `DIALOG_1`. Con esto hemos seleccionado nuestra ventana. Ahora la ventana de propiedades presenta el aspecto de la derecha. Hagamos un pequeño alto. Todo recurso tiene unas propiedades iniciales que pueden establecerse en el momento del diseño. Estas propiedades pueden ser cambiadas posteriormente en el momento de la programación o incluso en el momento de la ejecución. Entre esas propiedades ahora nos fijaremos en el título, o más técnicamente en **Caption**. Si pulsas en Caption o directamente en el texto `DIALOG_1`, podremos cambiar el texto. Hazlo de cualquiera de las dos maneras y pon **DIALOGO ABC**. Observa que a medida que editas, también cambia el título de la ventana. Veamos ahora la otra forma de cambiar el título de un diálogo.
- Haz doble clic en la parte azul de la ventana que ahora tiene por nombre **DIALOGO ABC**. Aparecerá ahora la siguiente ventana del programa:



En la parte superior izquierda aparece la opción **Caption**. Selecciónala y quita las letras ABC. En otra parte del tutorial veremos el resto de las opciones. Ahora pulsa en **OK**

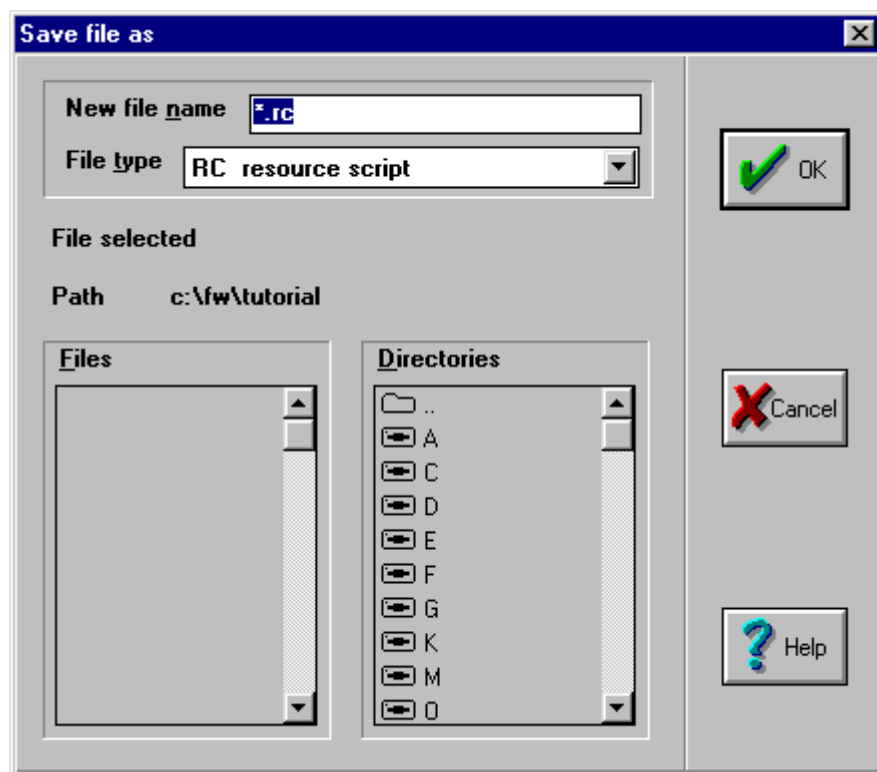
Bien ya tenemos una ventana de dialogo que vamos ha convertir en recurso. Lo primero será grabar los datos en un fichero que luego el compilador interprete e inserte en nuestro ejecutable. Para ello realizamos los siguientes pasos:

- Pinchamos en la opción del menú **File**.
- Aparecerá el siguiente submenú:



Como podemos ver ya no son las opciones iniciales donde solo se podía señalar New project. Ahora tenemos también las opciones de guardar el proyecto y de guardarlo con otro nombre. Elijamos **Save file as...**

De Nuevo aparece otra ventana con un aspecto similar a la siguiente:



Asegúrate seleccionando en **Directories**, que estás en el directorio **IFW/TUTORIAL** (o el que tu hayas elegido para hacer el tutorial). Después escribe en **New file name** el nombre TUTOR.RC (es importante que el proyecto donde están los recursos tenga el mismo nombre que el fichero EXE de nuestro programa). Por último pulsa **OK**. Con esto hemos grabado el proyecto pero seguimos en el programa. Vuelve a entrar por File y selecciona Exit (o pulsa directamente [**Alt-F4**]).

Ya hemos hecho nuestro primer recorrido por Workshop. Es el momento de comprobar que ha servido para algo. De todas formas antes de seguir con el programa

haz una pequeña comprobación que solo sirve para estos momentos iniciales, luego no será necesario en absoluto. Comprueba que en el directorio de trabajo existe un fichero denominado TUTOR.RC. Si entras con un editor podrás ver su contenido, pero no debes modificarlo. Será similar a esto:

```

/*****
tutor.rc
produced by Borland Resource Workshop
*****/

#define DIALOG_1 1
#include "winapi.ch"
DIALOG_1 DIALOG 6, 15, 207, 111
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "DIALOGO"
FONT 8, "MS Sans Serif"
{
    DEFPUSHBUTTON "OK", IDOK, 148, 6, 50, 14
    PUSHBUTTON "Cancel", IDCANCEL, 148, 24, 50, 14
    PUSHBUTTON "Help", IDHELP, 148, 42, 50, 14
}

```

Como ves se parece a clipper. Esto es porque tanto el fuente RC como los fuentes PRG se parecen a su vez a **C**. Las primeras líneas son comentarios. Luego viene un `#define` que pone `DIALOG_1` al número 1. Como ves se sigue llamando `DIALOG_1` y no `DIALOGO`. Este es el nombre interno que da el programa, pero nosotros y sobre todos los usuarios verá `DIALOGO`. A continuación viene el `#include` que pusimos al principio, cuando estábamos creando el proyecto. Después viene la definición propiamente dicha de nuestro `DIALOGO`, que ya podemos ir llamando recurso. Entre otras cosas viene la cláusula `CAPTION` donde si viene el nombre `DIALOGO`. Por último, entre llaves, vienen las definiciones de los tres botones que puso el programa automáticamente tras nuestro consentimiento.

Si has llegado hasta aquí, ya tienes la mitad del camino hecho. Ahora tenemos que incluir este recurso en nuestro fichero EXE y ver como funciona. Dejamos el programa TUTOR de forma que presentaba una ventana vacía. Hagamos dos cambios. Uno en el programa PRG y otro en la mecánica de compilación-enlace.

- En el programa debemos indicar que se quiere utilizar un recurso. Al ser del tipo RC, es decir NO ser DLL, los recursos están incluidos en el ejecutable, de forma que no hay que decir de donde tomarlos, pero SI hay que decir que recurso de todos los que puede haber en el ejecutable queremos utilizar. Para ello modificamos el programa que ahora quedará de la siguiente forma:

```

* TUTOR
#include 'romewin.inc'
static oWnd

```

```

*****

```

```

function tutor
local oBru
DEFINE BRUSH oBru STYLE BRICKS
DEFINE WINDOW oWnd TITLE 'TUTOR 2' BRUSH oBru
SET MESSAGE OF oWnd TO 'Tutorial' CLOCK DATE KEYBOARD
ACTIVATE WINDOW oWnd MAXIMIZED ON INIT dialogo()
return NIL
/*****
DIALOGO Visualiza una caja de dialogo
*/
function dialogo
#define DIALOGO 1
local oDlg

DEFINE DIALOG oDlg RESOURCE DIALOGO OF oWnd
ACTIVATE DIALOG oDlg CENTERED

return NIL

```

La única diferencia son las cuatro líneas indicadas con una raya vertical a la izquierda.

- Primero definimos DIALOGO como 1. Recuerda que workshop puso DIALOG_1 a 1. Esto es porque para referirse a los recursos ha de hacerse por número y no por nombre. Para evitar esto echamos mano del preprocesador de clipper y asignamos un nombre al recurso 1. Luego se verá que esto se puede complicar bastante y lo mejor será hacer un #include donde se pongan todos los #define de recursos de un proyecto.
- Asignamos como variable local el objeto que contendrá el recurso. Como veremos en otro momento, los recursos toman vida en el programa. Su control se hace mediante el objeto que se asigna al recurso.
- La tercera línea es la que define el recurso. El formato, como luego veremos al ir tratando los distintos recursos desde la perspectiva de la programación, es el siguiente:

```

DEFINE DIALOG <objeto> ;           //definición del recurso DIALOGO.
RESOURCE <numero_del_recurso> ; //enlace con el fuente RC de workshop.
OF <objeto_que_lo_contiene>      //objeto que contendrá el recurso.

```

- Como en otras partes de Fivewin, después de definir un objeto hay que activarlo mediante la cuarta línea. En este caso de forma centrada (CENTERED)
- La otra modificación que hay que hacer es en la **mecánica de compilación-enlazado**. Ahora habrá que compilar-enlazar-insertar. La compilación por medio de **CLIPPER.EXE**, con un fichero ***.MAK** y el enlace con **BLINKER** y un fichero ***.LNK** sigue igual, pero una vez obtenido el fichero ***.EXE**, por ejemplo TUTOR.EXE, debemos incluirle los recursos diseñados con Workshop. Para ello hacemos la “compilación” RC y además le indicamos que el fichero binario desarrollado lo incluya en nuestro ejecutable.

Para hacer todo esto nos basamos en el compilador que también se suministra con el workshop **BRC.EXE**.

Además de BRC.EXE hace falta los siguientes ficheros:
RTM.EXE, BRCC.EXE, WORKOPT.DOS y RLINK.EXE
Que debemos tener en **\\CLIWIN\\BIN**

La ejecución es mediante la instrucción:

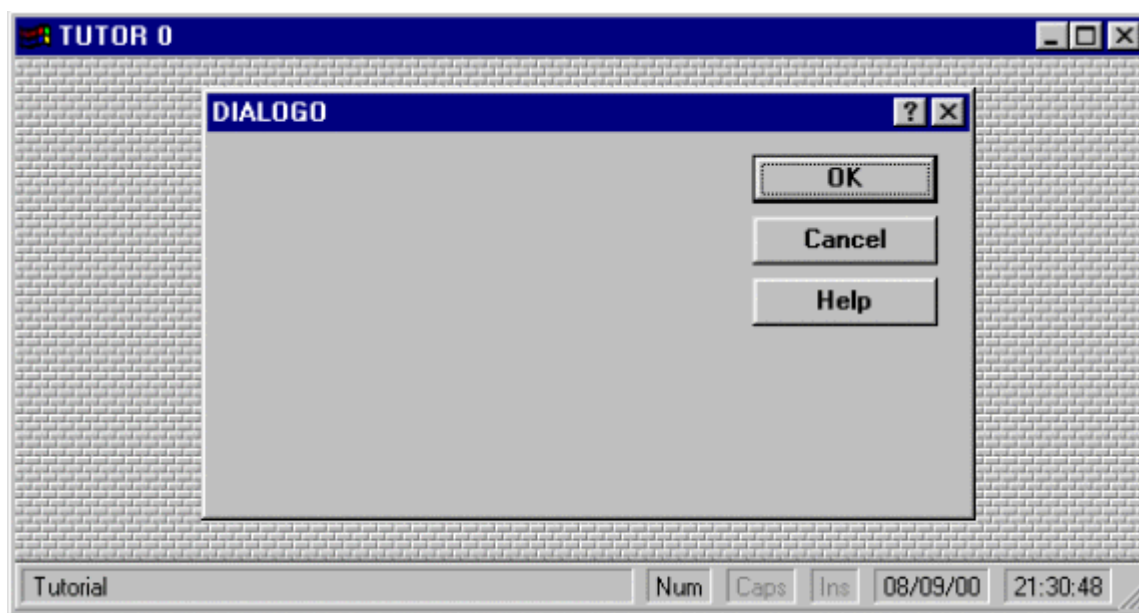
BRC -k <nombre_del_ejecutable>

En nuestro caso será **BRC -k tutor**. Ahora se puede comprender porque a nuestro proyecto en workshop le dimos el mismo nombre que a nuestro ejecutable clipper. El programa **BRC** solo admite un parámetro como nombre de fichero, y este ha de ser el mismo para los dos (ejecutable y recursos)

Como eres un programador profesional estoy seguro de que sabrás como añadir esta última instrucción en tu proceso de compilación de forma que quede:

- Compilar.
- Enlazar
- Insertar

Una vez ejecutado nuestro nuevo programa, quedará similar a lo siguiente:



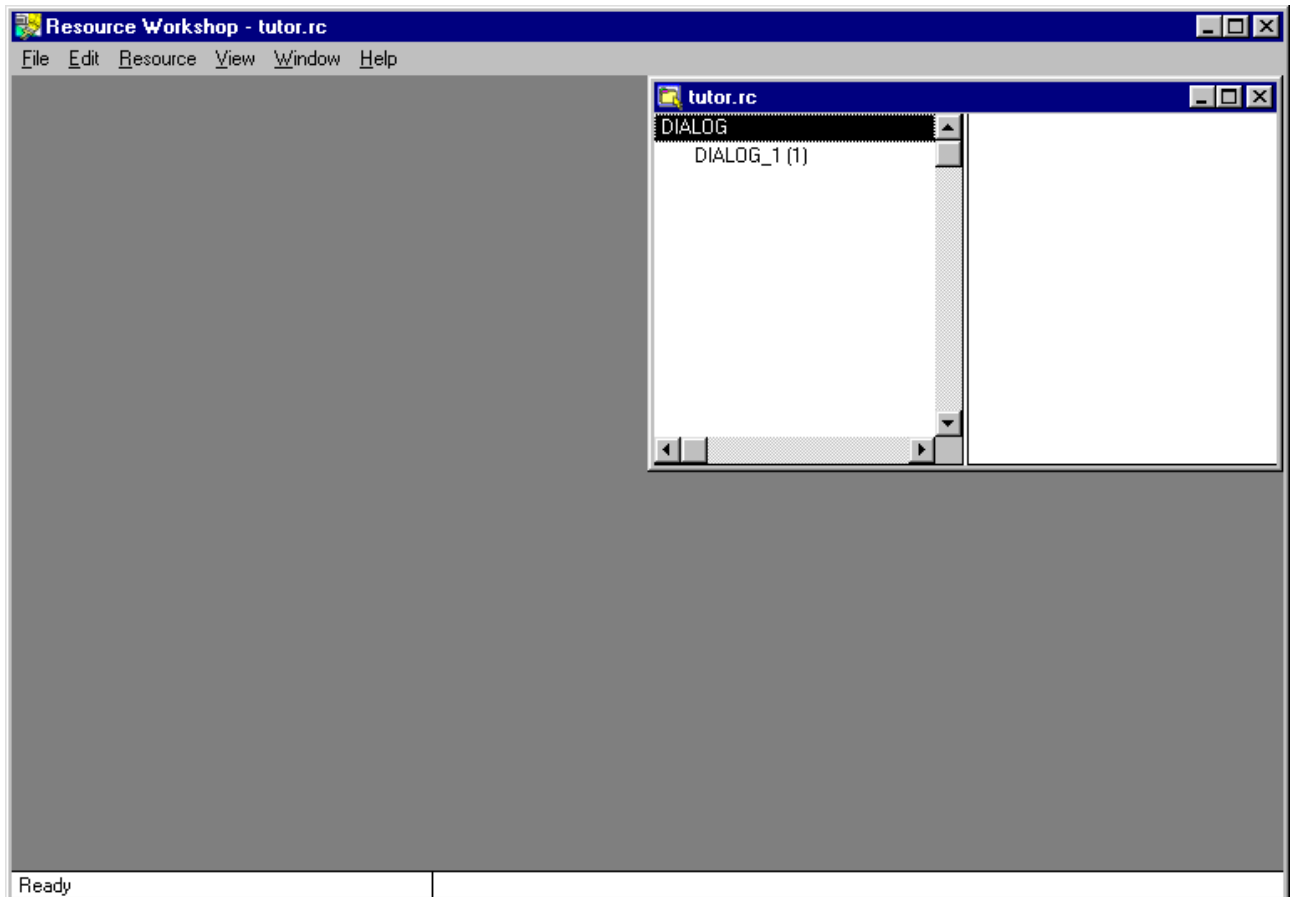
(Aquí se representa una composición, puesto que la ventana contenedor ocupará toda la pantalla y la ventana DIALOGO solo una parte relativamente pequeña)

Inclusión de recursos

Hasta aquí hemos visto el hilo conductor que nos lleva desde un programa sin recursos a un programa con una ventana de diálogo generada mediante un recurso. Siendo una parte muy pequeña en el desarrollo normal de una aplicación, para el que empieza es muy importante, puesto que sin esta secuencia de acciones es difícil dar con la mecánica de trabajo mediante workshop.

Ahora vamos a dar unos pasitos mas por el mundo de los recursos siempre vistos desde la perspectiva del programa workshop. Con cada nuevo recurso que añadamos a nuestro pequeño programa TUTOR, iremos también viendo nuevas posibilidades del programa. Comencemos.

Llamemos al programa WS. Nos aparecerá la pantalla de entrada. Ahora en lugar de elegir **New project...**, elegimos **Open project...** En la ventana que se presenta ya aparece nuestro proyecto anterior **tutor.rc**. Lo señalamos y pinchamos en **OK**, o hacemos doble clic sobre **tutor.rc**. De nuevo aparece la ventana con la lista de los recursos dentro del proyecto. Esta tendrá un aspecto similar a:



Para no cansarte con demasiados detalles y puesto que vas siendo un experto en workshop, no bajaremos mucho de nivel y solo indicaremos pantallas cuando sean imprescindibles.

Al hacer doble clic en **DIALOG_1**, aparecerá nuestro anterior recurso **DIALOGO** con los tres botones. Pincha en uno cualquiera de ellos, por ejemplo en el de **OK**. Aparecerá

la tabla de propiedades y quedará remarcado. Como queremos hacer nuestros propios recursos, vamos a borrarlo. Simplemente pulsa la tecla [**Supr**]. Como veras borrar es relativamente fácil. Si te equivocas existe la posibilidad de restaurar lo borrado pulsando [**Alt-retroceder**]. Borra los otros dos botones. Graba el proyecto, compila de nuevo el programa (esta vez no habrá cambio en el PRG) y activa el nuevo ejecutable. Verás que sigue apareciendo el recurso del diálogo, pero esta vez no aparecen los botones.

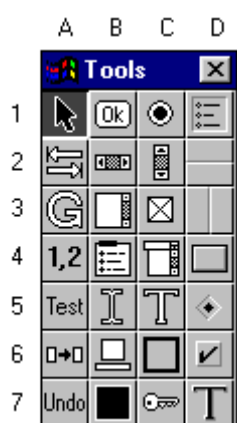
Algo tan simple como esto demuestra que puedes cambiar el aspecto de tu aplicación sin cambiar ninguna línea del programa fuente. Solo has cambiado el recurso. Vamos ahora a ir añadiendo nuevos recursos. Para cada uno de ellos, el proceso es siempre el mismo:

- Ejecutar WS.
- Abrir el proyecto.
- Hacer las modificaciones a los recursos, añadiendo, borrando, etc.
- Grabar el proyecto.
- Salir (si no se ha grabado el programa pregunta si graba la modificación).
- Compilar-enlazar-insertar.
- Comprobar los resultados.

Vamos ahora a incluir un texto. En clipper sería un **SAY**, por ejemplo la palabra 'Nombre'. Estos textos se utilizarán con el equivalente a un **GET**. En recursos no se puede hacerla combinación SAY-GET. Hay que hacer primero el SAY como un **texto fijo**, y luego el GET como un **texto editado**.

Llama al WS y llega hasta el recurso DIALOGO. Fijate en la ventana de herramientas **tools**. En esta ventana tienes todos los recursos que pueden incluirse. Recuerda que cada uno de estos recursos tiene múltiples propiedades que pueden ser cambiadas, de forma que la lista de recursos aparentes es muy grande. Vamos a darla un ligero repaso.

Debido a que no todas las herramientas tienen un nombre de una sola palabra y con el fin de poder entendernos y no tener que representar cada uno de los botones vamos a llegar a un acuerdo en cuanto a la representación de cada uno de ellos. Se ha dividido la ventana en filas de la **1** a la **7** y en columnas de la **A** a la **D**.



Cada vez que se selecciona una herramienta, se puede deseleccionar pulsando sobre otra o sobre la flecha (1-A). Vamos a poner nuestro texto. Para ello hay dos herramientas señaladas con una **T**. La **T** en blanco (5-C) y la **T** en negro (7-D). Vamos a poner las dos y ver sus diferencias.

Borlan ha dividido la caja de herramientas en cuatro columnas. La primera (A) son controles del propio workshop, NO son recursos. La segunda (B) y tercera (C) son recursos standard de Windows. La cuarta columna (D) son recursos propios de Borlan y para que puedan ser ejecutados desde nuestro programa, tenemos que cargar **BWCC.DLL** y **CTL3DV2.DLL** como ya veremos en otro momento.

Pincha en la T blanca (5-C). El cursor aparecerá con una pequeña cruz (el puntero) y un recuadro con la palabra **text**. Llévalo sobre cualquier parte de la ventana DIALOGO. Pinchando y arrastrando veras que aparece un recuadro. Cuando consideres que es suficientemente grande, pero no demasiado suelta. Aparecerá un rectángulo con borde y la palabra **Text** en su interior. También aparecerá en el cuadro de propiedades.

Cambia el texto pulsando dos veces sobre el rectángulo recién creado o sobre el texto en las propiedades y escribe “**T blanca**”.

Ahora haz lo mismo con la T negra (7-D), pero pon de texto “**T negra**”. Si tienes curiosidad antes de seguir, genera el programa y ejecútalo. Fivewin dará un error. Veamos la causa. Con el WS quita la T negra (señalando y Suprimiendo) y compila de nuevo. Ahora no da error. Pon de nuevo el recurso con la T negra y analicemos un poco.

Tendrás ahora dos textos. En uno pone **T blanca** y en otro **T negra**. Haz doble clic en **T blanca**. Te aparecerá una ventana con título **Static style**. Pulsa **Cancel** y ahora haz doble clic en la **T Negra**. Saldrá una ventana de otra forma titulada **Borlan static text style**. La **T blanca** (5-C) es un **texto normal**. La **T negra** (7-D) es un texto con un estilo propio de Borlan que no es reconocido por Fivewin. Una vez visto esto, borra la **T Negra** y cambia la **T blanca** por el texto “**Nombre**”.

La primera columna de la caja de herramientas son controles de edición. La segunda y tercera son controles standard de windows. La cuarta columna son controles de Borlan que necesitan cargar el fichero BWCC.DLL mediante la instrucción:

```
local hBorlan := loadLibrary('\\....\BWCC.DLL')
```

También puedes poner el nombre sin indicar el directorio, pero entonces tendrás que copiar la librería donde resida el ejecutable:

```
local hBorlan := loadLibrary('BWCC.DLL')
```

Otro truco consiste en cambiar el nombre a la librería, y ponerla junto al resto con la aplicación, de modo que quedaría algo similar a:

```
local hBorlan := loadLibrary('GESTION.DLL')
```

Puedes jugar un poco con el recurso, cambiándolo de posición, cambiando el font, etc. Con estos cambios si no haces nada en el programa, saldrá el texto con fondo blanco. Para poder tenerlo con fondo gris debes reconocerlo dentro del PRG. También desde el PRG puedes cambiar colores, font, etc. etc. Veamos como modificar el programa para tener dos recursos: la ventana de DIALOGO y el texto fijo.

Con respecto al programa del último ejemplo haz ahora la siguiente modificación:

(se representa solo la función dialogo. El resto sigue igual)

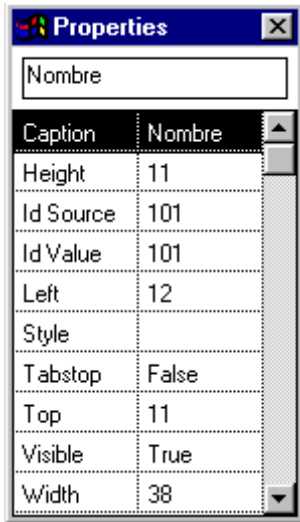
```

/*****
DIALOGO Visualiza una caja de dialogo
*/
function dialogo
#define DIALOGO 1
local oDlg,oSay1

DEFINE DIALOG oDlg RESOURCE DIALOGO OF oWnd
REDEFINE SAY oSay1 ; // redefinir el recurso type SAY
ID 101 OF oDlg // identificado dentro workshop como 101
ACTIVATE DIALOG oDlg CENTERED
return NIL

```

En el workshop debes cambiar el identificador. Entra en él y haz doble clic en el texto. Aparecerá una ventana que tiene en **Control ID** el número -1. Cámbialo por 101. De ahora en adelante (mientras no se cambie explícitamente) este recurso tiene el número 101. Cuando tenía el número -1, no podía ser reconocido desde el programa. Ahora ya puede ser REDEFINIDO. Por este simple hecho, ya aparece con fondo gris. Si miramos en el resto de las propiedades vemos por orden desde arriba:



- **Caption.**- Es el texto que aparecerá en la ventana.
- **Height.**- Son los píxel de alto.
- **Id Source.**- Es la denominación dentro del fuente RC.
- **Id Value.**- Es el número por el que debemos referirnos dentro de nuestros programas PRG's.
- **Left.**- Es la posición dentro de la ventana del lado izquierdo del texto.
- **Style.**- En blanco
- **Tabstop.**- Indica si al pulsar el tabulador, este recurso va a tomar el foco o no. Como es un texto, este valor es Falso.

- **Top.**- Es la posición dentro de la ventana del lado superior del texto
- **Visible.**- Es un valor lógico que indica si será visible en la ventana o no. Normalmente será True (cierto).
- **Width.**- Es el ancho del texto.

Los valores de posición y tamaño solo sirven para ajuste finos. Normalmente todo se hace con el ratón y con las herramientas de alineación. Esta es la principal ventaja de usar un sistema de ayuda visual.

Ahora vamos a poner el equivalente a un GET. Entra al WS y en la ventana de herramientas elige la 5-B que parece una doble T. El cursor cambia a la pequeña cruz que sirve de puntero y una doble T blanca. Señala un rectángulo enfrente de Nombre, como para poner el GET. Con muy poca práctica podrás hacerlo fácilmente. Observa las propiedades.

- Ahora en **Caption** no hay nada. Si pones algo será un texto predefinido que puede ser editado, pero esto no tiene sentido puesto que este recurso normalmente se asigna a una variable que tendrá un texto (o espacio) y que recogerá el texto editado.
- En **Id Value** el nuevo número será 102. El programa incrementa en uno el número mas alto dentro del contenedor, es decir dentro de la ventana de DIALOGO. Toma nota por que es lo que necesitas para redefinir el recurso.
- En **TabStop** ahora hay **True**. Es lógico puesto que el cursor se debe parar en él para que el operador pueda editar el texto. El resto de las propiedades son similares con el recurso anterior.

El programa debe quedar como sigue:

```

/*****
DIALOGO Visualiza una caja de dialogo
*/
function dialogo
#define DIALOGO 1
local cTexto := 'Hola Mundo.....'
local oDlg,oSay1,oGet1

DEFINE DIALOG oDlg RESOURCE DIALOGO OF oWnd
REDEFINE SAY oSay1 ;
    ID 101 OF oDlg

```

```

REDEFINE GET oGet VAR cTexto;
  ID 102 OF oDlg ;
  MESSAGE 'Pregunta sobre el nombre' ;
  COLOR nRGB(0,0,0),nRGB(0,255,0)

```

```
Return NIL
```

A este programa le hemos añadido dos líneas con conceptos nuevos. **MESSAGE** que indicará un texto en la barra de tareas de la ventana general y **COLOR** que nos presentará el texto dentro del GET en negro con el fondo en verde. (no te preocupes ahora de la función **nRGB()** si no la conoces)

Ya con estos dos recursos (tres si tenemos en cuenta el DIALOGO), puedes hacer muchas cosas y empezar a dar rienda suelta a tu imaginación. Ahora iremos todavía mas deprisa viendo los recursos que faltan. Luego volveremos a ellos pero desde el punto de vista de la programación y no como ahora que solo lo vemos desde el punto de vista del propio workshop.

Pongamos ahora un botón de pulsar. Este tipo de botones al ser pulsados llaman a una función que será ejecutada. Son los clásicos botones de OK, Ayuda, Imprimir, etc. Entra al WS a nuestro proyecto. En los botones pasa algo parecido a lo que ocurre con la T blanca y la T negra. Uno de ellos, el 4-D, es solo de Borlan. Selecciona la herramienta 1-B y señala un rectángulo que representará al botón. Aparecerá un botón con el texto Button escrito en él. Las propiedades son las normalmente esperadas. Dentro de **Id Value**, tendrá el número siguiente, es decir, el 103. Cambia el **Caption** y pon 'Hola'. El programa quedará:

```

/*****
DIALOGO Visualiza una caja de dialogo
*/
function dialogo
#define DIALOGO 1
local cTexto := 'Hola Mundo.....'
local oDlg,oSay1,oGet1,oBut1

DEFINE DIALOG oDlg RESOURCE DIALOGO OF ownd
REDEFINE SAY oSay1 ;
  ID 101 OF oDlg
REDEFINE GET oGet VAR cTexto;
  ID 102 OF oDlg ;
  MESSAGE 'Pregunta sobre el nombre' ;
  COLOR nRGB(0,0,0),nRGB(0,255,0)
REDEFINE BUTTON oBut1 ;
  ID 103 OF oDlg ;
  MESSAGE 'Boton numero 103' ;
  ACTION msginfo(cTexto)
ACTIVATE DIALOG oDlg CENTERED
return NIL

```

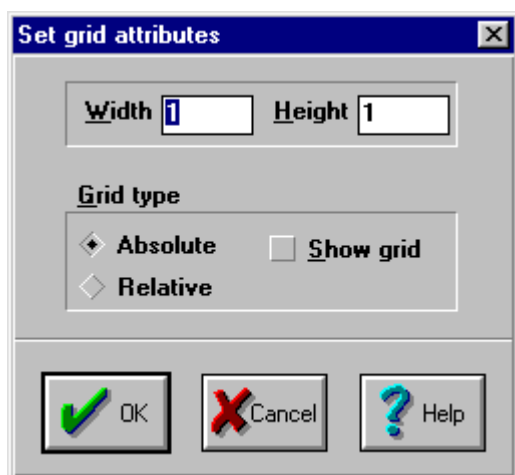
En el programa, si pulsas el botón 'Hola', te aparecerá el texto que haya en el GET.

Ahora va siendo el momento de que aprendas el tema de redimensionamiento y alineación. Inicialmente, la ventana de DIALOGO es de un tamaño predefinido por el programa workshop. Desde el programa PRG podemos dar nuevos valores de tamaño y posición, así como en tiempo de ejecución, pero ahora solo veremos las modificaciones

en tiempo de diseño. Nuestra ventana DIALOGO se esta quedando pequeña para todos los recursos que queremos introducir en ella. Para agrandarla simplemente pulsamos una vez en la banda azul del título **DIALOGO**. Aparecen las propiedades y se remarca la ventana. Podemos hacer dos cosas. Ponernos con el ratón en los bordes y ajustar el tamaño (esto es rápido, pero no preciso), o editar el valor de Alto y Ancho en las propiedades (esto es preciso, pero no rápido). Pon Alto= 200 y Ancho=400. Si no te cabe todo en la ventana del workshop, maximiza todo lo que sea necesario y redistribuye las ventanas de Alineación, Herramientas y Propiedades. Observa al dimensionarla con el ratón que en la parte inferior va poniendo simultáneamente los valores **X** e **Y**.

Con dos recursos únicamente es fácil alinear la palabra Nombre con el espacio GET correspondiente. Pero imagínate una ficha que tenga nombre, dirección, población, país, etc. etc. Lo más probable es que algún texto o get se salga de su sitio. Para ayudarnos en esto tenemos dos herramientas: La regilla y la Alineación. Veamos primero que es la regilla.

Con la ventana redimensionada a 200 x 400 y con workshop maximizado, selecciona la opción **Align** del menú superior. Te aparecerá un submenú donde solo hay dos opciones señaladas: **SIZE** y **Grid**. Pulsa en **Grid** y aparecerá esta ventana:



Nos indica que la rejilla NO se visualice, pero que de hacerse será de 1 pixel de lado. La práctica está alrededor de 5 pixel, pero esto depende del monitor, del tipo de font, de tus gustos personales, etc. Por ahora pon **Width 5** y **Height 5**. También activa **Show grid**. Por último pulsa **OK**. Ahora verás la ventana DIALOGO con una rejilla. Recuerda que ésta solo se ve en tiempo de diseño. El usuario del programa NO la verá.

Si tratas de mover alguno de los recursos, por ejemplo el GET, verás que cambia a saltitos, es decir se sincroniza siempre con la rejilla. De esta forma es fácil hacer que varios GET coincidan, etc. No solamente va a saltos la posición, sino también el dimensionado. Es decir el GET solo podrá tener de alto 5,10,15,20,etc. píxel, NO podrá tener 7 ni 13, etc. En mi opinión este sistema es bastante práctico.

Ahora veremos el sistema de alineamiento. Suponte que tienes varios SAY y GET y que quieres ponerlos todos con el lado izquierdo en una línea vertical. Haz lo siguiente:

- En la ventana de alineación tienes dos grupos de cuatro flechas:



- La parte superior es la alineación vertical (ajusta desplazando a izquierda y derecha para conseguir una línea vertical). La parte inferior es la alineación horizontal (ajusta desplazando arriba y abajo para conseguir una línea horizontal)
- Pon los tres recursos actuales (SAY, GET y Button) uno encima de otro pero sin preocuparte demasiado.
- Pincha en el primer elemento a alinear.

- Con la tecla de Shif pulsada, pincha en el segundo, en el tercero, etc.
- Veras como se va creando un cuadro que los cubre a todos. También puedes crear un recuadro con el ratón que incluya a todos los elementos.
- Ahora pincha el la ventana de alineación en la flecha de la izquierda. Verás que todos se alinean justificando por la izquierda. Haz lo mismo con la doble flecha y se alinearán centrados y con la flecha de la derecha se alinean por la derecha.
- Si pulsas en la doble flecha que tiene dos líneas verticales, alinea el conjunto, sin modificar la posición relativa entre los elementos, en el centro de la ventana DIALOGO.
- Estando en posición uno encima de otro, de las cuatro flechas de abajo, solo debes pulsar las flechas arriba-abajo que tienen las dos líneas pequeñas. El resto de las flechas superpone los componentes.
- Ahora cambia cada recurso y ponlos uno al lado de otro, mas o menos en línea.
- Pulsa en las flechas verticales y tu mismo saca las conclusiones pertinentes.



Vamos a seguir con mas herramientas. Veamos los **Check button**. Como sabes son pequeños botones que solo admiten dos estados: marcado y no marcado o si lo prefieres, Cierto/Falso. Elige el elemento 3-C. Llévelo a la ventana DIALOGO y marca un rectángulo con él. Verá que se compone de dos partes: El cuadrito donde pone la marca y un texto asociado. El texto puede cambiarlo de la forma habitual por medio de **Caption**. Observa el identificador para luego ponerlo en el PRG. (Los elementos 5-D y 6-D son de Borlan y Fivewin no los reconoce). En el programa debes poner las líneas:



```
REDEFINE CHECKBOX oCkb ;
ID 104 OF oDlg ;
MESSAGE 'Check box'
```

Recuerda declarar la variable oCkb como local)

Ahora veremos los **Radio button**. No te voy a explicar que tipo de elemento es este, se supone que ya lo sabes. Por tanto sabes que se necesita al menos dos para poder alternar entre ellos. Elige el elemento 1-C. y ponlo sobre el DIALOGO. Tendrás que poner al menos dos. Pon tres. Para este caso lo mejor es poner uno y copiarlo con el Copiar y Pegar [Ct-C], [Ct-V]. También será casi necesario la alineación.



Sal del WS y modifica el programa con las siguientes líneas:

```
REDEFINE RADIO oRad ;
ID 105,106,107 OF oDlg ;
MESSAGE 'Radio Button 105, 106 y 107'
```

Observa que en ID se han puesto los identificadores de los tres Radio Button. Esto permite relacionarlos de forma que si se activa uno se desactivan los otros tres. Como ahora no es el momento de analizar el programa, lo dejamos para otra ocasión. Si tu quieres practicar, puede hacer varios grupos, etc.

Pongamos ahora dos barras de desplazamiento una vertical y otra horizontal. Selecciona los elementos 2-B y 2-C. No te voy a cansar con la mecánica, solo indicarte dos detalles. Al principio, el programa deshabilita la parada del tabulador. Si quieres puedes activarla poniendo Tabstop a True. Puesto que este tipo de recurso esta relacionado con datos, por ejemplo puntero de matrices, número de registro en una base de datos, contador en un proceso, etc, en el momento de ejecución de nuestro pequeño programa, la marca de posición de la barra siempre tenderá a ir al principio. Veamos ahora la parte del programa que debes añadir:

```
REDEFINE SCROLLBAR oScrH ;
    ID 108 OF oDlg ;
    MESSAGE 'Barra de scroll horizontal'
REDEFINE SCROLLBAR oScrV ;
    ID 109 OF oDlg ;
    MESSAGE 'Barra de scroll vertical'
```

Para no hacer monótono el seguimiento del tutorial, vamos a ver algunas cosas relacionadas con la ventana de herramientas y luego seguimos con algunos elementos más. Habrás observado que cuando ejecutamos el programa, al pulsar Intro o Tabulador, el “foco” lo toman los recursos por el orden de creación, es decir, si has seguido el proceso, sera : el GET, el Button, el Checkbox, el Radibutton, la barra horizontal y luego la vertical (estas últimas si tienes el Tabstop a cierto). Supongamos ahora que queremos cambiar el orden de tabulador. Para ello disponemos de la herramienta señalada con **1,2** es decir, la 4-A.

- Con la ventana de DIALOGO en la pantalla del WS pincha en esta herramienta (4-A).
- Cambiará el fondo de la ventana y el cursor se pondrá con los números 1,2.
- Cada elemento que hemos ido poniendo se representará con un rectángulo en cuyo interior hay un número. Este número indica el orden de tabulación. Como puedes ver el texto fijo también tiene número aunque su valor es irrelevante puesto que el tabulador no se detiene en él.
- Pincha con el cursor **1,2** en cualquier elemento, por ejemplo en Button.
- Verás que ahora toma el número 1, y todos los demás elementos se renumeran automáticamente.
- Sigue pulsando en el resto de los recursos hasta que estén todos numerados.
- Vuelve a compilar el programa y observa el efecto.



Si te equivocas en el orden, debes empezar de nuevo, desactivando la selección y volviendo a activarla (pulsas en la flecha (1-A) y luego de nuevo en **1,2** (4-A))

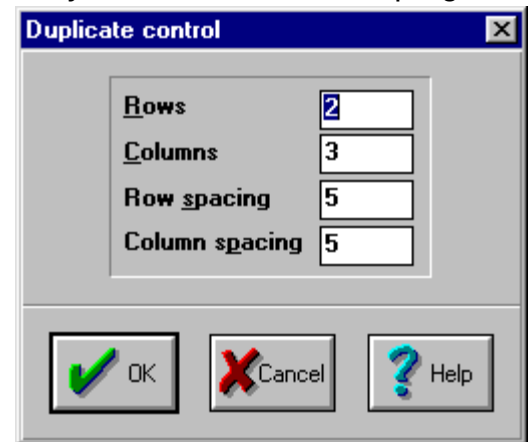
Si durante el diseño quieres ver como va quedando tu ventana de DIALOGO, pulsas en el apartado que pone **Test** (5-A). Se quitará la rejilla y aparecerá todo igual a como se verá en el programa. Hay que tener en cuenta que presenta los valores por defecto respecto a font, color, etc. Si luego tu lo cambias en los REDEFINE, naturalmente el programa ahora no lo sabe.

Para quitar el test, pulsas **[Alt-F4]** o seleccionas de nuevo la flecha (1-A)

Otro elemento en la ventana de herramientas es la duplicación en forma de matriz. (6-A). Veamos como funciona:

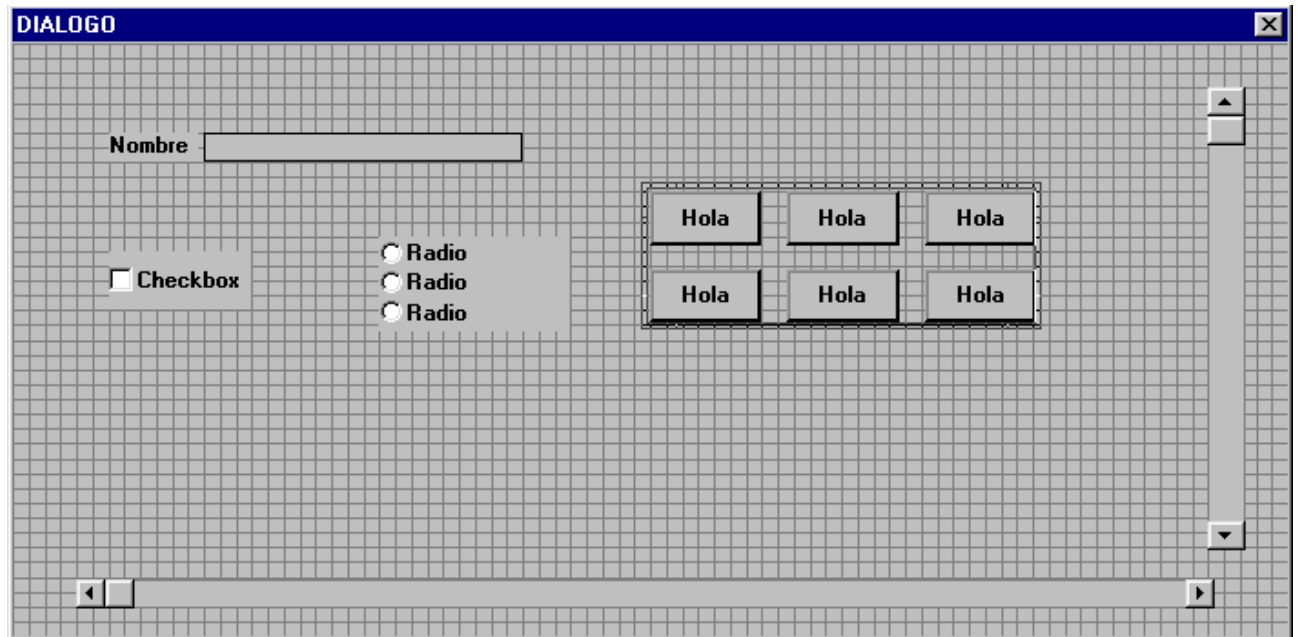


- Señala un elemento con el cursor en forma de flecha, por ejemplo el Botón. Ahora pincha en 6-A. Aparecerá un diálogo donde pregunta cuantas filas y columnas. También pregunta por el espaciado entre filas y el espaciado entre columnas. Pon 2 filas, 3 columnas y 5 pixel de separación.
- Pulsa en **OK**
- Aparecerá el Botón como se indica en la siguiente figura de la ventana de



DIALOGO.

- Cada uno de los nuevos botones tiene su propio número de identidad, pero todos los demás parámetros son iguales al primero (excepto la posición, naturalmente)

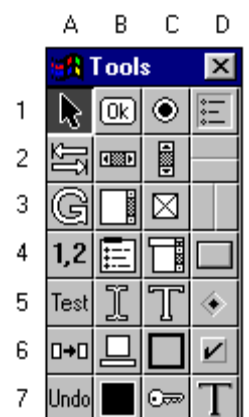


Una vez que has visto el significado de la copia, borra los botones nuevos para no liar la ventana con tantos elementos. Déjala como estaba antes de la copia.

Para deshacer un cambio puedes pulsar [**Alt-Retroceder**] o hacer click en la opción 7-A. Para rehacer cambios puedes pulsar [**Shif-Alt-Retroceder**].

Sigamos con los recursos. Ahora vamos a ver como se hace un List box.

Pinchemos en la herramienta 3-B y marquemos un rectángulo



en la caja de DIALOGO. Debe ser lo suficientemente grande como para poder poner cinco líneas como luego veremos.

En el programa añadamos las siguientes líneas:

```
local oLbx,vLisBox
...
REDEFINE LISTBOX oLbx ;
  VAR vLisBox ;
  ITEMS {'Africa','America','Asia','Europa','Oceania'} ;
  ID 110 OF oDLg ;
  MESSAGE 'List box'
```

Si quieres puedes modificar la cláusula del Boton hecho anteriormente y poner **msginfo(vLisBox)**. Una vez seleccionado una línea de la lista, pulsa el botón y veras el la línea seleccionada.

Muy parecida al control List Box es **Combo Box**. La diferencia ya la pudimos ver la introducción. Ahora solo veremos como crearla con WS. Pincha en la herramienta 4-C y señala un rectángulo que sea el doble de alto que la anterior List box. En el programa pon las siguientes líneas:

```
local oCbx,vComBox
...
REDEFINE COMBOBOX oCbx ;
  VAR vComBox ;
  ITEMS {'Africa','America','Asia','Europa','Oceania'} ;
  ID 111 OF oDLg ;
  MESSAGE 'Combo box'
```

Hagamos un repaso a la paleta de controles. De arriba abajo y de izquierda a derecha, los controles de la paleta de herramientas son los siguientes:

Columna A:



Se utiliza para volver el cursor a su situación normal, deshaciendo cualquier selección previa.



¿?



Agrupar controles.



Reordena las paradas del tabulador señalando en cada recurso.



Prueba la ventana en curso. Para salir de la prueba, pulsar **[Alt-F4]**



Duplica un recurso de la ventana, en forma de matriz.



Deshace la operación anterior. El máximo de operaciones esta predeterminado.

Columna **B**:

Control de tipo **Button**. El texto puede ser editado.



Control de tipo **Barra de scroll horizontal**.



Control de tipo **List box**.



Control de tipo **GET**.

Columna **C**:

Control de tipo **Radio Button**. Normalmente se pondrán dos o más.



Control de tipo **Barra de scroll vertical**.



Control de tipo **Check box**.



Control de tipo **Combo box**



Control de tipo **SAY**.



¿?



Control de tipo **externo**. El programador debe indicar que control ha de ponerse.

Columna **D**: [\(Necesitan BWCC.DLL\)](#).

Control Borlan.



Dibujo de una línea horizontal.



Dibujo de una línea vertical.



Control de tipo Button.



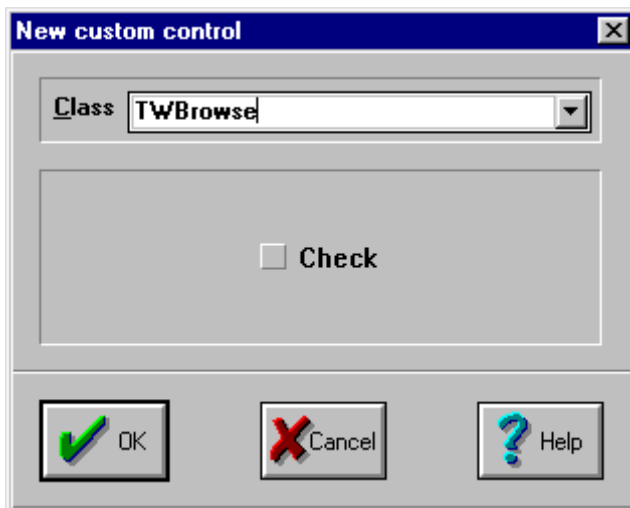
Control de tipo Check box.



Control de tipo Check box.



Control de tipo **SAY**.

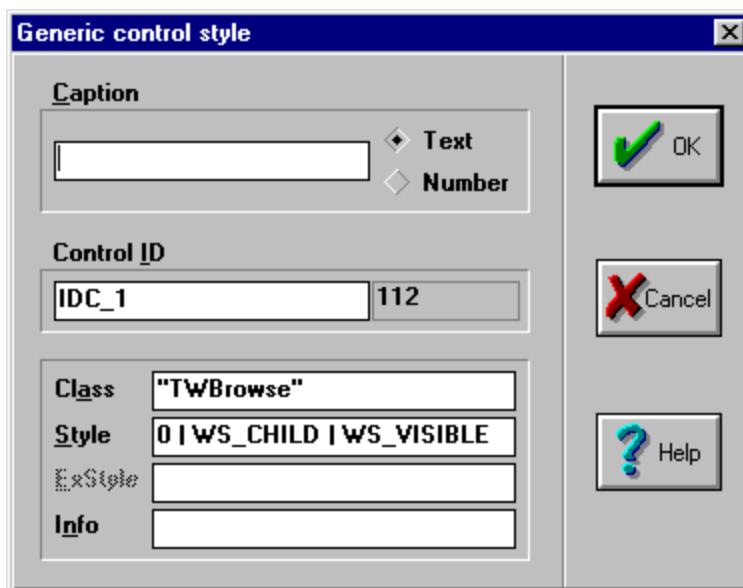


Vamos a ver ahora como se ponen controles externos. El ejemplo clásico, es un recurso equivalente a un Tbrowse de clipper.

Lo primero que hay que hacer con el WS es preparar la ventana DIALOGO para que quepa el browse que vamos a diseñar. Una vez organizados los recursos, pulse en la paleta de herramientas, el señalado con una llave  (7-C). Automáticamente aparece una caja de diálogo como la mostrada a la izquierda. El texto en **Class** será uno predefinido. Debe cambiarlo a mano escribiendo TWBrowse que es como

se llamara el objeto creado desde Fivewin. Pulse **OK** y aparecerá el cursor para que selecciones un rectángulo donde deberá ir el control que estamos creando. Haga un sitio bastante grande como para caber un browse con código, nombre y dirección. Al terminar aparece el rectángulo recién creado. Haga doble clic en él y aparecerá la clásica caja de diálogo del workshop.

La parte importante es el GET que tiene la palabra **Style**. Inicialmente tiene los



valores que se indican en la imagen. Tienes que añadir los siguientes estilos (no te preocupes por el sitio puesto que hará un scroll horizontal)

WS_VSCROLL

WS_HSCROLL

WS_BORDER

WS_TABSTOP

La separación entre estilos debe ser la barra vertical [**AltGr-1**]. Pulsa **OK** y deja el cuadro como está. El resto hay que hacerlo desde el programa.

Antes de seguir con el PRG hazte una base de datos que tenga por ejemplo, **CODIGO, NOMBRE, DIRECCIÓN** (o cualquier otra cosa que se te ocurra, como artículos, etc.)

En el programa, en la parte de la función raíz, abre la base de datos, por ejemplo con USE CLIENTES. Ya en la función **dialogo()** que nos esta sirviendo de patrón, añade las líneas:

```
local oTbr
....
REDEFINE LISTBOX oTbr ;
  FIELDS str(clientes->CODIGO),clientes->NOMBRE,clientes->DIRECCIÓN ;
  HEAD 'Codigo','Nombre','Direccion' ;
  SIZES 70,150 ;
  ID 112 OF oDlg
```

Ahora ejecuta el programa y verás un browse con las barras horizontal y vertical, etc. Si quieres ajustar las columnas cambia los valores de la línea **SIZE 70,150**. Este recurso tiene muchas variaciones en el programa pero ahora no es el momento de verlas.

Otro ejemplo de controles externos es la clase Tbitmap. Por medio de ella podemos visualizar ficheros BMP en una parte de la ventana de DIALOGO. Para ello debemos limpiar la función **dialogo()** de los REDEFINES que hemos ido poniendo excepto el de la propia ventana DIALOGO. Con el WS, borra todos los recursos que hay dentro de la ventana. Es como volver a la pantalla de la página @#@.

Pulsa en la paleta de herramientas en la llave . Ahora escribe en Class la palabra Tbitmap. Señala un rectángulo bastante grande. Ya tenemos el sitio que ocupara nuestro BMP. Ahora ve al programa. Añade las líneas:

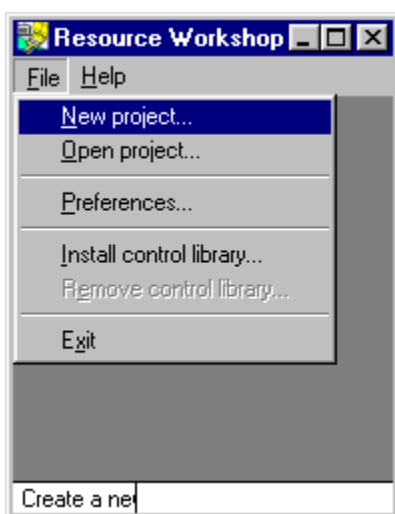
```
Local oBtm
....
REDEFINE BITMAP oBtm ;
FILE '\windows\nubes.bmp' ;
ADJUST ;
ID 101 OF oDlg
```

La cláusula **ADJUST** sirve para ajustar la imagen al sitio que hemos predefinido en el WS en el momento de diseño. Si se quiere utilizar las barras de scroll, debes hacer dos cosas. En el PRG sustituir ADJUST por **SCROLL** y con el WS modificar el estilo del recurso creado añadiendo **WS_VSCROLL** y **WS_HSCROLL**. Como se hizo en el recurso de más arriba. Puedes probar con otros BMP como fotos de personas, etc. etc.

Recursos en DLL

Programa WORKSHOP

Al cargar el programa, después de una pantalla de bienvenida, aparece la siguiente caja de diálogo.



En el submenú **File** aparece la siguiente selección.

New project.- Se utiliza para hacer nuevos proyectos. Un proyecto es la colección en un solo fichero de todos los recursos a utilizar por una aplicación (normalmente suele corresponder a un ejecutable o DLL completa.)

Open Project.- Se utiliza para “abrir” un proyecto ya creado en otra sesión anterior.

Preferences... .- Determina algunos parámetros que se quieren utilizar por defecto. Luego lo veremos ampliado.

Install control library.-

Exit.- Como siempre, se utiliza para abandonar el WS. Si ha habido cambios en el proyecto (o era nuevo), pregunta si se quieren guardar los cambios. Normalmente

contestaremos que **Yes**.

Veamos ahora algunas de estas opciones:

New project.- Presenta la pantalla de la derecha. En ella se nos esta pidiendo que definimos el tipo de proyecto en el que se basarán nuestros recursos. Este puede ser:

.RC.- Basará el proyecto en un fichero script que será compilado antes de incluirlo en el ejecutable. Es el mas normal como ahora veremos.

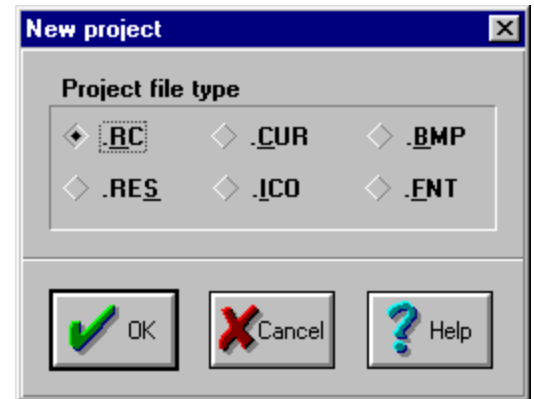
.RES.- Realizará el proyecto en un fichero fuente binario. Este tipo de fichero contiene uno o mas recursos ya compilados.

.CUR.- Basa el proyecto en un fichero de cursor. Este tipo de fichero solo puede tener una imagen.

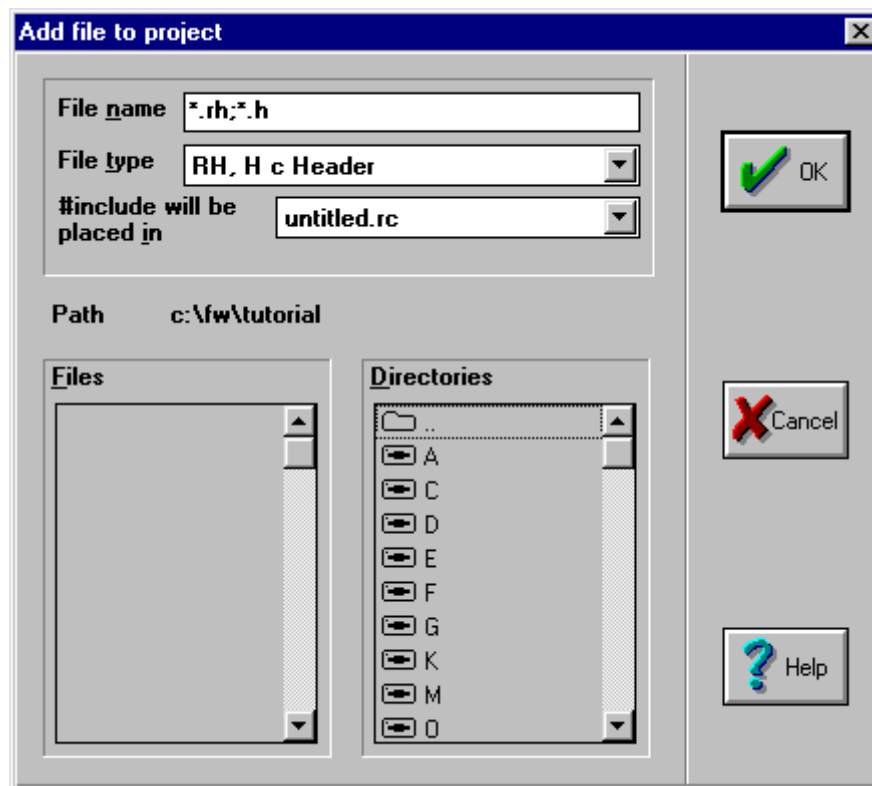
.ICO.- El proyecto sera un fichero de iconos. Este tipo de ficheros puede tener varios iconos.

.BMP.- El proyecto será una imagen de tipo bitmap. Un fichero de este tipo solo podrá tener una imagen.

.FNT.- El proyecto será un font de letras. Un fichero de este tipo puede tener mas de un font.



New project - .RC.- Si elegimos esta opción aparecerá la siguiente pantalla.



Donde cada apartado significa lo siguiente:

The Add File to Project dialog box is where you specify the name of an external file to add to your project. Display this dialog box with File|Add File to Project.

Note: You cannot mix Win32 and non-Win32 files in a project. If you're working with a Win32 project (that is, if you've turned on the Win32 radio button - one of the Target Windows Version radio buttons in the Preferences dialog box), you can't add a non-Win32 file to your project. To add existing Windows 3.x .RES or .EXE files to a Win32 project, first save the Windows 3.x files as .RC files. You can then add the .RC files to your project, since .RC files are version independent.

File Name Es el nombre del fichero que estas añadiendo a tu actual proyecto. Normalment se utiliza para incluir el fichero winapi.ch.

File Type Seleccionar en la lista desplegable el tipo de fichero que vamos a añadir. En caso de winapi.ch, será un tipo **RH, H c Header** (es decir, lo que viene por defecto).

RCINCLUDE will be Placed In Indicar donde sera puesto el fichero incluido. Si es un Nuevo proyecto, el WS presenta **untitled.rc**. Dejarlo de momento. Al salir del programa y dar nombre definitivo, también cambia este dato.

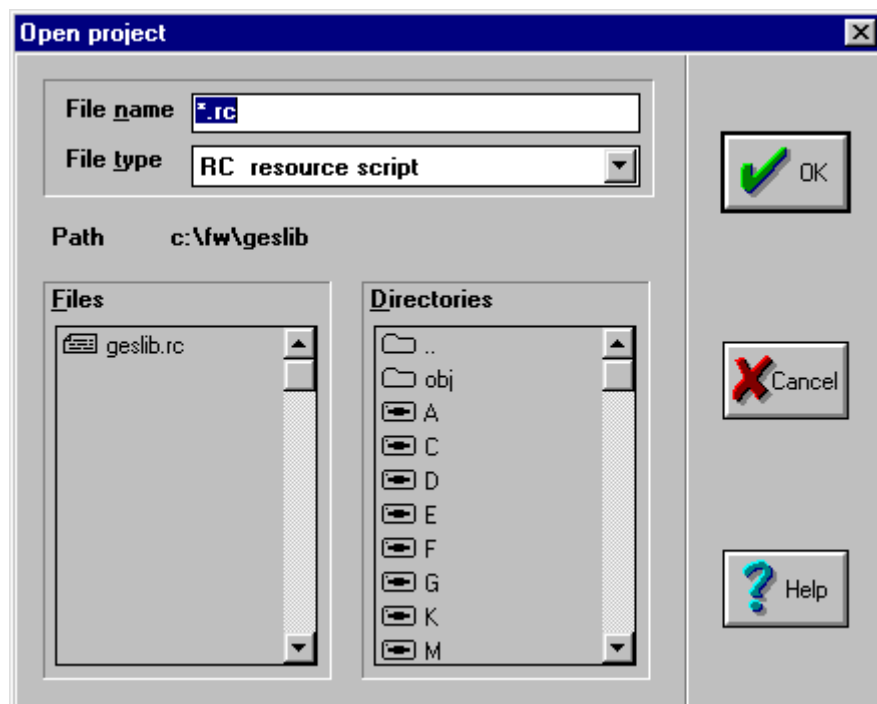
Path Indica (no se puede cambiar directamente) el directorio o senda donde estará el nuevo proyecto..

Files Aparecerán los proyecto actuales. Si es nuevo el que hacemos, no prestar atención a este dato..

Directories Indicar la senda donde se grabará el nuevo proyecto. Puede utilizarse tanto la opción .. para subir de nivel, como cambiar de directorio o de unidad.

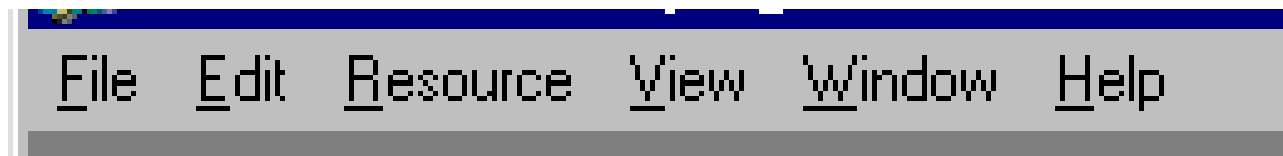
El resto es similar a cuando abrimos un proyecto existente, de modo que veamos ahora esta opción.

Open Project.- La diferencia de esta pantalla si entramos por New project u Open project, es pequeña como puede apreciarse ahora:



No pregunta por el include. Presenta una relación de proyecto en el último directorio grabado. De esta relación hay que elegir uno. Una vez elegido seguir los pasos comunes para la creación y para la continuación de proyectos.

Una vez pasadas estas ventanas, el menú principal del WS cambia con las opciones y contenidos siguientes:



File:

New project.- Cierra el proyecto actual y abre uno nuevo.

Open project.- Cierra el proyecto actual y abre uno existente.

Save project.- Guarda el proyecto actual. Es muy útil para ir haciendo copias de seguridad de vez en cuando. También se utiliza para guardar el proyecto y compilar nuestro programa sin tener que salir del WS.

Save file as....- Se utiliza para guardar el proyecto con otro nombre, como una copia, o para hacer otro proyecto basado en el actual.

Close all.- Cierra todo y se queda a la espera de New u Open.

Add tp project... Sirve para añadir, por ejemplo iconos al proyecto actual.

Remove from project.- Sirve para quitar un control, diálogo, icono, etc. del proyecto

Preferences....- Se utiliza para poner algunos funcionamientos del WS por defecto.

Install control library.- ¿?

Remove control library.- ¿?

Exit.- Se utiliza para salir del WS. Si ha habido cambios en el proyecto, pregunta si se graba antes de salir.

Edit:

Undo: Permite deshacer la última acción efectuada.

Redo: Permite rehacer el último "undo" hecho

Cortar: Permite retirar un dato, almacenándolo en memoria.

Copia: Permite copiar en memoria un dato.

Pegar: Permite jalar de memoria el dato previamente almacenado.

Deletear: Borra un dato.

Deletear item: retira un dato.

Duplicar: Duplica un dato.

Seleccionar todo: Permite seleccionar todos los datos en el recurso activo.

Nota que al referirnos a un dato, lo hacemos en forma global. Y es que un dato puede ser, segun el recurso en que estemos trabajando, un bitmap, un control, texto, etc.

Recurso:

Nuevo: Permite definir un nuevo tipo de recurso. Una vez que seleccionamos la opción, aparecerán los tipos de recurso que podemos definir.

Editar: Permite editar un recurso. (ta cual si presiónamos Intro).

Editar como texto: Cada recurso se puede almacenar como un dato binario (DLL), o como una cadena de texto (RC).

Ver: Permite ver el recurso.

Grabar recurso como: Permite grabar el recurso como un archivo. Recordemos que un DLL contiene distintos tipos de archivos, por lo que también puede grabar distintos tipos de archivos.

Renombrar: permite cambiar de nombre al recurso con el que estamos trabajando.

Opciones de memoria: Modo de cargar/descargar de memoria un recurso. No tocar.

Lenguaje: Permite seleccionar el lenguaje que usará al recurso DLL.

Mover: Permite mover un recurso a otro.

Identificadores: Permite definir identificadores.

Ver:

Por tipo: Permite ver los recursos por tipo (Bitmaps, Diálogos, etc)

Por archivo: Permite ver los recursos segun el archivo que los contiene.

Mostrar item: Mostrara los detalles de cada recurso.

Mostrar tipos sin usar: Muestra todos los tipos de recursos, aun los que no se esten usando

Prevista vertical: Permite dividir ver en forma vertical la pantalla, una con la prevista.

Prevista Horizontal: Permite dividir ver en forma vertical la pantalla, una con la prevista.

No mostrar Prevista: No muestra una prevista

Ventana

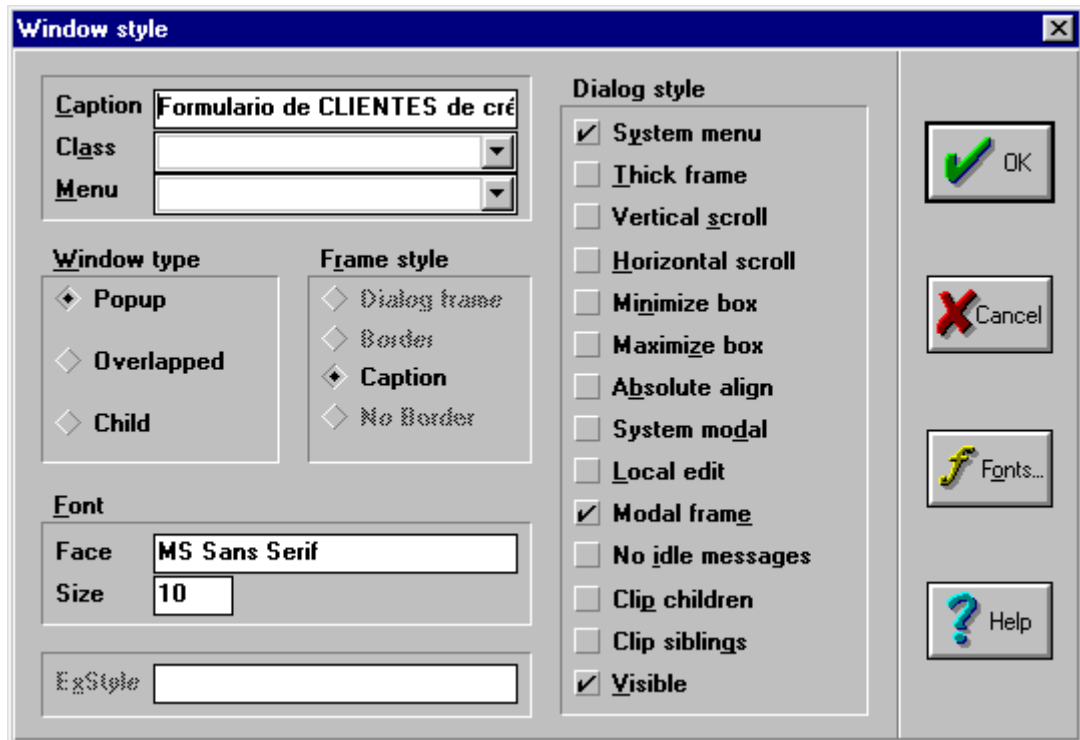
Pegadas: Permite ver las ventanas calzando (ajustandose a) la pantalla.

Cascada: Permite ver las ventanas una detras de otra, en forma de cascada.

Ayuda

Muestra los ítems de ayuda del Resource WorkShop.

Windows style



Desde esta ventana se ponen los parámetros que por defecto tendrá nuestro recursos. Posteriormente, en el programa (PRG) se pueden modificar algunas de estas características.

Use the Window Style dialog box to specify the details of your dialog box. To open this dialog box, select the dialog box and press Enter or double-click the dialog box's title bar or outer edge.

Caption.-The Caption input box is where you enter the caption of the dialog box. Tiene que estar activada **Caption** en **Frame style**. Si se pone algo, el resto de las opciones de **Frame style** queda deshabilitadas.

Class.-The Class input box is where you assign a custom class to your dialog box.

A custom class designation lets your application process dialog box messages with your own windows procedures instead of using Windows procedures.

If you enter `bordlg` in this input box, your dialog box is implemented as a Borland-style dialog box, like the ones that are used in Resource Workshop. Use the Borland custom control tools to put Borland-style controls in your dialog box.

Using the Borland custom control class improves the appearance of your dialog window by painting the background with a brush that varies according to the target display device. For screens of VGA and higher resolution, the background is a fine grid of perpendicular white lines, giving the effect of "chiseled steel." For EGA and monochrome screens, the background is white.

Choose `Bordlg_Gray` to achieve the same effect as the "bordlg" class, except the background is solid gray instead of chiseled. Using this background can prevent flickering with some screen drivers and monitors.

The Borland custom control class also optimizes the drawing of dialog boxes by calling the custom control routines directly, instead of waiting for Windows to paint the controls.

Menu.-The Menu input box puts a menu in your dialog box.

Just type the menu's resource identifier or resource ID.

The Dialog editor won't display the menu resource exactly as you define it. Instead, it displays a pop-up called "Menu" that includes a single menu item called "Item."

Window Type.-The Window Type radio buttons choose a type for your dialog box.

Popup.- Turn the Popup radio button on if your dialog box is a pop-up window. Because most dialog boxes are pop-ups, this is the default.

Overlapped.-Turn the Overlapped radio button on to define your dialog box as an overlapped pop-up window that can be partially covered by another. Define a dialog box as overlapped only when it's the main window in the application.

Child.-Turn the Child radio button on if the dialog box is a child of the current window.

Iconic Popup.-Turn the Iconic Popup radio button on to define your dialog box as a pop-up window that's originally displayed as an icon.

Frame Style.-The Frame Style radio buttons choose your dialog box's frame style.

Face.-The Face input box is where you enter the typeface for the text in your dialog box.

Size.-The Size input box is where you enter the font's point size.

ExStyle.-The ExStyle input box is where you enter the window's extended style. Choose one or more of the Extended window style constants. You can only enter an extended style if you have selected Win32 as the Target Windows Version in the Preferences dialog box.

Dialog Style.-The Dialog Style check boxes determine what the dialog box looks like and how the user can work with it.

System Menu.-The System Menu check box inserts a system menu on the left side of the title bar. A System menu is also called a Control menu.

- The system menu appears only if you turn on the Caption radio button (one of the Frame Style radio buttons).
- If you turn on the Child radio button (one of the Window Type radio buttons), you see a close box instead of a Control menu.

Thick Frame.-The Thick Frame check box places a thick frame around the dialog box.

- Vertical Scroll.**-The Vertical Scroll check box inserts a vertical scroll bar in the dialog box frame.
- Horizontal Scroll.**-The Horizontal Scroll check box inserts a horizontal scroll bar in the dialog box frame.
- Minimize Box.**-The Minimize Box check box adds a Minimize button to the right side of the dialog box title bar. The minimize box appears only if you turn on the Caption radio button (one of the Frame Style radio buttons).
- Maximize Box.**-The Maximize Box check box adds a Maximize button to the right side of the dialog box title bar. The maximize box appears only if you turn on the Caption radio button (one of the Frame Style radio buttons).
- Absolute Align.**-The Absolute Align check box makes the dialog box coordinates relative to the display screen rather than the parent window.
- System Modal.**-Turn on the System Modal check box if you want to make the dialog box modal. This means that the user can't switch to anything else until the dialog box is closed.
- Local Edit.**-The Local Edit check box allocates any edit text controls to the application's local heap. Turn this check box on if your application uses the EM_SETHANDLE and EM_GETHANDLE messages.
- Modal Frame.**-The Modal Frame check box frames the window with a combination of the dialog frame and the caption styles. This option also lets users move the dialog box.
- No Idle Messages.**-The No Idle Messages check box suppresses sending idle (WM_ENTERIDLE) messages to the application's main window. For this option to have any effect, you must also turn on the System Modal check box.
- Clip Children.**-The Clip Children check box protects the client area of child windows from being drawn on by the dialog box window.
- Clip Siblings.**-The Clip Siblings check box protects the siblings of this window, and restricts drawing to this window. This option is not required for pop-up windows, but can be useful for child dialog windows.
- Visible.**-The Visible check box makes a modeless dialog box visible before the return from CreateDialog. This option has no effect on modal dialog boxes (the usual kind of dialog box). By default, this option is not checked (NOT WS_VISIBLE).
- Fonts button.**-Press the Fonts button to display the Select Font dialog box, where you choose the font for the text in your dialog box.
- Face Name.**-The Face Name drop-down box is where you choose the typeface for the text in your resource.
- Size.**-The Size input box is where you choose the font point size.
- Style.**-The Style check boxes select the font style. The only style that you can select for dialog boxes is Bold.

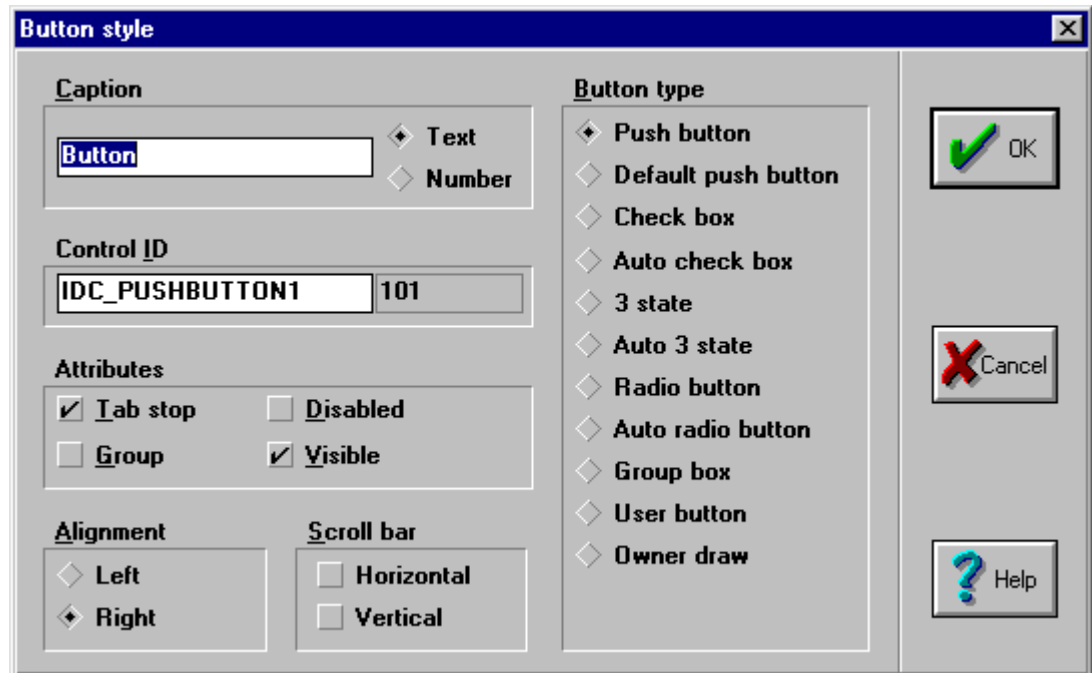
Bold.-The Bold check box makes the font bold.

Italic.-The Italic check box italicizes the font.

Underline.-The Underline check box underlines the font.

Strikeout.-The Strikeout check box strikes out the font.

Button style



Caption.- The Caption input box is where you enter the caption you want displayed with the control. The Caption radio buttons choose how the caption is displayed.

Type the caption you want displayed with the control in the Caption input box. The type of control determines where the caption displays. For example, in a group box, the caption displays at the top left. In a push button, the caption displays inside the button.

Once you enter a caption, turn on one of the Caption radio buttons to choose how the caption displays. (These radio buttons don't apply to Borland custom controls.)

Not all controls display a caption. For example, a list box does not display the text specified in its caption.

Control ID.-The Control ID input box is where you enter the control's identifier.

Enter the control's identifier in the Control ID input box. Control IDs can be a short integer or an integer expression. By convention, static controls that are not modified at run time are assigned a control ID of -1.

If you enter an alphanumeric identifier, Resource Workshop checks to see if a #define or a constant declaration has already been created for that identifier. If not, Resource Workshop asks if you want to create an identifier.

Attributes.- The Attributes check boxes select the control's attributes.

Auto Horizontal.- The Auto Horizontal check box automatically scrolls text to the left when it exceeds the width of the control. This attribute applies only to combo box controls.

Border.- The Border check box draws a border around the control. This attribute applies only to list boxes, edit text, and static controls.

Disabled.- The Disabled check box disables the control by graying it. This prevents the control from responding to user input. This attribute applies to all controls.

Group.- The Group check box identifies the first control in a group.

Turn the Group check box on to indicate the first control within a group. The user can then press the arrow keys to access all controls in the group. This attribute applies to all controls.

You can also use the Set Groups tool to define a group of controls.

Integral Height.- The Integral Height check box sizes the list box at run time so all items in the list are completely displayed (the default). If you need to precisely control the height of the list box, turn this option off. This attribute applies only to combo box controls.

OEM Conversion.- The OEM Conversion check box converts text the user enters to the current OEM character set, then reconverts the text to ANSI. This option is useful in file input boxes because it ensures that any file name entered will be translatable into the OEM character set, which is used by the DOS file system. This attribute applies only to combo box controls.

Sorted.- The Sorted check box automatically sorts items in a list box alphabetically. This attribute applies only to combo box controls.

Tab Stop.- The Tab Stop check box lets the user press Tab to access this control.

Turn the Tab Stop check box on if you want the user to be able to press Tab to access this control. When you set this attribute, all controls in the dialog box are tab stops.

You can also use the Tab Set tool to specify which controls are tab stops.

Vertical Scroll Always.- Turn the Vertical Scroll Always check box on to always place a vertical scroll bar in the list box. The scroll bar is disabled when the list box doesn't contain enough items to scroll. This attribute applies only to combo box controls. This

check box is displayed only when you run Resource Workshop under Windows 3.1.

Vertical Scroll.-The Vertical Scroll check box puts a vertical scroll bar in the list box. This attribute applies only to combo box controls.

Visible.-The Visible check box determines whether the control is visible when the dialog box is first displayed. By default, the option is checked (WS_VISIBLE). If the option is not checked (NOT WS_VISIBLE), the control does not appear. The application can call the ShowWindow function at run time to make the control appear.

Alignment.- Use the Alignment radio buttons to align text for radio buttons, check box buttons, and edit text. For edit text controls, these options apply to multiple-line text.

Left.-Turn the Left radio button on to align text to the left or place text to the left of the button.

Right.-Turn the Right radio button on to align text to the right or place text to the right of the button.

Center.-Turn the Center radio button on to center text. This option applies only to edit text controls.

Scroll Bar.- The Scroll Bar check boxes place scroll bars in your controls. This attribute applies to edit text and list boxes. Buttons can include scroll bars, although they are more commonly found elsewhere in a dialog box.

Horizontal.-The Horizontal check box places a horizontal scroll bar in the control.

Vertical.-The Vertical check box places a vertical scroll bar in the control.

Button Type.-The Button Type radio buttons define the type of buttons to include in your dialog box.

Push Button.-Turn on the Push Button radio button to define a button containing text. When the user clicks the button, a BN_CLICKED message is sent to the parent window.

Default Push Button.-Turn the Default Push Button radio button to define a button containing text. It's identical to a push button, but also includes a bold border indicating that it's the default response when the user presses Enter.

Check Box.-Turn the Check Box radio button on to define a small, rectangular button that can include text to the left or right of the button. The box is marked with an X when it's selected. It's the application's responsibility to check and uncheck the box when the user turns the box on or off.

Auto Check Box.-Turn the Auto Check Box radio button on to define a check box that's identical to a regular check box, but Windows does the checking and unchecking instead of the application.

3-State.-Turn the 3-state radio button on to define a button that's identical to a check box. In addition to being on or off, the button has a third state - it can be grayed to show that its state

is unknown or indeterminate. It's the application's responsibility to check, uncheck, and dim the box.

Auto 3-State.-Turn the Auto Three State radio button on to define a button that's identical to a 3-state button, but Windows does the checking, unchecking, and dimming instead of the application program.

Radio Button.-Turn the Radio Button radio button on to define a small, circular button that has identifying text to the left or right. When it's selected, the circle is filled with a solid dot and all other mutually exclusive choices are turned off. It's the application's responsibility to fill or clear the dot when the user turns the button on or off.

Radio buttons must appear in groups. Usually, a group of radio buttons presents the user with a group of mutually exclusive options.

When a the user clicks a radio button, the button sends a BN_CLICKED message to the parent window.

Auto Radio Button.-Turn the Auto Radio Button radio button on to define a button that's identical to a radio button, but Windows fills or clears the dot instead of the application.

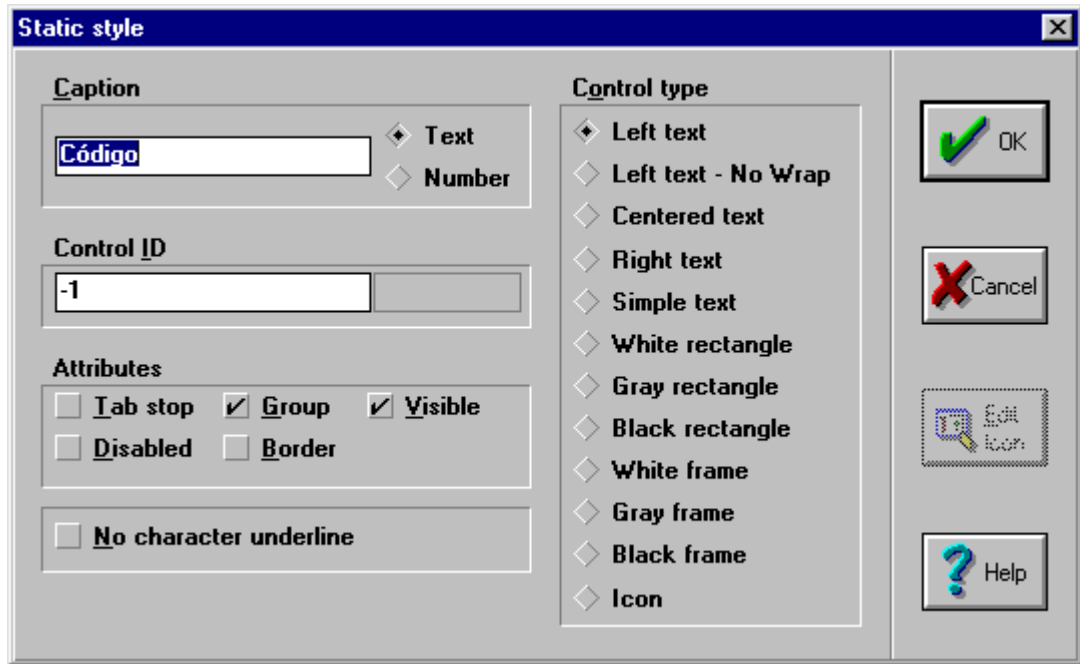
Group Box.- Turn the Group Box radio button on to define a box that groups buttons. You can also include a caption in the upper left corner of the group box.

User Button.- Turn the User Button radio button on to customize buttons for Windows 2.0 compatibility. Try to avoid using user button controls with Windows 3.x. Instead, use owner draw buttons.

Owner Draw.-Turn the Owner Draw radio button on to define a radio button that allows the application to paint the button. When the button needs painting, it sends a WM_DRAWITEM message to its parent.

Static style:

Este estilo corresponde a los texto literales que se ponen en las cajas de diálogo. Para internos, los SAY, pero que aquí tiene muchas más posibilidades. Si hacemos doble clic en un say, aparece la siguiente pantalla.



donde sus distintos controles son:

Caption.-The Caption input box is where you enter the caption you want displayed with the control. The Caption radio buttons choose how the caption is displayed.

Type the caption you want displayed with the control in the Caption input box. The type of control determines where the caption displays. For example, in a group box, the caption displays at the top left. In a push button, the caption displays inside the button.

Once you enter a caption, turn on one of the Caption radio buttons to choose how the caption displays. (These radio buttons don't apply to Borland custom controls.)

Not all controls display a caption. For example, a list box does not display the text specified in its caption.

Control ID.-The Control ID input box is where you enter the control's identifier.

Attributes.- The Attributes check boxes select the control's attributes.

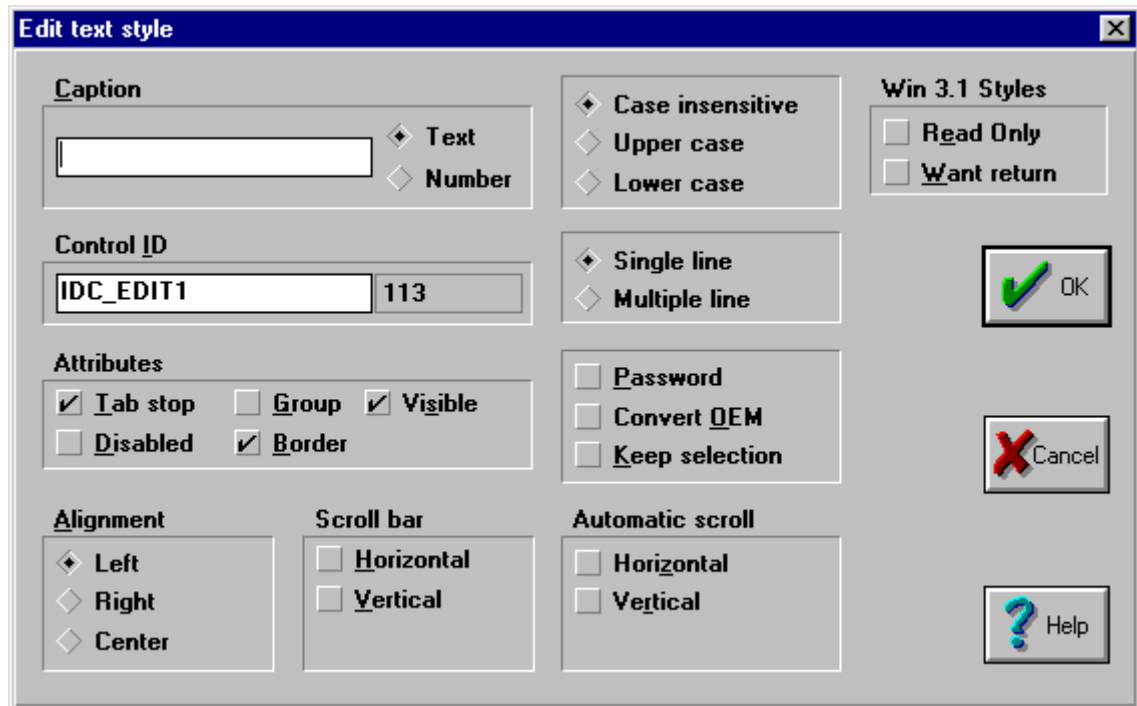
No Character Underline.-The No Character Underline check box turns off character underlining. If your static control contains text, you can underline a character by preceding it with an ampersand (&). If you turn this check box on, underlining is disabled and ampersands are displayed as literal characters.

Control Type.-The Control Type radio buttons define what's displayed by the static control.

Edit Icon button.-The Edit Icon Button starts the Bitmap editor, where you edit your icon. This button is enabled after you enter a caption for the icon in the Caption input box.

Edit text style:

Corresponde a los controles que permiten editar el contenido de una variable, o para entendernos, los GET's normales de clipper. Si hacemos doble clic en un **get**, aparece la siguiente pantalla.



donde sus distintos controles son:

Caption.- The Caption input box is where you enter the caption you want displayed with the control. The Caption radio buttons choose how the caption is displayed.

Control ID.- The Control ID input box is where you enter the control's identifier.

Attributes.- The Attributes check boxes select the control's attributes.

Alignment.- The Alignment radio buttons align text.

Scroll Bar.- The Scroll Bar check boxes put scroll bars in your control.

Case.- The Case radio buttons determine how edit text is displayed when it's typed.

Line.- The Line radio buttons choose how the user will type edit text.

Text Conversion.- The Text Conversion check boxes choose how text is converted when it's typed.

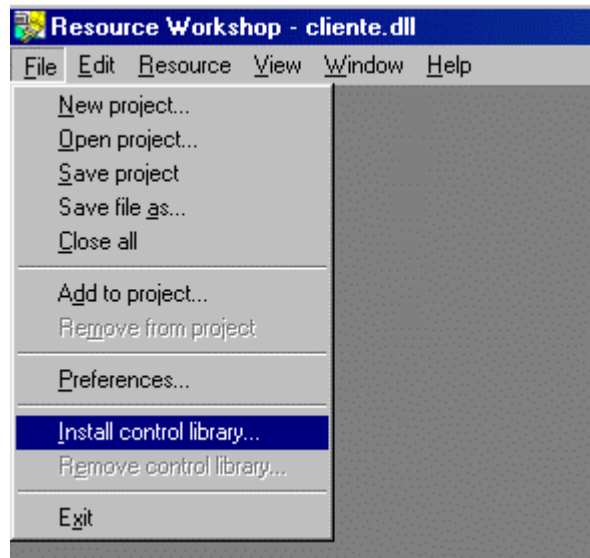
Automatic Scroll.- The Automatic Scroll check boxes choose how edit text will scroll.

Win 3.1 Styles.- The Win 3.1 Styles check boxes assign attributes specific to 3.1. These check boxes are available only if you're running under Windows 3.1.

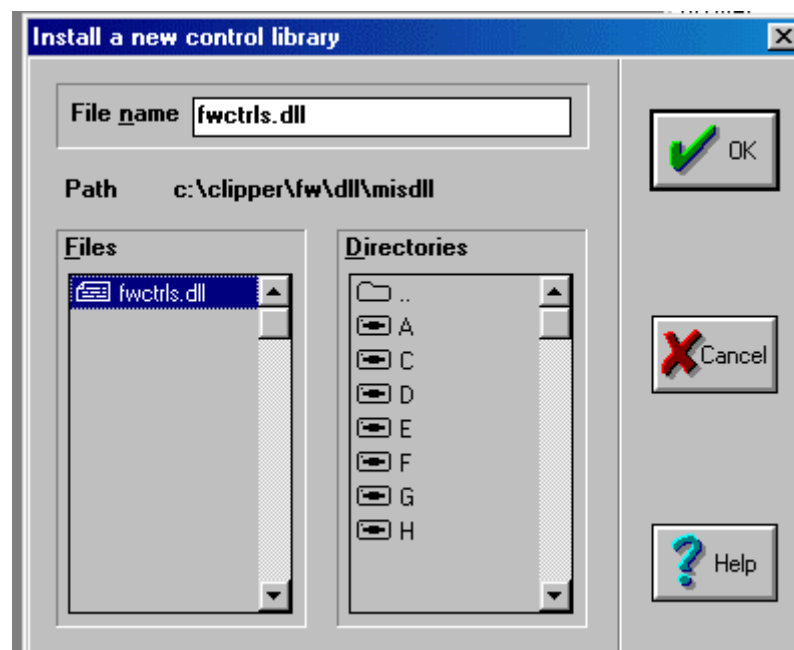
Añadir controles

Gracias a nuestro amigo Ricardo Ramírez y su control [Fwctrls](#) ahora podemos incorporar el control Folder directamente a la barra de herramientas de Workshop.

Para ello hemos de hacernos con el fichero FWCTRLS.DLL. A continuación seleccionaremos la opción *Install control library* del menú *file*



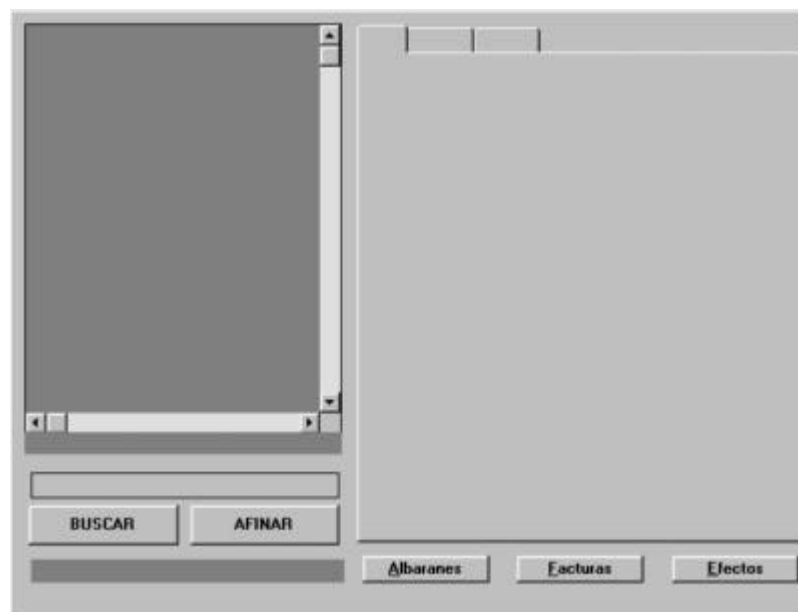
Desde la caja de selección de fichero, accederemos al directorio donde hemos guardado la librería Fwctrl.dll y la seleccionaremos



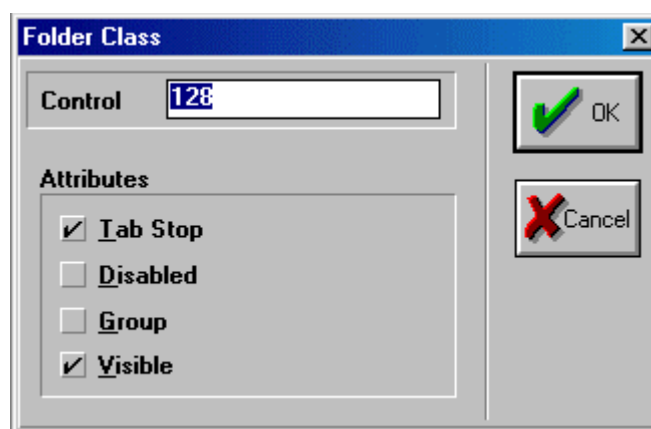
A partir de ese momento, en la caja de herramientas aparecerá un nuevo control con el cual podremos crear nuestras folders directamente.



Además, al crear por ejemplo, un folder, lo distinguiremos visualmente dentro del dialog



Y al acceder a las propiedades del folder, éstas se nos mostrarán de forma mucho más cómoda e intuitiva



Fenomenal !!! . Gracias Ricardo.

Programación Orientada a Objetos

Vamos a ver en este apartado una introducción a la Programación Orientada a Objetos, que de ahora en adelante, simplificaremos con las siglas en Ingles **OOP**. Si quieres ver el capítulo donde se profundiza en este tipo de programación pasa a la pág. 275.

Me resisto a explicar los objetos como objetos de la vida, tipo pelota, coche, etc. Esa definición la tendrás en muchos otros artículos sobre OOP. A mi no me sirvieron. Voy a tratar de explicártela como yo la entendí: haciendo equivalencias con la programación clásica que es la que conocemos.

Imagínate un sistema que le llamásemos **Programación Orientada a Matrices** (POM). Las matrices (o arrais, arreglos, o como quieras llamarlo) se supone que las conoces. Hagamos una matriz sobre un documento de tipo factura que llamaremos **mFactura**. En ella ponemos en el elemento 1, mFactura[1] otra matriz con el nombre del cliente mFactura[1,1], la dirección mFactura[1,2], etc. en mFactura[2] los datos de la propia factura como el número de documento mFactura[2,1], la fecha mFactura[2,2], etc. para terminar en otro elemento, por ejemplo, el 20, ponemos los datos del pie, como mFactura[20,1] el bruto, en mFactura[20,2] el neto y en mFactura[20,3] el total.

Elemento nivel 1	Elemento nivel 2	Contenido
Mfactura[1]		Datos del cliente
	mFactura[1,2]	Nombre
	mFactura[1,2]	Dirección
mFactura[2]		Datos de la factura
	mFactura[2,1]	Número
	mFactura[2,2]	Fecha
mFactura[20]		Datos del pie
	mFactura[20,1]	Importe Bruto
	mFactura[20,2]	Importe Neto
	mFactura[20,3]	Importe TOTAL

Esta matriz será el “objeto” factura. Para cambiar un dato simplemente escribiremos **mFactura[x,y] := lo_que_sea** . Una vez puestos los datos en el objeto, lo único que tenemos que hacer es llamar a la función de imprimir factura, pasando como parámetro la matriz, es decir, **ImprimeFactura(mFactura)**. Hasta aquí puedes ver como estamos construyendo un objeto que a su vez es una matriz. Pero la OOP es mucho mas que eso. Sigamos con nuestro ejercicio imaginativo.

Las facturas pueden imprimirse y al ser impresas calcular los valores de Bruto, Neto y Total, es decir no solamente hemos de aportar datos, sino que también podemos recibir valores una vez efectuada la operación. Veamos primero la función.

Hemos dicho que la función de impresión se llama ImprimeFactura(), pero quizás queramos imprimir unas facturas con una función y otras con otra, por ejemplo para variar el formato. Para ello es posible definir en otro elemento, por ejemplo mFactura[30,1] un codeblock con la función que se utilizará para imprimir realmente, por ejemplo ListaTipo1() o ListaTipo2(). Cuando llamamos a nuestra factura, la función ImprimeFactura(mFactura) llamará a su vez a la función indicada en mFactura[30,1] y le pasará los datos. Estas funciones pondrán los resultados de los cálculos de las líneas de la factura en mFactura[20,1], [20,2] y [20,3] y de esas posiciones nosotros podremos tomarlos para hacer lo que se quiera, por ejemplo, grabarlos en la base de datos. El programa quedaría así:

```
mFactura[1,1] := clientes->NOMBRE
mFactura[1,2] := clientes->DIRECCIÓN
mFactura[2,1] := contador->FACTURA
....
mFactura[30,1] := {|| ListaTipo1() }
....
ImprimeFactura(mFactura)
replace factura->BRUTO with mFactura[20,1]
etc.
```

Quizás la explicación sea un poco extensa, pero si comprendes esto, tienes el 50 por ciento ganado. Sigamos.

Como habrás visto, en las líneas del programa anterior falta declarar la variable. Esto es lo que en OOP se conoce como **constructor**. Por ejemplo.

```
mFactura := ConstruyeFactura()      // crea la matriz con los elementos vacíos.
....
mFactura := NIL                     // recupera memoria destruyendo la matriz.
```

Este constructor puede crear la matriz vacía o con elementos predefinidos en el programa objeto. Por ejemplo, puede poner la fecha del objeto factura igual a **date()**. Luego podemos modificarla, pero si no lo hacemos por lo menos ya tiene algo.

Bien, ya tenemos un sistema por el que creamos nuestro objeto. Un proceso por el que podemos poner valores o modificar los que se hubiesen creado por defecto, así como obtener datos del propio objeto. Un sistema por el que podemos ejecutar funciones que hay grabadas en el propio objeto y por último un sistema para destruir el objeto. Si solo fuese esto tendríamos nuestro OOP dirigido a matrices de una forma realmente operativa. Concretamente, antes de programar con objetos, muchos de mi sistemas eran Programación Orientada a Matrices.

El siguiente paso es crear un compilador para que el programador haga todo esto de una forma transparente. Ese compilador lo han creado prácticamente todos los fabricantes de lenguajes de programación y Clipper no podía ser menos. Al principio

incorporó pocos objetos propios, pero si dejó la puerta abierta para que terceros fabricante, como el caso de Fivewin pudiese crear sus propios sistemas de OOP e incluirlos como parte de Clipper. Como ocurre con otros aspectos de la vida, cuando el fabricante de un buen sistema deja la puesta abierta, se llena enseguida de otros que continúan con la labor. Un ejemplo es el propio ordenador PC, que IBM dejó abierto, tanto en hardware como un bus local donde otros fabricantes podían conectar sus propias tarjetas. Gracias a ello, hoy el PC es quien es. También es el caso de Windows, donde Microsoft ha hecho público su API para que todos los fabricantes de software puedan hacer programas en el entorno gráfico.

Uno de los factores que hace el sistema OOP práctico, es la forma de programar. Al igual que las matrices se componen de elementos que son definidos por su posición entre corchetes, como `mFactura[1,2]`, en la OOP los elementos son definidos por el nombre del objeto, el signo dos puntos y el dato concreto. Por ejemplo:

```
oFactura:nombre := clientes->NOMBRE
```

Detengámonos un momento. Aquí vemos una autentica instrucción OOP. Si fuese en matrices sería `mFactura[1,2] := clientes->NOMBRE` y utilizando el preprocesador y las constantes simbólicas, incluso podríamos poner

```
mFactura[NOMBRE] := clientes->NOMBRE
```

o

```
FACTURA_NOMBRE := clientes->NOMBRE
```

Lo que ha hecho el compilador es dar una forma estándar a la manera de acceder al dato nombre, dentro de la factura.

Ten en cuenta que dentro de un dato, puede haber todo un objeto, de modo que si hubiesemos creado un objeto para la cabecera y luego éste fuese parte de la factura, sería posible poner:

```
oFactura:oCabecera:nombre := clientes->NOMBRE
```

También podemos obtener datos del objeto, poniendo:

```
replace BRUTO with oFactura:bruto
```

(nota que las variables de los objetos empiezan por la letra "o", según la notación hungara)

Para ejecutar una función dentro del objeto, por ejemplo imprimir factura la sintaxis sería algo como:

```
oFactura:imprime()
```

Para crear un objeto la sintaxis suele ser:

```
Local oFactura := oFactura():new(parametro1,parametro2,parametro N)
```

Donde los parámetros son modificadores de la construcción o valores por defecto para las variables internas.

Para destruirla se utiliza normalmente:

`oFatura:end()`

No vamos a entrar ahora en herencia, polimorfismo, etc. Solo vamos a indicar algunos términos que has de conocer perfectamente. En clipper ya sabes sin necesidad de pensar lo que es una variable o una función. Ahora tienes que aprender algunos conceptos más, que a mi personalmente me costo trabajo aprender y que quiero resumir aquí para ver si a ti te es más fácil:

Equivalente Clipper	Terminología OOP	Descripción
Variable	Variable de instancia	Lugar donde están los datos. Se puede modificar (no siempre) o leer. Hay algunas variables que solo son internas del objeto y no pueden verse ni modificarse. Es como si no existiesen
Función	Método	Es la función que se puede ejecutar desde nuestro programa para que el objeto haga una determinada cosa. Por ejemplo <code>oTbr:up()</code> sube una línea en un objeto Browse. Los métodos pueden devolver datos a nuestro programa, o pueden modificar variables de instancia que después podemos analizar, o simplemente ejecutar algo sin modificar ningún dato.

¿ Como se ve todo esto desde Fivewin ?. Pues veamos un ejemplo, el objeto ventana de diálogo, es decir la clásica ventana de windows donde se hacen los get, etc.

Herencia.- En OOP constantemente estamos oyendo decir algo parecido a esto: “El objeto TBrowse es heredado del objeto TWindows, por lo tanto tiene las características de Twindows y además las suyas”.

Volvamos al ejemplo con matrices: nosotros podemos hacer:

```
mDocumento := creaDocumento()
mFactura    := creaFactura(mDocumento)
mAlbarán    := creaAlbaran(mDocumento)
```

La primera línea crea la matriz de documentos en general. La segunda crea la matriz de facturas pero basándose en una matriz ya creada, de forma que podemos hacer que tenga los mismo datos y además los suyos propios. La tercera línea hará igual con albaranes. mFactura y mAlbaran son matrices heredadas de mDocumento. Es fácil pensar como hacerlo en OOP.

Veamos ahora un objeto real (de programación) mucho mas simple pero que podemos crear y trabajar con él. El problema es encontrar un ejemplo que sea sencillo de entender, fácil de realizar y al que se pueda poner todas las particularidades de este tipo de programación. Pues bien, he elegido una caja de tipo

@ lin_superior,col_izquierda,lin_inferior,col_derecha BOX cuadro+relleno

que se realiza en clipper normal, sin windows. De esta forma, ahora que todavía no conoces bien Fivewin, te será mas facil. Como acabo de decirte este objeto es en DOS mientras no te diga lo contrario.

Necesitamos de dos PRG. Uno contiene el objeto y otro es el programa que creará y utilizará el objeto. Veamos el primero. Para ganar tiempo he creado en el mismo PRG dos funciones o hablando con mayor propiedad, dos métodos. Uno para crear el objeto y otro para dibujarlo. Con el primero hubiese sido suficiente para crearlo, pero no hubiese servido para nada, a menos que además de crearlo lo visualizara. Si recuerdas browse de windoes necesita primero crearse y luego visualizarse. Nosotros hacemos igual. Al final del ejemplo estarás en condiciones de comprender todo esto e incluso modificarlo a tu gusto. Como siempre te indico el programa y un pequeño comentario en cada línea.

```
#include 'objects.ch'
```

```
Este es el fichero de cabecera o include que necesitas para adaptar la
terminologia de sistema OBJECTS. Puede que en tus programas reales
necesites otros includes, pero para este ejemplo es suficiente este.
```

```
CLASS Tventana // comienza la definición de la clase.
```

```
Todo objeto pertenece a una clase, aunque esta se confunda aparentemente
con el propio objeto. Lo que definimos aquí es la clase. El objeto es lo
que se crea al invocar al constructor de la clase. Por ejemplo, un coche,
pertenece a la clase de coches, pero un coche no es la clase. Otro
ejemplo, una matriz corresponde al conjunto, sistema, clase, etc. de
matrices
```

```
DATA nLS // línea superior
```

```
En esta línea y las siguientes, con la instrucción DATA, definimos las
variables de instancia que podrán ser accedidas desde fuera del programa,
es decir, las variables que luego podremos modificar en nuestro programa
para modificar el funcionamiento del objeto. Ahora concretamente definimos
nLS como línea superior.
```

```
DATA nCI // columna izquierda
```

```
DATA nLI // línea inferior
```

```
DATA nCD // columna derecha
```

```
DATA cMarco // strig que definirá el marco.
```

```
DATA cRelleno // strug que definirá el fondo.
```

```
METHOD New(nLS,nCI,nLI,nCD,cMarco,cRelleno) CONSTRUCTOR
```

```
Ahora vienen los métodos. Aquí indicamos los metodos (funciones) que
podremos ejecutar desde nuestro programa. Como en cualquier función
clipper hay que indicar los parámetros que tienen. La cláusula CONSTRUCTOR
indica al compilador que este método es el encargado de crear el objeto.
Es decir, no se comporta como una función más, sino que nos devolverá el
objeto creado. Normalmente al metrodo que crea la clase se llama NEW() y
se dan unos valores por defecto. Pero podrá también llamarse de otra
manera y no crear valores ninguno.
```

```
METHOD Dibuja()
```

```
Este es otro método. Concretamente es que el dibujará el cuadro.
```

```
ENDCLASS // termina la definición
```

```
Ahora viene la programación en si de lo que es la clase y harán los
objetos pertenecientes a esta clase.
```

```
METHOD New(nLS,nCI,nLI,nCD,cMarco,cRelleno) CLASS TVentana
```

Comienza con la función. Como ves es parecida a las funciones normales de clipper, pero a primera vista tiene dos diferencias. Ha de ir precedida de la palabra METHOD y al terminar ha de indicarse a la clase que pertenece, en este caso CLASS TVentana. El resto es igual, el nombre de la función y los parámetros.

```
DEFAULT nLS := 0
```

Esta instrucción es del preprocesador normal e indica que en caso de que la que la variable perteneciente al parámetro sea NIL, que ponga el valor indicado. En esca caso 0

```
DEFAULT nCI := 0
```

```
DEFAULT nLI := maxrow()
```

```
DEFAULT nCD := maxcol()
```

```
DEFAULT cMarco := chr(218)+chr(196)+chr(191)+;
                 chr(179)+chr(217)+chr(196)+;
                 chr(192)+chr(179)
```

Este es un string(también puedes hacerlo directamente) para indicar los caracteres que compondrán la caja.

```
DEFAULT cRelleno := ' '
```

Este es el caracter de relleno, en esta caso, un espacio en blanco.

Ahora hemos de poner las variables de instancia del método igual a las variables locales correspondientes a los parámetros. Observa que se llaman igual, pero no son las mismas. El doble signo de dos puntos (::), indica que pertenece al objeto, no al programa. De esta forma viene ha decir: **pon la variable nLS del objeto igual a la variable nLS del programa**. Podían haberse llamado distinto, pero las hemos llamado igual.

```
::nLS      := nLS
::nCI      := nCI
::nLI      := nLI
::nCD      := nCD
::cMarco   := cMarco
::cRelleno := cRelleno
```

```
RETURN NIL
```

Aunque pongamos NIL, el constructor siempre retorna el objeto construido.

```
METHOD Dibuja() CLASS TVentana
```

Este es el método que dibuja la ventana.

```
@ ::nLS,::nCI,::nLI,::nCD BOX ::cMarco+::cRelleno
```

Como verás, todas las variables son de instancia (recuerda, "::")

```
RETURN NIL
```

Compila simplemente poniendo:

clipper Tventana -m-n-a-w

Ahora vamos a escribir el programa que usará la ventana:

```
function test
```

```
local oVen := Tventana():new()
```

Esto nos crea y nos devuelve el objeto Tventana y lo pone en la variable local **oVen**. Como puedes ver en el programa de la clase, la construcción no lleva consigo el dibujo.

```
oVen:dibuja()
```

Ahora lo dibujamos. Lo hará con los valores por defecto o los que hubiesemos pasado cuando construimos la ventana.

```
inkey(0)    // esperamos una tecla.
```

Ahora modificamos una o varias de sus variables de instancia.

```
oVen:nLS := 10
oVen:nCI := 10
oVen:nLI := 20
oVen:nCD := 20
```

```
oVen:dibuja()
```

La dibujamos de nuevo. Como verás en la ejecución, ahora se mostrará una nueva ventana.

```
inkey(0)    // esperamos una tecla
```

```
return NIL
```

Compila simplemente poniendo:

clipper test -m-n-a-w

Para enlazar pon:

blinker fi test,tventana lib objects

Como el programa no es windows, no tienes que preocuparte de más. Ejecuta el programa y ya tienes tu primer objeto.

2

CONTROLES

Aquí veremos los distintos controles que se pueden hacer por medio del preprocesador para convertir clipper, por medio de Fivewin, en un entorno Windows. Cada control esta redactado de la misma forma, de modo que todos tienen los mismos apartados con mayor o menor explicación.

PROGRAMACIÓN DE CONTROLES

En este capítulo vamos a descender de nivel, es decir, profundicar un poco más en el conocimiento de Fivewin. Verás una estructura similar al capítulo de introducción, pero ahora los detalles de cada uno de los controles se hace exhaustivamente. Podemos decir que este capítulo es el necesario para poder hacer una aplicación real. Espero que llegues aquí con los conocimientos necesarios como para poder desarrollar por ti mismo lo que pudiese faltar en alguna explicación. Ya debes ser capaz de entender el fichero de cabecera **fivewin.ch**. En cada control se hará una visión profunda desde el aspecto de workshop y desde el aspecto del programa.

Al pretender ver en cada control todos los detalles, no podremos seguir el tutorial con ejemplos para cada una de las variantes, de modo que debes ser tu mismo quien en propio interés te pongas ejemplos reales. En un próximo capítulo incluimos el desarrollo de una aplicación real, como es el caso del control de nuestra librería casera o de empresa.

En los controles que son utilizados con Workshop se indica someramente el sistema por programación. En los que normalmente se hacen por programación, como es el caso de la barra de menu, se hace al revés.

Puesto que este capítulo es de consulta, no vamos a seguir un orden pedagógico, sino un orden alfabético. El problema es en que idioma. Muchos de los controles los conocemos sin traducción, por ejemplo SAY,GET o SCROLL BAR. Otros tienen perfecta traducción, como BARRA DE BOTONES o CAJA DE DIALOGO. Puesto que al programar tanto con el editor, como con WorkShop hemos de poner la palabra en inglés, no nos queda mas remedio que conocerlas. Muy a mi pesar he de poner el orden en inglés. Los controles que tenemos en este capítulos son:

BITMAP

Browses

BTNBMP

BUTTON

BUTTONBAR

CHECKBOX

COMBOBOX
FOLDER
DIALOG
FOLDER
GET
GET (memo)
ICON
IMAGEN
LISTBOX
MENUBAR
MESSAGE
METER
PAGES
RADIO
SAY
SCROLL
TABS
VBX
VIDEO
WINDOWS

Un resumen genérico de todas las posibles cláusulas es el siguiente:

(En cada caso puede cambiar un poco)

- **@** irá seguido de las coordenadas de posición. Si no se especifica la cláusula PIXEL, los números serán línea y columna. Indicando PÍXEL, los valores serán en pixels. OJO, en caso de ser línea y columna, la pantalla se divide en **30** líneas y 80 columnas
- **3D** sirve para dar aspecto de tres dimensiones. Se usa en muy pocos controles.
- **ACTION** será la función que se ejecuta al hacer clic sobre el control.
- **ADJUST** Ajusta la imagen completa, al espacio asignado. Hay que tener en cuenta que el programa hace un ajuste matemático independiente para el ancho y el largo. Si el recuadro asignado por programa no guarda la misma proporción de ancho y largo que la imagen real, saldrá distorsionada. También hay que tener en cuenta que si se pretende ampliación, se pierde definición.
- **ALIAS** indica el alias del fichero relacionado con el control. Si no se pone, la base de datos será la activa en ese momento.
- **BORDER** indica que saque un rectángulo marcando el borde del control en función de SIZE.

- **BRUSH** indica el objeto brocha que se utilizará como fondo del objeto.
- **BOX**
- **BUTTONSIZE** es el tamaño del botón en pixels.
- **CANCEL** permite al usuario cancelar la operación del control.
- **CENTER, CENTERED** indica que centre el texto en el espacio marcado por SIZE. Esta cláusula es excluyente con RIGHT.
- **COLOR, COLORS** indica el color del texto. Si se pone un segundo color, será el del fondo. El color se define mediante la función nRGB() o similar.
- **CURSOR** indica que cambie el cursor cuando este pase por encima.
- **DEFINE**
- **DESIGN** añade un borde al control, redimensionable en tiempo de ejecución. También permite que el control se pueda mover con el ratón.
- **DLL** indica el fichero DLL relacionado con el control.
- **FIELDS** indica los campos que se utilizarán en el control.
- **FILE** indica el fichero relacionado con el control.
- **FONT** indica el objeto fuente, en caso de ser distinto al indicado en momento del diseño con el workshop.
- **FROM...TO...** indica las coordenadas del rectángulo.
- **GROUP** indica si pertenece a un grupo.
- **HEAD** indica el texto o textos que irán en la cabecera.
- **HORIZONTAL** indica si el control será en su versión horizontal.
- **HELP**
- **ID** indica el indicador en workshop. Puede ser una variable definida.
- **MEMO** se utiliza en el control para indicar que ha de editarse un campo memo o una variable de texto largo
- **MENU**
- **MESSAGE** es el mensaje que aparecerá en la barra de mensajes cuando el cursor pase por encima del control.
- **NAME**
- **OF, WINDOWS, DIALOG** es el marco que lo contine. Normalmente será una caja de diálogo.
- **ON CHANGE** indica la función que se ejecutará cuando cambie el contenido o aspecto del control.
- **ON DRAWITEN**
- **ON DROP**
- **ON EDIT**
- **OPTION**
- **PAGE**

- **PICTURE** es la máscara igual que en DOS.
- **PÍXEL, PIXELS**. Normalmente la posición se indica en líneas y columnas, y el tamaño en píxels. Con esta cláusula, todo será en píxels
- **PROMPT, VAR** seguidas del texto literal o el nombre de la variable que lo contiene. El font y el color será el indicado en las cláusulas **FONT** y **COLOR**. En caso de no tenerlas, podrá los valores por defecto del marco o ventana que lo contiene.
- **RAISED**
- **REDEFINE**
- **RIGHT** indica que visualice el texto justificado por la derecha, en el espacio marcado por SIZE. Esta cláusula es excluyente con CENTERED o CENTER
- **SELECT**
- **SHADOW, SHADES** indica que el texto tenga una pequeña sombra para simular 3D.
- **SIZE** indica el tamaño en píxels del campo que ocupará el texto. No indica el campo del texto, que eso está marcado por el propio texto y el font utilizado. Este tamaño se utiliza, por ejemplo para el color de fondo, el borde o la superposición sobre otros controles.
- **TITLE**
- **TOP/LEFT/RIGHT/BOTTOM/FLOAD**
- **TOOLTIP** indica el pequeño texto de ayuda que sale junto al control al dener unos segundos el cursor sobre él.
- **UPDATE** se utiliza para actualizar el texto cuando se refresca la caja o ventana de diálogo por medio de la instrucción <objeto diálogo>:refresh().
- **VALID** es la función que debe devolver .T. para que el control pueda dejar el foco y continuar con el programa. Es similar a la versión en DOS
- **VAR** es la variable asociada al control.
- **VERTICAL** indica que el control se presentará en su versión vertical
- **WHEN** es la función que debe devolver .T. para que el control pueda tomar el foco. Es similar a la versión en DOS

En la página siguiente se incluye un cuadro resumen de los controles y sus respectivas cláusulas. No todas las cláusulas significan lo mismo en todos los controles, pero normalmente sí. Las últimas cláusulas están separadas por pertenecer a un solo control. También puede ocurrir en esta última parte que se agrupen varias cláusulas distintas, por ejemplo CANCEL, GROUP quiere decir que en BUTTON existen estas dos cláusulas diferentes que solo están en BUTTON.

[illegible]

BIPMAPS

Concepto

Por medio de este control se visualizan ficheros de imágenes en formato BMP en cajas de diálogo. Como es lógico, entre las cláusulas esta la definición del fichero donde esta el BMP. Las imágenes pueden ocupar parte o toda la caja. Si existen otros controles, el BMP se pondrá “debajo” del resto. Si hay varios BITMAP, se pondrán por el orden que aparecen en el programa. Los bitmaps permiten a una aplicación windows hacer cosas que eran imposibles en DOS. Hay otros controles como los SAY, GETS e incluso LISTBOX, etc. que podian ser realizados con mas o menos formuna en DOS, pero los bitmap solo podian ser hechos si entrabamos en modo gráfico. Ahora desde windows, este control no permite mostrar fotografias, digujos pasados a BMP, etc. Las posibilidades son inagotables. Una aplicación mediocre pero que dispone de fotografias gana bastante de cara al usuario final.

Fivewin crea la clase TBitmap para controlar este tipo de recurso

Por programa...

fivewin.ch

```
#xcommand @ <nRow>, <nCol> BITMAP [ <oBmp> ] ;
    [ <resource: NAME, RESNAME, RESOURCE> <cResName> ] ;
    [ <file: FILENAME, FILE, DISK> <cBmpFile> ] ;
    [ <NoBorder:NOBORDER, NO BORDER> ] ;
    [ SIZE <nWidth>, <nHeight> ] ;
    [ <of: OF, WINDOW, DIALOG> <oWnd> ] ;
    [ <lClick: ON CLICK, ON LEFT CLICK> <uLClick> ] ;
    [ <rClick: ON RIGHT CLICK> <uRClick> ] ;
    [ <scroll: SCROLL> ] ;
    [ <adjust: ADJUST> ] ;
    [ CURSOR <oCursor> ] ;
    [ <pixel: PIXEL> ] ;
    [ MESSAGE <cMsg> ] ;
    [ <update: UPDATE> ] ;
    [ WHEN <uWhen> ] ;
    [ VALID <uValid> ] ;
    [ <lDesign: DESIGN> ] ;
=> ;
[ <oBmp> := ] TBitmap():New( <nRow>, <nCol>, <nWidth>, <nHeight>;,
    <cResName>, <cBmpFile>, <.NoBorder.>, <oWnd>;,
    [\{ |nRow,nCol,nKeyFlags| <uLClick> \} ],,
    [\{ |nRow,nCol,nKeyFlags| <uRClick> \} ], <.scroll.>;,
    <.adjust.>, <oCursor>, <cMsg>, <.update.>;,
    <{uWhen}>, <.pixel.>, <{uValid}>, <.lDesign.> )
```

```

¿?
#xcommand DEFINE BITMAP [<oBmp>] ;
    [ <resource: RESOURCE, NAME, RESNAME> <cResName> ] ;
    [ <file: FILE, FILENAME, DISK> <cBmpFile> ] ;
    [ <of: OF, WINDOW, DIALOG> <oWnd> ] ;
=> ;
    [ <oBmp> := ] TBitmap():Define( <cResName>, <cBmpFile>, <oWnd> )

```

Comando

```

@ nRow, nCol BITMAP oBmp
NAME, RESNAME, RESOURCE cResName
FILENAME, FILE, DISK cBmpFile
NOBORDER, NO BORDER
SIZE nwidth, nHeight
OF, WINDOW, DIALOG oWnd
ON CLICK, ON LEFT CLICK uLClick
ON RIGHT CLICK uRClick
SCROLL
ADJUST
CURSOR oCursor
PIXEL
MESSAGE cMsg
UPDATE
WHEN uWhen
VALID uValid
DESIGN

```

Cláusulas

- **@** irá seguido de las coordenadas de posición. Si no se especifica la cláusula **PIXEL**, los números serán línea y columna. Indicando **PÍXEL**, los valores serán en pixels..
- **ADJUST** ajusta la imagen completa, al espacio asignado. Hay que tener en cuenta que el programa hace un ajuste matemático independiente para el ancho y el largo. Si el recuadro asignado por programa no guarda la misma proporción de ancho y largo que la imagen real, saldrá distorsionada. También hay que tener en cuenta que si se pretende ampliación, se pierde definición.
- **CURSOR** indica que cambie el cursor cuando este pase por encima.
- **DESIGN** añade un borde al control, redimensionable en tiempo de ejecución. También permite que el control se pueda mover con el ratón.
- **FILE** indica el fichero relacionado con el control.
- **MESSAGE** es el mensaje que aparecerá en la barra de mensajes cuando el cursor pase por encima del control.
- **OF, WINDOWS, DIALOG** es el marco que lo contine. Normalmente será una caja de diálogo.
- **PIXEL, PIXELS**. Normalmente la posición se indica en líneas y columnas, y el tamaño en píxels. Con esta cláusula, todo será en píxels

- **SIZE** indica el tamaño en pixels del campo que ocupará el texto. No indica el campo del texto, que eso está marcado por el propio texto y el font utilizado. Este tamaño se utiliza, por ejemplo para el color de fondo, el borde o la superposición sobre otros controles.
- **UPDATE** se utiliza para actualizar el texto cuando se refresca la caja o ventana de diálogo por medio de la instrucción <objeto diálogo>.refresh().
- **VALID** es la función que debe devolver .T. para que el control pueda dejar el foco y continuar con el programa. Es similar a la versión en DOS
- **WHEN** es la función que debe devolver .T. para que el control pueda tomar el foco. Es similar a la versión en DOS

Ejemplos

Por recursos...

fivewin.ch

```
#xcommand REDEFINE BITMAP [ <oBmp> ] ;
    [ ID <nId> ] ;
    [ <of: OF, WINDOW, DIALOG> <ownd> ] ;
    [ <resource: NAME, RESNAME, RESOURCE> <cResName> ] ;
    [ <file: FILE, FILENAME, DISK> <cBmpFile> ] ;
    [ <lClick: ON CLICK, ON LEFT CLICK> <uLClick> ] ;
    [ <rClick: ON RIGHT CLICK> <uRClick> ] ;
    [ <scroll: SCROLL> ] ;
    [ <adjust: ADJUST> ] ;
    [ CURSOR <oCursor> ] ;
    [ MESSAGE <cMsg> ] ;
    [ <update: UPDATE> ] ;
    [ WHEN <uWhen> ] ;
    [ VALID <uValid> ] ;
    [ <transparent: TRANSPAREN> ] ;
=> ;
    [ <oBmp> := ] TBitmap().ReDefine( <nId>, <cResName>, <cBmpFile>,;
        <ownd>, [\{ |nRow,nCol,nKeyFlags| <uLClick> \}],;
        [\{ |nRow,nCol,nKeyFlags| <uRClick> \}],;
        <.scroll.>, <.adjust.>, <oCursor>, <cMsg>, <.update.>,;
        <{uWhen}>, <{uValid}>, <.transparent.> )
```

Workshop

En WS para definir un control tipo BMP, hay que hacerlo por recursos externos, es decir, con el icono de la llave en la caja de herramientas. Una vez seleccionada la llave, en la ventana de **New custom control** escribir en **Class** la palabra **Tbitmap**. Señalar el rectángulo que contendrá la imagen. Si NO pones ADJUST ni SCROLL y el espacio reservado es menor que la imagen real, se mostrará la esquina superior izquierda. Si pones **ADJUST**, se ajustará automáticamente al espacio reservado. Esto es muy útil, pero hay que tener precaución de mantener las proporciones de alta y ancho o de lo contrario

saldrá la imagen distorsionada. Si quieres utilizar las barras de scroll para mover la imagen dentro de su recuadro, pon la cláusula **SCROLL** en el comando (quita **ADJUST** si estuviese puesto) y en el WS modificar el recurso (doble clic) añadiendo en Style los estilos **| WS_VSCROLL** y **| WS_HSCROLL**. De esta forma aparecerá a su tamaño real, pero solo se vera el espacio reservado.

Comando

```
REDEFINE BITMAP oBmp
  ID nId
  OF, WINDOW, DIALOG oWnd
  NAME, RESNAME, RESOURCE cResName
  FILE, FILENAME, DISK cBmpFile
  ON CLICK, ON LEFT CLICK uLClick
  ON RIGHT CLICK uRClick
  SCROLL
  ADJUST
  CURSOR oCursor
  MESSAGE cMsg
  UPDATE
  WHEN uWhen
  VALID uValid
  TRANSPAREN
```

Cláusulas

- **ADJUST** ajusta la imagen completa, al espacio asignado. Hay que tener en cuenta que el programa hace un ajuste matemático independiente para el ancho y el largo. Si el recuadro asignado por programa no guarda la misma proporción de ancho y largo que la imagen real, saldrá distorsionada. También hay que tener en cuenta que si se pretende ampliación, se pierde definición.
- **CURSOR** indica que cambie el cursor cuando éste pase por encima.
- **FILE** indica el fichero relacionado con el control.
- **ID** indica el indicador en workshop. Puede ser una variable definida.
- **MESSAGE** es el mensaje que aparecerá en la barra de mensajes cuando el cursor pase por encima del control.
- **NAME** ¿?
- **OF, WINDOWS, DIALOG** es el marco que lo contine. Normalmente será una caja de diálogo.
- **TRANSPAREN** ¿?
- **UPDATE** se utiliza para actualizar el texto cuando se refresca la caja o ventana de diálogo por medio de la instrucción <objeto diálogo>.refresh().
- **VALID** es la función que debe devolver .T. para que el control pueda dejar el foco y continuar con el programa. Es similar a la versión en DOS
- **WHEN** es la función que debe devolver .T. para que el control pueda tomar el foco. Es similar a la versión en DOS

Ejemplos**Trucos****Conclusión**

Es un excelente control para poner imágenes de fondo en nuestras cajas de diálogo, ventanas, etc. También es el mejor método para mostrar fotografías de personas o artículos. Su utilización correcta dará un toque muy profesional a las aplicaciones

BROWSE

Concepto

Realmente no existe el control BROWSE, es mas bien un control LISTBOX modificado. El preprocesador al ver que se incluye o no las cláusulas PROMPT o FIELDS, genera un código llamando a una función o a otra. Estas funciones generadas por el preprocesador corresponderán a la clase TListBox o TWBrowse. Ahora vamos a ver el control que genera TWBrowse. Debido a su uso extensivo en programación se ha preferido darle entidad propia y hacer un apartado exclusivo para él dentro del libro.

Pienso a nivel personal, que Fivetech podía haber creado el comando BROWSE. De hecho tu puedes hacerlo en tu fichero de cabecera copiando de Fivewin.ch el apartado de listbox y cambiando el nombre a BROWSE. De todas formas, en este libro seguiremos con los criterios de Fivetech para no liarlo demasiado. Si hay problemas con el browse de clipper, se puede poner otra palabra equivalente.

Si controlas los colores de texto y de fondo podrás hacer browses muy llamativos.

Por programa...

fivewin.ch

```
#xcommand @ <nRow>, <nCol> LISTBOX [ <oBrw> ] FIELDS [<Flds,...>] ;
[ ALIAS <cAlias> ] ;
[ <sizes:FIELDsizes, sizes, COLsizes> <aColSizes,...> ] ;
[ <head:HEAD,HEADER,HEADERS,TITLE> <aHeaders,...> ] ;
[ SIZE <nWidth>, <nHeigth> ] ;
[ <dlg:OF,DIALOG> <oDlg> ] ;
[ SELECT <cField> FOR <uValue1> [ TO <uValue2> ] ] ;
[ ON CHANGE <uChange> ] ;
[ ON [ LEFT ] CLICK <uLClick> ] ;
[ ON [ LEFT ] DBLCLICK <uLDbClick> ] ;
[ ON RIGHT CLICK <uRClick> ] ;
[ FONT <oFont> ] ;
[ CURSOR <oCursor> ] ;
[ <color: COLOR, COLORS> <nClrFore> [,<nClrBack>] ] ;
[ MESSAGE <cMsg> ] ;
[ <update: UPDATE> ] ;
[ <pixel: PIXEL> ] ;
[ WHEN <uWhen> ] ;
[ <design: DESIGN> ] ;
[ VALID <uValid> ] ;
[ ACTION <uAction,...> ] ;
=> ;
[ <oBrw> := ] TWBrowse():New( <nRow>, <nCol>, <nWidth>, <nHeigth>;
    [\{|\{<Flds> \} \}], ;
    [\{<aHeaders>\}], [\{<aColSizes>\}], ;
    <oDlg>, <(cField)>, <uValue1>, <uValue2>;,
    [<{uChange}>],;
    [\{|nRow,nCol,nFlags|<uLDbClick>\}],;
    [\{|nRow,nCol,nFlags|<uRClick>\}],;
```

```
<oFont>, <oCursor>, <nClrFore>, <nClrBack>, <cMsg>;  
<.update.>, <cAlias>, <.pixel.>, <{uWhen}>;  
<.design.>, <{uValid}>, <{uLClick}>;  
[\{<{uAction}>\}] )
```

Comando

```
@ nRow, nCol LISTBOX oBrw  
  FIELDS flds,...  
  ALIAS cAlias  
  FIELDIZES, SIZES, COLSIZES aColSizes,...  
  HEAD,HEADER,HEADERS,TITLE aHeaders,...  
  SIZE nWidth, nHeight  
  OF,DIALOG oDlg  
  SELECT cField FOR uValue1 TO uValue2  
  ON CHANGE uChange  
  ON [ LEFT ] CLICK uLClick  
  ON [ LEFT ] DBLCLICK uLDbClick  
  ON RIGHT CLICK uRClick  
  FONT oFont  
  CURSOR oCursor  
  COLOR, COLORS nClrFore[,nClrBack]  
  MESSAGE cMsg  
  UPDATE  
  PIXEL  
  WHEN uWhen  
  DESIGN  
  VALID uValid  
  ACTION uAction,...
```

Cláusulas

- **@** irá seguido de las coordenadas de posición. Si no se especifica la cláusula PIXEL, los números serán línea y columna. Indicando PÍXEL, los valores serán en pixels..
- **ACTION** será la función que se ejecuta al hacer clic sobre el control.
- **ALIAS** indica el alias del fichero relacionado con el control. Si no se pone, la base de datos será la activa en ese momento.
- **COLOR, COLORS** indica el color del texto. Si se pone un segundo color, será el del fondo. El color se define mediante la función nRGB() o similar.
- **CURSOR** indica que cambie el cursor cuando este pase por encima.
- **DESIGN** añade un borde al control, redimensionable en tiempo de ejecución. También permite que el control se pueda mover con el ratón.
- **FIELDS** indica los campos que se utilizarán en el control.
- **FONT** indica el objeto fuente, en caso de ser distinto al indicado en momento del diseño con el workshop.
- **HEAD** indica el texto o textos que irán en la cabecera.
- **MESSAGE** es el mensaje que aparecerá en la barra de mensajes cuando el cursor pase por encima del control.

- **OF, WINDOWS, DIALOG** es el marco que lo contine. Normalmente será una caja de diálogo.
- **ON CHANGE** indica la función que se ejecutará cuando cambie el contenido o aspecto del control.
- **ON CLICK**
- **ON DBLCLICK**
- **ON RIGHT CLICK**
- **PIXEL, PIXELS**. Normalmente la posición se indica en líneas y columnas, y el tamaño en píxels. Con esta cláusula, todo será en píxels
- **SELECT**
- **SIZE** indica el tamaño en píxels del campo que ocupará el texto. No indica el campo del texto, que eso esta marcado por el propio texto y el font utilizado. Este tamaño se utiliza, por ejemplo para el color de fondo, el borde o la superposición sobre otros controles.
- **SIZES**
- **UPDATE** se utiliza para actualizar el texto cuando se refresca la caja o ventana de diálogo por medio de la instrucción <objeto diálogo>:refresh().
- **VALID** es la función que debe devolver .T. para que el control pueda dejar el foco y continuar con el programa. Es similar a la versión en DOS
- **WHEN** es la función que debe devolver .T. para que el control pueda tomar el foco. Es similar a la versión en DOS

Ejemplos

Por recursos...

fivewin.ch

```
#xcommand REDEFINE LISTBOX [ <oLbx> ] FIELDS [<Flds,...>] ;
[ ALIAS <cAlias> ] ;
[ ID <nId> ] ;
[ <dlg:OF,DIALOG> <oDlg> ] ;
[ <sizes:FIELDIZES, SIZES, COLSIZES> <aColSizes,...> ] ;
[ <head:HEAD,HEADER,HEADERS,TITLE> <aHeaders,...> ] ;
[ SELECT <cField> FOR <uValue1> [ TO <uValue2> ] ] ;
[ ON CHANGE <uChange> ] ;
[ ON [ LEFT ] CLICK <uLClick> ] ;
[ ON [ LEFT ] DBLCLICK <uLDbClick> ] ;
[ ON RIGHT CLICK <uRClick> ] ;
[ FONT <oFont> ] ;
[ CURSOR <oCursor> ] ;
[ <color: COLOR, COLORS> <nClrFore> [,<nClrBack>] ] ;
[ MESSAGE <cMsg> ] ;
[ <update: UPDATE> ] ;
[ WHEN <uWhen> ] ;
[ VALID <uValid> ] ;
```

```

=> [ ACTION <uAction,...> ] ;
    [ <oLbx> := ] TWBrowse():ReDefine( <nId>, ;
    [ \{ | \{ <Flds> \} \} ], <oDlg>, ;
    [ \{ <aHeaders> \} ], [ \{ <aColSizes> \} ], ;
    <(cField)>, <uValue1>, <uValue2>, ;
    [ <{uChange}> ], ;
    [ \{ | nRow, nCol, nFlags | <uLDbClick> \} ], ;
    [ \{ | nRow, nCol, nFlags | <uRClick> \} ], ;
    <oFont>, <oCursor>, <nClrFore>, <nClrBack>, <cMsg>, <.update.>, ;
    <cAlias>, <{uWhen}>, <{uValid}>, ;
    [ \{ | nRow, nCol, nFlags | <uLClick> \} ], [ \{ <{uAction}> \} ] )

```

Workshop

En WS para definir un control tipo BROWSE, hay que hacerlo por recursos externos, es decir, con el icono de la llave en la caja de herramientas. Una vez seleccionada la llave, en la ventana de **New custom control** escribir en **Class** la palabra **TWBrowse**. Señalar el rectángulo que contendrá el browse. En el WS modificar el recurso (doble clic) añadiendo en Style los estilos | **WS_VSCROLL** | **WS_HSCROLL** | **WS_BORDER** | **WS_TABSTOP**.

Comando

```

REDEFINE LISTBOX oLbx
  FIELDS Flds,...
  ALIAS cAlias
  ID nId
  OF,DIALOG oDlg
  FIELD SIZES, SIZES, COLSIZES aColSizes,...
  HEAD,HEADER,HEADERS,TITLE aHeaders,...
  SELECT cField FOR uValue1 [TO uValue2]
  ON CHANGE uChange
  ON [ LEFT ] CLICK uLClick
  ON [ LEFT ] DBLCLICK uLDbClick
  ON RIGHT CLICK uRClick
  FONT oFont
  CURSOR oCursor
  COLOR, COLORS nClrFore [,nClrBack]
  MESSAGE cMsg
  UPDATE
  WHEN uWhen
  VALID uValid
  ACTION uAction,...

```

Cláusulas

- **ACTION** será la función que se ejecuta al hacer clic sobre el control.
- **ALIAS** indica el alias del fichero relacionado con el control. Si no se pone, la base de datos será la activa en ese momento.
- **COLOR, COLORS** indica el color del texto. Si se pone un segundo color, será el del fondo. El color se define mediante la función nRGB() o similar. La cabecera

y las barras de scroll mantienen el color por defecto de windows sin poder modificarlas.

- **CURSOR** indica que cambie el cursor cuando este pase por encima.
- **FIELDS** indica los campos que se utilizarán en el control.
- **FONT** indica el objeto fuente, en caso de ser distinto al indicado en momento del diseño con el workshop.
- **HEAD** indica el texto o textos que irán en la cabecera. Pueden ir separados por comas o ser una matriz de textos.
- **ID** indica el indicador en workshop. Puede ser una variable definida.
- **MESSAGE** es el mensaje que aparecerá en la barra de mensajes cuando el cursor pase por encima del control.
- **OF, WINDOWS, DIALOG** es el marco que lo contine. Normalmente será una caja de diálogo.
- **ON CHANGE** indica la función que se ejecutará cuando cambie el contenido o aspecto del control.
- **ON CLICK**
- **ON DBLCLICK**
- **ON RIGHT CLICK**
- **SELECT**
- **SIZES**
- **UPDATE** se utiliza para actualizar el texto cuando se refresca la caja o ventana de diálogo por medio de la instrucción <objeto diálogo>:refresh().
- **VALID** es la función que debe devolver .T. para que el control pueda dejar el foco y continuar con el programa. Es similar a la versión en DOS
- **WHEN** es la función que debe devolver .T. para que el control pueda tomar el foco. Es similar a la versión en DOS

Ejemplos

Trucos

Conclusión

BRUSH

Concepto

BRUSH que traducido viene a querer decir **brocha**, permite colocar un fondo en las ventanas distinto al clásico fondo gris. Es una opción similar a la que hace Windows con el fondo del escritorio. Puede ser un fondo de los predefinidos en Fivewin y Borlan o incluso una imagen de un bitmap.

No es un recurso propiamente dicho, por lo que no puede ser controlado mediante workshop. Es mas bien un modificador de los recursos **WINDOWS** y **DIALOG**.

Tiene dos comandos. Uno para definir la brocha. Este comando crea el objeto. Una vez creado se puede activar de dos formas:

- 1.- Al definir la ventana poner de cláusula BRUSH <objeto_brush>
- 2.- Definir la ventana sin la cláusula BRUSH y luego poner:

SET BRUSH OF <objeto_ventana> TO <objeto_brush>

Este segundo caso utiliza un método del objeto ventana como puede verse en el fichero Fivewin.ch: **<oWnd>:SetBrush(<objeto_brush>)**. De esta forma se pueden cambiar los fondos de las ventanas de una forma dinámica en tiempo de ejecución. Dejamos los ejemplos de este método a tu imaginación.

Por programa...

fivewin.ch

```
#xcommand DEFINE BRUSH [ <oBrush> ] ;
    [ STYLE <cStyle> ] ;
    [ COLOR <nRGBColor> ] ;
    [ <file:FILE,FILENAME,DISK> <cBmpFile> ] ;
    [ <resource:RESOURCE,NAME,RESNAME> <cBmpRes> ] ;
=> ;
    [ <oBrush> := ] TBrush():New( [ Upper(<cStyle>) ], <nRGBColor>;
        <cBmpFile>, <cBmpRes> )

#xcommand SET BRUSH ;
    [ OF <oWnd> ] ;
    [ TO <oBrush> ] ;
=> ;
    <oWnd>:SetBrush( <oBrush> )
```

Comando

```
DEFINE BRUSH oBrush  
    STYLE cStyle  
    COLOR nRGBColor  
    FILE,FILENAME,DISK cBmpFile  
    RESOURCE,NAME,RESNAME cBmpRes
```

```
SET BRUSH OF oWnd TO oBrush
```

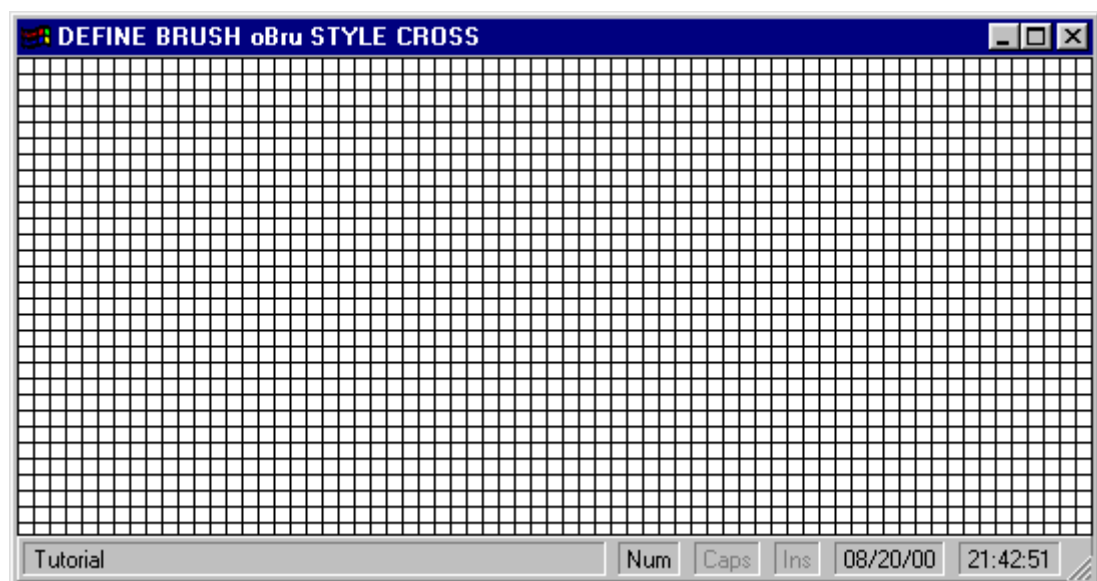
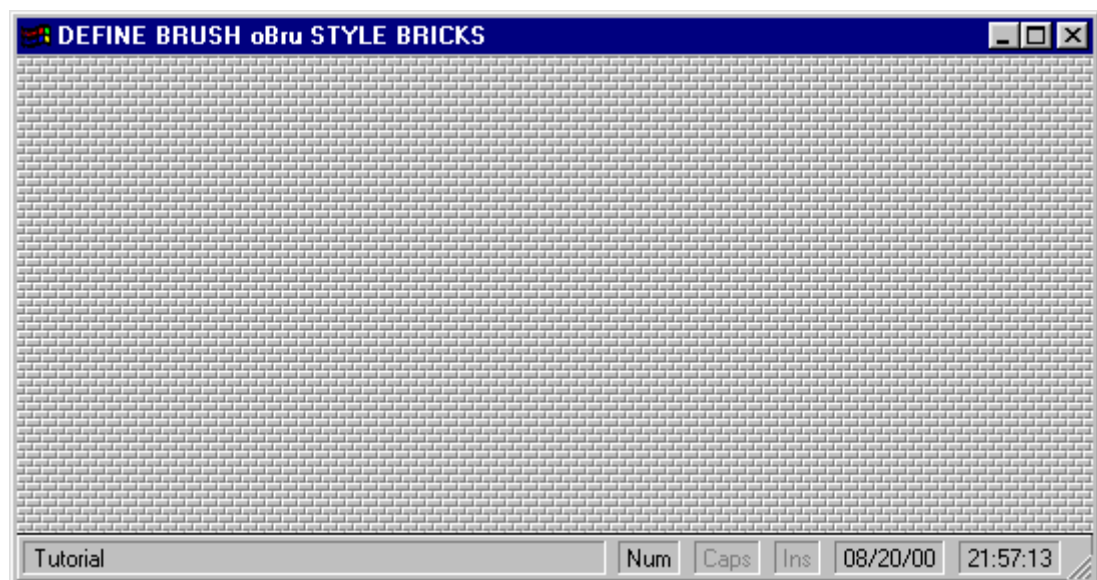
Cláusulas

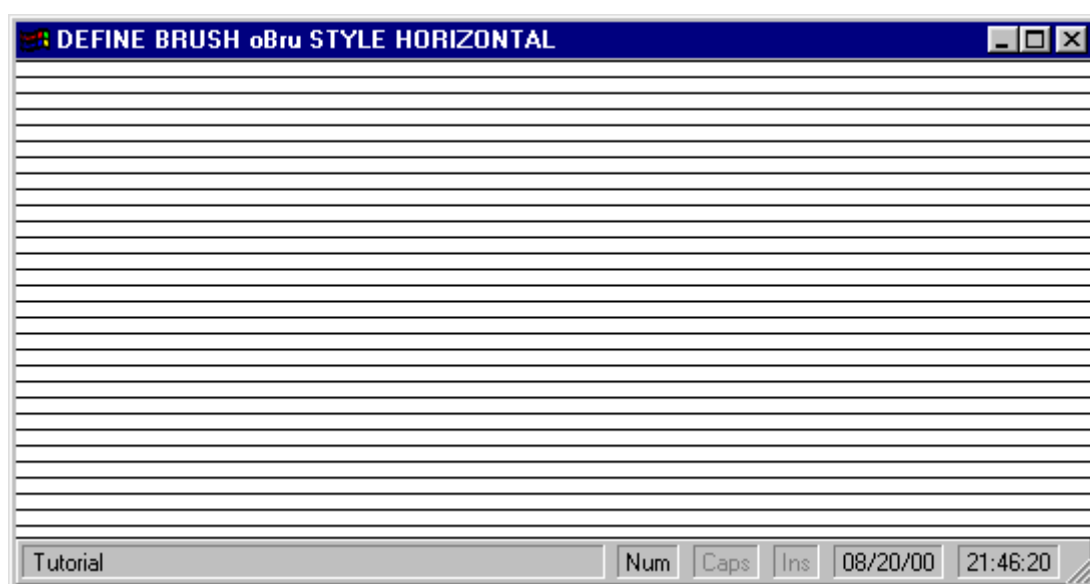
- **STYLE** marca el estilo del fondo. Esta cláusula es incompatible con COLOR.
- **COLOR** indica el color del fondo liso. Si se indica COLOR no se puede indicar STYLE.
- **FILE, FILENAME, DISK** es el fichero BMP si se utiliza
- **RESOURCE, NAME, RESNAME** es el nombre del recurso

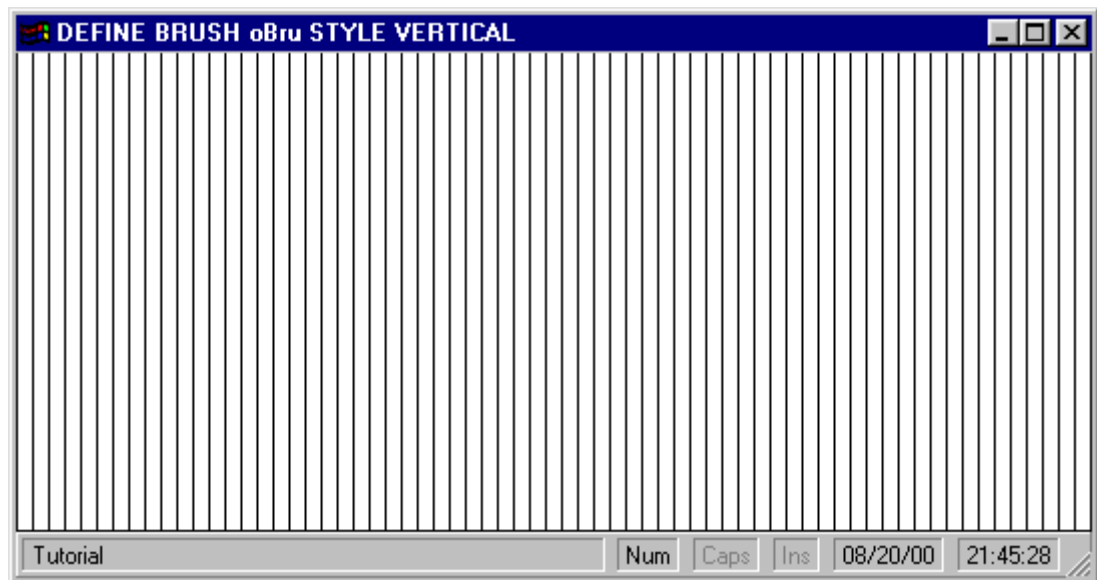
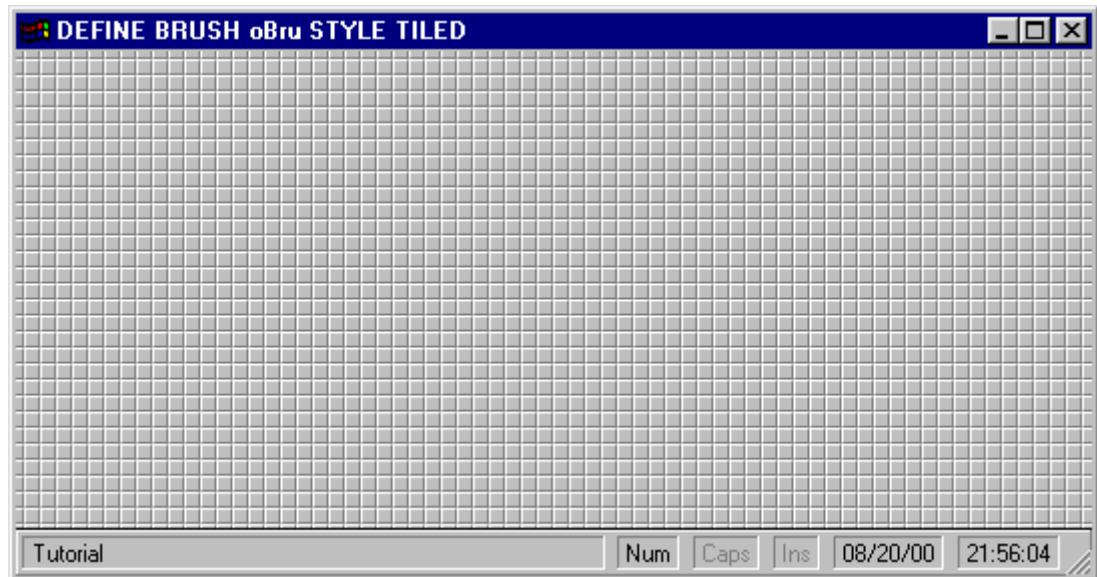
Ejemplos

Los posibles estilos se muestran a continuación. La cláusula COLOR no se ha utilizado. Fijate en el título (Caption) de la ventana donde podrás ver el comando que define la brocha. Primero veremos por orden alfabético, las que trae predefinidas el sistema





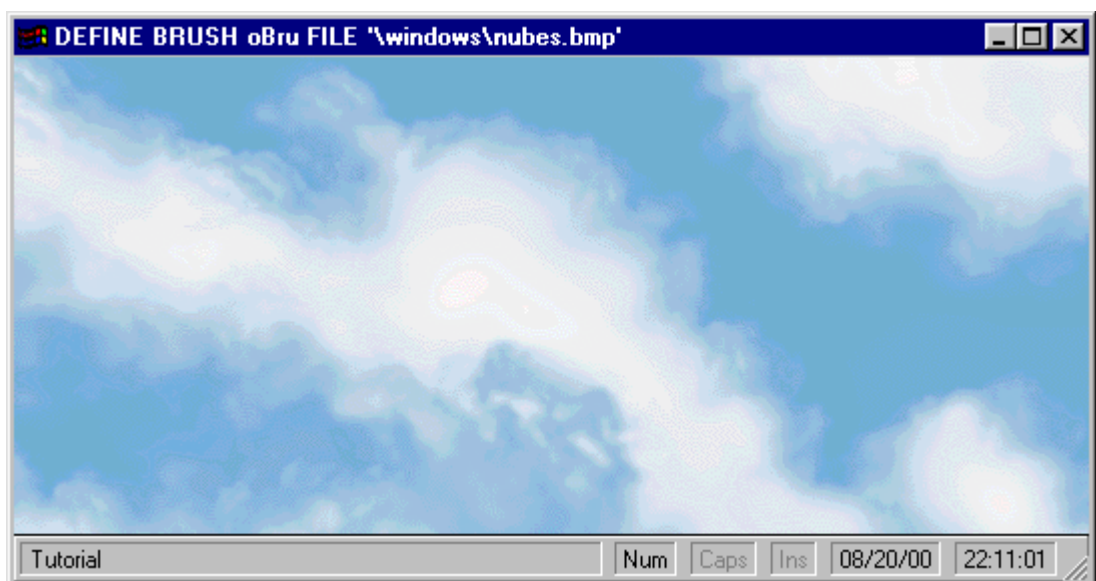


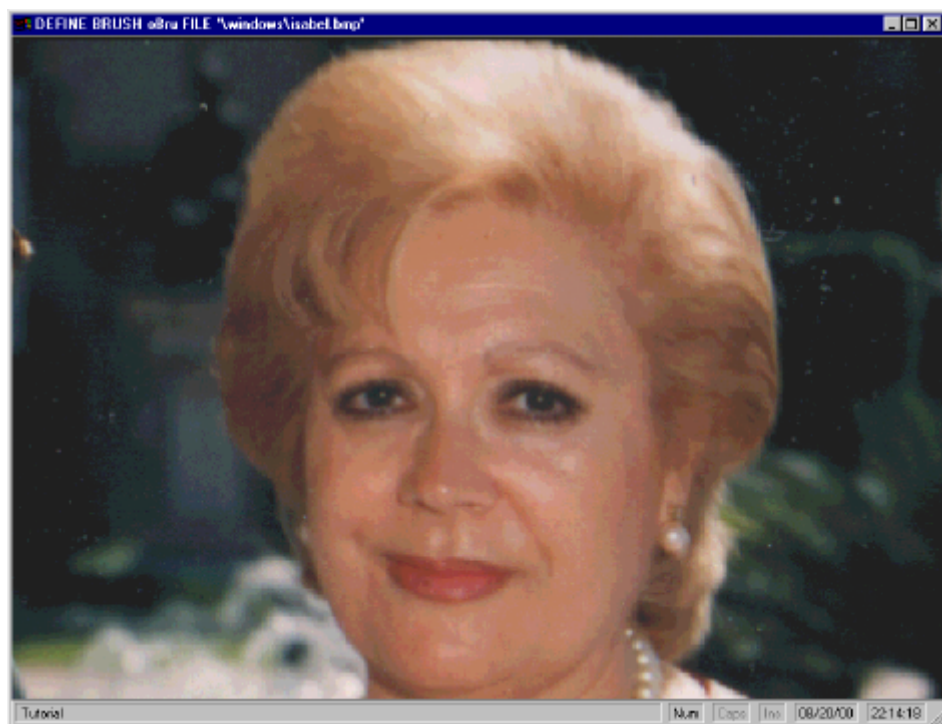
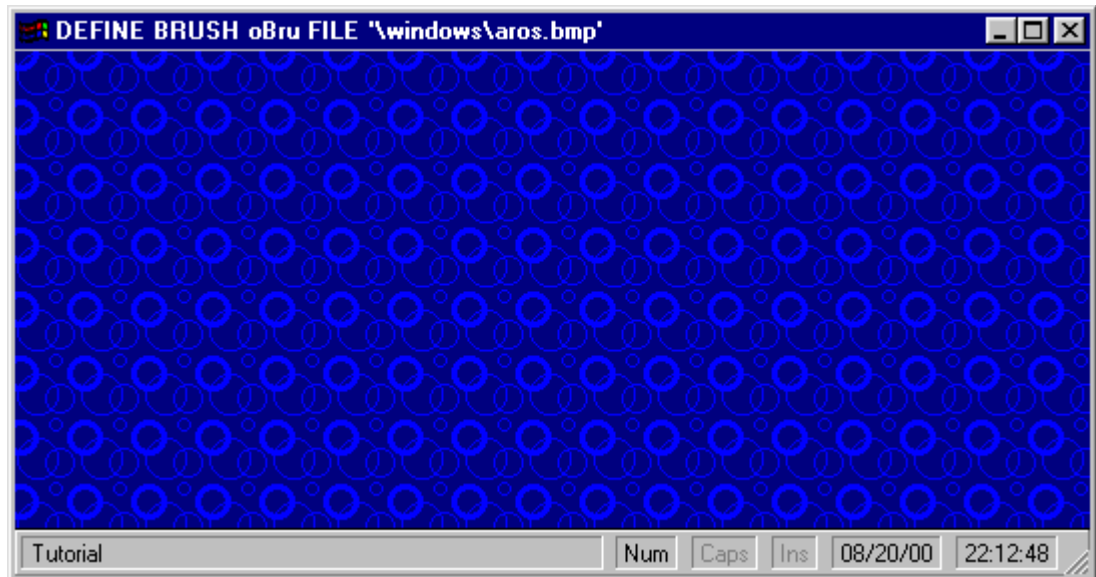


Ahora veamos una ventana con la indicación de color (se ha quitado STYLE) en este caso verde **nRGB(0,255,0)**



Ahora veamos ejemplos con BMP. Hay que hacer constar que si las imágenes son mas pequeñas que las ventanas, automáticamente se hace un mosaico. Si es una fotografía, hay que tener en cuenta el tamaño para poder ajustarla manualmente, puesto que aquí NO existe la cláusula ADJUST como en otros comandos.





Existe tambien el estilo NULL, que hace un fondo transparente. No es muy útil, pero si curioso y quizás te pueda servir para algo.

Conclusión

BRUSH es un elemento meramente decorativo. Pero ¿ Qué es Windows ? sino un sistema "bonito". Hemos de sacar el máximo provecho a nuestras aplicaciones. La mayoría de nuestros clientes si ven una aplicación bonita, probablemente no entrarán en detalles, o al menos perdonarán algunas imperfecciones.

BTNBMP

Concepto

Este comando o recurso es el típico botón que muestra en su interior un BMP (no confundir con BUTTON que muestra un texto). No confundir con los botones que muestran iconos. El botón puede ser de cualquier tamaño, para poder adaptarse a la imagen.

Se puede controlar por programa o por recursos con el workshop.

Si el BMP es mas pequeño que el botón, se centra en éste, pero NO se hace mosaico. Si es mas grande la imagen que el botón se visualiza la parte central.

Echando una ojeada al cuadro resumen de comandos y cláusulas, podemos apreciar entre BTNBMP y BUTTON las siguientes diferencias:

CLAUSULA	BTNBMP	BUTTON
FONT	SI	NO
HELP	NO	SI
NOBORDER	SI	NO
ON CHANGE	NO	SI
ON DROP	NO	SI
SIZES	PRG	NO
TOOLTIP	RECUR.	SI
CANCEL	NO	SI
GROUP	NO	SI

El resto de las opciones son las mismas.

Por programa...

fivewin.ch

```
#xcommand @ <nRow>, <nCol> BTNBMP [<oBtn>] ;
[ <resource: NAME, RESNAME, RESOURCE> <cResName1> ;
  [, <cResName2> [, <cResName3>] ] ] ;
[ <file: FILE, FILENAME, DISK> <cBmpFile1> ;
  [, <cBmpFile2> [, <cBmpFile3>] ] ] ;
[ SIZE <nWidth>, <nHeight> ] ;
[ ACTION <uAction, ...> ] ;
[ <of: OF, WINDOW, DIALOG> <oWnd> ] ;
[ MESSAGE <cMsg> ] ;
[ WHEN <uWhen> ] ;
[ <adjust: ADJUST> ] ;
```

```

[ <lUpdate: UPDATE> ] ;
[ PROMPT <cPrompt> ] ;
[ FONT <oFont> ] ;
[ <lNoBorder: NOBORDER> ] ;
=> ;
[ <oBtn> := ] TBtnBmp():New( <nRow>, <nCol>, <nWidth>, <nHeight>;
    <cResName1>, <cResName2>, <cBmpFile1>, <cBmpFile2>;
    [{|Self|<uAction>}], <oWnd>, <cMsg>, <{uWhen}>, <.adjust.>;
    <.lUpdate.>, <cPrompt>, <oFont>, [<cResName3>], [<cBmpFile3>],
    !<.lNoBorder.> )

```

Comando

```

@ nRow, nCol BTNBMP oBtn
NAME, RESNAME, RESOURCE ResName1[,cResName2[,cResName3] ]
FILE, FILENAME, DISK cBmpFile1[,cBmpFile2[,cBmpFile3] ]
SIZE nWidth, nHeight
ACTION uAction,...
OF, WINDOW, DIALOG oWnd
MESSAGE cMsg
WHEN uWhen
ADJUST
UPDATE
PROMPT cPrompt
FONT oFont
NOBORDER

```

Cláusulas

- **@** irá seguido de las coordenadas de posición. Los valores son siempre en píxel. No existe la cláusula PÍXEL como en otros comandos similares.
- **ACTION** será la función que se ejecuta al hacer clic sobre el control.
- **ACJUST** permite al operador modificar el tamaño y posición del control.
- **FILE** indica el fichero relacionado con el control.
- **FONT** indica el objeto fuente, en caso de ser distinto al indicado en momento del diseño con el workshop.
- **MESSAGE** es el mensaje que aparecerá en la barra de mensajes cuando el cursor pase por encima del control.
- **NAME**
- **NO BORDER**
- **OF, WINDOWS, DIALOG** es el marco que lo contine. Normalmente será una caja de diálogo.
- **PROMPT, VAR** seguidas del texto literal o el nombre de la variable que lo contiene. El font y el color será el indicado en las cláusulas **FONT** y **COLOR**. En caso de no tenerlas, podrá los valores por defecto del marco o ventana que lo contiene.
- **SIZE** indica el tamaño en pixels del campo que ocupará el texto. No indica el campo del texto, que eso esta marcado por el propio texto y el font utilizado.

Este tamaño se utiliza, por ejemplo para el color de fondo, el borde o la superposición sobre otros controles.

- **UPDATE** se utiliza para actualizar el texto cuando se refresca la caja o ventana de diálogo por medio de la instrucción <objeto diálogo>:refresh().
- **WHEN** es la función que debe devolver .T. para que el control pueda tomar el foco. Es similar a la versión en DOS

Ejemplos

Por recursos...

fivewin.ch

```
#xcommand REDEFINE BTNBMP [<oBtn>] ;
[ ID <nId> ] ;
[ <bar: OF, BUTTONBAR > <oBar> ] ;
[ <resource: NAME, RESNAME, RESOURCE> <cResName1> ;
  [, <cResName2>[, <cResName3>] ] ] ;
[ <file: FILE, FILENAME, DISK> <cBmpFile1> ;
  [, <cBmpFile2>[, <cBmpFile3>] ] ] ;
[ <action: ACTION, EXEC, ON CLICK> <uAction, ...> ] ;
[ MESSAGE <cMsg> ] ;
[ <adjust: ADJUST > ] ;
[ WHEN <uWhen> ] ;
[ <lUpdate: UPDATE> ] ;
[ TOOLTIP <cToolTip> ] ;
[ PROMPT <cPrompt> ] ;
[ FONT <oFont> ] ;
[ <lNoBorder: NOBORDER> ] ;
=> ;
[ <oBtn> := ] TBtnBmp():ReDefine( <nId>, <cResName1>, <cResName2>;
  <cBmpFile1>, <cBmpFile2>, <cMsg>, [{|Self|<uAction>}],;
  <oBar>, <.adjust.>, <{uWhen}>, <.lUpdate.>, <cToolTip>;
  <cPrompt>, <oFont>, [<cResName3>], [<cBmpFile3>], [!<.lNoBorder.>] )
```

Workshop

Comando

```
REDEFINE BTNBMP oBtn
  ID nId
  OF, BUTTONBAR oBar
  NAME, RESNAME, RESOURCE cResName1[, cResName2[, cResName3] ]
  FILE, FILENAME, DISK cBmpFile1[, cBmpFile2[, cBmpFile3] ]
  ACTION, EXEC, ON CLICK uAction, ...
  MESSAGE cMsg
  ADJUST
  WHEN uWhen
  UPDATE
  TOOLTIP cToolTip
  PROMPT cPrompt
```

FONT oFont
NOBORDER

Cláusulas

- **ACTION** será la función que se ejecuta al hacer clic sobre el control.
- **ACJUST** permite al operador modificar el tamaño y posición del control.
- **FILE** indica el fichero relacionado con el control.
- **FONT** indica el objeto fuente, en caso de ser distinto al indicado en momento del diseño con el workshop.
- **ID**
- **MESSAGE** es el mensaje que aparecerá en la barra de mensajes cuando el cursor pase por encima del control.
- **NAME**
- **NO BORDER**
- **OF, WINDOWS, DIALOG** es el marco que lo contine. Normalmente será una caja de diálogo.
- **PROMPT, VAR** seguidas del texto literal o el nombre de la variable que lo contiene. El font y el color será el indicado en las cláusulas **FONT** y **COLOR**. En caso de no tenerlas, podrá los valores por defecto del marco o ventana que lo contiene.
- **TOOTIP** indica el pequeño texto de ayuda que sale junto al control al dener unos segundos el cursor sobre él.
- **UPDATE** se utiliza para actualizar el texto cuando se refresca la caja o ventana de diálogo por medio de la instrucción <objeto diálogo>.refresh().
- **WHEN** es la función que debe devolver .T. para que el control pueda tomar el foco. Es similar a la versión en DOS

Ejemplos

Trucos

Conclusión

Se utiliza para hacer botones con imágenes. Ver el comando **BUTTON** como complemento a **BTNBMP**.

BUTTON

Concepto

Este comando o recurso es el típico botón que muestra en su interior un texto (no confundir con BTNBMP que muestra una imagen).

Se puede controlar por programa o por recursos con el workshop.

Echando una ojeada al cuadro resumen de comandos y cláusulas, podemos apreciar entre **BTNBMP** y **BUTTON** las siguientes diferencias:

CLAUSULA	BTNBMP	BUTTON
FONT	SI	NO
HELP	NO	SI
NOBORDER	SI	NO
ON CHANGE	NO	SI
ON DROP	NO	SI
SIZES	PRG	NO
TOOLTIP	RECUR.	SI
CANCEL	NO	SI
GROUP	NO	SI

El resto de las opciones son las mismas.

Por programa...

fivewin.ch

```
#xcommand @ <nRow>, <nCol> BUTTON [ <oBtn> PROMPT ] <cCaption> ;
    [ SIZE <nWidth>, <nHeight> ] ;
    [ ACTION <uAction> ] ;
    [ <default: DEFAULT> ] ;
    [ <of:OF, WINDOW, DIALOG> <oWnd> ] ;
    [ <help:HELP, HELPID, HELP ID> <nHelpId> ] ;
    [ FONT <oFont> ] ;
    [ <pixel: PIXEL> ] ;
    [ <design: DESIGN> ] ;
    [ MESSAGE <cMsg> ] ;
    [ <update: UPDATE> ] ;
    [ WHEN <WhenFunc> ] ;
    [ VALID <uValid> ] ;
    [ <lCancel: CANCEL> ] ;
=> ;
[ <oBtn> := ] TButton():New( <nRow>, <nCol>, <cCaption>, <oWnd>;
```

```
<{uAction}>, <nWidth>, <nHeight>, <nHelpId>, <oFont>, <.default.>,;  
<.pixel.>, <.design.>, <cMsg>, <.update.>, <{WhenFunc}>,;  
<{uValid}>, <.lCancel.> )
```

Comando

```
@ nRow, nCol BUTTON oBtn  
  PROMPT cCaption  
  SIZE nWidth, nHeight  
  ACTION uAction  
  DEFAULT  
  OF, WINDOW, DIALOG oWnd  
  HELP, HELPID, HELP ID nHelpId  
  FONT oFont  
  PIXEL  
  DESIGN  
  MESSAGE cMsg  
  UPDATE  
  WHEN WhenFunc  
  VALID uValid  
  CANCEL
```

Cláusulas

- **@** irá seguido de las coordenadas de posición. Si no se especifica la cláusula **PIXEL**, los números serán línea y columna. Indicando **PÍXEL**, los valores serán en pixels..
- **ACTION** será la función que se ejecuta al hacer clic sobre el control. Si se quieren hacer mas de una, separarlas por comas y encerrarlas entre paréntesis.
- **CANCEL** permite al usuario cancelar la operación del control.
- **DEFAULT** marcará el botón como **por defecto**. Esto implica que al pulsar Intro en cualquier parte de la caja de diálogo, aunque no tenga el foco, se ejecutará la o las funciones **ACTION** de este botón. Naturalmente solo puede haber un botón con esta cláusula.
- **DESIGN** añade un borde al control, redimensionable en tiempo de ejecución. También permite que el control se pueda mover con el ratón.
- **FONT** indica el objeto fuente, en caso de ser distinto al indicado en momento del diseño con el workshop.
- **HELP**
- **MESSAGE** es el mensaje que aparecerá en la barra de mensajes cuando el cursor pase por encima del control.
- **OF, WINDOWS, DIALOG** es el marco que lo contine. Normalmente será una caja de diálogo.
- **PIXEL, PIXELS**. Normalmente la posición se indica en líneas y columnas, y el tamaño en píxels. Con esta cláusula, todo será en píxels
- **PROMPT, VAR** seguidas del texto literal o el nombre de la variable que lo contiene. El font y el color será el indicado en las cláusulas **FONT** y **COLOR**. En

caso de no tenerlas, podrá los valores por defecto del marco o ventana que lo contiene.

- **SIZE** indica el tamaño en pixels del campo que ocupará el texto. No indica el campo del texto, que eso esta marcado por el propio texto y el font utilizado. Este tamaño se utiliza, por ejemplo para el color de fondo, el borde o la superposición sobre otros controles.
- **UPDATE** se utiliza para actualizar el texto cuando se refresca la caja o ventana de diálogo por medio de la instrucción <objeto diálogo>.refresh().
- **VALID** es la función que debe devolver .T. para que el control pueda dejar el foco y continuar con el programa. Es similar a la versión en DOS
- **WHEN** es la función que debe devolver .T. para que el control pueda tomar el foco. Es similar a la versión en DOS

Ejemplos

Por recursos...

fivewin.ch

```
#xcommand DEFINE BUTTON [ <oBtn> ] ;
    [ <bar: OF, BUTTONBAR > <oBar> ] ;
    [ <resource: NAME, RESNAME, RESOURCE> <cResName1> ;
      [, <cResName2>[, <cResName3>] ] ] ;
    [ <file: FILE, FILENAME, DISK> <cBmpFile1> ;
      [, <cBmpFile2>[, <cBmpFile3>] ] ] ;
    [ <action: ACTION, EXEC> <uAction, ...> ] ;
    [ <group: GROUP > ] ;
    [ MESSAGE <cMsg> ] ;
    [ <adjust: ADJUST > ] ;
    [ WHEN <WhenFunc> ] ;
    [ TOOLTIP <cToolTip> ] ;
    [ <lPressed: PRESSED> ] ;
    [ ON DROP <bDrop> ] ;
    [ AT <nPos> ] ;
    [ PROMPT <cPrompt> ] ;
    [ FONT <oFont> ] ;
    [ <lNoBorder: NOBORDER> ] ;
    [ MENU <oPopup> ] ;
=> ;
[ <oBtn> := ] TBtnBmp():NewBar( <cResName1>, <cResName2>;
  <cBmpFile1>, <cBmpFile2>, <cMsg>, [{|This|<uAction>}];
  <.group.>, <oBar>, <.adjust.>, <{WhenFunc}>;
  <cToolTip>, <.lPressed.>, [{|<bDrop>|}], [\'<uAction>\'], <nPos>;
  <cPrompt>, <oFont>, [<cResName3>], [<cBmpFile3>], [!<.lNoBorder.>];
  [<oPopup>] )

#xcommand REDEFINE BUTTON [ <oBtn> ] ;
    [ ID <nId> [ <of: OF, WINDOW, DIALOG> <oDlg> ] ] ;
    [ ACTION <uAction, ...> ] ;
    [ <help: HELP, HELPID, HELP ID> <nHelpId> ] ;
    [ MESSAGE <cMsg> ] ;
    [ <update: UPDATE> ] ;
```

```

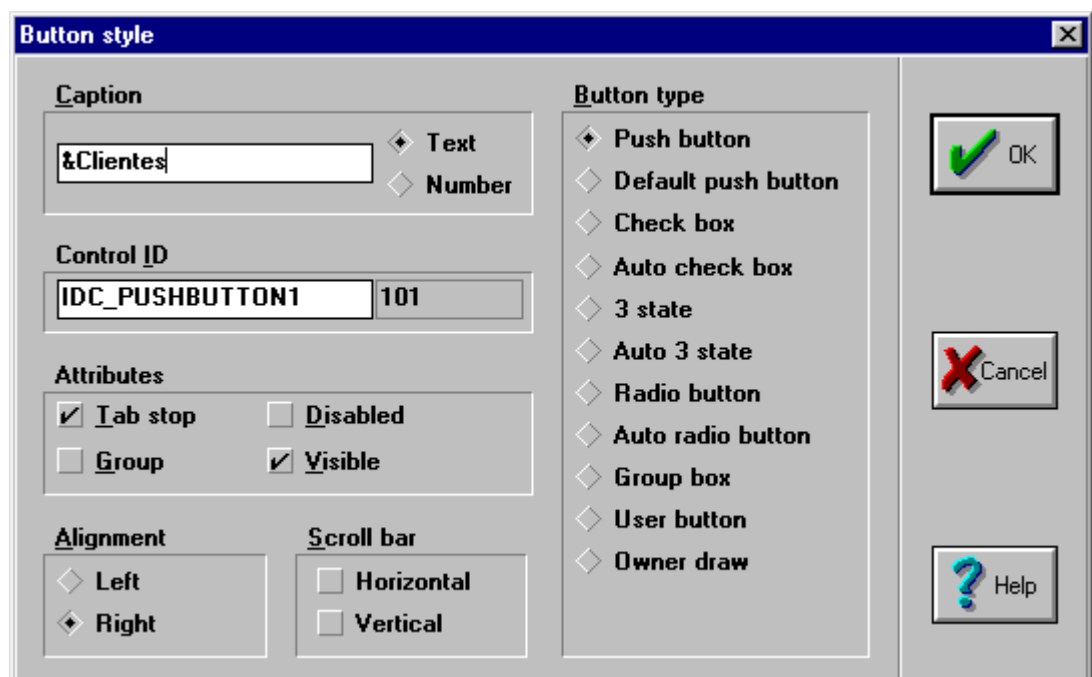
[ WHEN <WhenFunc> ] ;
[ VALID <uValid> ] ;
[ PROMPT <cPrompt> ] ;
[ <lCancel: CANCEL> ] ;
=> ;
[ <oBtn> := ] TButton():ReDefine( <nId>, [\{|<uAction>\}], <oDlg>;
  <nHelpId>, <cMsg>, <.update.>, <{WhenFunc}>, <{uValid}>;
  <cPrompt>, <.lCancel.> )

```

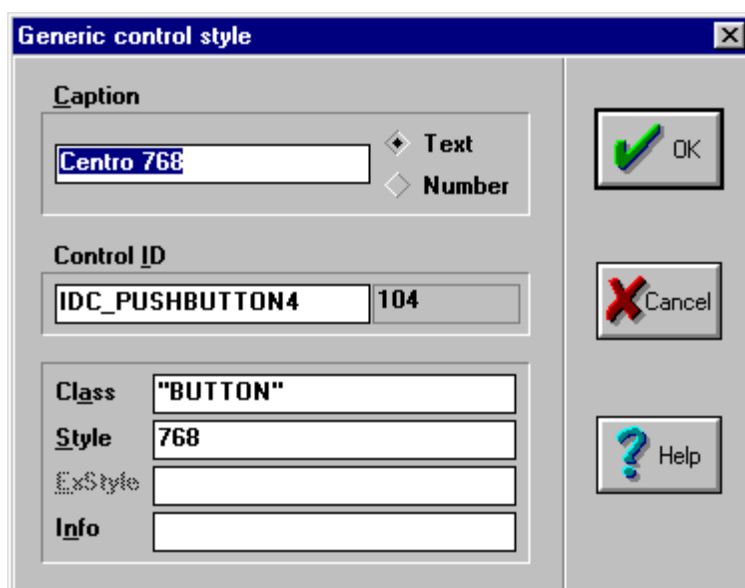
Workshop

Desde el WS el recurso corresponde a las herramientas **1-B** y **4-D**. Con 1-B se puede dimensionar en tiempo de diseño, pero con 4-D el tamaño es fijo. Al contrario que desde el programa, con recursos se puede insertar un BMP en el botón.

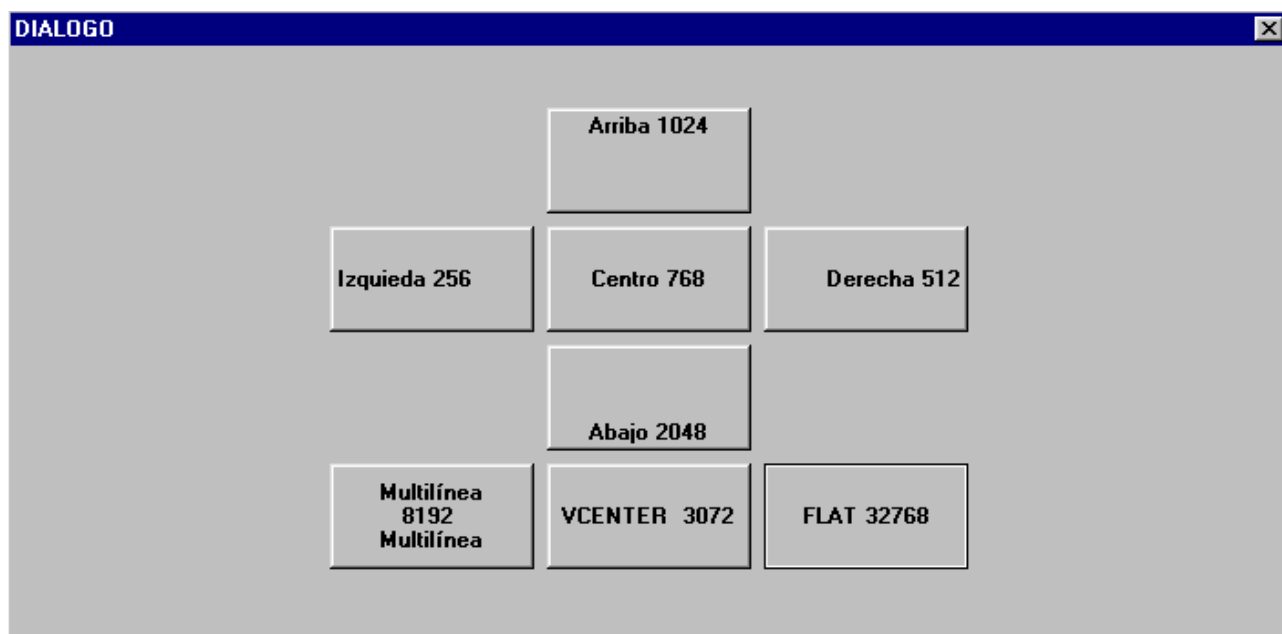
Eligiendo el botón estándar 1-B el cuadro de propiedades es el siguiente:



En **Caption** se ha puesto el texto que irá dentro del botón. El signo **&**, como siempre, indica el acelerador, de modo que para activar el botón no es necesario pinchar sobre él. En el ejemplo, es suficiente pulsar **Alt-C**.



Los estilos del botón se pueden controlar de la siguiente forma. Pulsa dos veces en el botón manteniendo la tecla Control. Esto hará aparecer la ventana indicada a la izquierda. Pon en Style el número que indico en la imagen de más abajo, para posicionar el texto. Los estilos pueden mezclarse, separando los números por el carácter |. Naturalmente hay estilos excluyentes.



Si se elige el botón estilo Borlan **4-D**, estos pueden ser de tres tipos: Predefinidos, Texto, y Gráficos.

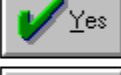
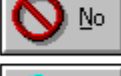
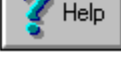
Los botones predefinido se consiguen poniendo en ID el valor indicado en el gráfico siguiente. Las opciones, según la ayuda del propio WS, son:



Borland Button Style Values

[See Also](#)

These are the values for the predefined Borland-style bitmap buttons:

ID Name	ID Value	Button
IDOK	1	 OK
IDCANCEL	2	 Cancel
IDABORT	3	 Abort
IDRETRY	4	 Retry
IDIGNORE	5	 Ignore
IDYES	6	 Yes
IDNO	7	 No
IDHELP	998	 Help

Como puede verse, en español el único válido podría ser el 1, de todas formas vamos a ver como se hacen. Al crear el recurso, cambiar el **ID value** al número indicado en la imagen de la ayuda, es decir, 1 para OK, 2 para Cancel, etc. La ventaja de estos botones es que ya tienen los dibujos de Inactivo, con Foco y Pulsado. Tienen un tamaño fijo de 62 x 40 y no se pueden redimensionar.

Los valores que no correspondan a ninguno de los aquí señalados, harán que el botón se comporte como uno normal de texto.

Para hacer botones con tres gráficos (Inactivo, Foco y Pulsado), hacer lo siguiente:

1. Crear tres BMP que serán los dibujos que tendrá el botón para los tres estados.
2. Importar los BMP al workshop modificando el ID según el punto siguiente.
3. Si por ejemplo el ID del botón es 101, ponel el ID de cada BMP antemoniando 1,2 y 5, de modo que serán 1101, 1101 y 5101.

De los tres tipos de BMP solo el primero es obligatorio, de forma si no están los otros dos, se muestran como el primero. Aunque no aparece texto, los botones siguen manteniendo el acelerador que se puso en **Caption** al crearlo.

Comando

```

DEFINE BUTTON oBtn
  OF, BUTTONBAR oBar
  NAME, RESNAME, RESOURCE cResName1 [,cResName2[,cResName3] ]
  FILE, FILENAME, DISK cBmpFile1 [,cBmpFile2[,cBmpFile3] ]
  ACTION,EXEC uAction,...
  GROUP
  MESSAGE cMsg
  ADJUST
  WHEN WhenFunc
  TOOLTIP cToolTip
  PRESSED
  ON DROP bDrop
  AT nPos
  PROMPT cPrompt
  FONT oFont
  NOBORDER
  MENU oPopup

REDEFINE BUTTON oBtn
  ID nId
  OF, WINDOW, DIALOG oDlg
  ACTION uAction,...
  HELP, HELPID, HELP ID nHelpId
  MESSAGE cMsg
  UPDATE
  WHEN WhenFunc
  VALID uValid
  PROMPT cPrompt
  CANCEL

```

Cláusulas

- **ACTION** será la función que se ejecuta al hacer clic sobre el control.

- **ACJUST** permite al operador modificar el tamaño y posición del control.
- **CANCEL** permite al usuario cancelar la operación del control.
- **FILE** indica el fichero relacionado con el control.
- **FONT** indica el objeto fuente, en caso de ser distinto al indicado en momento del diseño con el workshop.
- **GROUP** indica que pertenece a un grupo distinto y el dibujo se separa un poco del resto de botones especificados.
- **ID** indica el indicador en workshop. Puede ser una variable definida.
- **MENU**
- **MESSAGE** es el mensaje que aparecerá en la barra de mensajes cuando el cursor pase por encima del control.
- **NAME**
- **OF, WINDOWS, DIALOG** es el marco que lo contine. Normalmente será una caja de diálogo.
- **ON DROP**
- **PROMPT, VAR** seguidas del texto literal o el nombre de la variable que lo contiene. El font y el color será el indicado en las cláusulas **FONT** y **COLOR**. En caso de no tenerlas, podrá los valores por defecto del marco o ventana que lo contiene.
- **TOOTIP** indica el pequeño texto de ayuda que sale junto al control al dener unos segundos el cursor sobre él.
- **UPDATE** se utiliza para actualizar el texto cuando se refresca la caja o ventana de diálogo por medio de la instrucción <objeto diálogo>:refresh().
- **VALID** es la función que debe devolver .T. para que el control pueda dejar el foco y continuar con el programa. Es similar a la versión en DOS
- **WHEN** es la función que debe devolver .T. para que el control pueda tomar el foco. Es similar a la versión en DOS

Ejemplos

Trucos

Conclusión

BUTTONBAR

Concepto

Con este comando se define una barra de botones en una ventana. El elemento construido será el contenedor de los botones y podrá ser colocado en cualquiera de los cuatro lados de la ventana o incluso formar un recuadro flotante con los botones en su interior.

Una barra de botones debe llevar al menos un botón para que sea práctica, pero puede diseñarse sin ninguno. Realmente la barra solo sirve de contenedor. Todas las funciones de ACTION, etc. han de ser en cada botón concreto. Como cláusula propia de un botón dentro de una barra de botones, esta **GROUP** que indica que debe separarse un poco del botón precedente.

Por programa...

fivewin.ch

```
#xcommand @ <nRow>, <nCol> BUTTONBAR [ <oBar> ] ;
    [ SIZE <nWidth>, <nHeight> ] ;
    [ BUTTONSIZE <nBtnWidth>, <nBtnHeight> ] ;
    [ <_3d: 3D, 3DLOOK, _3DLOOK> ] ;
    [ <mode: TOP, LEFT, RIGHT, BOTTOM, FLOAT> ] ;
    [ <wnd: OF, WINDOW, DIALOG> <oWnd> ] ;
    [ CURSOR <oCursor> ] ;
=> ;
    [ <oBar> := ] TBar():NewAt( <nRow>, <nCol>, <nWidth>, <nHeight>;
        <nBtnWidth>, <nBtnHeight>, <oWnd>, <._3d.>, [ Upper(<(mode)>) ],;
        <oCursor> )
```

Comando

```
@ nRow, nCol BUTTONBAR oBar
SIZE nWidth, nHeight
BUTTONSIZE nBtnWidth, nBtnHeight
_3D, 3D, 3DLOOK, _3DLOOK
TOP, LEFT, RIGHT, BOTTOM, FLOAT
OF, WINDOW, DIALOG oWnd
CURSOR oCursor
```

Cláusulas

- **@** irá seguido de las coordenadas de posición. Si no se especifica la cláusula PIXEL, los números serán línea y columna. Indicando PÍXEL, los valores serán en pixels..
- **3D** sirve para dar aspecto de tres dimensiones. Sin esto queda muy fea.
- **BUTTONSIZE** es el tamaño del botón en pixels.
- **CURSOR** indica que cambie el cursor cuando este pase por encima.
- **OF, WINDOWS, DIALOG** es el marco que lo contine. Normalmente será una caja de diálogo.
- **SIZE** indica el tamaño en pixels de los botones contenidos en la barra.
- **TOP/LEFT/RIGHT/BOTTOM/FLOAT** se utiliza para colocar inicialmente la barra. Normalmente será **TOP**. Posteriormente, en tiempo de ejecución el operador puede modificar su colocación pulsando con el botón derecho sobre la barra.

Ejemplos

Por recursos...

fivewin.ch

```
#xcommand DEFINE BUTTONBAR [ <oBar> ] ;
    [ <size: SIZE, BUTTONSIZE, SIZEBUTTON > <nWidth>, <nHeight> ] ;
    [ <_3d: _3D, 3D, 3DLOOK, _3DLOOK> ] ;
    [ <mode: TOP, LEFT, RIGHT, BOTTOM, FLOAT> ] ;
    [ <wnd: OF, WINDOW, DIALOG> <oWnd> ] ;
    [ CURSOR <oCursor> ] ;
=> ;
    [ <oBar> := ] TBar():New( <oWnd>, <nWidth>, <nHeight>, <._3d.>,;
    [ Upper(<(mode)>) ], <oCursor> )
```

Workshop

Comando

```
DEFINE BUTTONBAR oBar
    SIZE, BUTTONSIZE, SIZEBUTTON nWidth, nHeight
    _3D, 3D, 3DLOOK, _3DLOOK
    TOP, LEFT, RIGHT, BOTTOM, FLOAT
    OF, WINDOW, DIALOG oWnd
    CURSOR oCursor
```

Cláusulas

- **3D** sirve para dar aspecto de tres dimensiones. Sin esto queda muy fea.
- **CURSOR** indica que cambie el cursor cuando éste pase por encima.
- **OF, WINDOWS, DIALOG** es el marco que lo contine. Normalmente será una caja de diálogo.
- **SIZE** indica el tamaño en pixels de los botones contenidos en la barra.
- **TOP/LEFT/RIGHT/BOTTOM/FLOAD**

Ejemplos

Trucos

Conclusión

CHECKBOX

Concepto

Es una pequeña marca que se hace para indicar que una condición esta **activa** o **inactiva**, **cierta** o **falsa**, **si** o **no**, etc. es decir, cualquier situación que admita dos estados.

Puede hacerse por programa o por recursos. Si se hace con el WS se pueden definir varios tipos de marcos para la señal (cuadrado o tipo rombo)

El CHECKBOX siempre tiene asociado una variable lógica. Cuando se inicia el programa esta variable tendrá .T. o .F. y según esto, aparecerá o no la marca. Una vez que el operador cambia la marca, también cambia el contenido de la variable.

Mediante la cláusula ON CLICK podemos actuar directamente sobre el programa sin esperar a salir del CHECKBOX. Al hacer click sobre el checkbox, se ejecutará la función indicada. También podemos controlar el programa por medio de la cláusula VALID.

Por programa...

fivewin.ch

```
#xcommand @ <nRow>, <nCol> CHECKBOX [ <oCbx> VAR ] <lVar> ;
    [ PROMPT <cCaption> ] ;
    [ <of:OF, WINDOW, DIALOG> <oWnd> ] ;
    [ SIZE <nWidth>, <nHeight> ] ;
    [ <help:HELPID, HELP ID> <nHelpId> ] ;
    [ FONT <oFont> ] ;
    [ <change: ON CLICK, ON CHANGE> <uClick> ] ;
    [ VALID <ValidFunc> ] ;
    [ <color: COLOR, COLORS> <nClrFore> [,<nClrBack>] ] ;
    [ <design: DESIGN> ] ;
    [ <pixel: PIXEL> ] ;
    [ MESSAGE <cMsg> ] ;
    [ <update: UPDATE> ] ;
    [ WHEN <WhenFunc> ] ;
=> ;
[ <oCbx> := ] TCheckBox():New( <nRow>, <nCol>, <cCaption>;
    [bSETGET(<lVar>)], <oWnd>, <nWidth>, <nHeight>, <nHelpId>;
    [<{uClick}>], <oFont>, <{ValidFunc}>, <nClrFore>, <nClrBack>;
    <.design.>, <.pixel.>, <cMsg>, <.update.>, <{WhenFunc}> )
```

Comando

```
@ nRow, nCol CHECKBOX oCbx
VAR lVar
PROMPT cCaption
OF, WINDOW, DIALOG oWnd
```

```

SIZE nWidth, nHeight
HELPID, HELP ID nHelpId
FONT oFont
ON CLICK, ON CHANGE uClick
VALID ValidFunc
COLOR, COLORS nClrFore [,nClrBack
DESIGN
PIXEL
MESSAGE cMsg
UPDATE
WHEN WhenFunc

```

Cláusulas

- **@** irá seguido de las coordenadas de posición. Si no se especifica la cláusula **PIXEL**, los números serán línea y columna. Indicando **PÍXEL**, los valores serán en pixels..
- **COLOR, COLORS** indica el color del texto. Si se pone un segundo color, será el del fondo. El color se define mediante la función **nRGB()** o similar.
- **DESIGN** añade un borde al control, redimensionable en tiempo de ejecución. También permite que el control se pueda mover con el ratón.
- **FONT** indica el objeto fuente, en caso de ser distinto al indicado en momento del diseño con el workshop.
- **HELP**
- **MESSAGE** es el mensaje que aparecerá en la barra de mensajes cuando el cursor pase por encima del control.
- **OF, WINDOWS, DIALOG** es el marco que lo contine. Normalmente será una caja de diálogo.
- **PIXEL, PIXELS**. Normalmente la posición se indica en líneas y columnas, y el tamaño en píxels. Con esta cláusula, todo será en pixels
- **PROMPT, VAR** seguidas del texto literal o el nombre de la variable que lo contiene. El font y el color será el indicado en las cláusulas **FONT** y **COLOR**. En caso de no tenerlas, podrá los valores por defecto del marco o ventana que lo contiene.
- **SIZE** indica el tamaño en pixels del campo que ocupará el texto. No indica el campo del texto, que eso esta marcado por el propio texto y el font utilizado. Este tamaño se utiliza, por ejemplo para el color de fondo, el borde o la superposición sobre otros controles.
- **UPDATE** se utiliza para actualizar el texto cuando se refresca la caja o ventana de diálogo por medio de la instrucción <objeto diálogo>:refresh().
- **VALID** es la función que debe devolver .T. para que el control pueda dejar el foco y continuar con el programa. Es similar a la versión en DOS
- **VAR** es la variable lógica asociada al control. Tendrá inicialmente uno de los valores .T. o .F. Al cambiar el usuario la marca, cambia el contenido de esta variable.

- **WHEN** es la función que debe devolver .T. para que el control pueda tomar el foco. Es similar a la versión en DOS

Ejemplos

Por recursos...

fivewin.ch

```
#xcommand REDEFINE CHECKBOX [ <oCbx> VAR ] <lVar> ;
    [ ID <nId> ] ;
    [ <of:OF, WINDOW, DIALOG> <oWnd> ] ;
    [ <help:HELPID, HELP ID> <nHelpId> ] ;
    [ <click:ON CLICK, ON CHANGE> <uClick> ] ;
    [ VALID <uValid> ] ;
    [ <color: COLOR, COLORS> <nClrFore> [,<nClrBack>] ] ;
    [ MESSAGE <cMsg> ] ;
    [ <update: UPDATE> ] ;
    [ WHEN <uWhen> ] ;
=> ;
[ <oCbx> := ] TCheckBox():ReDefine( <nId>, bSETGET(<lVar>),,
    <oWnd>, <nHelpId>, [<{uClick}>], <{uValid}>, <nClrFore>,
    <nClrBack>, <cMsg>, <.update.>, <{uWhen}> )
```

Workshop

Con WS podemos elegir dos tipos de checkbox. El icono 3-C o el 6-D. El primero es el normal. El segundo es de estilo Borlan. El funcionamiento en los dos es el mismo, cambiando únicamente el aspecto.

Comando

```
REDEFINE CHECKBOX oCbx
VAR lVar
ID nId
OF, WINDOW, DIALOG oWnd
HELPID, HELP ID nHelpId
ON CLICK, ON CHANGE uClick
VALID uValid
COLOR, COLORS nClrFore [,nClrBack]
MESSAGE cMsg
UPDATE
WHEN uWhen
```

Cláusulas

- **COLOR, COLORS** indica el color del texto. Si se pone un segundo color, será el del fondo. El color se define mediante la función nRGB() o similar.
- **HELP**
- **ID** indica el indicador en workshop. Puede ser una variable definida.
- **MESSAGE** es el mensaje que aparecerá en la barra de mensajes cuando el cursor pase por encima del control.
- **OF, WINDOWS, DIALOG** es el marco que lo contine. Normalmente será una caja de diálogo.
- **UPDATE** se utiliza para actualizar el texto cuando se refresca la caja o ventana de diálogo por medio de la instrucción <objeto diálogo>:refresh().
- **VALID** es la función que debe devolver .T. para que el control pueda dejar el foco y continuar con el programa. Es similar a la versión en DOS
- **VAR** es la variable lógica asociada al control. Tendrá inicialmente uno de los valores .T. o .F. Al cambiar el usuario la marca, cambia el contenido de esta variable.
- **WHEN** es la función que debe devolver .T. para que el control pueda tomar el foco. Es similar a la versión en DOS

Ejemplos

Trucos

Conclusión

CLIPBOARD

Concepto

Por programa...

fivewin.ch

```
#xcommand DEFINE CLIPBOARD <oClp> ;  
    [ FORMAT <format:TEXT,OEMTEXT,BITMAP,DIF> ] ;  
    [ OF <oWnd> ] ;  
=> ;  
    <oClp> := TClipboard():New( [Upper(<(format)>)], <oWnd> )  
  
#xcommand ACTIVATE CLIPBOARD <oClp> => <oClp>:Open()
```

Comando

Cláusulas

Ejemplos

Help

Conclusión

COMBOBOX

Concepto

Por programa...

fivewin.ch

```
#xcommand @ <nRow>, <nCol> COMBOBOX [ <oCbx> VAR ] <cVar> ;
    [ <it: PROMPTS, ITEMS> <aItems> ] ;
    [ SIZE <nWidth>, <nHeight> ] ;
    [ <dlg:OF,WINDOW,DIALOG> <oWnd> ] ;
    [ <help:HELPID, HELP ID> <nHelpId> ] ;
    [ ON CHANGE <uChange> ] ;
    [ VALID <uValid> ] ;
    [ <color: COLOR,COLORS> <nClrText> [,<nClrBack>] ] ;
    [ <pixel: PIXEL> ] ;
    [ FONT <oFont> ] ;
    [ <update: UPDATE> ] ;
    [ MESSAGE <cMsg> ] ;
    [ WHEN <uWhen> ] ;
    [ <design: DESIGN> ] ;
    [ BITMAPS <acBitmaps> ] ;
    [ ON DRAWITEM <uBmpSelect> ] ;
    [ STYLE <nStyle> ] ;
    [ PICTURE <cPicture> ] ;
    [ ON EDIT CHANGE <uEChange> ] ;
=> ;
    [ <oCbx> := ] TComboBox():New( <nRow>, <nCol>, bSETGET(<cVar>),;
    <aItems>, <nWidth>, <nHeight>, <oWnd>, <nHelpId>;
    [{|Self|<uChange>}], <{uValid}>, <nClrText>, <nClrBack>;
    <.pixel.>, <oFont>, <cMsg>, <.update.>, <{uWhen}>;
    <.design.>, <acBitmaps>, [{|nItem|<uBmpSelect>}], <nStyle>;
    <cPicture>, [<{uEChange}>] )
```

Comando

```
@ nRow, nCol COMBOBOX oCbx
VAR cVar
PROMPTS, ITEMS> aItems
SIZE nWidth, nHeight
OF,WINDOW,DIALOG oWnd
HELPID, HELP ID nHelpId
ON CHANGE uChange
VALID uValid
COLOR,COLORS nClrText[,nClrBack]
PIXEL
FONT oFont
UPDATE
MESSAGE cMsg
WHEN uWhen
```

DESIGN
BITMAPS acBitmaps
ON DRAWITEM uBmpSelect
STYLE nStyle
PICTURE cPicture
ON EDIT CHANGE uEChange

Cláusulas

- **@** irá seguido de las coordenadas de posición. Si no se especifica la cláusula **PIXEL**, los números serán línea y columna. Indicando **PÍXEL**, los valores serán en pixels..
- **COLOR, COLORS** indica el color del texto. Si se pone un segundo color, será el del fondo. El color se define mediante la función **nRGB()** o similar.
- **DESIGN** añade un borde al control, redimensionable en tiempo de ejecución. También permite que el control se pueda mover con el ratón.
- **FONT** indica el objeto fuente, en caso de ser distinto al indicado en momento del diseño con el workshop.
- **HELP**
- **MESSAGE** es el mensaje que aparecerá en la barra de mensajes cuando el cursor pase por encima del control.
- **OF, WINDOWS, DIALOG** es el marco que lo contine. Normalmente será una caja de diálogo.
- **ON CHANGE** indica la función que se ejecutará cuando cambie el contenido o aspecto del control.
- **ON DRAWITEM**
- **ON EDIT**
- **PICTURE** es la máscara igual que en DOS.
- **PIXEL, PIXELS**. Normalmente la posición se indica en líneas y columnas, y el tamaño en píxels. Con esta cláusula, todo será en pixels
- **PROMPT, VAR** seguidas del texto literal o el nombre de la variable que lo contiene. El font y el color será el indicado en las cláusulas **FONT** y **COLOR**. En caso de no tenerlas, podrá los valores por defecto del marco o ventana que lo contiene.
- **SIZE** indica el tamaño en pixels del campo que ocupará el texto. No indica el campo del texto, que eso esta marcado por el propio texto y el font utilizado. Este tamaño se utiliza, por ejemplo para el color de fondo, el borde o la superposición sobre otros controles.
- **STYLE**
- **UPDATE** se utiliza para actualizar el texto cuando se refresca la caja o ventana de diálogo por medio de la instrucción `<objeto diálogo>.refresh()`.
- **VALID** es la función que debe devolver `.T.` para que el control pueda dejar el foco y continuar con el programa. Es similar a la versión en DOS

- **VAR** es la variable asociada al control.
- **WHEN** es la función que debe devolver .T. para que el control pueda tomar el foco. Es similar a la versión en DOS

Ejemplos

Por recursos...

fivewin.ch

```
#xcommand REDEFINE COMBOBOX [ <oCbx> VAR ] <cVar> ;
    [ <items: PROMPTS, ITEMS> <aItems> ] ;
    [ ID <nId> ] ;
    [ <dlg:OF,WINDOW,DIALOG> <oWnd> ] ;
    [ <help:HELPID, HELP ID> <nHelpId> ] ;
    [ ON CHANGE <uChange> ] ;
    [ VALID <uValid> ] ;
    [ <color: COLOR,COLORS> <nClrText> [,<nClrBack>] ] ;
    [ <update: UPDATE> ] ;
    [ MESSAGE <cMsg> ] ;
    [ WHEN <uWhen> ] ;
    [ BITMAPS <acBitmaps> ] ;
    [ ON DRAWITEM <uBmpSelect> ] ;
    [ STYLE <nStyle> ] ;
    [ PICTURE <cPicture> ];
    [ ON EDIT CHANGE <uEChange> ] ;
=> ;
    [ <oCbx> := ] TComboBox():ReDefine( <nId>, bSETGET(<cVar>),;
    <aItems>, <oWnd>, <nHelpId>, <{uValid}>, [{|Self|<uChange>}],;
    <nClrText>, <nClrBack>, <cMsg>, <.update.>, <{uWhen}>,;
    <acBitmaps>, [{|nItem|<uBmpSelect>}], <nStyle>, <cPicture>,;
    [<{uEChange}>] )
```

Workshop

Comando

```
REDEFINE COMBOBOX oCbx
    VAR cVar
    PROMPTS, ITEMS aItems
    ID nId
    OF,WINDOW,DIALOG oWnd
    HELPID, HELP ID nHelpId
    ON CHANGE uChange
    VALID uValid
    COLOR,COLORS nClrText[,nClrBack]
    UPDATE
    MESSAGE cMsg
    WHEN uWhen
    BITMAPS acBitmaps
    ON DRAWITEM uBmpSelect
    STYLE nStyle
    PICTURE cPicture
```

ON EDIT CHANGE uEChange

Cláusulas

- **COLOR, COLORS** indica el color del texto. Si se pone un segundo color, será el del fondo. El color se define mediante la función nRGB() o similar.
- **HELP**
- **ID** indica el indicador en workshop. Puede ser una variable definida.
- **MESSAGE** es el mensaje que aparecerá en la barra de mensajes cuando el cursor pase por encima del control.
- **OF, WINDOWS, DIALOG** es el marco que lo contine. Normalmente será una caja de diálogo.
- **ON CHANGE** indica la función que se ejecutará cuando cambie el contenido o aspecto del control.
- **ON DRAWITEN**
- **ON EDIT**
- **PICTURE** es la máscara igual que en DOS.
- **PROMPT, VAR** seguidas del texto literal o el nombre de la variable que lo contiene. El font y el color será el indicado en las cláusulas **FONT** y **COLOR**. En caso de no tenerlas, podrá los valores por defecto del marco o ventana que lo contiene.
- **STYLE**
- **UPDATE** se utiliza para actualizar el texto cuando se refresca la caja o ventana de diálogo por medio de la instrucción <objeto diálogo>:refresh().
- **VALID** es la función que debe devolver .T. para que el control pueda dejar el foco y continuar con el programa. Es similar a la versión en DOS
- **VAR** es la variable asociada al control.
- **WHEN** es la función que debe devolver .T. para que el control pueda tomar el foco. Es similar a la versión en DOS

Ejemplos

Trucos

Conclusión

CURSOR

Concepto

Un cursor es un pequeño gráfico, normalmente de 32x32 pixel que nos indica la posición en la que se encuentra el ratón dentro de la pantalla. Windows utiliza distintos tipos de cursores para representar las tareas más características (redimensionar, mover, editar, esperar)

En Fivewin, cada ventana puede contener su propio cursor que se nos mostrará al colocar el ratón dentro de su área. Cuando trabajemos con cursores, conviene recordar que en Fivewin prácticamente todo son ventanas y por lo tanto pueden tener su propio cursor asociado (botones, listbox, gets, etc..)

Para asociar un cursor a una ventana lo indicaremos en la cláusula correspondiente al definirla

```
DEFINE WINDOW oWnd;
FROM 5,7 TO 15,75;
TITLE "TITULO DE LA VENTANA";
COLOR "B/R";
STYLE nOr( WS_POPUP, WS_VSCROLL );
BRUSH oBrush;
CURSOR oCursor
```

Windows incorpora una serie de cursores definidos por defecto que debe indicarse al definir el objeto:

```
DEFINE CURSOR <oCursor>;
[RESOURCE | NAME | RESNAME <cResName > ]
[ARROW | ICON | IBEAM | ...]
```



ARROW



IBEAM



WAIT



CROSS



UPARROW



SIZE



ICON

	SIZENWSE
	SIZENESW
	SIZEWS
	SIZENS

Por programa...

fivewin.ch

```
#xcommand DEFINE CURSOR <oCursor> ;  
    [ <resource: RESOURCE, RESNAME, NAME> <cResName> ] ;  
    [ <predef: ARROW, ICON, SIZENS, SIZEWE, SIZENWSE,;  
      SIZENESW, IBEAM, CROSS, SIZE, UPARROW, WAIT, HAND> ] ;  
=> ;  
    <oCursor> := TCursor():New( <cResName>, [ Upper(<(predef)>) ] )
```

Comando

Cláusulas

Ejemplos

Trucos

Conclusión

DATABASE

Concepto

Fivewin nos brinda la oportunidad de manejar las bases de datos como objetos. La principal ventaja es la de tener cada campo accesible directamente sin necesidad de crear variables temporales para editar sus contenidos.

Para pasar una base de datos a objeto basta con escribir **DATABASE oDbf** teniendo una base de datos activa en ese momento, por ejemplo nada mas abrirla. Fivewin se encargará automáticamente de rellenar los datos del objeto sacandolos de los datos internos de clipper.

Asumiendo que el objeto se ha grabado en la variable oDbf, los métodos principales para el manejo de la base de datos son:

oDbf:skip(nRegistros)	Salta para adelante o atrás nRegistros.
oDbf:gotop()	Salta al principio del fichero.
oDbf:gobottom()	Salta al final del fichero
oDbf:eof()	Devuelve .T. o .F. según este al final.
oDbf:bof()	Devuelve .T. o .F. según este al principio.
oDbf:load()	Carga los datos del registro en el buffer.
oDbf:save()	Graba el buffer en el registro.

El resto de métodos puede verse en la transcripción de la ayuda que acompaña a Fivewin.

Para actualizar la caja de diálogo de un registro, el método que usaremos dentro del objeto DIALOGO, será algo similar a **oDlg:update()**. Aquellos controles que tengan la cápsula UPDATE serán actualizados automáticamente.

Por programa...

fivewin.ch

```
#ifndef __XPP__
  #xcommand DATABASE <oDbf> => <oDbf> := TDataBase():New()
#else
  #xcommand DATABASE <oDbf> => TDataBase():New() ;;
                                <oDbf> := ClassObject( Alias() ):New()
#endif
```

Comando

Ejemplos

Trucos

Conclusión

DEFAULT**Concepto****Por programa...****fivewin.ch**

```
#xcommand DEFAULT <uVar1> := <uVal1> ;  
    [, <uVarN> := <uValN> ] => ;  
    <uVar1> := If( <uVar1> == nil, <uVal1>, <uVar1> ) ;;  
    [ <uVarN> := If( <uVarN> == nil, <uValN>, <uVarN> ); ]
```

Comando**Ejemplos****Help**

DIALOG

Concepto

Es uno de los comandos mas importantes debido a que servirá de contenedor para los controles de trabajo. Si no hay caja de diálogo solo podriamos poner controles en la ventana principal y esto limitaria el sistema a una sola pantalla.

Las cajas de diálogo tienen muchas cláusulas para su construcción, así como muchas variables de instancia y métodos para el control del objeto.

Al final de este apartado sobre DIALOG hablaremos un poco sobre el concepto **MODAL** y **NO-MODAL**. No se ha querido mezclar con el resto por tener mucha importancia en el sistema de ventanas Windows. Vaya por delante únicamente que una ventana MODAL es aquella que cuando se activa para trabajar con ella y los controles que contiene, no puede hacerse nada con otra ventana. Una NO-MODAL, permite trabajar con ella y pulsando en (o creando) otra, pasar el foco sin necesidad de cerrarla. Cuando se quiera, se puede volver a ella y continuar donde estabamos. En Windows es similar a lo que se consigue pulsando Alt-tabulador, pero desde dentro de la misma aplicación.

Las cajas de diálogo se distinguen de las ventanas normales en que no tienen botones de minimizar ni maximizar. Tampoco tienen barra de menús, ni de mensajes. Por el contrario puede haber varias cajas de diálogo simultaneas, cada una con sus controles.

La creación de una ventana, tanto si es por programa como por recursos necesita del comando DEFINE. Las cláusulas serán las que determinen si es por un sistema u otro. Es imprescindible declarar la variable donde se guardará el objeto puesto que posteriormente todos los controles harán referencia a dicho objeto.

Una vez creada un diálogo, es necesario activarlo por medio de **ACTIVATE DIALOG**. Este comando también activa los controles contenidos en el diálogo.

Por programa...

dialog.ch

```
#xcommand DEFINE DIALOG <oDlg> ;
[ <resource: NAME, RESNAME, RESOURCE> <cResName> ] ;
[ TITLE <cTitle> ] ;
[ FROM <nTop>, <nLeft> TO <nBottom>, <nRight> ] ;
[ SIZE <nWidth>, <nHeight> ] ;
[ <lib: LIBRARY, DLL> <hResources> ] ;
[ <vbx: VBX> ] ;
[ STYLE <nStyle> ] ;
[ <color: COLOR, COLORS> <nClrText> [, <nClrBack> ] ] ;
[ BRUSH <oBrush> ] ;
[ <of: WINDOW, DIALOG, OF> <oWnd> ] ;
[ <pixel: PIXEL> ] ;
[ ICON <oIco> ] ;
```

```

        [ FONT <oFont> ] ;
        [ <help: HELP, HELPID> <nHelpId> ] ;
=> ;
        <oDlg> = TDialog():New( <nTop>, <nLeft>, <nBottom>, <nRight>;
                               <cTitle>, <cResName>, <hResources>, <.vbX.>, <nStyle>;
                               <nClrText>, <nClrBack>, <oBrush>, <oWnd>, <.pixel.>;
                               <oIco>, <oFont>, <nHelpId>, <nWidth>, <nHeight> )

#xcommand ACTIVATE DIALOG <oDlg> ;
        [ <center: CENTER, CENTERED> ] ;
        [ <NonModal: NOWAIT, NOMODAL> ] ;
        [ WHEN <uWhen> ] ;
        [ VALID <uValid> ] ;
        [ ON [ LEFT ] CLICK <uClick> ] ;
        [ ON INIT <uInit> ] ;
        [ ON MOVE <uMoved> ] ;
        [ ON PAINT <uPaint> ] ;
        [ ON RIGHT CLICK <uRClicked> ] ;
=> ;
        <oDlg>:Activate( <oDlg>:bLClicked [ := {|nRow,nCol,nFlags|<uClick>}],
;
                               <oDlg>:bMoved [ := <{uMoved}> ], ;
                               <oDlg>:bPainted [ := {|hDC,cPS|<uPaint>}],;
                               <.center.>, [ {|Self|<uValid>}],;
                               [ ! <.NonModal.> ], [ {|Self|<uInit>}],;
                               <oDlg>:bRClicked [ :=
{|nRow,nCol,nFlags|<uRClicked>}],;
                               [ {|Self|<uWhen>} ] )

```

Comando

```

DEFINE DIALOG oDlg ;
    TITLE cTitle ;
    FROM nTop, nLeft TO nBottom, nRight ;
    SIZE nWidth, nHeight ;
    VBX ;
    STYLE nStyle ;
    COLOR, COLORS nClrText [,<nClrBack>] ;
    BRUSH oBrush ;
    WINDOW, DIALOG, OF oWnd ;
    PIXEL ;
    ICON oIco ;
    FONT oFont ;
    HELP, HELPID nHelpId

ACTIVATE DIALOG oDlg ;
    CENTER, CENTERED ;
    NOWAIT, NOMODAL ;
    WHEN uWhen ;
    VALID uValid ;
    ON [ LEFT ] CLICK uClick ;
    ON INIT uInit ;
    ON MOVE uMoved ;
    ON PAINT uPaint ;
    ON RIGHT CLICK uRClicked

```

Cláusulas

- **COLOR, COLORS.**-
 - **FONT.**-
 - **FROM / TO.**-
 - **HELP.**-
 - **ICO.**-
 - **OF.**-
 - **PIXEL.**-
 - **SIZE.**-
 - **STYLE.**- Indica el estilo de la ventana. Por ejemplo WS_MAXIMIZE hara la ventana maximizada.
 - **TITLE.**-
 - **VBX.**-
-
- **CENTER.**- Centra la caja entre los lados de la ventana o diálogo que la contiene.
 - **NOWAIT, NOMODAL.**- Activa la caja y el foco no permanence en ella. Se utiliza para hacer diálogos MDI
 - **ON CLICK.**- Función que se ejecuta cuando se pincha en ella directamente (no en un control de ella).
 - **ON INIT.**- Función que se ejecuta al inicializar la caja, una vez comprobada la cláusula **WEND**.
 - **ON MOVE.**-
 - **ON PAINT.**-
 - **ON RIGHT CLICK.**-
 - **VALID.**-
 - **WHEN.**- Función que se ejecuta para comprobar si ha de inicializarse la caja o no.

Ejemplos

Por recursos...

dialog.ch

```
#xcommand DEFINE DIALOG <oDlg> ;
```

```

[ <resource: NAME, RESNAME, RESOURCE> <cResName> ] ;
[ TITLE <cTitle> ] ;
[ FROM <nTop>, <nLeft> TO <nBottom>, <nRight> ] ;
[ SIZE <nWidth>, <nHeight> ] ;
[ <lib: LIBRARY, DLL> <hResources> ] ;
[ <vbx: VBX> ] ;
[ STYLE <nStyle> ] ;
[ <color: COLOR, COLORS> <nClrText> [,<nClrBack> ] ] ;
[ BRUSH <oBrush> ] ;
[ <of: WINDOW, DIALOG, OF> <oWnd> ] ;
[ <pixel: PIXEL> ] ;
[ ICON <oIco> ] ;
[ FONT <oFont> ] ;
[ <help: HELP, HELPID> <nHelpId> ] ;
=> ;
<oDlg> = TDialog():New( <nTop>, <nLeft>, <nBottom>, <nRight>,;
    <cTitle>, <cResName>, <hResources>, <.vbx.>, <nStyle>,;
    <nClrText>, <nClrBack>, <oBrush>, <oWnd>, <.pixel.>,;
    <oIco>, <oFont>, <nHelpId>, <nWidth>, <nHeight> )

#xcommand ACTIVATE DIALOG <oDlg> ;
[ <center: CENTER, CENTERED> ] ;
[ <NonModal: NOWAIT, NOMODAL> ] ;
[ WHEN <uWhen> ] ;
[ VALID <uValid> ] ;
[ ON [ LEFT ] CLICK <uClick> ] ;
[ ON INIT <uInit> ] ;
[ ON MOVE <uMoved> ] ;
[ ON PAINT <uPaint> ] ;
[ ON RIGHT CLICK <uRClicked> ] ;
=> ;
<oDlg>:Activate( <oDlg>:bLClicked [ := {|nRow,nCol,nFlags|<uClick>}],
;
    <oDlg>:bMoved [ := <{uMoved}> ], ;
    <oDlg>:bPainted [ := {|hDC,cPS|<uPaint>}],;
    <.center.>, [|Self|<uValid>}],;
    [ ! <.NonModal.> ], [|Self|<uInit>}],;
    <oDlg>:bRClicked [ :=
{|nRow,nCol,nFlags|<uRClicked>}],;
    [|Self|<uWhen>}] )

```

Workshop

Comando

```

DEFINE DIALOG oDlg ;
    NAME, RESNAME, RESOURCE> <cResName ;
    TITLE cTitle ;
    FROM nTop, nLeft TO nBottom, nRight ;
    SIZE nWidth, nHeight ;
    LIBRARY, DLL hResources ;
    VBX ;
    STYLE nStyle ;
    COLOR, COLORS nClrText [,<nClrBack> ] ;
    BRUSH oBrush ;
    WINDOW, DIALOG, OF oWnd ;
    PIXEL ;
    ICON oIco ;
    FONT oFont ;
    HELP, HELPID nHelpId

```

```
ACTIVATE DIALOG oDlg ;  
  CENTER, CENTERED ;  
  NOWAIT, NOMODAL ;  
  WHEN uWhen ;  
  VALID uValid ;  
  ON [ LEFT ] CLICK uClick ;  
  ON INIT uInit ;  
  ON MOVE uMoved ;  
  ON PAINT uPaint ;  
  ON RIGHT CLICK uRClicked
```

Cláusulas

- **COLOR, COLORS.**-
- **FONT.**-
- **FROM / TO.**-
- **HELP.**-
- **ICO.**-
- **OF.**-
- **PIXEL.**-
- **SIZE.**-
- **STYLE.**- Indica el estilo de la ventana. Por ejemplo WS_MAXIMIZE hara la ventana maximizada.
- **TITLE.**-
- **VBX.**-

- **CENTER.**- Centra la caja entre los lados de la ventana o diálogo que la contiene.
- **NOWAIT, NOMODAL.**- Activa la caja y el foco no permanence en ella. Se utiliza para hacer diálogos MDI
- **ON CLICK.**- Función que se ejecuta cuando se pincha en ella directamente (no en un control de ella).
- **ON INIT.**- Función que se ejecuta al inicializar la caja, una vez comprobada la cláusula **WEND**.
- **ON MOVE.**-
- **ON PAINT.**-
- **ON RIGHT CLICK.**-
- **VALID.**-
- **WHEN.**- Función que se ejecuta para comprobar si ha de inicializarse la caja o no.

Ejemplos

Trucos

Para hacer que un diálogo comience MAXIMIZADO, poner entre el DEFINE y el ACTIVATE,

```
oDlg:bStart := {ShowWindow(oDlg:hWnd,WS_MAXIMIZE)}
```

No olvidar poner WS_MAXIMIZE al definir el diálogo en el workshop.

Conclusión

FOLDER

Concepto

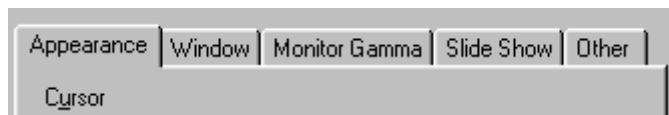
Los fólдер son esas carpetas que tienen varias solapas en la parte superior. Según la solapa que se active, la superficie de la carpeta presentará unos controles u otros. De esta forma es fácil poner en un mismo espacio muchos controles, con la única particularidad de que no se pueden visualizar todos al mismo tiempo.

Lo primero que hay que tener en cuenta desde el punto de vista de la programación, es que un fólдер es un espacio reservado en una caja de diálogo para tener superpuestas en el mismo sitio varias cajas hijas del diálogo contenedor. Cada una de estas cajas hijas, tendrá sus propios controles, incluso más fólдер. Cuando se programa directamente, sin recursos, no te has de preocupar de las cajas hijas, pero con recursos sí.

Este método de mostrar información se empezó a utilizar con Windows 3.0 y se ha mostrado muy eficaz ya que nos permite visualizar de forma organizada gran cantidad de información en espacios muy reducidos.

Existen varios tipos de folder:

Unilínea: Son los más usuales, y en ellos todas las pestañas que forman el folder se encuentran distribuidas en una única línea horizontal.



Multilínea: Surgen con ciertas aplicaciones de Windows 95 y se caracterizan por agrupar las distintas pestañas que forman el folder en varios grupos alineados horizontalmente.



Verticales: Cierta tipo de aplicaciones han diseñado variaciones sobre los folder originales, como el caso de Lotus Organizer, donde las pestañas de selección se encuentran alineadas verticalmente. Este tipo de folder también es muy típico del sistema operativo OS/2.

FiveWin nos va a permitir incluir en nuestros programas folder unilínea, si bien, a través de librerías de terceros, como Canal Five Folders tenemos la posibilidad de ampliar los tipos de folders con los que podemos desarrollar.

Por programa...

folder.ch

```
#xcommand @ <nRow>, <nCol> FOLDER [<oFolder>] ;
[ <of: OF, WINDOW, DIALOG> <oWnd> ] ;
[ <prm: PROMPT, PROMPTS, ITEMS> <cPrompt,...> ] ;
[ <dlg: DIALOG, DIALOGS, PAGE, PAGES> <cDlgName1> [,<cDlgNameN>]] ;
[ <lPixel: PIXEL> ] ;
[ <lDesign: DESIGN> ] ;
[ <color: COLOR, COLORS> <nClrFore> [,<nClrBack>] ] ;
[ OPTION <nOption> ] ;
[ SIZE <nWidth>, <nHeight> ] ;
[ MESSAGE <cMsg> ] ;
[ <lAdjust: ADJUST> ] ;
[ FONT <oFont> ];

=> ;
[<oFolder> := ] TFolder():New( <nRow>, <nCol>;
  [\{<cPrompt>\}], \{<cDlgName1> [,<cDlgNameN>]\},;
  <oWnd>, <nOption>, <nClrFore>, <nClrBack>, <.lPixel.>;
  <.lDesign.>, <nWidth>, <nHeight>, <cMsg>, <.lAdjust.>;
  <oFont> )
```

Comando

```
@ nRow, nCol FOLDER oFolder ;
OF, WINDOW, DIALOG oWnd ;
PROMPT, PROMPTS, ITEMS cPrompt,... ;
DIALOG, DIALOGS, PAGE, PAGES cDlgName1[, cDlgNameN] ;
PIXEL ;
DESIGN ;
COLOR, COLORS nClrFore[, nClrBack ] ;
OPTION nOption ;
SIZE nWidth, nHeight ;
MESSAGE cMsg ;
ADJUST ;
FONT oFont
```

Cláusulas

- **@** irá seguido de las coordenadas de posición. Si no se especifica la cláusula PIXEL, los números serán línea y columna. Indicando PÍXEL, los valores serán en pixels. OJO, en caso de ser línea y columna, la pantalla se divide en **30** líneas y 80 columnas
- **ADJUST** Ajusta la imagen completa, al espacio asignado. Hay que tener en cuenta que el programa hace un ajuste matemático independiente para el ancho y el largo. Si el recuadro asignado por programa no guarda la misma proporción de ancho y largo que la imagen real, saldrá distorsionada. También hay que tener en cuenta que si se pretende ampliación, se pierde definición.

- **COLOR, COLORS** indica el color del texto. Si se pone un segundo color, será el del fondo. El color se define mediante la función `nRGB()` o similar.
- **DESIGN** añade un borde al control, redimensionable en tiempo de ejecución. También permite que el control se pueda mover con el ratón.
- **FONT** indica el objeto fuente, en caso de ser distinto al indicado en momento del diseño con el workshop.
- **MESSAGE** es el mensaje que aparecerá en la barra de mensajes cuando el cursor pase por encima del control.
- **OF, WINDOWS, DIALOG** es el marco que lo contine. Normalmente será una caja de diálogo.
- **ON CHANGE** indica la función que se ejecutará cuando cambie el contenido o aspecto del control.
- **ON DROP**
- **OPTION**
- **PAGE**
- **PÍXEL, PIXELS**. Normalmente la posición se indica en líneas y columnas, y el tamaño en píxels. Con esta cláusula, todo será en píxels
- **PROMPT, VAR** seguidas del texto literal o el nombre de la variable que lo contiene. El font y el color será el indicado en las cláusulas **FONT** y **COLOR**. En caso de no tenerlas, podrá los valores por defecto del marco o ventana que lo contiene.

Ejemplos

```

/*****
DIALOGO Visualiza una caja de dialogo
*/
static function dialogo

local oDlg
local oFol,oBut1,oBut2

DEFINE DIALOG oDlg OF oWnd TITLE 'Tutorial' FROM 0,0 TO 20,80

@ 1,1 FOLDER oFol OF oDlg ;
    PROMPT 'Uno','Dos' ;
    SIZE 300,100

@ 1,1 BUTTON oBut1 PROMPT 'Boton 1' OF oFol:aDialogs[1]
@ 1,1 BUTTON oBut2 PROMPT 'Boton 2' OF oFol:aDialogs[2]

ACTIVATE DIALOG oDlg CENTER

return NIL

```

Por recursos...

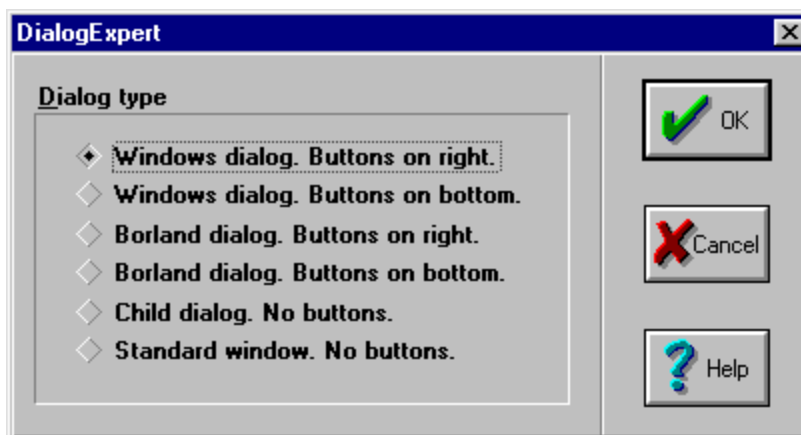
folder.ch

```
#xcommand REDEFINE FOLDER [<oFolder>] ;
    [ ID <nId> ] ;
    [ <of: OF, WINDOW, DIALOG> <oWnd> ] ;
    [ <prm: PROMPT, PROMPTS, ITEMS> <cPrompt,...> ] ;
    [ <dlg: DIALOG, DIALOGS, PAGE, PAGES> <cDlgName1> [,<cDlgNameN>]] ;
    [ <color: COLOR, COLORS> <nClrFore> [,<nClrBack>] ] ;
    [ OPTION <nOption> ] ;
    [ ON CHANGE <uChange> ] ;
    [ <lAdjust: ADJUST> ] ;
=> ;
[<oFolder> := ] TFolder():ReDefine( <nId>, [\{<cPrompt>\}],;
    \{ <cDlgName1> [,<cDlgNameN>] \}, <oWnd>;
    <nOption>, <nClrFore>, <nClrBack>;
    [{|nOption,nOldOption| <uChange>}], <.lAdjust.> )
```

Workshop

La realización de un f6lder desde el WS se hace b6sicamente en tres etapas:

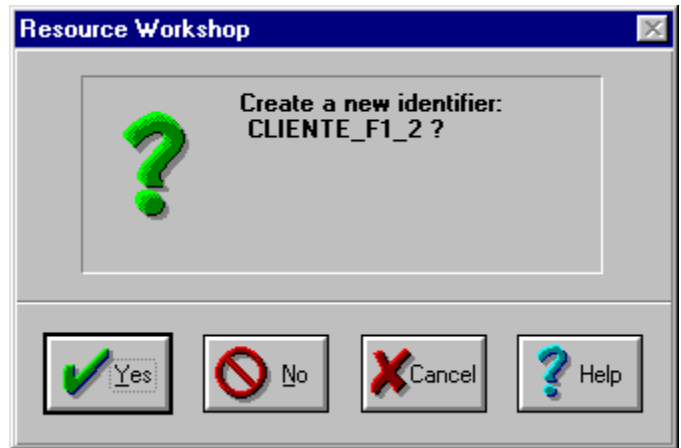
- 1.- **Definir el espacio que ocuparán las carpetas.**- Para ello elegir la herramienta indicada con una llave e indicar en **Class** la palabra **TFolder**. Marcar a continuaci6n el rect6ngulo que ocupara la carpeta (incluyendo las solapas). Fijate en el tama1o que indica en **Height** y **Width** (toma nota en un papel). Luego lo usar6s para cada caja de di6logo hija. Cierra la ventana donde est6s construyendo el di6logo, pero no salgas del WS.
- 2.- **Definir tantas cajas de di6logo como solapas tenga la carpeta.** En la opci6n del men6 del WS **Resource**, marca New y en la ventana emergente se1ala **DIALOG**. En **Place identifiers in** elige el proyecto actual. Pulsa **OK**. Saldr6 algo similar a la ventana indicada mas abajo. Seleccionar **Child dialog. No button** y pulsa **OK**. Aparecer6 una nueva caja de di6logo para su edici6n. Recordando el tama1o que anotastes en el paso anterior, pon las dimensiones correctas asegurandote que las ventanas hijas caben en el di6logo contenedor.



- 3.- **Poner en cada caja de di6logo hija los controles que pertenecer6n a cada solapa.**- Pon los controles que quieras en cada una de las nuevas cajas de di6logo. Como ver6s en cada caja nueva, los ID comienzan por 101. No importa que haya un 101 en cada caja, puesto que luego, adem6s del n6mero, nos referiremos a la caja concreta.

Como veras, los nuevos di6logos se llaman algo as6 como DIALOG_1, DIALOG_2,

etc. Puedes dejarlo así o renombrarlo. Supongamos que quieres que se llame CLIENTE_F1_1, CLIENTE_F1_2. Para ello, pulsa con el derecha en la barra azul de la ventana que pondra DIALOG: DIALOG_1. Se abrirá un menú emergente. Elige la opción **Rename resource** y pon el nombre que quieras. Al pulsar OK aparecerá una ventana preguntando si quieres un nuevo identificador. Indica que **No**. Ahora puedes poner los controles que quieras. Para hacer las otras cajas de diálogo, puedes repetir el proceso, o bien, antes de poner ningún control, salir a la ventana donde están todos los elementos y realizar una operación de **copiar y pegar** para cada una de las nuevas ventanas



Comando

```
REDEFINE FOLDER oFolder ;
  ID nId ;
  OF, WINDOW, DIALOG oWnd ;
  PROMPT, PROMPTS, ITEMS cPrompt,... ;
  DIALOG, DIALOGS, PAGE, PAGES cDlgName1[, cDlgNameN];
  COLOR, COLORS nClrFore [, nClrBack] ;
  OPTION nOption ;
  ON CHANGE uChange ;
  ADJUST
```

Cláusulas

- **ADJUST** Ajusta la imagen completa, al espacio asignado. Hay que tener en cuenta que el programa hace un ajuste matemático independiente para el ancho y el largo. Si el recuadro asignado por programa no guarda la misma proporción de ancho y largo que la imagen real, saldrá distorsionada. También hay que tener en cuenta que si se pretende ampliación, se pierde definición.
- **COLOR, COLORS** indica el color del texto. Si se pone un segundo color, será el del fondo. El color se define mediante la función nRGB() o similar.
- **ID** indica el indicador en workshop. Puede ser una variable definida.
- **OF, WINDOWS, DIALOG** es el marco que lo contine. Normalmente será una caja de diálogo.
- **ON CHANGE** indica la función que se ejecutará cuando cambie el contenido o aspecto del control.
- **OPTION**
- **PAGE**
- **PROMPT, VAR** seguidas del texto literal o el nombre de la variable que lo contiene. El font y el color será el indicado en las cláusulas **FONT** y **COLOR**. En

caso de no tenerlas, podrá los valores por defecto del marco o ventana que lo contiene.

Ejemplos

```
// CARPETAS .....
REDEFINE FOLDER oFol ID 140 OF oDia ;
  ITEMS 'Comentarios',OTA('Facturación'), 'Recibos' ;
  DIALOGS 'clien_1','clien_2','clien_3'

// Carpeta 1 .....
REDEFINE GET oTexto VAR cTexto MEMO ID 101 OF oFol:aDialogs[1] ;
  UPDATE ;
  MESSAGE 'Comentarios'

// Carpeta 2 .....
REDEFINE GET oDbf:CUOTA ;
  ID 101 OF oFol:aDialogs[2] ;
  UPDATE ;
  MESSAGE 'Importe para mantenimiento'

// Carpeta 3 .....
REDEFINE GET oDbf:A_NOMBRE ;
  ID 101 OF oFol:aDialogs[3] ;
  UPDATE ;
  MESSAGE ''
```

Trucos

Conclusión

Los fólder son unas herramientas muy útiles cuando se quieren poner muchos datos en poco sitio. Son similares a los controles **PAGES** y **TABS**, pero estos últimos están menos extendidos.

GET (no memo)

Concepto

El comando GET es similar al mismo en DOS, pero solo en apariencia.

Por programa...

fivewin.ch

```
#command @ <nRow>, <nCol> GET [ <oGet> VAR ] <uVar> ;
[ <dlg: OF, WINDOW, DIALOG> <oWnd> ] ;
[ PICTURE <cPict> ] ;
[ VALID <ValidFunc> ] ;
[ <color: COLOR, COLORS> <nClrFore> [, <nClrBack>] ] ;
[ SIZE <nWidth>, <nHeight> ] ;
[ FONT <oFont> ] ;
[ <design: DESIGN> ] ;
[ CURSOR <oCursor> ] ;
[ <pixel: PIXEL> ] ;
[ MESSAGE <cMsg> ] ;
[ <update: UPDATE> ] ;
[ WHEN <uWhen> ] ;
[ <lCenter: CENTER, CENTERED> ] ;
[ <lRight: RIGHT> ] ;
[ ON CHANGE <uChange> ] ;
[ <readonly: READONLY, NO MODIFY> ] ;
[ <pass: PASSWORD> ] ;
[ <lNoBorder: NO BORDER, NOBORDER> ] ;
[ <help: HELPID, HELP ID> <nHelpId> ] ;
=> ;
[ <oGet> := ] TGet():New( <nRow>, <nCol>, bSETGET(<uVar>),;
[ <oWnd>], <nWidth>, <nHeight>, <cPict>, <{ValidFunc}>;,
<nClrFore>, <nClrBack>, <oFont>, <.design.>;,
<oCursor>, <.pixel.>, <cMsg>, <.update.>, <{uWhen}>;,
<.lCenter.>, <.lRight.>;,
[ \{ | nKey, nFlags, Self | <uChange> \} ], <.readonly.>;,
<.pass.>, [ <.lNoBorder.> ], <nHelpId> )

#command @ <nRow>, <nCol> GET [ <oGet> VAR ] <uVar> ;
[ <dlg: OF, WINDOW, DIALOG> <oWnd> ] ;
[ PICTURE <cPict> ] ;
[ VALID <ValidFunc> ] ;
[ <color: COLOR, COLORS> <nClrFore> [, <nClrBack>] ] ;
[ SIZE <nWidth>, <nHeight> ] ;
[ FONT <oFont> ] ;
[ <design: DESIGN> ] ;
[ CURSOR <oCursor> ] ;
[ <pixel: PIXEL> ] ;
[ MESSAGE <cMsg> ] ;
[ <update: UPDATE> ] ;
[ WHEN <uWhen> ] ;
```

```

[ <lCenter: CENTER, CENTERED> ] ;
[ <lRight: RIGHT> ] ;
[ ON CHANGE <uChange> ] ;
[ <readonly: READONLY, NO MODIFY> ] ;
[ <help:HELPID, HELP ID> <nHelpId> ] ;
[ <spin: SPINNER> [ON UP <SpnUp>] [ON DOWN <SpnDn>] [MIN <Min>] [MAX
<Max>] ] ;
=> ;
[ <oGet> := ] TGet():New( <nRow>, <nCol>, bSETGET(<uVar>),,
[<oWnd>], <nWidth>, <nHeight>, <cPict>, <{ValidFunc}>,;
<nClrFore>, <nClrBack>, <oFont>, <.design.>,;
<oCursor>, <.pixel.>, <cMsg>, <.update.>, <{uWhen}>,;
<.lCenter.>, <.lRight.>,;
[\\|nKey, nFlags, Self| <uChange>\\], <.readonly.>,;
.f., .f., <nHelpId>,;
<.spin.>, <{SpnUp}>, <{SpnDn}>, <{Min}>, <{Max}> )

```

Comando

```

@ nRow, nCol GET oGet
VAR uVar
OF, WINDOW, DIALOG oWnd
PICTURE cPict
VALID ValidFunc
COLOR,COLORS nClrFore[,nClrBack]
SIZE nWidth, nHeight
FONT oFont
DESIGN
CURSOR oCursor
PIXEL
MESSAGE cMsg
UPDATE
WHEN uWhen
CENTER, CENTERED
RIGHT
ON CHANGE uChange
READONLY, NO MODIFI
PASSWORD
NO BORDER, NOBORDER
HELPID, HELP ID nHelpId

```

```

@ nRow, nCol GET oGet
VAR uVar
OF, WINDOW, DIALOG oWnd
PICTURE cPict
VALID ValidFunc
COLOR,COLORS nClrFore[,nClrBack]
SIZE nWidth, nHeight
FONT oFont
DESIGN
CURSOR oCursor
PIXEL
MESSAGE cMsg
UPDATE
WHEN uWhen
CENTER, CENTERED
RIGHT
ON CHANGE uChange
READONLY, NO MODIFY
HELPID, HELP ID nHelpId
SPINNER [ON UP SpnUp] [ON DOWN SpnDn] [MIN Min] [MAX Max]

```

Cláusulas

- **@** irá seguido de las coordenadas de posición. Si no se especifica la cláusula **PIXEL**, los números serán línea y columna. Indicando **PÍXEL**, los valores serán en pixels..
- **CENTER, CENTERED** indica que centre el texto en el espacio marcado por **SIZE**. Esta cláusula es excluyente con **RIGHT**.
- **COLOR, COLORS** indica el color del texto. Si se pone un segundo color, será el del fondo. El color se define mediante la función **nRGB()** o similar.
- **CURSOR** indica que cambie el cursor cuando este pase por encima.
- **DESIGN** añade un borde al control, redimensionable en tiempo de ejecución. También permite que el control se pueda mover con el ratón.
- **FONT** indica el objeto fuente, en caso de ser distinto al indicado en momento del diseño con el workshop.
- **HELP**
- **MESSAGE** es el mensaje que aparecerá en la barra de mensajes cuando el cursor pase por encima del control.
- **OF, WINDOWS, DIALOG** es el marco que lo contine. Normalmente será una caja de diálogo.
- **ON CHANGE** indica la función que se ejecutará cuando cambie el contenido o aspecto del control.
- **ON DROP**
- **PICTURE** es la máscara igual que en DOS.
- **PIXEL, PIXELS**. Normalmente la posición se indica en líneas y columnas, y el tamaño en píxels. Con esta cláusula, todo será en pixels
- **RIGHT** indica que visualice el texto justificado por la derecha, en el espacio marcado por **SIZE**. Esta cláusula es excluyente con **CENTERED** o **CENTER**
- **SIZE** indica el tamaño en pixels del campo que ocupará el texto. No indica el campo del texto, que eso está marcado por el propio texto y el font utilizado. Este tamaño se utiliza, por ejemplo para el color de fondo, el borde o la superposición sobre otros controles.
- **UPDATE** se utiliza para actualizar el texto cuando se refresca la caja o ventana de diálogo por medio de la instrucción `<objeto diálogo>.refresh()`.
- **VALID** es la función que debe devolver .T. para que el control pueda dejar el foco y continuar con el programa. Es similar a la versión en DOS
- **VAR** es la variable asociada al control.
- **WHEN** es la función que debe devolver .T. para que el control pueda tomar el foco. Es similar a la versión en DOS

Ejemplos

Por recursos...

fivewin.ch

```
#xcommand REDEFINE GET [ <oGet> VAR ] <uVar> ;
    [ ID <nId> ] ;
    [ <dlg: OF, WINDOW, DIALOG> <oDlg> ] ;
    [ <help:HELPID, HELP ID> <nHelpId> ] ;
    [ VALID <ValidFunc> ] ;
    [ PICTURE <cPict> ] ;
    [ <color:COLOR,COLORS> <nClrFore> [,<nClrBack>] ] ;
    [ FONT <oFont> ] ;
    [ CURSOR <oCursor> ] ;
    [ MESSAGE <cMsg> ] ;
    [ <update: UPDATE> ] ;
    [ WHEN <uWhen> ] ;
    [ ON CHANGE <uChange> ] ;
    [ <readonly: READONLY, NO MODIFY> ] ;
    [ <spin: SPINNER> [ON UP <SpnUp>] [ON DOWN <SpnDn>] [MIN <Min>]
[ MAX <Max>] ] ] ;
    => ;
    [ <oGet> := ] TGet():ReDefine( <nId>, bSETGET(<uVar>), <oDlg>;,
    <nHelpId>, <cPict>, <{ValidFunc}>, <nClrFore>, <nClrBack>;,
    <oFont>, <oCursor>, <cMsg>, <.update.>, <{uWhen}>;,
    [ \{|nKey,nFlags,Self| <uChange> \}], <.readonly.>;,
    <.spin.>, <{SpnUp}>, <{SpnDn}>, <{Min}>, <{Max}>)
```

Workshop

Comando

```
REDEFINE GET oGet
VAR uVar
ID nId
OF, WINDOW, DIALOG oDlg
HELPID, HELP ID nHelpId
VALID ValidFunc
PICTURE cPict
COLOR,COLORS nClrFore [,nClrBack]
FONT oFont
CURSOR oCursor
MESSAGE cMsg
UPDATE
WHEN uWhen
ON CHANGE uChange
READONLY, NO MODIFY
SPINNER [ON UP SpnUp] [ON DOWN SpnDn] [MIN Min] [MAX Max]
```

Cláusulas

- **CENTER, CENTERED** indica que centre el texto en el espacio marcado por SIZE. Esta cláusula es excluyente con RIGHT.
- **COLOR, COLORS** indica el color del texto. Si se pone un segundo color, será el del fondo. El color se define mediante la función nRGB() o similar.
- **CURSOR** indica que cambie el cursor cuando este pase por encima.
- **FONT** indica el objeto fuente, en caso de ser distinto al indicado en momento del diseño con el workshop.
- **HELP**
- **ID** indica el indicador en workshop. Puede ser una variable definida.
- **MESSAGE** es el mensaje que aparecerá en la barra de mensajes cuando el cursor pase por encima del control.
- **OF, WINDOWS, DIALOG** es el marco que lo contine. Normalmente será una caja de diálogo.
- **ON CHANGE** indica la función que se ejecutará cuando cambie el contenido o aspecto del control.
- **ON DROP**
- **PICTURE** es la máscara igual que en DOS.
- **RIGHT** indica que visualice el texto justificado por la derecha, en el espacio marcado por SIZE. Esta cláusula es excluyente con CENTERED o CENTER
- **UPDATE** se utiliza para actualizar el texto cuando se refresca la caja o ventana de diálogo por medio de la instrucción <objeto diálogo>.refresh().
- **VALID** es la función que debe devolver .T. para que el control pueda dejar el foco y continuar con el programa. Es similar a la versión en DOS
- **VAR** es la variable asociada al control.
- **WHEN** es la función que debe devolver .T. para que el control pueda tomar el foco. Es similar a la versión en DOS

Ejemplos

Trucos

Para que un GET coja el foco cuando nosotros queramos existe el método **oGet:SetFocus()**. De esta forma se evita la secuencia normal de Intros y Tabuladores.

Un truco muy útil en cajas de diálogo con muchos get's es cambiar el color de aquel que tenga el foco. Para hacer que el color de un get cambie según tenga el foco o no, se puede hacer la siguiente modificación a la clase TGet:

Al tomar el foco:

```
METHOD GotFocus() CLASS TGet

    if ! ::lDrag
        ::oGet:KillFocus()    // to properly initialize internal status
        ::oGet:SetFocus()
        ::setColor( RGB(0,0,0), RGB(150,255,150) )
        ::DispText()
        ::oGet:Pos = ::nPos
        ::SetPos( ::nPos )
        CallWindowProc( ::nOldProc, ::hWnd, WM_SETFOCUS )

        if Set( _SET_INSERT )
            DestroyCaret()
            CreateCaret( ::hWnd, 0, 6, ::nGetChrHeight() )
            ShowCaret( ::hWnd )
        endif
    else
        HideCaret( ::hWnd )
    endif

    Super:GotFocus()

return 0
```

Al dejar el foco:

```
METHOD LostFocus( hCtlFocus ) CLASS TGet

    Super:LostFocus( hCtlFocus )

    ::oGet:SetFocus()

    if ! ::oGet:BadDate .and. ! ::lReadOnly .and. ;
        ( ::oGet:changed .or. ::oGet:unTransform() <> ::oGet:original )
        ::oGet:Assign()    // for adjust numbers
        ::oGet:UpdateBuffer()
    endif

    ::setColor( RGB(0,0,0), RGB(255,255,255) )
    ::DispText()

    if ! ::oGet:BadDate
        ::oGet:KillFocus()
    else
        ::oGet:Pos = 1
        ::nPos = 1
        #ifdef __XPP__
            ::oGet:TypeOut = .f.
        #endif
    endif

return nil
```

Conclusión

GET multilinea (memos)

Concepto

Por programa...

fivewin.ch

```
#command @ <nRow>, <nCol> GET [ <oGet> VAR ] <uVar> ;
    [ <dlg: OF, WINDOW, DIALOG> <oWnd> ] ;
    [ <memo: MULTILINE, MEMO, TEXT> ] ;
    [ <color: COLOR, COLORS> <nClrFore> [, <nClrBack>] ] ;
    [ SIZE <nWidth>, <nHeight> ] ;
    [ FONT <oFont> ] ;
    [ <hscroll: HSCROLL> ] ;
    [ CURSOR <oCursor> ] ;
    [ <pixel: PIXEL> ] ;
    [ MESSAGE <cMsg> ] ;
    [ <update: UPDATE> ] ;
    [ WHEN <uWhen> ] ;
    [ <lCenter: CENTER, CENTERED> ] ;
    [ <lRight: RIGHT> ] ;
    [ <readonly: READONLY, NO MODIFY> ] ;
    [ VALID <uValid> ] ;
    [ ON CHANGE <uChange> ] ;
    [ <lDesign: DESIGN> ] ;
    [ <lNoBorder: NO BORDER, NOBORDER> ] ;
    [ <lNoVScroll: NO VSCROLL> ] ;
=> ;
    [ <oGet> := ] TMultiGet():New( <nRow>, <nCol>, bSETGET(<uVar>),;
        [ <oWnd>], <nWidth>, <nHeight>, <oFont>, <.hscroll.>;
        <nClrFore>, <nClrBack>, <oCursor>, <.pixel.>;
        <cMsg>, <.update.>, <{uWhen}>, <.lCenter.>;
        <.lRight.>, <.readonly.>, <{uValid}>;
        [ \{ | nKey, nFlags, Self | <uChange> \} ], <.lDesign.>;
        [ <.lNoBorder.> ], [ <.lNoVScroll.> ] )
```

Comando

```
@ nRow, nCol GET oGet VAR uVar
    OF, WINDOW, DIALOG oWnd
    MULTILINE, MEMO, TEXT
    COLOR, COLORS nClrFore[, nClrBack]
    SIZE nWidth, nHeight
    FONT oFont
    HSCROLL
    CURSOR oCursor
    PIXEL
    MESSAGE cMsg
    UPDATE
    WHEN uWhen
```

CENTER, CENTERED
RIGHT
READONLY, NO MODIFY
VALID uValid
ON CHANGE uChange
DESIGN
NO BORDER, NOBORDER
NO VSCROLL

Cláusulas

- **@** irá seguido de las coordenadas de posición. Si no se especifica la cláusula PIXEL, los números serán línea y columna. Indicando PÍXEL, los valores serán en pixels..
- **CENTER, CENTERED** indica que centre el texto en el espacio marcado por SIZE. Esta cláusula es excluyente con RIGHT.
- **COLOR, COLORS** indica el color del texto. Si se pone un segundo color, será el del fondo. El color se define mediante la función nRGB() o similar.
- **CURSOR** indica que cambie el cursor cuando este pase por encima.
- **DESIGN** añade un borde al control, redimensionable en tiempo de ejecución. También permite que el control se pueda mover con el ratón.
- **FONT** indica el objeto fuente, en caso de ser distinto al indicado en momento del diseño con el workshop.
- **HELP**
- **MEMO** se utiliza en el control para indicar que ha de editarse un campo memo o una variable de texto largo
- **MESSAGE** es el mensaje que aparecerá en la barra de mensajes cuando el cursor pase por encima del control.
- **OF, WINDOWS, DIALOG** es el marco que lo contine. Normalmente será una caja de diálogo.
- **ON CHANGE** indica la función que se ejecutará cuando cambie el contenido o aspecto del control.
- **ON DROP**
- **PIXEL, PIXELS**. Normalmente la posición se indica en líneas y columnas, y el tamaño en píxels. Con esta cláusula, todo será en pixels
- **RIGHT** indica que visualice el texto justificado por la derecha, en el espacio marcado por SIZE. Esta cláusula es excluyente con CENTERED o CENTER
- **SIZE** indica el tamaño en pixels del campo que ocupará el texto. No indica el campo del texto, que eso está marcado por el propio texto y el font utilizado. Este tamaño se utiliza, por ejemplo para el color de fondo, el borde o la superposición sobre otros controles.
- **UPDATE** se utiliza para actualizar el texto cuando se refresca la caja o ventana de diálogo por medio de la instrucción <objeto diálogo>:refresh().

- **VALID** es la función que debe devolver .T. para que el control pueda dejar el foco y continuar con el programa. Es similar a la versión en DOS
- **VAR** es la variable asociada al control.
- **WHEN** es la función que debe devolver .T. para que el control pueda tomar el foco. Es similar a la versión en DOS

Ejemplos

Por recursos...

fivewin.ch

```
#xcommand REDEFINE GET [ <oGet> VAR ] <uVar> ;
    [ <memo: MULTILINE, MEMO, TEXT> ] ;
    [ ID <nId> ] ;
    [ <dlg: OF, WINDOW, DIALOG> <oDlg> ] ;
    [ <help:HELPID, HELP ID> <nHelpId> ] ;
    [ <color: COLOR,COLORS> <nClrFore> [,<nClrBack>] ] ;
    [ FONT <oFont> ] ;
    [ CURSOR <oCursor> ] ;
    [ MESSAGE <cMsg> ] ;
    [ <update: UPDATE> ] ;
    [ WHEN <uWhen> ] ;
    [ <readonly: READONLY, NO MODIFY> ] ;
    [ VALID <uValid> ] ;
    [ ON CHANGE <uChange> ] ;
=> ;
[ <oGet> := ] TMultiGet():ReDefine( <nId>, bSETGET(<uVar>),;
    <oDlg>, <nHelpId>, <nClrFore>, <nClrBack>, <oFont>, <oCursor>,;
    <cMsg>, <.update.>, <{uWhen}>, <.readonly.>, <{uValid}>,;
    [\|nKey, nFlags, Self| <uChange>\] )
```

Workshop

Definir en las características del recursos **Want return** para que admita la tecla de retorno y haga una nueva línea.

Comando

```
REDEFINE GET oGet VAR uVar
    MULTILINE, MEMO, TEXT
    ID nId
    OF, WINDOW, DIALOG oDlg
    HELPID, HELP ID nHelpId
    COLOR,COLORS nClrFore [,nClrBack]
    FONT oFont
    CURSOR oCursor
```

MESSAGE cMsg
UPDATE
WHEN uWhen
READONLY, NO MODIFY
VALID uValid
ON CHANGE uChange

Cláusulas

- **CENTER, CENTERED** indica que centre el texto en el espacio marcado por SIZE. Esta cláusula es excluyente con RIGHT.
- **COLOR, COLORS** indica el color del texto. Si se pone un segundo color, será el del fondo. El color se define mediante la función nRGB() o similar.
- **CURSOR** indica que cambie el cursor cuando este pase por encima.
- **FONT** indica el objeto fuente, en caso de ser distinto al indicado en momento del diseño con el workshop.
- **HELP**
- **ID** indica el indicador en workshop. Puede ser una variable definida.
- **MEMO** se utiliza en el control para indicar que ha de editarse un campo memo o una variable de texto largo
- **MESSAGE** es el mensaje que aparecerá en la barra de mensajes cuando el cursor pase por encima del control.
- **OF, WINDOWS, DIALOG** es el marco que lo contine. Normalmente será una caja de diálogo.
- **ON CHANGE** indica la función que se ejecutará cuando cambie el contenido o aspecto del control.
- **ON DROP**
- **RIGHT** indica que visualice el texto justificado por la derecha, en el espacio marcado por SIZE. Esta cláusula es excluyente con CENTERED o CENTER
- **UPDATE** se utiliza para actualizar el texto cuando se refresca la caja o ventana de diálogo por medio de la instrucción <objeto diálogo>.refresh().
- **VALID** es la función que debe devolver .T. para que el control pueda dejar el foco y continuar con el programa. Es similar a la versión en DOS
- **VAR** es la variable asociada al control.
- **WHEN** es la función que debe devolver .T. para que el control pueda tomar el foco. Es similar a la versión en DOS

Ejemplos

Trucos

Conclusión

GROUPS

Concepto

Por programa...

fivewin.ch

```
#xcommand @ <nTop>, <nLeft> [ GROUP <oGroup> ] TO <nBottom>, <nRight > ;
    [ <label:LABEL,PROMPT> <cLabel> ] ;
    [ OF <oWnd> ] ;
    [ COLOR <nClrFore> [,<nClrBack>] ] ;
    [ <lPixel: PIXEL> ] ;
    [ <lDesign: DESIGN> ] ;
=> ;
    [ <oGroup> := ] TGroup():New( <nTop>, <nLeft>, <nBottom>, <nRight>;
    <cLabel>, <oWnd>, <nClrFore>, <nClrBack>, <.lPixel.>;
    [<.lDesign.>] )
```

Comando

```
@ nTop, nLeft GROUP oGroup TO nBottom, nRight
  LABEL,PROMPT cLabel
  OF oWnd
  COLOR nClrFore[,nClrBack]
  PIXEL
  DESIGN
```

Cláusulas

Ejemplos

Por recursos...

fivewin.ch

```
#xcommand REDEFINE GROUP [ <oGroup> ] ;
    [ <label:LABEL,PROMPT> <cLabel> ] ;
    [ ID <nId> ] ;
    [ <of: OF, WINDOW, DIALOG> <oWnd> ] ;
    [ COLOR <nClrFore> [,<nClrBack>] ] ;
=> ;
    [ <oGroup> := ] TGroup():ReDefine( <nId>, <cLabel>, <oWnd>;
```

```
<nClrFore>, <nClrBack> )
```

Workshop

Comando

```
REDEFINE GROUP oGroup  
  LABEL,PROMPT cLabel  
  ID nId  
  OF, WINDOW, DIALOG oWnd  
  COLOR nClrFore[,nClrBack]
```

Cláusulas

Ejemplos

Trucos

Conclusión

ICONS

Concepto

Los iconos se comportan como un elemento gráfico de resolución fija (32x32 pixels) y cualquier profundidad de color. Se pueden poner en cualquier parte de la ventana o caja de diálogo. Se puede utilizar como un botón gracias a la cláusula ON CLIC.

Este tipo de objeto solo maneja iconos de ficheros o recursos, pero no BMP, JPG, cursores, etc.

Aunque al diseñar el icono, en la paleta de colores existe el “transparente”, posteriormente no se utiliza en tiempo de ejecución.

Por programa...

fivewin.ch

```
#xcommand @ <nRow>, <nCol> ICON [ <oIcon> ] ;
    [ <resource: NAME, RESOURCE, RESNAME> <cResName> ] ;
    [ <file: FILE, FILENAME, DISK> <cIcoFile> ] ;
    [ <border:BORDER> ] ;
    [ ON CLICK <uClick> ] ;
    [ <of: OF, WINDOW, DIALOG> <ownd> ] ;
    [ <update: UPDATE> ] ;
    [ WHEN <uWhen> ] ;
    [ COLOR <nClrFore> [,<nClrBack>] ] ;
=> ;
    [ <oIcon> := ] TIcon():New( <nRow>, <nCol>, <cResName>;
    <cIcoFile>, <.border.>, <{uClick}>, <ownd>, <.update.>;
    <{uWhen}>, <nClrFore>, <nClrBack> )
```

Comando

```
@ nRow, nCol ICON oIcon
NAME, RESOURCE, RESNAME cResName
FILE, FILENAME, DISK cIcoFile
BORDER
ON CLICK uClick
OF, WINDOW, DIALOG oWnd
UPDATE
WHEN uWhen
COLOR nClrFore[,nClrBack]
```

Cláusulas

- **@** irá seguido de las coordenadas de posición. Si no se especifica la cláusula PIXEL, los números serán línea y columna. Indicando PÍXEL, los valores serán en pixels..
- **BORDER** indica que saque un rectángulo marcando el borde del control en función de SIZE.
- **COLOR, COLORS** indica el color del texto. Si se pone un segundo color, será el del fondo. El color se define mediante la función nRGB() o similar.
- **FILE** indica el fichero relacionado con el control.
- **MESSAGE** es el mensaje que aparecerá en la barra de mensajes cuando el cursor pase por encima del control.
- **NAME**
- **OF, WINDOWS, DIALOG** es el marco que lo contine. Normalmente será una caja de diálogo.
- **ON CLICK**
- **UPDATE** se utiliza para actualizar el texto cuando se refresca la caja o ventana de diálogo por medio de la instrucción <objeto diálogo>.refresh().
- **WHEN** es la función que debe devolver .T. para que el control pueda tomar el foco. Es similar a la versión en DOS

Ejemplos

Por recursos...

fivewin.ch

```
#xcommand DEFINE ICON <oIcon> ;
    [ <resource: NAME, RESOURCE, RESNAME> <cResName> ] ;
    [ <file: FILE, FILENAME, DISK> <cIcoFile> ] ;
    [ WHEN <WhenFunc> ] ;
=> ;
    <oIcon> := TIcon():New( ,, <cResName>, <cIcoFile>, <{WhenFunc}> )

#xcommand REDEFINE ICON <oIcon> ;
    [ ID <nId> ] ;
    [ <resource: NAME, RESOURCE, RESNAME> <cResName> ] ;
    [ <file: FILE, FILENAME, DISK> <cIcoFile> ] ;
    [ ON CLICK <uClick> ] ;
    [ <of: OF, WINDOW, DIALOG> <oWnd> ] ;
    [ <update: UPDATE> ] ;
    [ WHEN <uWhen> ] ;
=> ;
    [ <oIcon> := ] TIcon():ReDefine( <nId>, <cResName>, <cIcoFile>,;
    <{uClick}>, <.update.>, <oWnd>, <{uWhen}> )
```

Workshop

Como la clase TIcon no dispone de herramienta propia en el workshop, se recurre al control genérico que se representa con una llave. En la ventana que se nos abre (New custom control), en **Class** poner **TIcon** y pulsar OK o Intro. El resto es como con otros controles del WS.

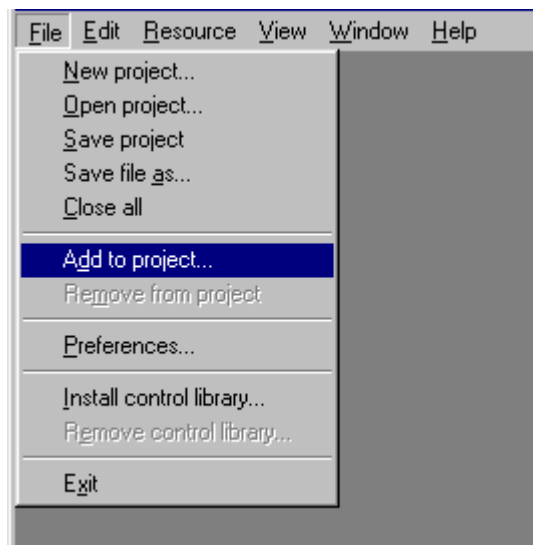
Lo dicho anteriormente crea el objeto, pero hay que suministrarle el propio icono, es decir, el dibujito. Para ello se pueden seguir dos caminos:

- Crear el icono desde cero, definiéndolo y dibujándolo directamente con el WS.
- Importar un icono ya generado y presente en un fichero.

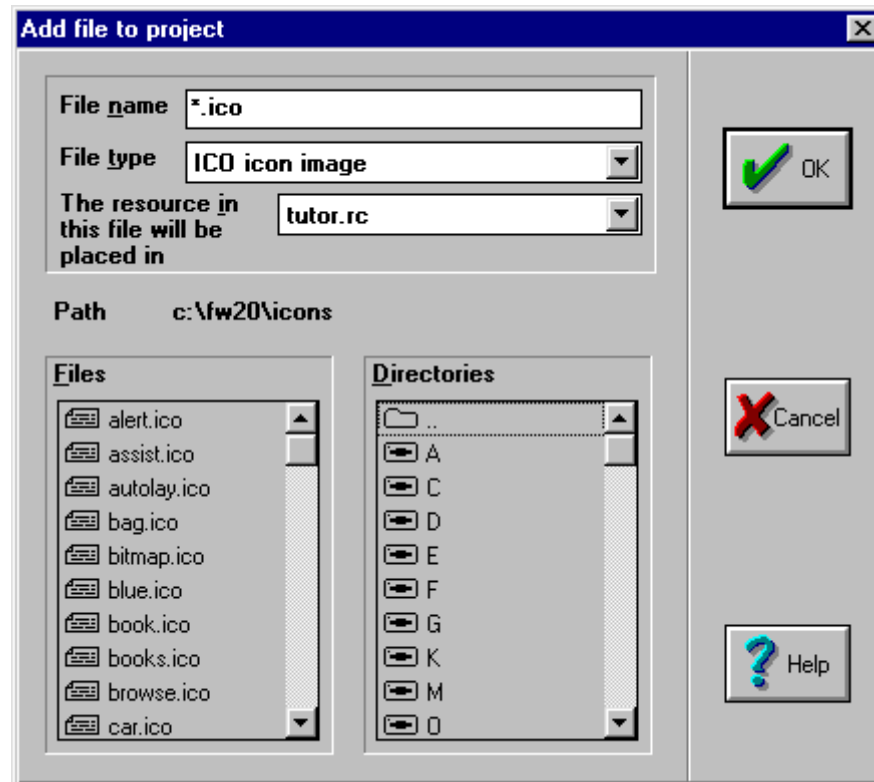
Vamos a ver la segunda, puesto que normalmente todos tenemos herramientas de dibujo mas potentes que las del WS.

Para importar, seguir los siguientes pasos:

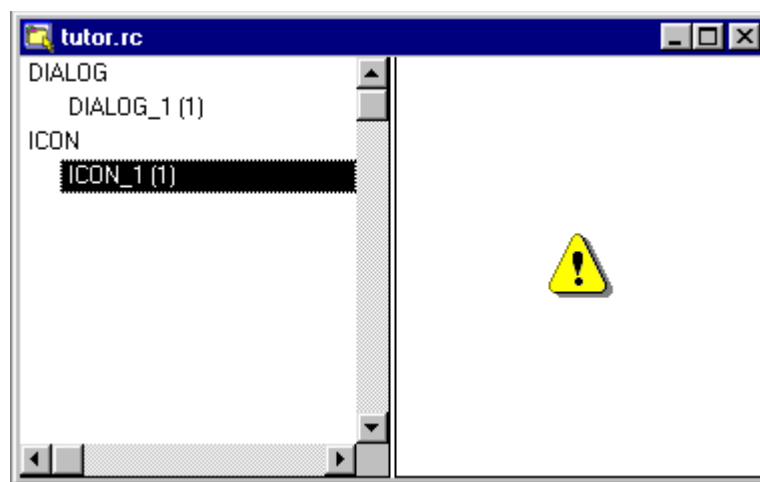
1. Abrir un proyecto para poder tener todas las opciones marcadas.
2. En la opción **File** del menú, seleccionar la opción **Add to project...**



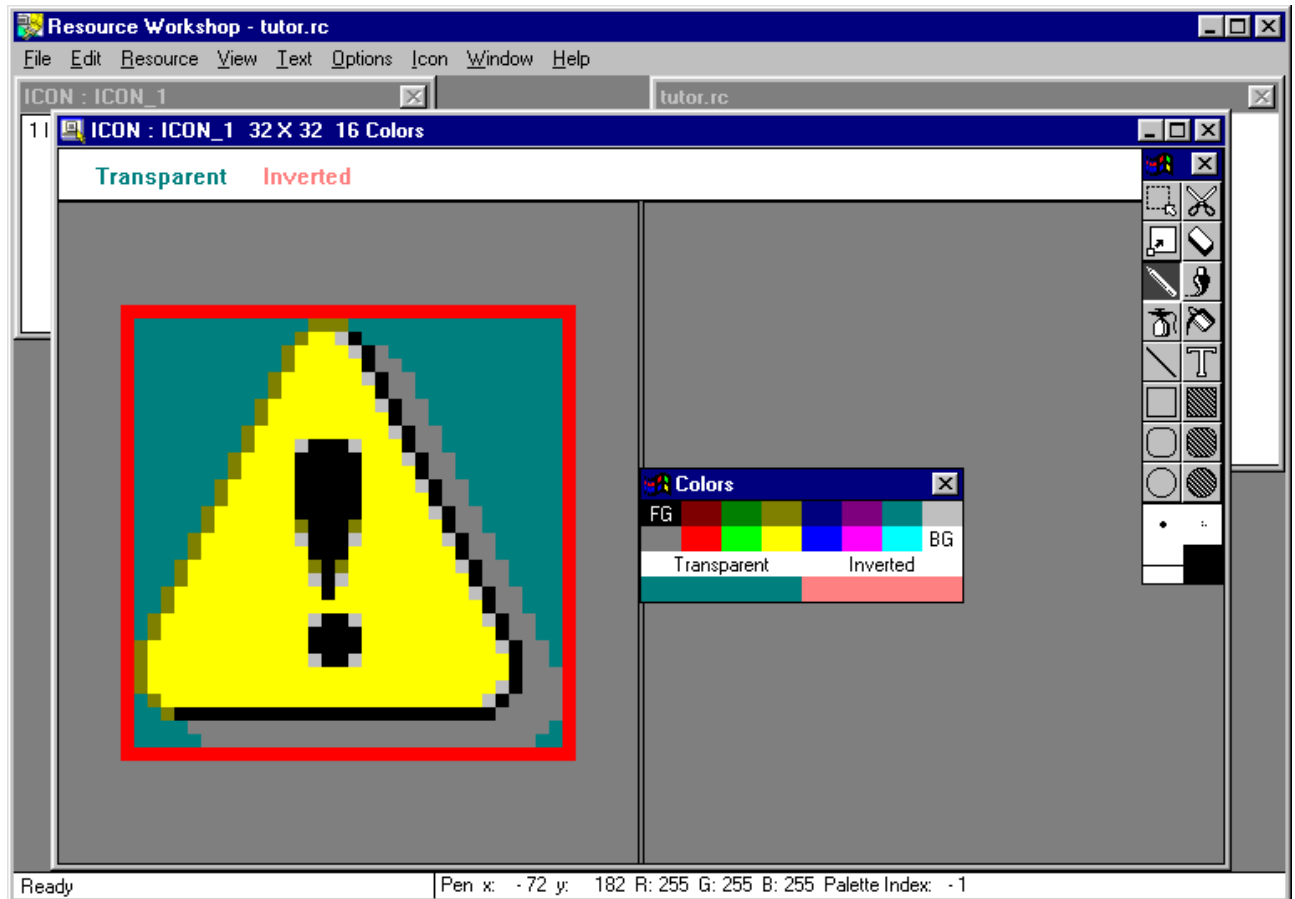
3. Aparecerá una ventana como la siguiente:



4. Elegir el icono que se quiera. Lamentablemente no se puede ver antes se seleccionar, de modo que tenemos que saber de antemano el que queremos.
5. En la ventana de recursos aparecera bajo el grupo ICON, el icono recién importado. Si nos ponemos en él aparecerá a la derecha de una forma similar a esta:



6. Una vez en el proyecto podemos trabajarlo. Haciendo doble click, podemos editarlo, etc.



Comando

```

DEFINE ICON oIcon
    NAME, RESOURCE, RESNAME cResName
    FILE, FILENAME, DISK cIcoFile
    WHEN WhenFunc

REDEFINE ICON oIcon
    ID nId
    NAME, RESOURCE, RESNAME cResName
    FILE, FILENAME, DISK cIcoFile
    ON CLICK uClick
    OF, WINDOW, DIALOG oWnd
    UPDATE
    WHEN uWhen
  
```

Cláusulas

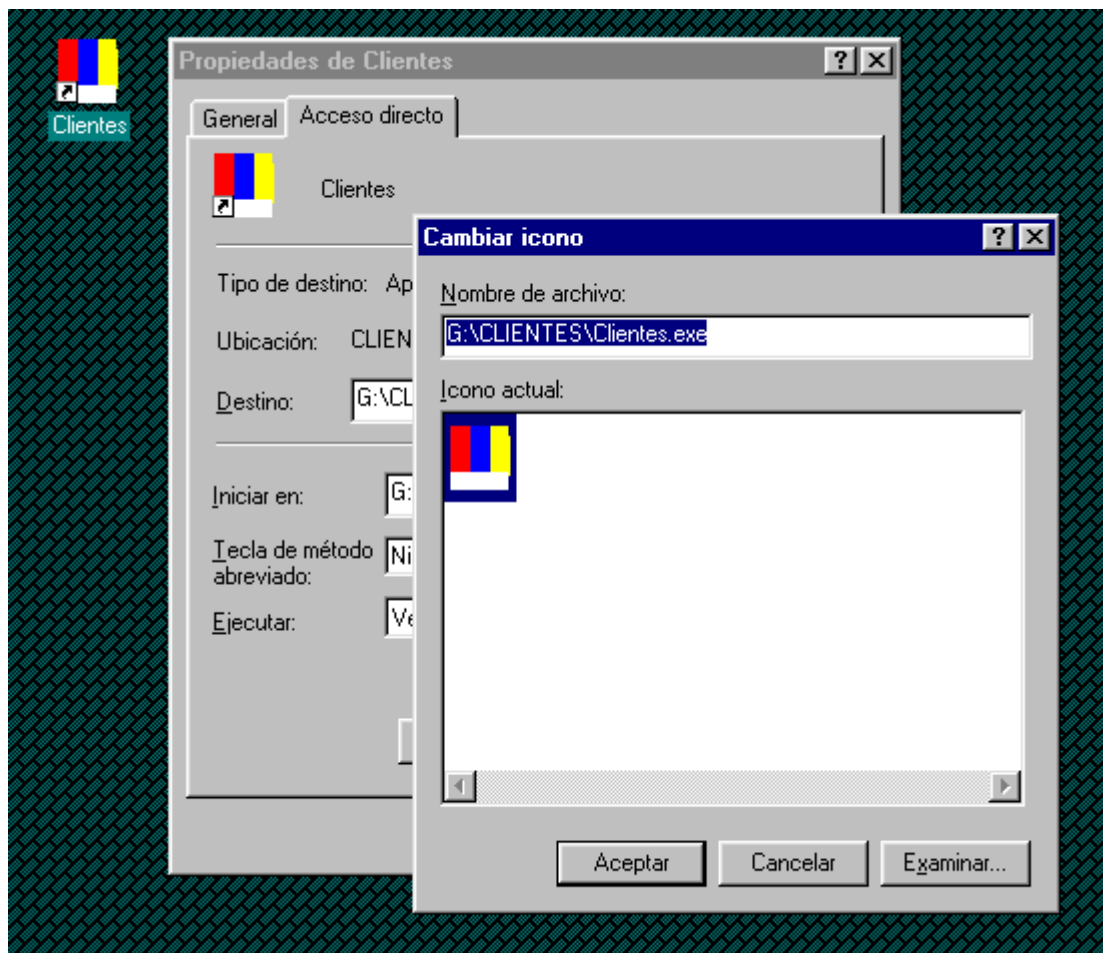
- **FILE** indica el fichero relacionado con el control.
- **ID** indica el indicador en workshop. Puede ser una variable definida.
- **NAME**
- **OF, WINDOWS, DIALOG** es el marco que lo contine. Normalmente será una caja de diálogo.
- **ON CLICK**

- **UPDATE** se utiliza para actualizar el texto cuando se refresca la caja o ventana de diálogo por medio de la instrucción <objeto diálogo>.refresh().
- **WHEN** es la función que debe devolver .T. para que el control pueda tomar el foco. Es similar a la versión en DOS

Ejemplos

Trucos

Si se crea el icono con recursos y estos se ponen en el ejecutable, se pueden ver con las propiedades del enlace directo del escritorio, en la opción de cambiar icono.



Conclusión

IDLEACTION

Concepto

Por programa...

fivewin.ch

```
#xcommand SET IDLEACTION TO <uIdleAction> => SetIdleAction( <{uIdleAction}> )
```

Comando

Ejemplos

IMAGEN

Concepto

Por programa...

fivewin.ch

```
#xcommand @ <nRow>, <nCol> IMAGE [ <oBmp> ] ;
[ <resource: NAME, RESNAME, RESOURCE> <cResName> ] ;
[ <file: FILENAME, FILE, DISK> <cBmpFile> ] ;
[ <NoBorder:NOBORDER, NO BORDER> ] ;
[ SIZE <nWidth>, <nHeight> ] ;
[ <of: OF, WINDOW, DIALOG> <oWnd> ] ;
[ <lClick: ON CLICK, ON LEFT CLICK> <uLClick> ] ;
[ <rClick: ON RIGHT CLICK> <uRClick> ] ;
[ <scroll: SCROLL> ] ;
[ <adjust: ADJUST> ] ;
[ CURSOR <oCursor> ] ;
[ <pixel: PIXEL> ] ;
[ MESSAGE <cMsg> ] ;
[ <update: UPDATE> ] ;
[ WHEN <uWhen> ] ;
[ VALID <uValid> ] ;
[ <lDesign: DESIGN> ] ;
=> ;
[ <oBmp> := ] TImage():New( <nRow>, <nCol>, <nWidth>, <nHeight>;,
<cResName>, <cBmpFile>, <.NoBorder.>, <oWnd>;,
[ \{ |nRow,nCol,nKeyFlags| <uLClick> \} ],,
[ \{ |nRow,nCol,nKeyFlags| <uRClick> \} ], <.scroll.>;,
<.adjust.>, <oCursor>, <cMsg>, <.update.>;,
<{uWhen}>, <.pixel.>, <{uValid}>, <.lDesign.> )
```

Comando

```
@ nRow, nCol IMAGE oBmp
NAME, RESNAME, RESOURCE cResName
FILENAME, FILE, DISK cBmpFile
NOBORDER, NO BORDER
SIZE nWidth, nHeight
OF, WINDOW, DIALOG oWnd
ON CLICK, ON LEFT CLICK uLClick
ON RIGHT CLICK uRClick
SCROLL
ADJUST
CURSOR oCursor
PIXEL
MESSAGE cMsg
UPDATE
WHEN uWhen
VALID uValid
DESIGN
```

Cláusulas

- **@** irá seguido de las coordenadas de posición. Si no se especifica la cláusula **PIXEL**, los números serán línea y columna. Indicando **PÍXEL**, los valores serán en pixels..
- **ACJUST** permite al operador modificar el tamaño y posición del control.
- **CURSOR** indica que cambie el cursor cuando este pase por encima.
- **DESIGN** añade un borde al control, redimensionable en tiempo de ejecución. También permite que el control se pueda mover con el ratón.
- **FILE** indica el fichero relacionado con el control.
- **MESSAGE** es el mensaje que aparecerá en la barra de mensajes cuando el cursor pase por encima del control.
- **NAME**
- **NOBORDER**
- **OF, WINDOWS, DIALOG** es el marco que lo contine. Normalmente será una caja de diálogo.
- **ON CLICK**
- **ON RIGHT CLICK**
- **PIXEL, PIXELS**. Normalmente la posición se indica en líneas y columnas, y el tamaño en píxels. Con esta cláusula, todo será en pixels
- **SCROLL**
- **SIZE** indica el tamaño en pixels del campo que ocupará el texto. No indica el campo del texto, que eso está marcado por el propio texto y el font utilizado. Este tamaño se utiliza, por ejemplo para el color de fondo, el borde o la superposición sobre otros controles.
- **UPDATE** se utiliza para actualizar el texto cuando se refresca la caja o ventana de diálogo por medio de la instrucción <objeto diálogo>:refresh().
- **VALID** es la función que debe devolver .T. para que el control pueda dejar el foco y continuar con el programa. Es similar a la versión en DOS
- **WHEN** es la función que debe devolver .T. para que el control pueda tomar el foco. Es similar a la versión en DOS

Ejemplos

Trucos

Conclusión

LISTBOX

Concepto

Puesto que el control LISTBOX genera dos funciones distintas, el preprocesador al ver que se incluye o no las cláusulas PROMPT o FIELDS, genera un código llamando a una función o a otra. Estas funciones generadas por el preprocesador corresponderán a la clase **TListBox** o TWBrowse. Ahora vamos a ver el control que genera TListBox.

Si controlas los colores de texto, de fondo y además incluyes iconos, podrás hacer listas muy llamativas.

Entre las características de este control podemos destacar:

- Posibilidad de incorporar imágenes gráficas a cada elemento de la lista.
- Selección de múltiples elementos.
- Posibilidad de añadir o borrar elementos.
- Búsqueda “sensitiva” al pulsar la primera letra del elemento a buscar.
- Posibilidad de indicar el primer elemento seleccionado.

Por programa...

fivewin.ch

```
#xcommand @ <nRow>, <nCol> LISTBOX [ <oLbx> VAR ] <cnVar> ;
[ <items: PROMPTS, ITEMS> <aList> ] ;
[ SIZE <nWidth>, <nHeight> ] ;
[ ON CHANGE <uChange> ] ;
[ ON [ LEFT ] DBLCLICK <uLDbLClick> ] ;
[ <of: OF, WINDOW, DIALOG > <oWnd> ] ;
[ VALID <uValid> ] ;
[ <color: COLOR, COLORS> <nClrFore> [, <nClrBack>] ] ;
[ <pixel: PIXEL> ] ;
[ <design: DESIGN> ] ;
[ FONT <oFont> ] ;
[ MESSAGE <cMsg> ] ;
[ <update: UPDATE> ] ;
[ WHEN <uWhen> ] ;
[ BITMAPS <aBitmaps> ] ;
[ ON DRAWITEM <uBmpSelect> ] ;
[ <multi: MULTI, MULTIPLE, MULTISEL> ] ;
[ <sort: SORT> ] ;
=> ;
[ <oLbx> := ] TListBox():New( <nRow>, <nCol>, bSETGET(<cnVar>),;
<aList>, <nWidth>, <nHeight>, <{uChange}>, <oWnd>, <{uValid}>,;
<nClrFore>, <nClrBack>, <.pixel.>, <.design.>, <{uLDbLClick}>,;
<oFont>, <cMsg>, <.update.>, <{uWhen}>, <aBitmaps>,;
```

```
[{|nItem|<uBmpSelect>}], <.multi.>, <.sort.> )
```

Comando

```
@ nRow, nCol LISTBOX oLbx
  VAR cnVar
  PROMPTS, ITEMS aList
  SIZE nWidth, nHeight
  ON CHANGE uChange
  ON [ LEFT ] DBLCLICK uLDbClick
  OF, WINDOW, DIALOG oWnd
  VALID uValid
  COLOR,COLORS nClrFore[,nClrBack]
  PIXEL
  DESIGN
  FONT oFont
  MESSAGE cMsg
  UPDATE
  WHEN uWhen
  BITMAPS aBitmaps
  ON DRAWITEM uBmpSelect
  MULTI, MULTIPLE, MULTISEL
  SORT
```

Cláusulas

- **@** irá seguido de las coordenadas de posición. Si no se especifica la cláusula **PIXEL**, los números serán línea y columna. Indicando **PÍXEL**, los valores serán en pixels..
- **ACTION** será la función que se ejecuta al hacer clic sobre el control.
- **ALIAS** indica el alias del fichero relacionado con el control. Si no se pone, la base de datos será la activa en ese momento.
- **BITMAPS** matriz de ficheros BMP a mostrar junto a cada elemento.
- **COLOR, COLORS** indica el color del texto. Si se pone un segundo color, será el del fondo. El color se define mediante la función **nRGB()** o similar.
- **CURSOR** indica que cambie el cursor cuando este pase por encima.
- **DESIGN** añade un borde al control, redimensionable en tiempo de ejecución. También permite que el control se pueda mover con el ratón.
- **FONT** indica el objeto fuente, en caso de ser distinto al indicado en momento del diseño con el workshop.
- **HEAD** indica el texto o textos que irán en la cabecera.
- **MESSAGE** es el mensaje que aparecerá en la barra de mensajes cuando el cursor pase por encima del control.
- **MULTI, MULTIPLE, MULTISEL** permite realizar multiples selecciones.
- **OF, WINDOWS, DIALOG** es el marco que lo contine. Normalmente será una caja de diálogo.

- **ON CHANGE** indica la función que se ejecutará cada vez que se cambie el elemento seleccionado.
- **ON DBLCLICK** función a ejecutar cuando se hace doble click sobre un elemento.
- **ON DRAWITEN** función a ejecutar antes de mostrar cada BMP en cada línea.
- **PIXEL, PIXELS**. Normalmente la posición se indica en líneas y columnas, y el tamaño en píxels. Con esta cláusula, todo será en píxels
- **PROMPT,ITEMS** seguidas de la matriz literal o el nombre de la variable que la contiene.
- **SIZE** indica el tamaño en píxels del campo que ocupará el texto. No indica el campo del texto, que eso esta marcado por el propio texto y el font utilizado. Este tamaño se utiliza, por ejemplo para el color de fondo, el borde o la superposición sobre otros controles.
- **SORT** los elementos de la matriz se ordenan antes de ser visualizados. Esto evita tener que hacerlo nosotros en el programa con **asort()**.
- **TITLE**
- **UPDATE** se utiliza para actualizar el texto cuando se refresca la caja o ventana de diálogo por medio de la instrucción <objeto diálogo>.refresh().
- **VALID** es la función que debe devolver .T. para que el control pueda dejar el foco y continuar con el programa. Es similar a la versión en DOS
- **VAR** es la variable asociada al control. Puede ser carácter (se almacenará el texto seleccionado) o numérica (se almacenara la posición del elemento seleccionado)
- **WHEN** es la función que debe devolver .T. para que el control pueda tomar el foco. Es similar a la versión en DOS

Ejemplos

Por recursos...

fivewin.ch

```
#xcommand REDEFINE LISTBOX [ <oLbx> VAR ] <cnVar> ;
    [ <items: PROMPTS, ITEMS> <aItems> ] ;
    [ <files: FILES, FILESPEC> <cFileSpec> ] ;
    [ ID <nId> ] ;
    [ ON CHANGE <uChange,...> ] ;
    [ ON [ LEFT ] DBLCLICK <uLDbLClick> ] ;
    [ <of: OF, WINDOW, DIALOG > <ownd> ] ;
    [ <help:HELPID, HELP ID> <nHelpId> ] ;
    [ VALID <uValid> ] ;
    [ <color: COLOR,COLORS> <nClrFore> [,<nClrBack>] ] ;
    [ MESSAGE <cMsg> ] ;
    [ <update: UPDATE> ] ;
    [ WHEN <uWhen> ] ;
```

```

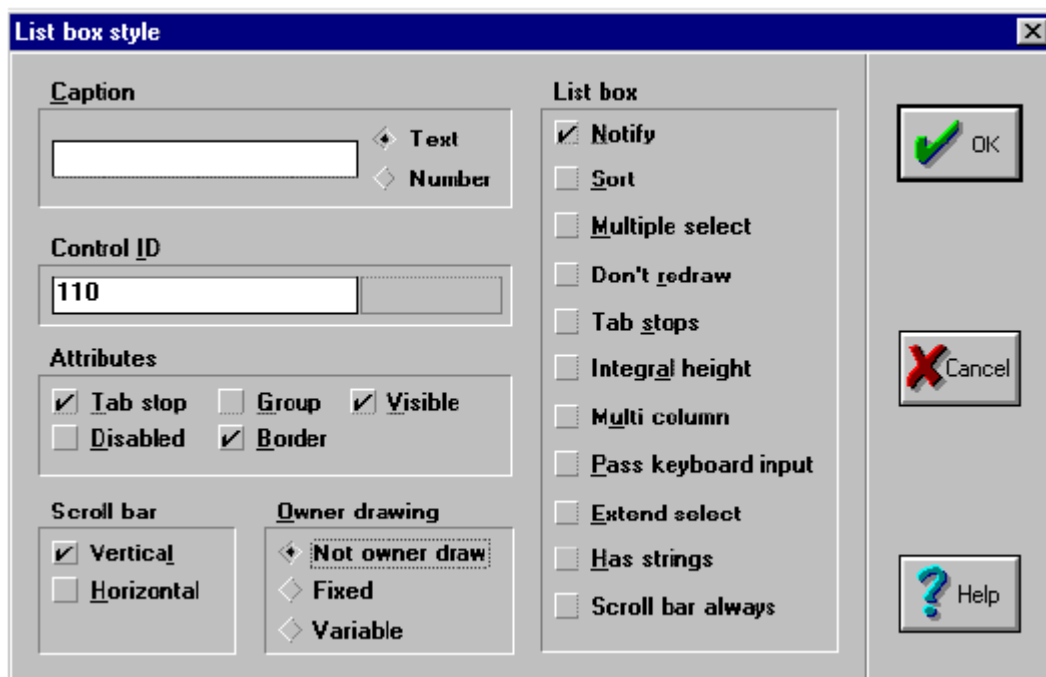
[ BITMAPS <acBitmaps> ] ;
[ ON DRAWITEM <uBmpSelect> ] ;
=> ;
[ <oLbx> := ] TListBox():ReDefine( <nId>, bSETGET(<cnVar>), <aItems>;,
[ \{ | <uChange> \} ], <oWnd>, <nHelpId>, <acBitmaps>;,
<{uValid}>, <cFileSpec>, <nClrFore>, <nClrBack>;,
<{uLdblClick}>, <cMsg>, <.update.>, <{uWhen}>;,
[ \{ | nItem | <uBmpSelect> \} ] )

```

Workshop

El control Listbox ... Items cuenta con su propio icono en la caja de herramientas de Workshop, representado por una caja blanca con una barra de scroll a la derecha.

Una vez marcado el control sobre la caja de diálogo, si hacemos doble clic nos aparece la ventana de propiedades:



donde cada apartado significa:

Attributes: Los controles del tipo Listbox cuenta con todos los atributos comunes a la mayoría de los controles y que ya hemos estudiado en anteriores capítulos del manual.

Scroll Bar: Inicialmente, este tipo de controles sólo maneja la barra de Scroll vertical. En caso de desmarcar esta opción, la barra de Scroll no aparecerá nunca, ni siquiera en el caso de que los elementos del array sobrepasen el espacio reservado en pantalla para el control, y sólo podremos desplazarnos utilizando las teclas del cursor.

Owner Drawing: Esta propiedad nos permite indicar a windows si el control es redibujado directamente por windows o por nuestra aplicación. Cuando incorporamos bitmaps a un ListBox, éste ha de ser controlado directamente por la aplicación. De tal forma, si queremos incluir bitmaps, será necesario marcar la opción "Fixed". Si el Listbox sólo va a contener texto hemos de marcar la opción "Not owner draw".

Los estilos aplicables a Listbox son:

Sort: Los elementos del array se mostrarán ordenados alfabéticamente.

Multiple Select: Permite la selección simultánea de más de un elemento del listbox.

Integral Height: Impide que el último elemento visible del Listbox se muestre parcialmente, redimensionando automáticamente el Listbox hasta hacerlo desaparecer.

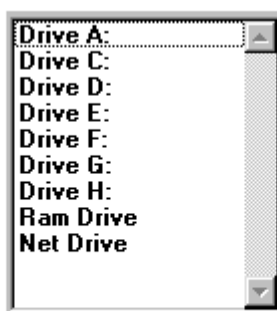


Extend Select: Este atributo activa es sistema de selección extendida de windows con las siguientes características:

- Posibilidad de seleccionar elementos correlativos arrastrando el ratón sobre ellos.
- Selección de múltiples elementos manteniendo pulsada la tecla control.
- Selección de un rango de elementos manteniendo pulsada la tecla mayúsculas.

Para el correcto funcionamiento de este atributo, siempre lo seleccionaremos acompañado de la opción "Multiple Select" indicada más arriba.

Scroll bar Allways: Nuestro Listbox siempre mostrará la barra de desplazamiento vertical. En el caso de que todos los elementos del Listbox quepan dentro del control, la barra de scroll se mostrará desactivada. Lógicamente, para que este atributo funcione, tiene que estar activada la barra de scroll vertical del grupo "Scroll Bar"



"Scroll bar allways" aplicado a un listbox donde se pueden visualizar todos sus elementos

Comando

```

REDEFINE LISTBOX oLbx
  VAR cnVar
  PROMPTS, ITEMS aItems
  FILES, FILESPEC cFileSpec
  ID nId
  ON CHANGE uChange,...
  ON [ LEFT ] DBLCLICK uLDblick
  OF, WINDOW, DIALOG oWnd
  HELPID, HELP ID nHelpId
  VALID uValid
  COLOR,COLORS nClrFore[,nClrBack]
  MESSAGE cMsg
  UPDATE
  WHEN uWhen
  BITMAPS acBitmaps
  ON DRAWITEM uBmpSelect

```

Cláusulas

- **ACTION** será la función que se ejecuta al hacer clic sobre el control.
- **ALIAS** indica el alias del fichero relacionado con el control. Si no se pone, la base de datos será la activa en ese momento.
- **BITMAPS** matriz de ficheros BMP a mostrar junto a cada elemento.
- **COLOR, COLORS** indica el color del texto. Si se pone un segundo color, será el del fondo. El color se define mediante la función nRGB() o similar.
- **CURSOR** indica que cambie el cursor cuando este pase por encima.
- **FILES** indica la máscara a utilizar si deseamos cargar nuestra matriz con los ficheros existentes en un directorio, por ejemplo "C:*.BAT".
- **FONT** indica el objeto fuente, en caso de ser distinto al indicado en momento del diseño con el workshop.
- **HEAD** indica el texto o textos que irán en la cabecera.
- **HELPID**
- **ID** indica el indicador en workshop. Puede ser una variable definida.
- **MESSAGE** es el mensaje que aparecerá en la barra de mensajes cuando el cursor pase por encima del control.
- **OF, WINDOWS, DIALOG** es el marco que lo contine. Normalmente será una caja de diálogo.
- **ON CHANGE** indica la función que se ejecutará cada vez que se cambie el elemento seleccionado.
- **ON DBLCLICK** función a ejecutar cuando se hace doble click sobre un elemento.
- **ON DRAWITEM** función a ejecutar antes de mostrar cada BMP en cada línea.
- **PROMPT, VAR** seguidas del texto literal o el nombre de la variable que lo contiene. El font y el color será el indicado en las cláusulas **FONT** y **COLOR**. En caso de no tenerlas, podrá los valores por defecto del marco o ventana que lo contiene.

- **TITLE**
- **UPDATE** se utiliza para actualizar el texto cuando se refresca la caja o ventana de diálogo por medio de la instrucción <objeto diálogo>.refresh().
- **VALID** es la función que debe devolver .T. para que el control pueda dejar el foco y continuar con el programa. Es similar a la versión en DOS
- **VAR** es la variable asociada al control.
- **WHEN** es la función que debe devolver .T. para que el control pueda tomar el foco. Es similar a la versión en DOS

Ejemplos

Programación

INCLUIR BITMAPS EN LISTBOX

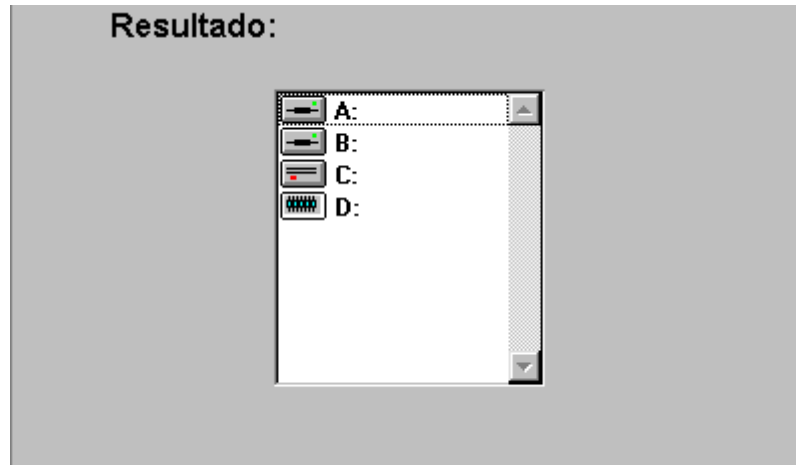
Como hemos comentado anteriormente, Fivewin nos ofrece la posibilidad de incorporar imágenes bitmap a los elementos de un control ListBox. Veamos con más detalle cómo hacerlo:

Las imágenes bitmap pueden ser tanto ficheros .BMP independientes, como bitmap almacenados en un archivo de recursos RC o DLL.

Un ListBox con imágenes asociadas ha de poseer obligatoriamente el estilo LBS_OWNERDRAWFIXED. Así, si estamos definiendo el control desde código fuente, el propio Fivewin se encargará de aplicar este estilo; por el contrario, si lo redefinimos desde recursos, seremos nosotros los encargados de aplicar dicho estilo a través de la opción "Owner Drawing / Fixed" del control LISTBOX de Workshop.

La incorporación de los bitmap al ListBox se realiza mediante la cláusula BITMAP del control, la cual contiene un vector con los iconos correspondientes a cada elemento del array.

```
REDEFINE LISTBOX oLbx VAR cDrive ;
  ID 101 OF oDlg ;
  ITEMS { "A:", "B:", "C:", "D:" } ;
  BITMAPS { "Floppy", ;
            "Floppy", ;
            "Hd", ;
            "C:\Cd.Bmp" }
```



Siendo, en este caso, “Floppy” y “Hd” recursos almacenados en un recurso DLL y CD.BMP un fichero independiente ubicado en el directorio raíz.

Dado que el estilo LBS_OWNERDRAWFIXED cede al programa la responsabilidad de actualizar el ListBox, Fivewin nos ofrece la cláusula **ON DRAWITEM**, a través de la cual podemos decidir qué elemento del vector de bitmaps va a ser mostrado junto a cada elemento del array PROMPT.

ON DRAWITEM se evalúa de forma automática cada vez que se refresca el ListBox, así como cada vez que hacemos un scroll dentro del control.

ON DRAWITEM recibe de forma implícita el parámetro **nItem**, que es la posición dentro del array ITEMS que estamos evaluando, y retorna el índice correspondiente al bitmap a mostrar para dicho item.

Veamos un ejemplo:

```
REDEFINE LISTBOX oLbx VAR cDrive ;
  ID 101 OF oDlg ;
  ITEMS { "A:", "B:", "C:", "D:" } ;
  BITMAPS { "Floppy", ;
            "Hd", ;
            "c:\cd.bmp" } ;
  ON DRAWITEM PonIcono( nItem, oLbx )
```

```
Function PonIcono( nItem, oLbx )
Local nSeleccion
Local Drive := oLbx:aItems[ nItem ]

DO CASE
  CASE Drive = "A:" .or. Drive = "B:"
    nSeleccion := 1
  CASE Drive = "C:"
    nSeleccion := 2
  OTHERWISE
    nSeleccion := 3
ENDCASE

Return nSeleccion
```

DANDO COLOR AL LISBOX ... ITEMS

Al igual que con el resto de los controles aportados por Fivewin, en un ListBox contamos con una serie de propiedades y métodos que nos permiten modificar los colores asociados por defecto al control.

En un ListBox podemos modificar:

- El color de fondo del ListBox (Método Setcolor(NIL, nColorFondo))
- El color fondo de los elementos (Propiedad nClrPane)
- El color del texto de los elementos (Propiedad nClrText)

No podemos modificar:

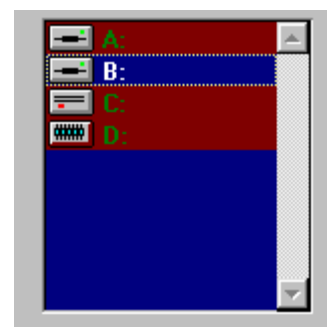
- El color de la barra de selección
- El color del texto del elemento seleccionado.

Realmente, los dos colores anteriores que no podemos modificar desde Fivewin, vienen determinados por las propiedades del sistema de Windows (Panel de control / Pantalla / Apariencia / Elemento / Elementos seleccionados)

A la hora de establecer los colores del ListBox es importante mantener el orden arriba indicado, pues de lo contrario los resultados pueden no ser los deseados como puede apreciarse en estos dos ejemplos.

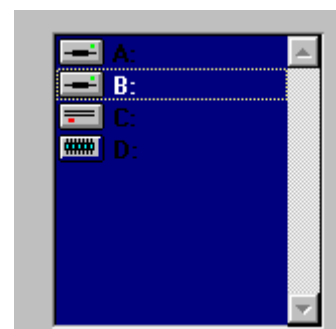
```
REDEFINE LISTBOX oLbx
.....
.....

oLbx:SetColor( NIL, nRgb(0,0,128 ) )
oLbx:nClrPane := nRgb( 128, 0, 0 )
oLbx:nClrText := nRgb( 0,128,0 )
```



```
REDEFINE LISTBOX oLbx
.....
.....

oLbx:nClrPane := nRgb( 128, 0, 0 )
oLbx:nClrText := nRgb( 0,128,0 )
oLbx:SetColor( NIL, nRgb(0,0,128 ) )
```



Trucos

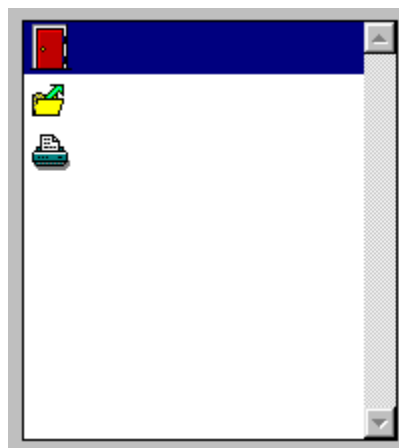
Cuando incorporamos bitmaps a nuestros ListBox es recomendable procurar que todos ellos tengan el mismo tamaño. De lo contrario Fivewin tomará como medida de referencia el tamaño del primer bitmap mostrado. Si existe otro bitmap de mayor tamaño,

tendremos que ajustar manualmente las propiedades nBmpHeight y nBmpWidth con los valores del bitmap más grande, o de lo contrario obtendremos unos resultados no deseados



Si deseamos confeccionar un ListBox en el que sólo aparezcan bitmaps, sólo tenemos que indicar en la cláusula ITEMS un array con tantos elementos nulos o vacíos como bitmaps queramos mostrar.

```
REDEFINE LISTBOX oLbx VAR cDrive ;
  ID 101 OF oDlg ;
  ITEMS { NIL, NIL, NIL } ;
  BITMAPS { "Salir", ;
            "Abrir", ;
            "Imprimir" }
```



Conclusión

MENU

Concepto

Bajo el concepto de MENU se encuentran varios comandos que por su afinidad vamos a ver todos juntos. **MENU**, **MENUITEM**, **MRU**, **SEPARATOR**, **ENDMENU**, **SYSMENU** y **ENDSYSMENU**.

MENU consiste en una barra horizontal en la parte superior de la ventana, en la que se desplegan de forma emergente unos submenús verticales. Cada una de las opciones pueden tener una acción en el programa o bien otro submenú con nuevas opciones, que a su vez pueden hacer lo mismo. Por cada MENU ha de haber al mismo nivel un comando **ENDMENU** (similar a **do while** con **enddo**, o **for** y **next**, etc.)

MENUITEM es cada opción de un MENU. Un MENUITEM puede tener como opción otro MENU que a su vez tiene MENUITEM. Como cada MENUITEM es independiente, no existe un ENDMENU.

En cada MENUITEM se puede poner:

- Una marca  de activación.
- Un dibujo.
- Una acción asociada.
- Teclas aceleradoras.
- Opciones habilitadas o deshabilitadas

Como se puede ver el ejemplo de menú de la página @#@, normalmente las opciones estarán en una función propia que es llamada desde el comando de definición de la ventana del tipo:

```
DEFINE WINDOWS oWnd TITLE 'Gestión de Stock' MENU funcionmenu()
```

El objeto MENUITEM es opcional para hacer las funciones básicas, pero es imprescindible si se quiere posteriormente trabajar con las variables de instancia y métodos de este objeto.

SYSMENU se utiliza para modificar el menu del sistema. Este menú se visualiza cuando se pincha en el pequeño icono que hay a la izquierda en la barra superior de la ventana. Normalmente ya tiene unas opciones por defecto como minimizar, maximizar, slir, etc. Con este comando lo que se hace es añadir opciones a este menu.

MENU FLOTANTE...

MENU DE HERRAMIENTAS...

Por programa...

Prácticamente será la única forma de utilizar los menús. Por recursos es más complicado y no merece la pena.

menu.ch

```
#xcommand MENU [ <oObjMenu> ] ;
    [ <popup: POPUP> ] ;
=> ;
    [ <oObjMenu> := ] MenuBegin( <.popup.> )

#xcommand MENUITEM [ <oMenuItem> PROMPT ] [ <cPrompt> ] ;
    [ MESSAGE <cMsg> ] ;
    [ <checked: CHECK, CHECKED, MARK> ] ;
    [ <enable: ENABLED, DISABLED> ] ;
    [ <file: FILE, FILENAME, DISK> <cBmpFile> ] ;
    [ <resource: RESOURCE, RESNAME, NAME> <cResName> ] ;
    [ ACTION <uAction,...> ] ;
    [ BLOCK <bAction> ] ;
    [ <of: OF, MENU, SYSMENU> <oMenu> ] ;
    [ ACCELERATOR <nState>, <nVirtKey> ] ;
    [ <help: HELP> ] ;
    [ <HelpId: HELP ID, HELPID> <nHelpId> ] ;
    [ WHEN <uWhen> ] ;
    [ <break: BREAK> ] ;
=> ;
    [ <oMenuItem> := ] MenuAddItem( <cPrompt>, <cMsg>;
    <.checked.>, [ Upper(<(enable)>) == "ENABLED" ],;
    [ \{|oMenuItem|<uAction>\} ],;
    <cBmpFile>, <cResName>, <oMenu>, <bAction>, <nState>, <nVirtKey>;
    <.help.>, <nHelpId>, [ <{uWhen}> ], <.break.> )

#xcommand MRU <oMru> ;
    [ <Ini: INI, ININame, FILENAME, NAME, DISK> <cIniFile> ] ;
    [ SECTION <cSection> ] ;
    [ <size: SIZE, ITEMS> <nItems> ] ;
    [ MESSAGE <cMsg> ] ;
    [ ACTION <uAction> ] ;
=> ;
    <oMru> := TMru():New( <cIniFile>, <cSection>, <nItems>, <cMsg>;
    [ { |cMruItem, oMenuItem| <uAction> } ] )

#xcommand SEPARATOR [ <oMenuItem> ] => [ <oMenuItem>:= ] MenuAddItem()

#xcommand ENDMENU => MenuEnd()

#xcommand REDEFINE SYSMENU [ <oMenu> ] ;
    [ <of: OF, WINDOW, DIALOG> <ownd> ] ;
=> ;
    [ <oMenu> := ] MenuBegin( .f., .t., <ownd> )

#xcommand ENDSYSMENU => MenuEnd()
```

Comandos

```

MENU [ <oObjMenu> ] [ <popup: POPUP> ] ;

MENUITEM oMenuItem PROMPT cPrompt ;
    MESSAGE cMsg ;
    CHECK, CHECKED, MARK ;
    ENABLED, DISABLED ;
    FILE, FILENAME, DISK cBmpFile ;
    RESOURCE, RESNAME, NAME cResName ;
    ACTION uAction,... ;
    BLOCK bAction ;
    OF, MENU, SYSMENU oMenu ;
    ACCELERATOR nState, nVirtKey ;
    HELP ;
    HELP ID, HELPID nHelpId ;
    WHEN uWhen ;
    BREAK

MRU oMru ;
    INI, ININame, FILENAME, NAME, DISK cIniFile ;
    SECTION cSection ;
    SIZE, ITEMS nItems ;
    MESSAGE cMsg ;
    ACTION uAction ;

SEPARATOR oMenuItem

ENDMENU

REDEFINE SYSMENU oMenu ;
    OF, WINDOW, DIALOG> oWnd ;

ENDSYSMENU

```

Cláusulas

- **ACCELERATOR**.-
- **ACTION**.- Indicar la acción a ejecutar cuando se pulsa en la opción. Normalmente será una función. Recuerda que se pueden encadenar funciones y/o acciones separándolas por comas y encerrando todo entre paréntesis. Si se quiere ejecutar un codeblock ver **BLOCK**.
- **BLOCK**.- Hace lo mismo que acción pero con codeblock en lugar de funciones. Las cláusulas **ACTION** y **BLOCK** son excluyentes.
- **BREAK**.-
- **CHECK**.- Hace que aparezca el signo ✓ en la opción. Solo funciona el los submenús. NO funciona en el primer nivel, es decir, el del menú horizontal.

- **ENABLED / DISABLED**.- Activa o desactiva la opción. Por defecto estará activada. Cuando se desactiva aparece el texto en gris y como es lógico no se puede pinchar sobre él.
- **FILE**.- Se utiliza para indicar el fichero de un BMP
- **HELP**.- Esta cláusula hace que la opción se separe de las demás y aparezca a la derecha de la barra. Los MENUITEM que se escriban en el programa a continuación, aparecerán a la derecha de éste.
- **HELPID**.-
- **MESSAGE**.-
- **OF**.-
- **PROMPT**.- Indica el texto que aparecerá como opción. En el texto se puede poner el signo & precediendo a una letra para formar una vía rápida, como por ejemplo, **&Clientes**, aparecerá como **C**lientes y se activará al pulsar Alt-C
- **RESOURCE**.- Si ha de visualizarse un BMP almacenado en recursos, indicar el nombre
- **WHEN**.-

Ejemplos

Reproduzco parte del ejemplo de la página @#@

```
local oBru
local oMenusys

DEFINE BRUSH oBru STYLE BRICKS
DEFINE WINDOW oWnd TITLE 'TUTOR 01_01' BRUSH oBru MENU menu()
REDEFINE SYSMENU oMenusys OF oWnd
    SEPARATOR
    MENUITEM 'Cualquier cosa' SYSMENU oMenusys
ENDSYSMENU
SET MESSAGE OF oWnd TO 'Tutorial' CLOCK DATE KEYBOARD
ACTIVATE WINDOW oWnd MAXIMIZED ON INIT dialogo()
return NIL

/*****
MENU
*/
static function Menu

local oMenu

MENU oMenu
    MENUITEM '&Archivos'
    MENU
        MENUITEM '&Empresa'          ACTION msginfo('Empresa')
        MENUITEM '&Ficheros'         ACTION msginfo('Ficheros')
        SEPARATOR
        MENUITEM '&Salida' ;
            ACTION If( MsgYesNo( oemtoansi(" Desea terminar ?")
                                ,oemtoansi('Elija opción SI/NO') ), oWnd:End(),) ;
            MESSAGE 'Salida del programa'
    ENDMENU
```

```

MENUITEM '&Clientes'
MENU
    MENUITEM '&Formulario'
    MENUITEM '&Listado'
ENDMENU

MENUITEM '&Utilidades'
MENU
    MENUITEM '&Calculadora'    ACTION WinExec('Calc')
    MENUITEM '&Write'          ACTION WinExec('Write')
ENDMENU

MENUITEM '&Ayuda'
MENU
    MENUITEM '&Ayuda'          ACTION msginfo('Ayuda')MESSAGE 'Ayuda'
    MENUITEM '&Acerca de'      ACTION msginfo('Acerca de')
ENDMENU

ENDMENU

return oMenu

```

Como se puede apreciar en la función raíz, se ha puesto un modificador al menú del sistema.

Por recursos...

No se utiliza prácticamente, de modo que solo pondré lo que ya esta en menu.ch

menu.ch

```

#xcommand DEFINE MENU <oMenu> ;
    [ <res: RESOURCE, NAME, RESNAME> <cResName> ] ;
    [ <popup: POPUP> ] ;
=> ;
    <oMenu> := TMenu():ReDefine( <cResName>, <.popup.> )

#xcommand REDEFINE MENUITEM [ <oMenuItem> PROMPT ] [<cPrompt>] ;
    [ ID <nId> <of: OF, MENU> <oMenu> ] ;
    [ ACTION <uAction> ] ;
    [ BLOCK <bAction> ] ;
    [ MESSAGE <cMsg> ] ;
    [ <checked: CHECK, CHECKED, MARK> ] ;
    [ <enable: ENABLED, DISABLED> ] ;
    [ <file: FILE, FILENAME, DISK> <cBmpFile> ] ;
    [ <resource: RESOURCE, RESNAME, NAME> <cResName> ] ;
    [ ACCELERATOR <nState>, <nVirtKey> ] ;
    [ <HelpId: HELP ID, HELPID> <nHelpId> ] ;
    [ WHEN <uWhen> ] ;
=> ;
    [ <oMenuItem> := ] TMenuItem():ReDefine( <cPrompt>, <cMsg>;,
        <.checked.>, [ Upper(<enable>) == "ENABLED" ], <{uAction}>;,
        <cBmpFile>, <cResName>, <oMenu>, <bAction>, <nId>;,
        <nState>, <nVirtKey>, <nHelpId>, [<{uWhen}>] )

```

```
#xcommand DEFINE MENU <oMenu> OF <oWnd> ;
    => ;
    <oMenu> := TMenu():New( .f., <oWnd> )

#xcommand SET MENU OF <oWnd> TO <oMenu> => <oWnd>:SetMenu( <oMenu> )

#xcommand ACTIVATE <menu:POPUP,MENU> <oMenu> ;
    [ AT <nRow>, <nCol> ] ;
    [ <of: OF, WINDOW, DIALOG> <oWnd> ] ;
    => ;
    <oMenu>:Activate( <nRow>, <nCol>, <oWnd> )

#xcommand REDEFINE SYSMENU [<oMenu>] ;
    [ <of: OF, WINDOW, DIALOG> <oWnd> ] ;
    => ;
    [<oMenu> :=] MenuBegin( .f., .t., <oWnd> )

#xcommand ENDSYSMENU => MenuEnd()
```

Workshop

No se utiliza.

Comando

```
DEFINE MENU oMenu ;
    RESOURCE, NAME, RESNAME cResName ;
    POPUP

REDEFINE MENUITEM oMenuItem PROMPT cPrompt ;
    ID nId ;
    OF, MENU oMenu ;
    ACTION uAction ;
    BLOCK bAction ;
    MESSAGE cMsg ;
    CHECK, CHECKED, MARK ;
    ENABLED, DISABLED ;
    FILE, FILENAME, DISK cBmpFile ;
    RESOURCE, RESNAME, NAME cResName ;
    ACCELERATOR nState, nVirtKey ;
    HELP ID, HELPID nHelpId ;
    WHEN uWhen

DEFINE MENU oMenu OF oWnd

SET MENU OF oWnd TO oMenu

ACTIVATE POPUP,MENU oMenu ;
    AT nRow, nCol ;
    OF, WINDOW, DIALOG oWnd
```

```
REDEFINE SYSMENU oMenu ;  
    OF, WINDOW, DIALOG oWnd
```

```
ENDSYSMENU
```

Cláusulas

Ejemplos

Conclusión

MESSAGE

Concepto

Por programa...

fivewin.ch

```
#xcommand SET MESSAGE [ OF <oWnd> ] ;
    [ TO <cMsg> ] ;
    [ <center: CENTER, CENTERED> ] ;
    [ <clock: CLOCK, TIME> ] ;
    [ <date: DATE> ] ;
    [ <kbd: KEYBOARD> ] ;
    [ FONT <oFont> ] ;
    [ <color: COLOR, COLORS> <nClrFore> [,<nClrBack> ] ] ;
    [ <inset: NO INSET, NOINSET> ] ;
=> ;
    <oWnd>:oMsgBar := TMsgBar():New( <oWnd>, <cMsg>, <.center.>;
                                     <.clock.>, <.date.>, <.kbd.>;
                                     <nClrFore>, <nClrBack>, <oFont>;
                                     [!<.inset.>] )
```

Comando

```
SET MESSAGE OF oWnd
TO cMsg
CENTER, CENTERED
CLOCK, TIME
DATE
KEYBOARD
FONT oFont
COLOR, COLORS nClrFore[,nClrBack]
NO INSET, NOINSET
```

Cláusulas

Ejemplos

Por recursos...

fivewin.ch

```
#xcommand DEFINE MESSAGE [ BAR ] [<oMsg>] ;
    [ OF <oWnd> ] ;
    [ PROMPT <cMsg> ] ;
    [ <center: CENTER, CENTERED> ] ;
    [ <clock: CLOCK, TIME> ] ;
    [ <date: DATE> ] ;
    [ <kbd: KEYBOARD> ] ;
    [ FONT <oFont> ] ;
    [ <color: COLOR, COLORS> <nClrFore> [,<nClrBack> ] ] ;
    [ <inset: NO INSET, NOINSET> ] ;
=> ;
[<oMsg>:=]    <oWnd>:oMsgBar    :=    TMsgBar():New(    <oWnd>,    <cMsg>,
<.center.>;,
                                     <.clock.>, <.date.>, <.kbd.>;,
                                     <nClrFore>, <nClrBack>, <oFont>;,
                                     [!<.inset.>] )
```

Workshop

Comando

```
DEFINE MESSAGE [ BAR ] oMsg
    OF oWnd
    PROMPT <cMsg>
    CENTER, CENTERED
    CLOCK, TIME
    DATE
    KEYBOARD
    FONT oFont
    COLOR, COLORS nClrFore[,nClrBack]
    NO INSET, NOINSET
```

Cláusulas

Ejemplos

Trucos

Conclusión

METAFILE

Concepto

Por programa...

fivewin.ch

```
#xcommand @ <nRow>, <nCol> METAFILE [<oMeta>] ;
    [ <file: FILE, FILENAME, DISK> <cMetaFile> ] ;
    [ <dlg: OF, WINDOW, DIALOG> <oWnd> ] ;
    [ SIZE <nWidth>, <nHeight> ] ;
    [ <color: COLOR, COLORS> <nClrFore> [,<nClrBack>] ] ;
=> ;
    [<oMeta> := ] TMetaFile():New( <nRow>, <nCol>, <nWidth>, <nHeight>;
    <cMetaFile>, <oWnd>, <nClrFore>, <nClrBack> )
```

Comando

```
@ nRow, nCol METAFILE oMeta
FILE, FILENAME, DISK cMetaFile
OF, WINDOW, DIALOG oWnd
SIZE nWidth, nHeight
COLOR, COLORS nClrFore[,nClrBack]
```

Cláusulas

Ejemplos

Por recursos...

fivewin.ch

```
#xcommand REDEFINE METAFILE [<oMeta>] ;
    [ ID <nId> ] ;
    [ <file: FILE, FILENAME, DISK> <cMetaFile> ] ;
    [ <dlg: OF, WINDOW, DIALOG> <oWnd> ] ;
    [ <color: COLOR, COLORS> <nClrFore> [,<nClrBack>] ] ;
=> ;
    [<oMeta> := ] TMetaFile():Redefine( <nId>, <cMetaFile>, <oWnd>;
    <nClrFore>, <nClrBack> )
```

Workshop**Comando**

```
REDEFINE METAFILE oMeta  
  ID nId  
  FILE, FILENAME, DISK cMetaFile  
  OF, WINDOW, DIALOG oWnd  
  COLOR, COLORS nClrFore[,nClrBack]
```

Cláusulas**Ejemplos****Trucos****Conclusión**

METER

Concepto

Por programa...

fivewin.ch

```
#xcommand @ <nRow>, <nCol> METER [ <oMeter> VAR ] <nActual> ;
[ TOTAL <nTotal> ] ;
[ SIZE <nWidth>, <nHeight> ];
[ OF <oWnd> ] ;
[ <update: UPDATE > ] ;
[ <lPixel: PIXEL > ] ;
[ FONT <oFont> ] ;
[ PROMPT <cPrompt> ] ;
[ <lNoPercentage: NOPERCENTAGE > ] ;
[ <color: COLOR, COLORS> <nClrPane>, <nClrText> ] ;
[ BARCOLOR <nClrBar>, <nClrBText> ] ;
[ <lDesign: DESIGN> ] ;
=> ;
[ <oMeter> := ] Tmeter():New( <nRow>, <nCol>, bSETGET(<nActual>),;
    <nTotal>, <oWnd>, <nWidth>, <nHeight>, <.update.>, ;
    <.lPixel.>, <oFont>, <cPrompt>, <.lNoPercentage.>,;
    <nClrPane>, <nClrText>, <nClrBar>, <nClrBText>, <.lDesign.> )
```

Comando

```
@ nRow, nCol METER oMeter
VAR nActual
TOTAL nTotal
SIZE nWidth, <nHeight>
OF oWnd
UPDATE
PIXEL
FONT oFont
PROMPT cPrompt
NOPERCENTAGE
COLOR, COLORS nClrPane, nClrText
BARCOLOR nClrBar, nClrBText
DESIGN
```

Cláusulas

- **@** irá seguido de las coordenadas de posición. Si no se especifica la cláusula PIXEL, los números serán línea y columna. Indicando PÍXEL, los valores serán en pixels..

- **BARCOLOR**
- **COLOR, COLORS** indica el color del texto. Si se pone un segundo color, será el del fondo. El color se define mediante la función nRGB() o similar.
- **DESIGN** añade un borde al control, redimensionable en tiempo de ejecución. También permite que el control se pueda mover con el ratón.
- **FONT** indica el objeto fuente, en caso de ser distinto al indicado en momento del diseño con el workshop.
- **NOPERCENTAJE**
- **OF, WINDOWS, DIALOG** es el marco que lo contine. Normalmente será una caja de diálogo.
- **PIXEL, PIXELS**. Normalmente la posición se indica en líneas y columnas, y el tamaño en píxels. Con esta cláusula, todo será en píxels
- **PROMPT, VAR** seguidas del texto literal o el nombre de la variable que lo contiene. El font y el color será el indicado en las cláusulas **FONT** y **COLOR**. En caso de no tenerlas, podrá los valores por defecto del marco o ventana que lo contiene.
- **SIZE** indica el tamaño en píxels del campo que ocupará el texto. No indica el campo del texto, que eso esta marcado por el propio texto y el font utilizado. Este tamaño se utiliza, por ejemplo para el color de fondo, el borde o la superposición sobre otros controles.
- **TOTAL**
- **UPDATE** se utiliza para actualizar el texto cuando se refresca la caja o ventana de diálogo por medio de la instrucción <objeto diálogo>:refresh().
- **VAR** es la variable asociada al control.
- **WHEN** es la función que debe devolver .T. para que el control pueda tomar el foco. Es similar a la versión en DOS

Ejemplos

Por recursos...

fivewin.ch

```
#xcommand REDEFINE METER [ <oMeter> VAR ] <nActual> ;
    [ TOTAL <nTotal> ] ;
    [ ID <nId> ];
    [ OF <oWnd> ] ;
    [ <update: UPDATE > ] ;
    [ FONT <oFont> ] ;
    [ PROMPT <cPrompt> ] ;
    [ <lNoPercentage: NOPERCENTAGE > ] ;
    [ <color: COLOR, COLORS> <nClrPane>, <nClrText> ] ;
    [ BARCOLOR <nClrBar>, <nClrBText> ] ;
=> ;
```

```
[ <oMeter> := ] TMeter():ReDefine( <nId>, bSETGET(<nActual>),;
    <nTotal>, <oWnd>, <.update.>, <oFont>, <cPrompt>,
<.lNoPercentage.>, ;
    <nClrPane>, <nClrText>, <nClrBar>, <nClrBText> )
```

Workshop

Comando

```
REDEFINE METER oMeter
VAR nActual
TOTAL nTotal
ID nId
OF oWnd
UPDATE
FONT oFont
PROMPT cPrompt
NOPERCENTAGE
COLOR, COLORS nClrPane, nClrText
BARCOLOR nClrBar, nClrBText
```

Cláusulas

- **BARCOLOR**
- **COLOR, COLORS** indica el color del texto. Si se pone un segundo color, será el del fondo. El color se define mediante la función nRGB() o similar.
- **FONT** indica el objeto fuente, en caso de ser distinto al indicado en momento del diseño con el workshop.
- **ID** indica el indicador en workshop. Puede ser una variable definida.
- **OF, WINDOWS, DIALOG** es el marco que lo contine. Normalmente será una caja de diálogo.
- **NOPERCENTAJE**
- **PROMPT, VAR** seguidas del texto literal o el nombre de la variable que lo contiene. El font y el color será el indicado en las cláusulas **FONT** y **COLOR**. En caso de no tenerlas, podrá los valores por defecto del marco o ventana que lo contiene.
- **TOTAL**
- **UPDATE** se utiliza para actualizar el texto cuando se refresca la caja o ventana de diálogo por medio de la instrucción <objeto diálogo>:refresh().
- **VAR** es la variable asociada al control.
- **VERTICAL** indica que el control se presentará en su versión vertical
- **WHEN** es la función que debe devolver .T. para que el control pueda tomar el foco. Es similar a la versión en DOS

Ejemplos

Trucos

Conclusión

MSGITEM

Concepto

Por programa...

fivewin.ch

```
#xcommand DEFINE MSGITEM [<oMsgItem>] ;
    [ OF <oMsgBar> ] ;
    [ PROMPT <cMsg> ] ;
    [ SIZE <nSize> ] ;
    [ FONT <oFont> ] ;
    [ <color: COLOR, COLORS> <nClrFore> [, <nClrBack> ] ] ;
    [ <bmp: BITMAP, BITMAPS> <cBitmap1> [, <cBitmap2> ] ] ;
    [ ACTION <uAction> ] ;
    [ TOOLTIP <cToolTip> ] ;
=> ;
    [<oMsgItem>:=] TMsgItem():New( <oMsgBar>, <cMsg>, <nSize>;
                                <oFont>, <nClrFore>, <nClrBack>, .t.,;
                                [<{uAction}>],;
                                [<cBitmap1>, <cBitmap2>], [<cToolTip>] )
```

Comando

```
DEFINE MSGITEM oMsgItem
    OF oMsgBar
    PROMPT cMsg
    SIZE nSize
    FONT oFont
    COLOR, COLORS nClrFore[, nClrBack]
    BITMAP, BITMAPS cBitmap1[, cBitmap2]
    ACTION uAction
    TOOLTIP cToolTip
```

Cláusulas

- **@** irá seguido de las coordenadas de posición. Si no se especifica la cláusula PIXEL, los números serán línea y columna. Indicando PÍXEL, los valores serán en pixels..
- **3D** sirve para dar aspecto de tres dimensiones.
- **ACTION** será la función que se ejecuta al hacer clic sobre el control.
- **ACJUST** permite al operador modificar el tamaño y posición del control.
- **ALIAS** indica el alias del fichero relacionado con el control. Si no se pone, la base de datos será la activa en ese momento.

- **BORDER** indica que saque un rectángulo marcando el borde del control en función de SIZE.
- **BRUSH** indica el objeto brocha que se utilizará como fondo del objeto.
- **BOX**
- **BUTTONSIZE** es el tamaño del botón en pixels.
- **CANCEL** permite al usuario cancelar la operación del control.
- **CENTER, CENTERED** indica que centre el texto en el espacio marcado por SIZE. Esta cláusula es excluyente con RIGHT.
- **COLOR, COLORS** indica el color del texto. Si se pone un segundo color, será el del fondo. El color se define mediante la función nRGB() o similar.
- **CURSOR** indica que cambie el cursor cuando este pase por encima.
- **DEFINE**
- **DESIGN** añade un borde al control, redimensionable en tiempo de ejecución. También permite que el control se pueda mover con el ratón.
- **DLL** indica el fichero DLL relacionado con el control.
- **FIELDS** indica los campos que se utilizarán en el control.
- **FILE** indica el fichero relacionado con el control.
- **FONT** indica el objeto fuente, en caso de ser distinto al indicado en momento del diseño con el workshop.
- **FROM...TO...** indica las coordenadas del rectángulo.
- **GROUP** indica si pertenece a un grupo.
- **HEAD** indica el texto o textos que irán en la cabecera.
- **HORIZONTAL** indica si el control será en su versión horizontal.
- **HELP**
- **ID** indica el indicador en workshop. Puede ser una variable definida.
- **MEMO** se utiliza en el control para indicar que ha de editarse un campo memo o una variable de texto largo
- **MENU**
- **MESSAGE** es el mensaje que aparecerá en la barra de mensajes cuando el cursor pase por encima del control.
- **NAME**
- **OF, WINDOWS, DIALOG** es el marco que lo contine. Normalmente será una caja de diálogo.
- **ON CHANGE** indica la función que se ejecutará cuando cambie el contenido o aspecto del control.
- **ON DRAWITEN**
- **ON DROP**
- **ON EDIT**

- **OPTION**
- **PAGE**
- **PICTURE** es la máscara igual que en DOS.
- **PIXEL, PIXELS**. Normalmente la posición se indica en líneas y columnas, y el tamaño en píxels. Con esta cláusula, todo será en píxels
- **PROMPT, VAR** seguidas del texto literal o el nombre de la variable que lo contiene. El font y el color será el indicado en las cláusulas **FONT** y **COLOR**. En caso de no tenerlas, podrá los valores por defecto del marco o ventana que lo contiene.
- **RAISED**
- **REDEFINE**
- **RIGHT** indica que visualice el texto justificado por la derecha, en el espacio marcado por SIZE. Esta cláusula es excluyente con CENTERED o CENTER
- **SELECT**
- **SHADOW, SHADES** indica que el texto tenga una pequeña sombra para simular 3D.
- **SIZE** indica el tamaño en píxels del campo que ocupará el texto. No indica el campo del texto, que eso está marcado por el propio texto y el font utilizado. Este tamaño se utiliza, por ejemplo para el color de fondo, el borde o la superposición sobre otros controles.
- **TITLE**
- **TOP/LEFT/RIGHT/BOTTOM/FLOAD**
- **TOOLTIP** indica el pequeño texto de ayuda que sale junto al control al dener unos segundos el cursor sobre él.
- **UPDATE** se utiliza para actualizar el texto cuando se refresca la caja o ventana de diálogo por medio de la instrucción <objeto diálogo>:refresh().
- **VALID** es la función que debe devolver .T. para que el control pueda dejar el foco y continuar con el programa. Es similar a la versión en DOS
- **VAR** es la variable asociada al control.
- **VERTICAL** indica que el control se presentará en su versión vertical
- **WHEN** es la función que debe devolver .T. para que el control pueda tomar el foco. Es similar a la versión en DOS

Ejemplos

Trucos

Conclusión

PAGES

Concepto

Por recursos...

[folder.ch](#)

```
#xcommand REDEFINE PAGES <oPag> ;  
    [ ID <nId> ] ;  
    [ OF <oWnd> ] ;  
    [ DIALOGS <DlgName,...> ] ;  
    [ OPTION <nOption> ] ;  
    [ ON CHANGE <uChange> ] ;  
    [ FONT <oFont> ] ;  
=> ;  
    <oPag> := TPages():Redefine( <nId>, <oWnd>, [{<DlgName>}], <nOption>;  
    [ bSETGET(<uChange>) ], <oFont> )
```

Workshop

Comando

Cláusulas

Ejemplos

Trucos

Conclusión

PEN

Concepto

Por programa...

fivewin.ch

```
#xcommand DEFINE PEN <oPen> ;
    [ STYLE <nStyle> ] ;
    [ WIDTH <nWidth> ] ;
    [ COLOR <nRGBColor> ] ;
    [ <of: OF, WINDOW, DIALOG> <oWnd> ] ;
=> ;
    <oPen> := TPen():New( <nStyle>, <nWidth>, <nRGBColor>, <oWnd> )

#xcommand ACTIVATE PEN <oPen> => <oPen>:Activate()
```

Comando

```
DEFINE PEN oPen
    STYLE nStyle
    WIDTH nWidth
    COLOR nRGBColor
    OF, WINDOW, DIALOG oWnd
```

Cláusulas

Ejemplos

Trucos

Conclusión

RADIO

Concepto

El **Radio Button** forma un conjunto de dos o mas opciones de las que solo podemos elegir una. La variable que se utiliza para controlar los Radio Button es una variable numérica que presenta el número de la opción elegida. Al activarse un Radio button, la opción señalada es la que corresponda al número contenido en la variable.

Por programa...

fivewin.ch

```
#xcommand @ <nRow>, <nCol> RADIO [ <oRadMenu> VAR ] <nVar> ;
    [ <prm: PROMPT, ITEMS> <cItems,...> ] ;
    [ <of: OF, WINDOW, DIALOG> <oWnd> ] ;
    [ <help:HELPID, HELP ID> <nHelpId,...> ] ;
    [ <change: ON CLICK, ON CHANGE> <uChange> ] ;
    [ COLOR <nClrFore> [,<nClrBack>] ] ;
    [ MESSAGE <cMsg> ] ;
    [ <update: UPDATE> ] ;
    [ WHEN <uWhen> ] ;
    [ SIZE <nWidth>, <nHeight> ] ;
    [ VALID <uValid> ] ;
    [ <lDesign: DESIGN> ] ;
    [ <lLook3d: 3D, _3D> ] ;
    [ <lPixel: PIXEL> ] ;
=> ;
    [ <oRadMenu> := ] TRadMenu():New( <nRow>, <nCol>, {<cItems>},;
    [bSETGET(<nVar>)], <oWnd>, [{<nHelpId>}], <{uChange}>;
    <nClrFore>, <nClrBack>, <cMsg>, <.update.>, <{uWhen}>;
    <nWidth>, <nHeight>, <{uValid}>, <.lDesign.>, <.lLook3d.>;
    <.lPixel.> )
```

Comando

```
@ nRow, nCol RADIO oRadMenu
VAR nVar
PROMPT, ITEMS cItems,...
OF, WINDOW, DIALOG oWnd
HELPID, HELP ID nHelpId,...
ON CLICK, ON CHANGE uChange
COLOR nClrFore[,nClrBack]
MESSAGE cMsg
UPDATE
WHEN
SIZE nwidth, nHeight
VALID uValid
DESIGN
```

3D, _3D
PIXEL

Cláusulas

- **@** irá seguido de las coordenadas de posición. Si no se especifica la cláusula **PIXEL**, los números serán línea y columna. Indicando **PÍXEL**, los valores serán en pixels..
- **3D** sirve para dar aspecto de tres dimensiones.
- **COLOR, COLORS** indica el color del texto. Si se pone un segundo color, será el del fondo. El color se define mediante la función **nRGB()** o similar.
- **DESIGN** añade un borde al control, redimensionable en tiempo de ejecución. También permite que el control se pueda mover con el ratón.
- **HELP**
- **MESSAGE** es el mensaje que aparecerá en la barra de mensajes cuando el cursor pase por encima del control.
- **OF, WINDOWS, DIALOG** es el marco que lo contine. Normalmente será una caja de diálogo.
- **ON CHANGE** indica la función que se ejecutará cuando cambie el contenido o aspecto del control.
- **ON CLICK**
- **PIXEL, PIXELS**. Normalmente la posición se indica en líneas y columnas, y el tamaño en píxels. Con esta cláusula, todo será en pixels
- **PROMPT, VAR** seguidas del texto literal o el nombre de la variable que lo contiene. El font y el color será el indicado en las cláusulas **FONT** y **COLOR**. En caso de no tenerlas, podrá los valores por defecto del marco o ventana que lo contiene.
- **SIZE** indica el tamaño en pixels del campo que ocupará el texto. No indica el campo del texto, que eso esta marcado por el propio texto y el font utilizado. Este tamaño se utiliza, por ejemplo para el color de fondo, el borde o la superposición sobre otros controles.
- **UPDATE** se utiliza para actualizar el texto cuando se refresca la caja o ventana de diálogo por medio de la instrucción **<objeto diálogo>:refresh()**.
- **VALID** es la función que debe devolver **.T.** para que el control pueda dejar el foco y continuar con el programa. Es similar a la versión en DOS
- **VAR** es la variable asociada al control.
- **WHEN** es la función que debe devolver **.T.** para que el control pueda tomar el foco. Es similar a la versión en DOS

Ejemplos

Por recursos...

fivewin.ch

```
#xcommand REDEFINE RADIO [ <oRadMenu> VAR ] <nVar> ;
    [ ID <nId,...> ] ;
    [ <of: OF, WINDOW, DIALOG> <oWnd> ] ;
    [ <help:HELPID, HELP ID> <nHelpId,...> ] ;
    [ <change: ON CHANGE, ON CLICK> <uChange> ] ;
    [ COLOR <nClrFore> [,<nClrBack>] ] ;
    [ MESSAGE <cMsg> ] ;
    [ <update: UPDATE> ] ;
    [ WHEN <uWhen> ] ;
    [ VALID <uValid> ] ;
=> ;
[ <oRadMenu> := ] TRadMenu():Redefine( [bSETGET(<nVar>)],;
    <oWnd>, [{<nHelpId>}], \{ <nId> \}, <{uChange}>, <nClrFore>;
    <nClrBack>, <cMsg>, <.update.>, <{uWhen}>, <{uValid}> )
```

Workshop

Comando

```
REDEFINE RADIO oRadMenu
VAR nVar
ID nId,...
OF, WINDOW, DIALOG oWnd
HELPID, HELP ID nHelpId,...
ON CHANGE, ON CLICK uChange
COLOR nClrFore[,nClrBack]
MESSAGE cMsg
UPDATE
WHEN uWhen
VALID uValid
```

Cláusulas

- **COLOR, COLORS** indica el color del texto. Si se pone un segundo color, será el del fondo. El color se define mediante la función nRGB() o similar.
- **HELP**
- **ID** indica el indicador en workshop. Puede ser una variable definida.
- **MESSAGE** es el mensaje que aparecerá en la barra de mensajes cuando el cursor pase por encima del control.
- **OF, WINDOWS, DIALOG** es el marco que lo contine. Normalmente será una caja de diálogo.
- **ON CHANGE** indica la función que se ejecutará cuando cambie el contenido o aspecto del control.
- **ON CLICK**

- **PROMPT, VAR** seguidas del texto literal o el nombre de la variable que lo contiene. El font y el color será el indicado en las cláusulas **FONT** y **COLOR**. En caso de no tenerlas, podrá los valores por defecto del marco o ventana que lo contiene.
- **UPDATE** se utiliza para actualizar el texto cuando se refresca la caja o ventana de diálogo por medio de la instrucción <objeto diálogo>:refresh().
- **VALID** es la función que debe devolver .T. para que el control pueda dejar el foco y continuar con el programa. Es similar a la versión en DOS
- **VAR** es la variable asociada al control.
- **WHEN** es la función que debe devolver .T. para que el control pueda tomar el foco. Es similar a la versión en DOS

Ejemplos

Trucos

Conclusión

RELEASE**Concepto****Por programa...****fivewin.ch**

```
#xcommand RELEASE <ClassName> <oObj1> [,<oObjN>] ;  
=> ;  
    <oObj1>:End() ; <oObj1> := nil ;  
    [ ; <oObjN>:End() ; <oObjN> := nil ]
```

Comando**Cláusulas****Ejemplos****Help****Conclusión**

SAY

Concepto

Este control visualiza un texto fijo en la caja de diálogo. Es muy similar al SAY de DOS pero con las posibilidades añadidas de un entorno Windows.

Se puede hacer por programa pero principalmente se utiliza por recursos. El objeto generado es de la clase **TSay**.

Por programa...

fivewin.ch

```
#xcommand @ <nRow>, <nCol> SAY [ <oSay> <label: PROMPT,VAR > ] <cText> ;
    [ PICTURE <cPict> ] ;
    [ <dlg: OF,WINDOW,DIALOG > <oWnd> ] ;
    [ FONT <oFont> ] ;
    [ <lCenter: CENTERED, CENTER > ] ;
    [ <lRight: RIGHT > ] ;
    [ <lBorder: BORDER > ] ;
    [ <lPixel: PIXEL, PIXELS > ] ;
    [ <color: COLOR,COLORS > <nClrText> [,<nClrBack> ] ] ;
    [ SIZE <nWidth>, <nHeight> ] ;
    [ <design: DESIGN > ] ;
    [ <update: UPDATE > ] ;
    [ <lShaded: SHADED, SHADOW > ] ;
    [ <lBox: BOX > ] ;
    [ <lRaised: RAISED > ] ;
=> ;
    [ <oSay> := ] TSay():New( <nRow>, <nCol>, <{cText}>;,
    [<oWnd>], [<cPict>], <oFont>, <.lCenter.>, ;
    <.lRight.>, <.lBorder.>;,
    <.lPixel.>, <nClrText>, <nClrBack>, <nWidth>, <nHeight>;,
    <.design.>, <.update.>, <.lShaded.>, <.lBox.>, <.lRaised.> )
```

Comando

```
@ nRow, nCol SAY oSay ;
    PROMPT,VAR cText ;
    PICTURE cPict ;
    OF,WINDOW,DIALOG oWnd ;
    FONT oFont ;
    CENTERED, CENTER ;
    RIGHT ;
    BORDER ;
    PIXEL, PIXELS ;
    COLOR,COLORS nClrText[, nClrBack ] ;
```

```
SIZE nWidth, nHeight ;  
DESIGN ;  
UPDATE ;  
SHADED, SHADOW ;  
BOX ;  
RAISED
```

Cláusulas

- **@** irá seguida de las coordenadas en formato línea,columna. Si está presente la cláusula **PIXEL**, las unidades serán píxel.
- **BORDER** indica que saque un rectángulo marcando el borde del texto en función de **SIZE**.
- **BOX** el objeto será una caja, sin texto. Junto con las cláusulas **SHADED** y **RAISED** se puede hacer en varios formatos. Ver mas abajo para los ejemplos.
- **CENTERED, CENTER** indica que centro el texto en el espacio marcado por **SIZE**. Esta cláusula es excluyente con **RIGHT**.
- **COLOR, COLORS** indica el color del texto. Si se pone un segundo color, será el del fondo.
- **DESIGN** añade un borde al texto, redimensionable en tiempo de ejecución. También permite que el texto se pueda mover con el ratón.
- **FONT** indica el objeto fuente. Si no se indica tomará el de la caja de diálogo que lo contiene.
- **OF, WINDOWS, DIALOG** es el marco que lo contine. Normalmente será una caja de diálogo.
- **PICTURE** es la máscara igual que en DOS.
- **PIXEL, PIXELS**. Normalmente la posición se indica en líneas y columnas, y el tamaño en píxels. Con esta cláusula, todo será en píxels
- **PROMPT, VAR** seguidas del texto literal o el nombre de la variable que lo contiene. El font y el color será el indicado en las cláusulas **FONT** y **COLOR**. En caso de no tenerlas, podrá los valores por defecto del marco o ventana que lo contiene.
- **RAISED** se utiliza para indicar la forma de la caja en caso de haber declarado **BOX**.
- **RIGHT** indica que visualice el texto justificado por la derecha, en el espacio marcado por **SIZE**. Esta cláusula es excluyente con **CENTERED** o **CENTER**
- **SHADOW** se utiliza para indicar la forma de la caja en caso de haber declarado **BOX**.
- **SIZE** indica el tamaño en píxels del campo que ocupará el texto. No indica el campo del texto, que eso esta marcado por el propio texto y el font utilizado. Este tamaño se utiliza, por ejemplo para el color de fondo, el borde o la superposición sobre otros controles.
- **UPDATE** se utiliza para actualizar el texto cuando se refresca la caja o ventana de diálogo por medio de la instrucción <objeto diálogo>.update().

- VAR

Ejemplos

Por recursos...

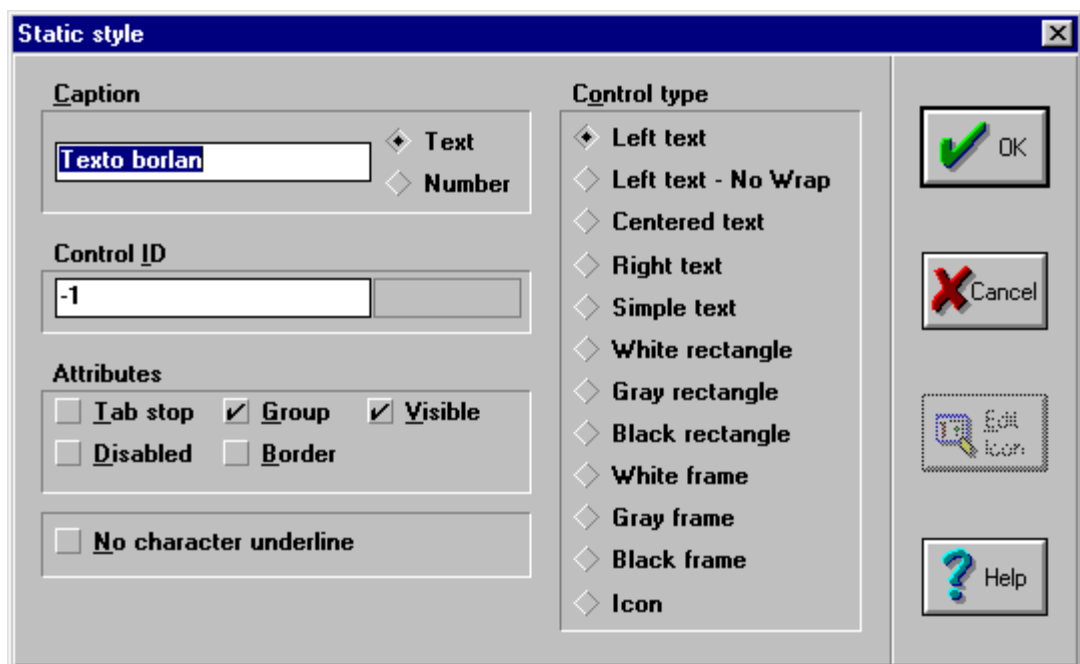
fivewin.ch

```
#xcommand REDEFINE SAY [<oSay>] ;
    [ <label: PROMPT, VAR> <cText> ] ;
    [ PICTURE <cPict> ] ;
    [ ID <nId> ] ;
    [ <dlg: OF,WINDOW,DIALOG > <ownd> ] ;
    [ <color: COLOR,COLORS > <nClrText> [,<nClrBack> ] ] ;
    [ <update: UPDATE > ] ;
    [ FONT <oFont> ] ;
=> ;
[ <oSay> := ] TSay():ReDefine( <nId>, <{cText}>, <ownd>, ;
    <cPict>, <nClrText>, <nClrBack>, <.update.>, <oFont> )
```

Workshop

Hay dos textos fijos. El texto con características normales, y el texto Borlan que necesita cargar BWCC.DLL.

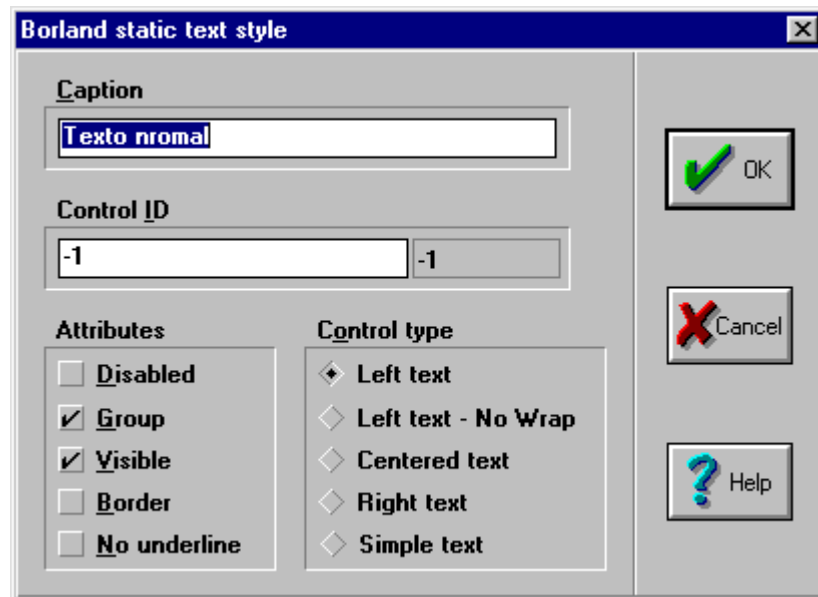
El estilo normal (se hace con la T blanca de la caja de herramientas), es el siguiente:



Entre las opciones de **Attributes** y **Control type** se pueden poner los formatos de borde, texto a la izquierda, centrado, derecha, cajas blanca, gris o negra y marcos blanco,

gris o negro. Por ejemplo para hacer un marco a un grupo de controles dentro de un diálogo, se puede poner: Border NO, White frame SI. En el orden **1,2** poner el marco con un número anterior a los controles (de no ser así los tapa)

El estilo Borlan (se hace con la T negra de la caja de herramientas), es el siguiente:



En el programa solo hará falta redefinir los objetos que vayan a tener alguna modificación dentro del PRG o en tiempo de ejecución.

Comando

```
REDEFINE SAY oSay ;
  PROMPT, VAR cText ;
  PICTURE cPict ;
  ID nId ;
  OF, WINDOW, DIALOG oWnd ;
  COLOR, COLORS nClrText[, nClrBack] ;
  UPDATE ;
  FONT oFont
```

Cláusulas

- **CENTER**
- **COLOR, COLORS** indica el color del texto. Si se pone un segundo color, será el del fondo.
- **FONT** indica el objeto fuente, en caso de ser distinto al indicado en momento del diseño con el workshop.
- **ID** es el identificador del WS. Como es lógico puede utilizarse una variable manifiesta que traducirá el preprocesador.
- **OF, WINDOWS, DIALOG** es el objeto del marco que lo contine. Normalmente será una caja de diálogo.

- **PICTURE** es la máscara igual que en DOS.
- **PROMPT**, **VAR** seguidas del texto literal o el nombre de la variable que lo contiene. El font y el color será el indicado en el momento del diseño con WS a menos que se indique en la cláusula **FONT** y **COLOR**.
- **RIGHT**
- **UPDATE** se utiliza para actualizar el texto cuando se refresca la caja de diálogo.
- **VAR**

Ejemplos

Como otros controles que se hacen por recursos, el resto de las características que pueden ser definidas y/o modificadas por el sistema de programación, han de ser definidas en tiempo de diseño con WS.

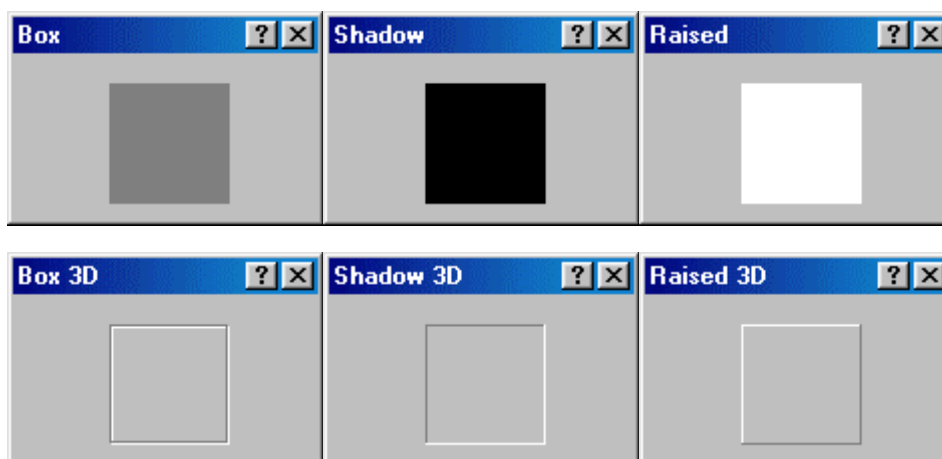
Trucos

Para hacer que una función se ejecute cuando se picha sobre un control Tsay, poner lo siguiente:

oSay:IWantClick := .T.

El resto es como en otro objeto.

Tres ejemplos de tipos de caja con y sin 3D.



También se puede definir la forma en el concepto **STYLE** del control en el workshop

Conclusión

SCROLL

Concepto

Por programa...

fivewin.ch

```
#xcommand @ <nRow>, <nCol> SCROLLBAR [ <oSbr> ] ;
[ <h: HORIZONTAL> ] ;
[ <v: VERTICAL> ] ;
[ RANGE <nMin>, <nMax> ] ;
[ PAGESIZE <nPgStep> ] ;
[ SIZE <nWidth>, <nHeight> ] ;
[ <up:UP, ON UP> <uUpAction> ] ;
[ <dn:DOWN, ON DOWN> <uDownAction> ] ;
[ <pgup:PAGEUP, ON PAGEUP> <uPgUpAction> ] ;
[ <pgdn:PAGEDOWN, ON PAGEDOWN> <uPgDownAction> ] ;
[ <pos: ON THUMBPOS> <uPos> ] ;
[ <pixel: PIXEL> ] ;
[ <color: COLOR,COLORS> <nClrText> [,<nClrBack>] ] ;
[ OF <oWnd> ] ;
[ MESSAGE <cMsg> ] ;
[ <update: UPDATE> ] ;
[ WHEN <uWhen> ] ;
[ VALID <uValid> ] ;
[ <lDesign: DESIGN> ] ;
=> ;
[ <oSbr> := ] TScrollBar():New( <nRow>, <nCol>, <nMin>, <nMax>,
<nPgStep>,;
    (.not.<.h.>) [.or. <.v.> ], <oWnd>, <nWidth>, <nHeight> ,;
    [<{uUpAction}>], [<{uDownAction}>], [<{uPgUpAction}>], ;
    [<{uPgDownAction}>], [\{|nPos| <uPos> \}], [<.pixel.>],;
    <nClrText>, <nClrBack>, <cMsg>, <.update.>, <{uWhen}>, <{uValid}>,;
    <.lDesign.> )

// for 'non-true ScrollBars' ( when using WS_VSCROLL or WS_HSCROLL styles )
```

Comando

```
@ nRow, nCol SCROLLBAR oSbr
HORIZONTAL
VERTICAL
RANGE nMin, nMax
PAGESIZE nPgStep
SIZE nwidth, nHeight
ON UP uUpAction
DOWN, ON DOWN uDownAction
PAGEUP, ON PAGEUP uPgUpAction
PAGEDOWN, ON PAGEDOWN uPgDownAction
ON THUMBPOS uPos
```

```
PIXEL  
COLOR, COLORS nClrText[, nClrBack]  
OF oWnd  
MESSAGE cMsg  
UPDATE  
WHEN  
VALID uValid  
DESIGN
```

Cláusulas

- **@** irá seguido de las coordenadas de posición. Si no se especifica la cláusula **PIXEL**, los números serán línea y columna. Indicando **PÍXEL**, los valores serán en pixels..
- **COLOR, COLORS** indica el color del texto. Si se pone un segundo color, será el del fondo. El color se define mediante la función **nRGB()** o similar.
- **DOWN**
- **DESIGN** añade un borde al control, redimensionable en tiempo de ejecución. También permite que el control se pueda mover con el ratón.
- **HORIZONTAL** indica si el control será en su versión horizontal.
- **ICON**
- **MESSAGE** es el mensaje que aparecerá en la barra de mensajes cuando el cursor pase por encima del control.
- **OF, WINDOWS, DIALOG** es el marco que lo contine. Normalmente será una caja de diálogo.
- **ON THUMBPOS**
- **PAGEDOWN**
- **PAGESTEP**
- **PAGEUP**
- **PIXEL, PIXELS**. Normalmente la posición se indica en líneas y columnas, y el tamaño en píxels. Con esta cláusula, todo será en pixels
- **RANGE**
- **SIZE** indica el tamaño en pixels del campo que ocupará el texto. No indica el campo del texto, que eso está marcado por el propio texto y el font utilizado. Este tamaño se utiliza, por ejemplo para el color de fondo, el borde o la superposición sobre otros controles.
- **UP**
- **UPDATE** se utiliza para actualizar el texto cuando se refresca la caja o ventana de diálogo por medio de la instrucción `<objeto diálogo>.refresh()`.
- **VALID** es la función que debe devolver `.T.` para que el control pueda dejar el foco y continuar con el programa. Es similar a la versión en DOS
- **VERTICAL** indica que el control se presentará en su versión vertical

- **WHEN** es la función que debe devolver .T. para que el control pueda tomar el foco. Es similar a la versión en DOS

Ejemplos

Por recursos...

fivewin.ch

```
#xcommand DEFINE SCROLLBAR [ <oSbr> ] ;
    [ <h: HORIZONTAL> ] ;
    [ <v: VERTICAL> ] ;
    [ RANGE <nMin>, <nMax> ] ;
    [ PAGESTEP <nPgStep> ] ;
    [ <up:UP, ON UP> <uUpAction> ] ;
    [ <dn:DOWN, ON DOWN> <uDownAction> ] ;
    [ <pgup:PAGEUP, ON PAGEUP> <uPgUpAction> ] ;
    [ <pgdn:PAGEDOWN, ON PAGEDOWN> <uPgDownAction> ] ;
    [ <pos: ON THUMBPOS> <uPos> ] ;
    [ <color: COLOR,COLORS> <nClrText> [,<nClrBack>] ] ;
    [ <of: OF, WINDOW, DIALOG> <ownd> ] ;
    [ MESSAGE <cMsg> ] ;
    [ <update: UPDATE> ] ;
    [ WHEN <uWhen> ] ;
    [ VALID <uValid> ] ;
=> ;

[ <oSbr> := ] TScrollBar():WinNew( <nMin>, <nMax>, <nPgStep>, ;
(.not.<.h.>) [.or. <.v.> ], <ownd>, [<{uUpAction}>], ;
[<{uDownAction}>], [<{uPgUpAction}>], ;
[<{uPgDownAction}>], [\{|nPos| <uPos> \}], ;
<nClrText>, <nClrBack>, <cMsg>, <.update.>, <{uWhen}>, <{uValid}> )

#xcommand REDEFINE SCROLLBAR [ <oSbr> ] ;
    [ ID <nID> ] ;
    [ RANGE <nMin>, <nMax> ] ;
    [ PAGESTEP <nPgStep> ] ;
    [ <up:UP, ON UP, ON LEFT> <uUpAction> ] ;
    [ <dn:DOWN, ON DOWN, ON RIGHT> <uDownAction> ] ;
    [ <pgup:PAGEUP, ON PAGEUP> <uPgUpAction> ] ;
    [ <pgdn:PAGEDOWN, ON PAGEDOWN> <uPgDownAction> ] ;
    [ <pos: ON THUMBPOS> <uPos> ] ;
    [ <color: COLOR,COLORS> <nClrText> [,<nClrBack>] ] ;
    [ OF <oDlg> ] ;
    [ MESSAGE <cMsg> ] ;
    [ <update: UPDATE> ] ;
    [ WHEN <uWhen> ] ;
    [ VALID <uValid> ] ;
=> ;

[ <oSbr> := ] TScrollBar():Redefine( <nID>, <nMin>, <nMax>,
<nPgStep>, ;
<oDlg>, [<{uUpAction}>], [<{uDownAction}>], [<{uPgUpAction}>], ;
[<{uPgDownAction}>], [\{|nPos| <uPos> \}], <nClrText>, ;
<nClrBack>, <cMsg>, <.update.>, <{uWhen}>, <{uValid}> )
```

Workshop

Comando

```

DEFINE SCROLLBAR oSbr
  HORIZONTAL
  VERTICAL
  RANGE nMin, nMax
  PAGESIZE nPgStep
  UP, ON UP uUpAction
  DOWN, ON DOWN uDownAction
  PAGEUP, ON PAGEUP uPgUpAction
  PAGEDOWN, ON PAGEDOWN uPgDownAction
  ON THUMBPOS uPos
  COLOR,COLORS nClrText[,nClrBack]
  OF, WINDOW, DIALOG oWnd
  MESSAGE cMsg
  UPDATE
  WHEN uWhen
  VALID uValid

REDEFINE SCROLLBAR oSbr
  ID nID
  RANGE nMin, nMax
  PAGESIZE nPgStep
  UP, ON UP, ON LEFT uUpAction
  DOWN, ON DOWN, ON RIGHT uDownAction
  PAGEUP, ON PAGEUP uPgUpAction
  PAGEDOWN, ON PAGEDOWN uPgDownAction
  ON THUMBPOS uPos
  COLOR,COLORS nClrText[,nClrBack]
  OF oDlg
  MESSAGE cMsg
  UPDATE
  WHEN uWhen
  VALID uValid

```

Cláusulas

- **COLOR, COLORS** indica el color del texto. Si se pone un segundo color, será el del fondo. El color se define mediante la función nRGB() o similar.
- **DOWN**
- **ID** indica el indicador en workshop. Puede ser una variable definida.
- **ICON**
- **MESSAGE** es el mensaje que aparecerá en la barra de mensajes cuando el cursor pase por encima del control.
- **OF, WINDOWS, DIALOG** es el marco que lo contine. Normalmente será una caja de diálogo.
- **ON THUMBPOS**
- **PAGEDOWN**

- **PAGESTEP**
- **PAGEUP**
- **RANGE**
- **UP**
- **UPDATE** se utiliza para actualizar el texto cuando se refresca la caja o ventana de diálogo por medio de la instrucción <objeto diálogo>.refresh().
- **VALID** es la función que debe devolver .T. para que el control pueda dejar el foco y continuar con el programa. Es similar a la versión en DOS
- **WHEN** es la función que debe devolver .T. para que el control pueda tomar el foco. Es similar a la versión en DOS

Ejemplos

Trucos

Conclusión

TABS

Concepto

Por programa...

folder.ch

```
#xcommand @ <nRow>, <nCol> TABS [<oTabs>] ;
[ <of: OF, WINDOW, DIALOG> <oWnd> ] ;
[ <prm: PROMPT, PROMPTS, ITEMS> <cPrompt,...> ] ;
[ <act: ACTION, EXECUTE, ON CHANGE> <uAction> ] ;
[ <lPixel: PIXEL> ] ;
[ <lDesign: DESIGN> ] ;
[ <color: COLOR, COLORS> <nClrFore> [,<nClrBack>] ] ;
[ OPTION <nOption> ] ;
[ SIZE <nWidth>, <nHeight> ] ;
[ MESSAGE <cMsg> ] ;
=> ;
[<oTabs> := ] TTabs():New( <nRow>, <nCol>;
[ \{<cPrompt>\}], [{|nOption|<uAction>}],;
<oWnd>, <nOption>, <nClrFore>, <nClrBack>, <.lPixel.>;
<.lDesign.>, <nWidth>, <nHeight>, <cMsg> )

#xcommand REDEFINE PAGES <oPag> ;
[ ID <nId> ] ;
[ OF <oWnd> ] ;
[ DIALOGS <DlgName,...> ] ;
[ OPTION <nOption> ] ;
[ ON CHANGE <uChange> ] ;
[ FONT <oFont> ] ;
=> ;
<oPag> := TPages():Redefine( <nId>, <oWnd>, [{<DlgName>}], <nOption>;
[ bSETGET(<uChange>) ], <oFont> )

#endif
```

Comando

Cláusulas

Ejemplos

Por recursos...**folder.ch**

```
#xcommand REDEFINE TABS [<oTabs>] ;
    [ ID <nId> ] ;
    [ <of: OF, WINDOW, DIALOG> <oWnd> ] ;
    [ <prm: PROMPT, PROMPTS, ITEMS> <cPrompt,...> ] ;
    [ <act: ACTION, EXECUTE> <uAction> ] ;
    [ <color: COLOR, COLORS> <nClrFore> [,<nClrBack>] ] ;
    [ OPTION <nOption> ] ;
=> ;
[<oTabs> := ] TTab():ReDefine( <nId>, [\{<cPrompt>\}],;
    [{|<nOption>|<uAction>}], <oWnd>, <nOption>, <nClrFore>, <nClrBack> )
```

Workshop**Comando****Cláusulas****Ejemplos****Trucos****Conclusión**

TIMER

Concepto

Por programa...

fivewin.ch

```
#xcommand DEFINE TIMER [ <oTimer> ] ;  
    [ INTERVAL <nInterval> ] ;  
    [ ACTION <uAction,...> ] ;  
    [ <of: OF, WINDOW, DIALOG> <oWnd> ] ;  
=> ;  
    [ <oTimer> := ] TTimer():New( <nInterval>, [ \{ | | <uAction> \} ], <oWnd> )  
  
#xcommand ACTIVATE TIMER <oTimer> => <oTimer>:Activate()
```

Comando

```
DEFINE TIMER oTimer  
    INTERVAL nInterval  
    ACTION uAction,...  
    OF, WINDOW, DIALOG oWnd
```

Cláusulas

Ejemplos

Trucos

UNTIL

Concepto

Por programa...

fivewin.ch

```
#xcommand DO                => while .t.  
#xcommand UNTIL <uExpr> => if <uExpr>; exit; end; end
```

Comando

Cláusulas

Ejemplos

Trucos

Conclusión

VBX

Concepto

Por programa...

fivewin.ch

```
#xcommand @ <nRow>, <nCol> VBX [<oVbx>] ;
    [ OF <oWnd> ] ;
    [ SIZE <nWidth>, <nHeight> ] ;
    [ <file: FILE, FILENAME, DISK> <cVbxFile> ] ;
    [ CLASS <cVbxClass> ] ;
    [ ON <cClause1> <uAction1> ;
      ON <cClauseN> <uActionN> ] ] ;
    [ WHEN <uWhen> ] ;
    [ VALID <uValid> ] ;
    [ <lPixel: PIXEL> ] ;
    [ <lDesign: DESIGN> ] ;
=> ;
[ <oVbx> := ] TVbControl():New( <nRow>, <nCol>, <nWidth>, <nHeight>;
  <oWnd>, <cVbxFile>, <cVbxClass>, ;
  \{ [ <(cClause1)>, _PARAM_BLOCK_10_( <uAction1> ) ] ;
    [, <(cClauseN)>, _PARAM_BLOCK_10_( <uActionN> ) ] ;
  \}, [ <{uWhen}> ], [ <{uValid}> ], <.lPixel.>, <.lDesign.> )
```

Comando

```
@ nRow, nCol VBX oVbx
  OF oWnd
  SIZE nWidth, nHeight
  FILE, FILENAME, DISK cVbxFile
  CLASS cVbxClass
  ON cClause1 uAction1
  ON cClauseN uActionN
  WHEN uWhen
  VALID uValid
  PIXEL
  DESIGN
```

Cláusulas

- @ irá seguido de las coordenadas de posición. Si no se especifica la cláusula PIXEL, los números serán línea y columna. Indicando PÍXEL, los valores serán en pixels..

- **CLASS**
- **DESIGN** añade un borde al control, redimensionable en tiempo de ejecución. También permite que el control se pueda mover con el ratón.
- **FILE** indica el fichero relacionado con el control.
- **OF, WINDOWS, DIALOG** es el marco que lo contine. Normalmente será una caja de diálogo.
- **ON**
- **PIXEL, PIXELS**. Normalmente la posición se indica en líneas y columnas, y el tamaño en píxels. Con esta cláusula, todo será en píxels
- **SIZE** indica el tamaño en píxels del campo que ocupará el texto. No indica el campo del texto, que eso está marcado por el propio texto y el font utilizado. Este tamaño se utiliza, por ejemplo para el color de fondo, el borde o la superposición sobre otros controles.
- **VALID** es la función que debe devolver .T. para que el control pueda dejar el foco y continuar con el programa. Es similar a la versión en DOS
- **WHEN** es la función que debe devolver .T. para que el control pueda tomar el foco. Es similar a la versión en DOS

Ejemplos

Por recursos...

fivewin.ch

```
#xcommand REDEFINE VBX [<oControl>] ;
    [ ID <nId> ] ;
    [ OF <oDlg> ] ;
    [ COLOR <nClrFore> [,<nClrBack>] ] ;
    [ ON <cClause1> <uAction1> ;
      [ ON <cClauseN> <uActionN> ] ] ;
=> ;
    [ <oControl> := ] TVbControl():ReDefine( <nId>, <oDlg>;
      <nClrFore>, <nClrBack>;
      \{ [ <(cClause1)>, _PARAM_BLOCK_10_( <uAction1> ) ] ;
        [,<(cClauseN)>, _PARAM_BLOCK_10_( <uActionN> ) ] ;
      \} )
```

Workshop

Comando

```
REDEFINE VBX oControl
  ID nId
  OF oDlg
  COLOR nClrFore[,nClrBack]
  ON cClause1 uAction1
```

ON cClauseN uActionN

Cláusulas

- **COLOR, COLORS** indica el color del texto. Si se pone un segundo color, será el del fondo. El color se define mediante la función nRGB() o similar.
- **ID** indica el indicador en workshop. Puede ser una variable definida.
- **OF, WINDOWS, DIALOG** es el marco que lo contine. Normalmente será una caja de diálogo.
- **ON**

Ejemplos

Trucos

Conclusión

WINDOWS

Concepto

Microsoft Windows es un sistema operativo dominado por las ventanas, de hecho, Window en inglés significa ventana. Para éste sistema operativo, al igual que para cualquier otro sistema basado en entornos gráficos, una ventana es el medio a través del cual se muestra la información al usuario, con la gran ventaja de que pueden coexistir múltiples ventanas trabajando de forma simultánea, mostrando información independiente de uno o más programas.

De este modo, es fundamental saber crear y gestionar las ventanas para empezar a programar en Fivewin. Además, la gran mayoría de controles existentes en Fivewin derivan del objeto window, por lo cual heredarán gran parte de sus características.

Para poder trabajar con una ventana será necesario realizar dos tipos de operaciones:

Declarar la ventana: Indicando sus características fundamentales como sus dimensiones, título, color, etc..

Activar la ventana: Nos permitirá mostrarla en pantalla e indicar cómo se comportará ante determinados eventos (cerrarla, hacer click, minimizarla, etc.).

Este capítulo sobre el objeto window se centrará en la primera de las operaciones a realizar, esto es, cómo definir una ventana y todas sus características.

SINTAXIS

```
DEFINE WINDOW <oWnd> ;  
    [ FROM <nTop>, <nLeft> TO <nBottom>, <nRight> ] ;  
    [ TITLE <cTitle> ] ;  
    [ <color: COLOR, COLORS> <nClrFore> [, <nClrBack>] ] ;  
    [ OF <oParent> ] ;  
    [ BRUSH <oBrush> ] ;  
    [ CURSOR <oCursor> ] ;  
    [ ICON <oIcon> ] ;  
    [ MENU <oMenu> ] ;  
    [ STYLE <nStyle> ] ;
```

```
[ BORDER [ NONE | SINGLE ] ] ;  
[ NOSYSMENU | NO SYSMENU ] ;  
[ NOCAPTION | NO CAPTION | NO TITLE ] ;  
[ NOICONIZE | NOMINIMIZE ] ;  
[ NOZOOM | NO ZOOM | NOMAXIMIZE | NO MAXIMIZE ] ;  
[ VSCROLL | VERTICAL SCROLL ] ;  
[ HSCROLL | HORIZONTAL SCROLL ]
```

DEFINIR EL OBJETO

Como todo en Fivewin, una ventana no deja de ser un objeto, por lo cual el primer paso para confeccionar una ventana será declarar la variable contenedora y dotarla de las características propias de un objeto window

Define Window oWnd

Al definir un objeto, como en este caso una ventana, hay una serie de variables (propiedades) que empiezan a tomar determinados valores y a las cuales podemos acceder a lo largo del resto del programa.

Nosotros iremos conociendo estas propiedades conforme vayan apareciendo a lo largo de la definición del objeto

- Propiedades:

hWnd:

Número correspondiente al manejador asociado al objeto. Podríamos decir que es la "matrícula" de la ventana.

- Métodos:

ClassName()

Devuelve el nombre de la clase a la que pertenece la ventana

SITUAR LA VENTANA

A continuación indicaremos su posición dentro de la pantalla y su tamaño.

From 1,1 TO 10,20

Al igual que en clipper, estas coordenadas se refieren a la esquina superior izquierda e inferior derecha de la ventana.

En Fivewin podemos trabajar con dos unidades de medida diferentes para indicar una posición dentro de la pantalla del ordenador.

Por defecto, Fivewin utiliza como unidad de medida en la definición de una ventana filas y columnas, exactamente igual que Clipper, pero con la diferencia de que al estar trabajando en un entorno gráfico que admite múltiples resoluciones de pantalla, no estamos limitados a 25 filas x 80 columnas, sino que dependerá del área de trabajo definido en nuestro sistema.

TABLA EQUIVALENCIAS ENTRE MODO GRAFICO Y TEXTO

VERTICAL		HORIZONTAL		FILAS		COLUMNAS
480	X	640		30	X	80
600	X	800		37.5	X	100
768	X	1024		48	X	128
1024	X	1280		64	X	160

Si nos fijamos con atención veremos que cada celda del modo texto corresponde a 8x16 pixel. Estos dos valores están declarados dentro del fichero *constant.ch* en las constantes WIN_CHARPIX_H y WIN_CHARPIX_W, de lo cual podemos deducir que internamente Fivewin siempre trabaja en pixel. Esto nos permite, por ejemplo, utilizar decimales y hacer un say en la posición 4.3 , 6.5

Si preferimos usar directamente como unidad de medida el pixel sólo hemos de indicar la cláusula PIXEL tras las coordenadas.

Así, si queremos trabajar en un entorno tipo "texto"

Define Window oWnd From 5,7 To 15,75

Si queremos trabajar en un entorno tipo "gráfico"

Define Window oWnd From 150,425 To 550,610 PIXEL

De igual modo que podemos definir la posición y el tamaño de una ventana a la hora de crear el objeto window, existen una serie de propiedades y métodos propios de esta clase que nos permiten modificar su situación y dimensiones a lo largo del resto del programa

- Propiedades:

nTop, nLeft, nbottom, nRight

Contienen la posición en pixel de la esquina superior izquierda y de la inferior derecha de la ventana. Si desplazamos o dimensionamos la ventana manualmente será necesario ejecutar :CoorsUpdate() para actualizar estos valores. Por el contrario si la desplazamos con los métodos aportados por el objeto window la actualización se hace de forma automática

nWidth, nHeight

Ancho y alto de nuestra ventana expresado en pixels

- Métodos:

nWidth(nAncho) , nHeight(nAlto)

Permiten modificar el ancho y alto de la ventana respectivamente. Para ello le pasaremos como parámetro el nuevo valor expresado en pixels

Move(nFila ,nCol)

Desplaza la ventana a una nueva posición

CoorsUpdate(nil)

Fuerza la actualización de los datos almacenados en nTop, nLeft, nBottom y nRight

GetCliRect(nil)

Devuelve un objeto de la clase **tRect** con las coordenadas interiores de la ventana, esto es, la zona de trabajo de la ventana (incluidas botonbar y messagebar).

Estas coordenadas son relativas a la ventana no a la pantalla, esto es, siempre empiezan en 0,0

Propiedades de la clase tRect

nTop, nLeft, nBottom, nRight

Métodos de la clase tRect

n width, nHeight

GetRect(nil)

Devuelve un objeto de la clase tRect con las coordenadas de la ventana completa respecto a la pantalla (incluida barra de título y menú). Este método devuelve los mismos valores que las propiedades nTop ... nRight de la ventana, pero en un objeto independiente

SetCoors(oRect)

Modifica el tamaño y posición de la ventana de acuerdo a los datos almacenados en el objeto oRect

nVertRes()

Devuelve la resolución vertical del escritorio de Windows

nHorzRes()

Devuelve la resolución horizontal del escritorio de Windows

- Funciones relacionadas

GetCoors(oWnd:hWnd)

Devuelve un array de 4 posiciones con las coordenadas de la ventana completa respecto a la pantalla (incluida barra de título y menú)

GetSysMetrics(nValor)

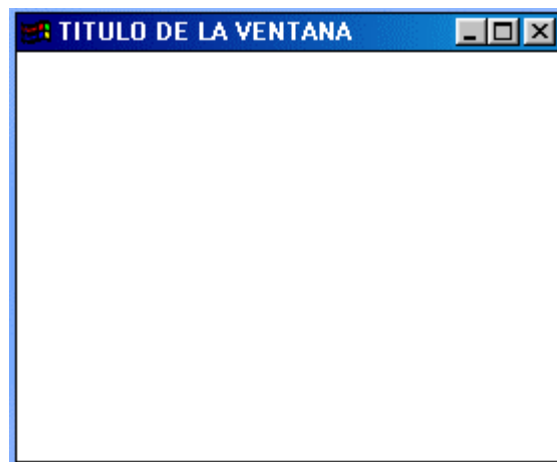
nValor := 0 Resolución horizontal

nValor := 1 Resolución vertical de la pantalla

TITULO DE LA VENTANA

Para ponerle título a nuestra ventana utilizaremos la cláusula TITLE

TITLE "TITULO DE LA VENTANA"



- Propiedades:

cCaption:

Variable tipo texto que contine el título (caption) de nuestra ventana

- Métodos:

cTitle(cNuevoTitulo), SetText(cNuevoTitulo):

Permite cambiar el caption de la ventana

DAR COLOR A LA VENTA

El siguiente paso a realizar para confeccionar nuestra ventana es indicar los colores a aplicar por defecto. El primer color es de escritura y el segundo el color del fondo de la ventana

COLOR "B/R"

Al igual que en muchos otros casos dentro de Fivewin, para indicar el color que deseamos aplicar podemos utilizar la sintaxis propia de clipper (<Texto>/<Fondo>) o usar los métodos proporcionados por Fivewin. En este último caso, los colores de texto y fondo vienen definidos por un nº de color dentro de una paleta de 16.000.000 de colores.

Para confeccionar un color determinado utilizaremos la función

RGB(nRojo, nVerde, nAzul)

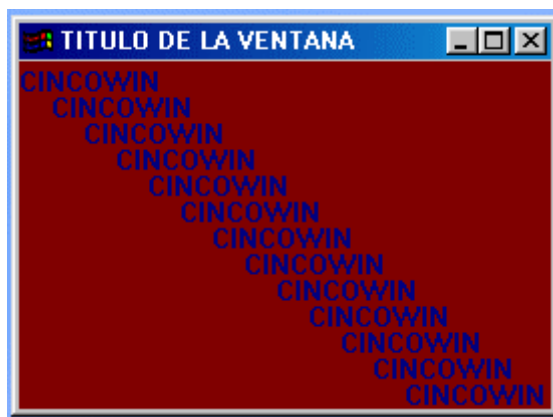
Devuelve el color correspondiente a la mezcla de los 3 colores básicos en un rango de 0 a 255 para cada color

Así, si utilizamos la sintaxis xBase de clipper

COLOR "B/R"

Si por el contrario optamos por la sintaxis Fivewin

COLOR rgb(0,0,128) , rgb(128,0,0)



Como podemos apreciar, la sintaxis Fivewin nos ofrece un rango mucho mayor de colores, algunos de los cuales se encuentran predefinidos en el fichero *colors.ch*

- Propiedades:

nClrText

Devuelve el código de color por defecto correspondiente al texto

nClrPane

Devuelve el código de color del fondo de la ventana

- Métodos

nSelColor(ltipo)

Permite seleccionar de forma interactiva tanto el color de texto como de fondo de la ventana

lTipo := .t. selección color texto

lTipo := .f. selección color fondo

nSetColor(nTexto, nFondo)

Permite modificar el color de texto y fondo de la ventana

- Funciones relacionadas

nGetForeRGB (cColorxBase)

Retorna el N° de color RGB correspondiente al color de texto indicado en formato xBase

nGetBackRGB (cColorxBase)

Retorna el N° de color RGB correspondiente al color de fondo indicado en formato xBase

nRGBred (nColor)

Devuelve el valor de los bytes correspondientes al color rojo dentro del color indicado

nRGBgreen (nColor)

Devuelve el valor de los bytes correspondientes al color verde dentro del color indicado

nRGBblue (nColor)

Devuelve el valor de los bytes correspondientes al color azul dentro del color indicado

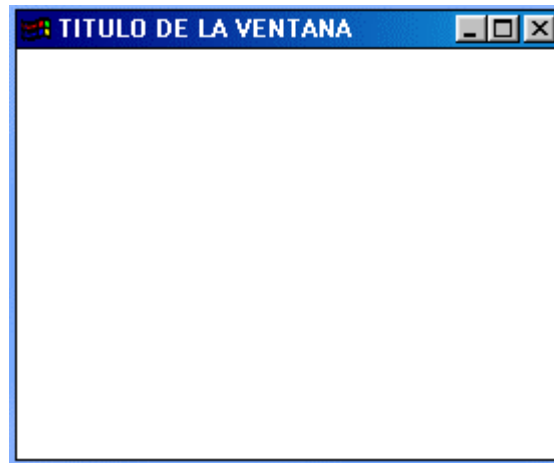
ESTILOS

Los estilos de una ventana determinan cómo se va a mostrar dicha ventana en la pantalla y qué opciones va a tener activas de todas las disponibles.

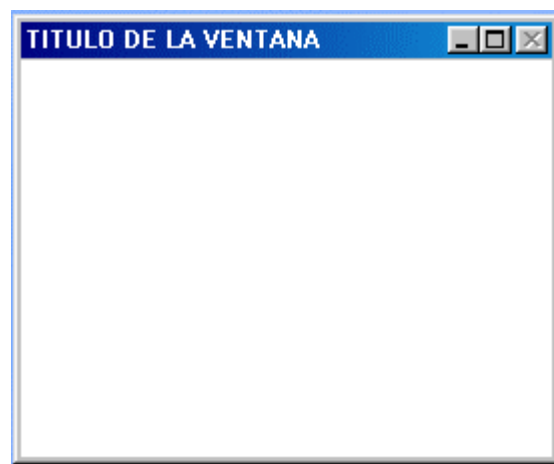
El estilo genérico de una ventana es `Overlapped_window`, lo que significa que posee título, menú de sistema, botones de maximizar, minimizar y cerrar, icono y es redimensionable.

Todas las características propias de una ventana `overlapper window` se pueden deshabilitar a través de una serie de cláusulas en el momento de definir la ventana

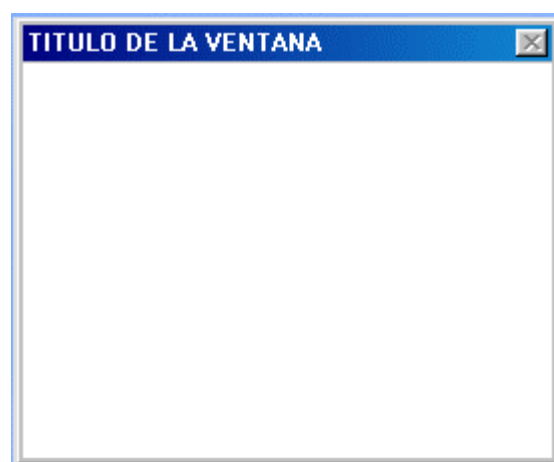
BORNER NONE: La ventana adquiere un borde consistente en una delgada línea casi imperceptible al mismo tiempo que impide que ésta sea redimensionada.



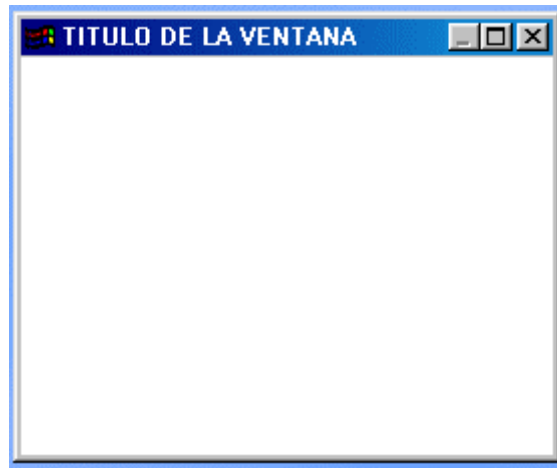
NOSYSMENU: Hace desaparecer el menú de sistema, el icono de la ventana e inhabilita el botón de cerrar la ventana



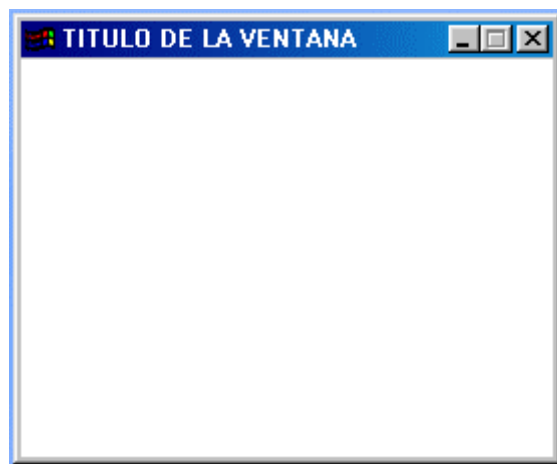
NOCAPTION: Al igual que NOSYSMENU hace desaparecer el menú de sistema, su icono e inhabilita el boton cerrar. Además hace desaparecer los botones maximizar y minimizar y la referencia de la barra de tareas. Curiosamente, mantiene el título o caption de la ventana.



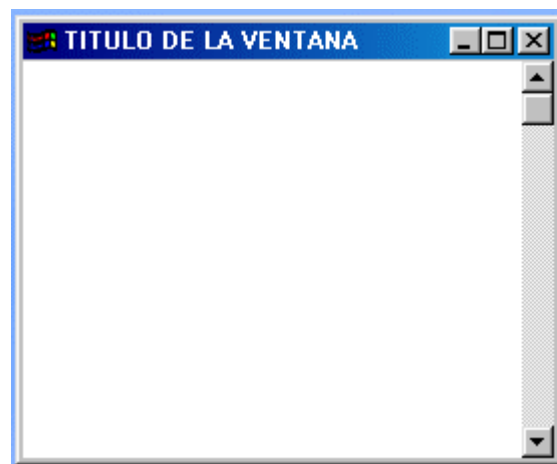
NOICONIZE: Inhabilita el botón minimizar.



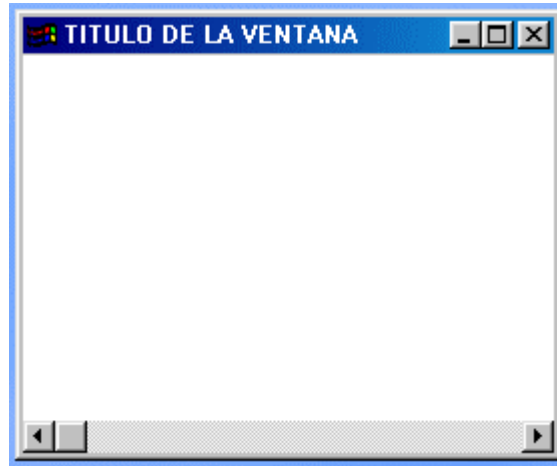
NOMAXIMIZE: Inhabilita el botón maximizar.



VSCROLL: Dota a la ventana de una barra de scroll vertical



HSCROLL: Dota a la ventana de una barra de scroll horizontal



Para asignar otros estilos a la ventana utilizaremos la cláusula STYLE, indicando tantos estilos como deseemos, SIEMPRE en mayúsculas. Si queremos indicar más de un estilo usaremos la función nOr() o los enviaremos como sumandos, siendo aconsejable el uso de la función

STYLE nOr(WS_POPUP, WS_VSCROLL)

o

STYLE WS_POPUP+WS_VSCROLL

- Propiedades:

nStyle

Contiene el valor numérico correspondiente a la suma de los estilos de la ventana

Por programa...

fivewin.ch

```
#xcommand DEFINE WINDOW [<oWnd>] ;
[ MDICHILD ] ;
[ FROM <nTop>, <nLeft> TO <nBottom>, <nRight> ] ;
[ TITLE <cTitle> ] ;
[ BRUSH <oBrush> ] ;
[ CURSOR <oCursor> ] ;
[ MENU <oMenu> ] ;
[ MENUINFO <nMenuInfo> ] ;
[ ICON <oIco> ] ;
[ OF <oParent> ] ;
[ <vscroll: VSCROLL, VERTICAL SCROLL> ] ;
[ <hscroll: HSCROLL, HORIZONTAL SCROLL> ] ;
[ <color: COLOR, COLORS> <nClrFore> [,<nClrBack>] ] ;
[ <pixel: PIXEL> ] ;
[ STYLE <nStyle> ] ;
[ <HelpId: HELPID, HELP ID> <nHelpId> ] ;
[ BORDER <border: NONE, SINGLE> ] ;
```

```

[ <NoSysMenu: NOSYSMENU, NO SYSMENU> ] ;
[ <NoCaption: NOCAPTION, NO CAPTION, NO TITLE> ] ;
[ <NoIconize: NOICONIZE, NOMINIMIZE> ] ;
[ <NoMaximize: NOZOOM, NO ZOOM, NOMAXIMIZE, NO MAXIMIZE> ] ;
=> ;
[<oWnd> := ] TMDiChild():New( <nTop>, <nLeft>, <nBottom>, <nRight>,;
    <cTitle>, <nStyle>, <oMenu>, <oParent>, <oIco>, <.vscroll.>,
<nClrFore>;,
    <nClrBack>, <oCursor>, <oBrush>, <.pixel.>, <.hscroll.>;,
    <nHelpId>, [Upper(<(border)>)], !<.NoSysMenu.>, !<.NoCaption.>;,
    !<.NoIconize.>, !<.NoMaximize.>, [<nMenuInfo>] )

#xcommand DEFINE WINDOW <oWnd> ;
[ FROM <nTop>, <nLeft> TO <nBottom>, <nRight> ] ;
[ TITLE <cTitle> ] ;
[ STYLE <nStyle> ] ;
[ MENU <oMenu> ] ;
[ BRUSH <oBrush> ] ;
[ ICON <oIcon> ] ;
[ MDI ] ;
[ <color: COLOR, COLORS> <nClrFore> [,<nClrBack>] ] ;
[ <vScroll: VSCROLL, VERTICAL SCROLL> ] ;
[ <hScroll: HSCROLL, HORIZONTAL SCROLL> ] ;
[ MENUINFO <nMenuInfo> ] ;
[ [ BORDER ] <border: NONE, SINGLE> ] ;
[ OF <oParent> ] ;
[ <pixel: PIXEL> ] ;
=> ;
<oWnd> := TMdiFrame():New( <nTop>, <nLeft>, <nBottom>, <nRight>;,
    <cTitle>, <nStyle>, <oMenu>, <oBrush>, <oIcon>, <nClrFore>;,
    <nClrBack>, [<.vScroll.>], [<.hScroll.>], <nMenuInfo>;,
    [Upper(<(border)>)], <oParent>, [<.pixel.>] )

#xcommand DEFINE WINDOW <oWnd> ;
[ FROM <nTop>, <nLeft> TO <nBottom>, <nRight> [<pixel: PIXEL>] ] ;
[ TITLE <cTitle> ] ;
[ <color: COLOR, COLORS> <nClrFore> [,<nClrBack>] ] ;
[ OF <oParent> ] ;
[ BRUSH <oBrush> ] ; // Contained Objects
[ CURSOR <oCursor> ] ;
[ ICON <oIcon> ] ;
[ MENU <oMenu> ] ;
[ STYLE <nStyle> ] ; // Styles
[ BORDER <border: NONE, SINGLE> ] ;
[ <NoSysMenu: NOSYSMENU, NO SYSMENU> ] ;
[ <NoCaption: NOCAPTION, NO CAPTION, NO TITLE> ] ;
[ <NoIconize: NOICONIZE, NOMINIMIZE> ] ;
[ <NoMaximize: NOZOOM, NO ZOOM, NOMAXIMIZE, NO MAXIMIZE> ] ;
[ <vScroll: VSCROLL, VERTICAL SCROLL> ] ;
[ <hScroll: HSCROLL, HORIZONTAL SCROLL> ] ;
=> ;
<oWnd> := TWindow():New( <nTop>, <nLeft>, <nBottom>, <nRight>;,
    <cTitle>, <nStyle>, <oMenu>, <oBrush>, <oIcon>, <oParent>;,
    [<.vScroll.>], [<.hScroll.>], <nClrFore>, <nClrBack>, <oCursor>;,
    [Upper(<(border)>)], !<.NoSysMenu.>, !<.NoCaption.>;,
    !<.NoIconize.>, !<.NoMaximize.>, <.pixel.> )

#xcommand ACTIVATE WINDOW <oWnd> ;
[ <show: ICONIZED, NORMAL, MAXIMIZED> ] ;
[ ON [ LEFT ] CLICK <uLClick> ] ;
[ ON LBUTTONDOWN <uLButtonUp> ] ;
[ ON RIGHT CLICK <uRClick> ] ;
[ ON MOVE <uMove> ] ;
[ ON RESIZE <uResize> ] ;

```

```

[ ON PAINT <uPaint> ] ;
[ ON KEYDOWN <uKeyDown> ] ;
[ ON INIT <uInit> ] ;
[ ON UP <uUp> ] ;
[ ON DOWN <uDown> ] ;
[ ON PAGEUP <uPgUp> ] ;
[ ON PAGEDOWN <uPgDn> ] ;
[ ON LEFT <uLeft> ] ;
[ ON RIGHT <uRight> ] ;
[ ON PAGELEFT <uPgLeft> ] ;
[ ON PAGERIGHT <uPgRight> ] ;
[ ON DROPFILES <uDropFiles> ] ;
[ VALID <uValid> ] ;
=> ;
    <oWnd>:Activate( [ Upper(<(show)>) ],;
    <oWnd>:bLClicked [ := \{ |nRow,nCol,nKeyFlags|
<uLClick> \} ], ;
    <oWnd>:bRClicked [ := \{ |nRow,nCol,nKeyFlags|
<uRClick> \} ], ;
    <oWnd>:bMoved [ := <{uMove}> ], ;
    <oWnd>:bResized [ := <{uResize}> ], ;
    <oWnd>:bPainted [ := \{ | hDC, cPS | <uPaint> \} ],
;
    <oWnd>:bKeyDown [ := \{ | nKey | <uKeyDown> \} ],;
    <oWnd>:bInit [ := \{ | Self | <uInit> \} ],;
    [<{uUp}>], [<{uDown}>], [<{uPgUp}>], [<{uPgDn}>],;
    [<{uLeft}>], [ <{uRight}>], [ <{uPgLeft}>],
[<{uPgRight}>],;
    [<{uValid}>], [\{ |nRow,nCol,aFiles|<uDropFiles>\}],;
    <oWnd>:bLButtonUp [ := <{uLButtonUp}> ] )

```

Comando

```

DEFINE WINDOW [<oWnd>] ;
    MDICHILD ] ;
    FROM <nTop>, <nLeft> TO <nBottom>, <nRight> ] ;
    TITLE <cTitle> ] ;
    BRUSH <oBrush> ] ;
    CURSOR <oCursor> ] ;
    MENU <oMenu> ] ;
    MENUINFO <nMenuInfo> ] ;
    ICON <oIco> ] ;
    OF <oParent> ] ;
    VSCROLL, VERTICAL SCROLL ] ;
    HSCROLL, HORIZONTAL SCROLL ] ;
    COLOR, COLORS <nClrFore> [, <nClrBack> ] ] ;
    PIXEL ] ;
    STYLE <nStyle> ] ;
    HELPID, HELP ID <nHelpId> ] ;
    BORDER <border: NONE, SINGLE> ] ;
    NOSYSMENU, NO SYSMENU ] ;
    NOCAPTION, NO CAPTION, NO TITLE ] ;
    NOICONIZE, NOMINIMIZE ] ;
    NOZOOM, NO ZOOM, NOMAXIMIZE, NO MAXIMIZE ] ;

DEFINE WINDOW <oWnd> ;
    FROM <nTop>, <nLeft> TO <nBottom>, <nRight> ] ;
    TITLE <cTitle> ] ;
    STYLE <nStyle> ] ;
    MENU <oMenu> ] ;
    BRUSH <oBrush> ] ;
    ICON <oIcon> ] ;

```

```

MDI ] ;
COLOR, COLORS> <nClrFore> [, <nClrBack>] ] ;
VSCROLL, VERTICAL SCROLL> ] ;
HSCROLL, HORIZONTAL SCROLL> ] ;
MENUINFO <nMenuInfo> ] ;
BORDER ] <border: NONE, SINGLE> ] ;
OF <oParent> ] ;
PIXEL> ] ;

DEFINE WINDOW <oWnd> ;
FROM <nTop>, <nLeft> TO <nBottom>, <nRight> [<pixel: PIXEL>] ] ;
TITLE <cTitle> ] ;
COLOR, COLORS> <nClrFore> [, <nClrBack>] ];
OF <oParent> ] ;
BRUSH <oBrush> ] ; // Contained Objects
CURSOR <oCursor> ] ;
ICON <oIcon> ] ;
MENU <oMenu> ] ;
STYLE <nStyle> ] ; // Styles
BORDER <border: NONE, SINGLE> ] ;
NOSYSMENU, NO SYSMENU> ] ;
NOCAPTION, NO CAPTION, NO TITLE> ] ;
NOICONIZE, NOMINIMIZE> ] ;
NOZOOM, NO ZOOM, NOMAXIMIZE, NO MAXIMIZE> ] ;
VSCROLL, VERTICAL SCROLL> ] ;
HSCROLL, HORIZONTAL SCROLL> ] ;

ACTIVATE WINDOW <oWnd> ;
ICONIZED, NORMAL, MAXIMIZED> ] ;
ON [ LEFT ] CLICK <uLClick> ] ;
ON LBUTTONDOWN <uLButtonUp> ] ;
ON RIGHT CLICK <uRClick> ] ;
ON MOVE <uMove> ] ;
ON RESIZE <uResize> ] ;
ON PAINT <uPaint> ] ;
ON KEYDOWN <uKeyDown> ] ;
ON INIT <uInit> ] ;
ON UP <uUp> ] ;
ON DOWN <uDown> ] ;
ON PAGEUP <uPgUp> ] ;
ON PAGEDOWN <uPgDn> ] ;
ON LEFT <uLeft> ] ;
ON RIGHT <uRight> ] ;
ON PAGELEFT <uPgLeft> ] ;
ON PAGERIGHT <uPgRight> ] ;
ON DROPFILES <uDropFiles> ] ;
VALID <uValid> ] ;

```

Cláusulas

- **@** irá seguido de las coordenadas de posición. Si no se especifica la cláusula PIXEL, los números serán línea y columna. Indicando PÍXEL, los valores serán en pixels..
- **3D** sirve para dar aspecto de tres dimensiones.
- **ACTION** será la función que se ejecuta al hacer clic sobre el control.
- **ACJUST** permite al operador modificar el tamaño y posición del control.

- **ALIAS** indica el alias del fichero relacionado con el control. Si no se pone, la base de datos será la activa en ese momento.
- **BORDER** indica que saque un rectángulo marcando el borde del control en función de SIZE.
- **BRUSH** indica el objeto brocha que se utilizará como fondo del objeto.
- **BOX**
- **BUTTONSIZE** es el tamaño del botón en pixels.
- **CANCEL** permite al usuario cancelar la operación del control.
- **CENTER, CENTERED** indica que centre el texto en el espacio marcado por SIZE. Esta cláusula es excluyente con RIGHT.
- **COLOR, COLORS** indica el color del texto. Si se pone un segundo color, será el del fondo. El color se define mediante la función nRGB() o similar.
- **CURSOR** indica que cambie el cursor cuando este pase por encima.
- **DEFINE**
- **DESIGN** añade un borde al control, redimensionable en tiempo de ejecución. También permite que el control se pueda mover con el ratón.
- **DLL** indica el fichero DLL relacionado con el control.
- **FIELDS** indica los campos que se utilizarán en el control.
- **FILE** indica el fichero relacionado con el control.
- **FONT** indica el objeto fuente, en caso de ser distinto al indicado en momento del diseño con el workshop.
- **FROM...TO...** indica las coordenadas del rectángulo.
- **GROUP** indica si pertenece a un grupo.
- **HEAD** indica el texto o textos que irán en la cabecera.
- **HORIZONTAL** indica si el control será en su versión horizontal.
- **HELP**
- **ID** indica el indicador en workshop. Puede ser una variable definida.
- **MEMO** se utiliza en el control para indicar que ha de editarse un campo memo o una variable de texto largo
- **MENU**
- **MESSAGE** es el mensaje que aparecerá en la barra de mensajes cuando el cursor pase por encima del control.
- **NAME**
- **OF, WINDOWS, DIALOG** es el marco que lo contine. Normalmente será una caja de diálogo.
- **ON CHANGE** indica la función que se ejecutará cuando cambie el contenido o aspecto del control.
- **ON DRAWITEN**
- **ON DROP**

- **ON EDIT**
- **OPTION**
- **PAGE**
- **PICTURE** es la máscara igual que en DOS.
- **PIXEL, PIXELS**. Normalmente la posición se indica en líneas y columnas, y el tamaño en píxels. Con esta cláusula, todo será en píxels
- **PROMPT, VAR** seguidas del texto literal o el nombre de la variable que lo contiene. El font y el color será el indicado en las cláusulas **FONT** y **COLOR**. En caso de no tenerlas, podrá los valores por defecto del marco o ventana que lo contiene.
- **RAISED** ¿?
- **REDEFINE**
- **RIGHT** indica que visualice el texto justificado por la derecha, en el espacio marcado por SIZE. Esta cláusula es excluyente con CENTERED o CENTER
- **SELECT**
- **SHADOW, SHADES** indica que el texto tenga una pequeña sombra para simular 3D.
- **SIZE** indica el tamaño en píxels del campo que ocupará el texto. No indica el campo del texto, que eso está marcado por el propio texto y el font utilizado. Este tamaño se utiliza, por ejemplo para el color de fondo, el borde o la superposición sobre otros controles.
- **TITLE**
- **TOP/LEFT/RIGHT/BOTTOM/FLOAD**
- **TOOTIP** indica el pequeño texto de ayuda que sale junto al control al dener unos segundos el cursor sobre él.
- **UPDATE** se utiliza para actualizar el texto cuando se refresca la caja o ventana de diálogo por medio de la instrucción <objeto diálogo>:refresh().
- **VALID** es la función que debe devolver .T. para que el control pueda dejar el foco y continuar con el programa. Es similar a la versión en DOS
- **VAR** es la variable asociada al control.
- **VERTICAL** indica que el control se presentará en su versión vertical
- **WHEN** es la función que debe devolver .T. para que el control pueda tomar el foco. Es similar a la versión en DOS

Ejemplos

Trucos

Si se quiere hacer que una ventana no se vea, pero si sus controles, ponerla fuera de la pantalla mediante las cláusulas **FOR y TO** (ejemplo, FROM -2,-2 TO -1,-1) y al activarla no poner **MAXIMIZED**

Conclusión

PAGINA PATRON**Concepto**

Por programa...

fivewin.ch

Comando

Cláusulas

Ejemplos

Por recursos...

fivewin.ch

Worksho

Comando

Cláusulas

Ejemplos

Conclusión

IMPRESION

Es la intención de esta capítulo aglutinar en él todos los temas sobre la impresión de documentos, tanto en impresoras controladas por Windows, como las controladas directamente por el programa en formato DOS.

CONTROL DE IMPRESORA

Windows no solo nos ha obligado a cambiar la forma de pensar y trabajar con respecto a las salidas y entradas por pantalla, sino también a la forma de pensar y trabajar con respecto a las salidas por impresora. Con Windows, nuestro programa no escribe en la hoja de papel de la impresora, escribe en una página virtual que no existe físicamente. Una vez terminado escribir en esta página, mandamos la orden de impresión real y será Windows quien se encargue de actuar físicamente sobre la impresora.

Si has trabajado con impresoras láser habrás comprobado que mientras no mandes un avance de página (**chr(12)**) puedes escribir lo que quieras. Puedes escribir en posiciones aleatorias, de abajo a arriba, luego a la derecha, etc. Todo esto se graba en el buffer que tiene la impresora en su interior. Cuando mandamos FormFeed entonces la impresora, no ya nuestro programa ni nuestro ordenador, será el que de arriba abajo y de izquierda a derecha vaya escribiendo el mapa de bits sobre el papel.

Windows lo que ha hecho es crear ese buffer que esta en las impresoras láser en su propia memoria RAM. Forma con los datos que vamos suministrando un mapa de bit al estilo de un BMP del tamaño del papel. Windows sabe el tamaño del papel, la orientación, el numero de puntos por pulgada, etc. Cuando le decimos que hemos acabado de mandar cosas a esta página, Windows y no nosotros, ni clipper, mandará el buffer creado, punto a punto a la impresora. Por este motivo las impresiones en Windows son mas lentas que en DOS. Todo se hace dibujando. En DOS para escribir una "A", simplemente mandábamos el carácter ASCII equivalente (chr(65)) que es un byte. Ahora, con windows hay que mandar tantos bytes como sean necesarios para dibujar la **A**. Esto en ordenadores rápidos casi no se nota, pero en equipos antiguos tipo Pentium 100, si se notaba.

Por el único inconveniente de la velocidad, todo lo demás son ventajas. Se pueden hacer todas las letras que tengan el fichero de windows, y no unicamente los font de la propia impresora, se pueden escalar, hacer impresión proporcional, etc. etc. No necesitamos preocuparnos de que impresora utilizará el cliente final, esa es una labor de Windows. Naturalmente no todo es tan bonito, pero se acerca a la perfección.

La impresión se puede hacer a tres niveles:

- Imprimir directamente, como se hacía en DOS. Casi no se usa pero ha veces es imprescindible, por ejemplo en una impresora de ticket de un comercio, donde no hay páginas de longitud constante.
- Imprimir con @ nl,nc SAY como en DOS, preocupándonos de donde poner los datos, en que forma, etc. De esta forma trabajaremos como con las impresoras láser en DOS.
- Para los datos que esten en ficheros DBF, Fivewin nos facilita un comando que permite su listado de forma preestablecida. A esta forma, nosotros podemos modificarla mediante cláusulas y metodos del objeto **TReport**.

A mi personalmente me gusta más el segundo método, aunque comprendo que da más trabajo y al cliente lo que le importa es el plazo de entrega.

GENERADOR DE INFORMES

Nota.- Este apartado esta copiado casi integramente de la documentación de dominio público aparecida en Internet, denominada **Introducción a la Programación Windows** de los autores Francisco García, Jesús Morán y Fernando Ballesteros. Me he limitado a formatear un poco el texto y modificar algunas frases. He preferido mantener las referencias a los párrafos, por ejemplo, **6.1 EL GENERADOR DE INFORMES....**

6.1 EL GENERADOR DE INFORMES DE Fivewin

Por muy bien diseñado que esté un programa, por sencillo que sea su manejo, por muchas formas de consultar la información que contenga, siempre aparecerán una serie de problemas que obligan a generar algún tipo de salida impresa:

- Se requerirá usar dicha información donde no hay ordenador.
- Se prefiere ver las cosas en papel mejor que en la pantalla de un ordenador.

La solución es la realización de una serie de programas que procesen la información contenida en las bases de datos y la vuelquen en papel.

El principal problema ha sido siempre que al diseñar los programas que generaban los informes había que prever el tipo de dispositivo (impresoras matriciales, impresoras láser, ...) en que se iba a realizar la impresión, eligiendo una de estas tres alternativas:

- El informe fuera lo más simple posible de forma que pudiera imprimirse en cualquier tipo de dispositivo. Esto implica el no poder utilizar ningún tipo de efecto de impresión o la utilización de distintos tipos de letras.
- Escribir el programa de modo que soporte un pequeño grupo de dispositivos de amplia difusión (generalmente IBM, Epson y HP) y esperar que si hay que imprimir en otra máquina, esta sea capaz de emular las soportadas por el programa (lo que no siempre ocurre).

- Dar soporte a un gran número de dispositivos existentes en el mercado, con el problema de conseguir información sobre todos esos dispositivos, y preparar el programa para soportarlos.

El sistema de generación de informes de FiveWin permite solucionar este problema al utilizar el sistema de impresión de Windows. De esta manera, sólo debemos indicar lo que queremos imprimir y Windows se encargará de todo el control de cambio de tipos de letras, impresión gráfica, etc.

Entre las principales características del generador de informes de FiveWin se encuentran:

- Es un sistema orientado a objetos, completamente personalizable, derivable y de fácil utilización.
- Independencia del dispositivo de impresión. Lo mismo puede trabajar con una impresora, un plotter o la pantalla.
- Completa gestión sobre columnas, cabeceras, totales, pies de página, ...
- Gestión automática de Grupos.
- Utilización de distintas fuentes de letra.
- Completo control sobre el proceso de impresión.
- Capacidad de impresión de gráficos.

6.2 Creación de un informe

La creación de un informe es sumamente sencilla:

- Se define un objeto informe.
- Se indican las columnas que componen el informe.
- Se imprime el informe.

Dicho así parece muy sencillo. Vamos a comprobar con un ejemplo que realmente es tan fácil como parece.

Supongamos que tenemos una base de datos GENTE.DBF con los campos NOMBRE, APELLIDO1, APELLIDO2, DIRECCION, y que deseamos realizar un listado de esta base de datos. El programa que realiza esto sería:

```
#include "Fivewin.ch"
#include "Report.ch"

// Creamos una función que imprima el informe
FUNCTION Informe()
LOCAL oInforme // El objeto informe

    // Abrimos una base de datos
    USE GENTE

    // Creamos el informe
```

```
REPORT oInforme

// Añadimos las columnas indicando que
// datos contienen y su cabecera
    COLUMN TITLE "Nombre"          DATA GENTE->NOMBRE
    COLUMN TITLE "Apell. 1"        DATA GENTE->APELLIDO1
    COLUMN TITLE "Apell. 2"        DATA GENTE->APELLIDO2
    COLUMN TITLE "Dirección"      DATA GENTE->DIRECCION

END REPORT // Fin de la definición del informe

ACTIVATE REPORT oInforme // Imprimimos el informe
CLOSE GENTE
RETURN NIL
```

Con esto obtendríamos un listado.

6.3 Informes a pantalla, disco o impresora

El generador de informes de FiveWin permite la realización de informes cuyo destino final no ha de ser la impresora. Para ello, a la hora de definir el informe, se ha de indicar cual va a ser su destino (si no se indica nada el destino es la impresora).

Para obtener un informe por pantalla, es necesario especificar la cláusula PREVIEW (vista previa) en la creación del informe:

```
// Creamos un informe por pantalla
REPORT oInforme PREVIEW
.....
.....
END REPORT // Fin de la definición del informe
ACTIVATE REPORT oInforme // mostramos el informe
```

Cuando se genera un informe **por pantalla**, el generador crea todas las páginas del informe antes de presentarlas, y muestra un cuadro de dialogo en el que informa del proceso del trabajo. Si se trata de un informe muy largo puede ser interrumpido pulsando el botón de Preview, con lo que veremos las páginas que se han generado hasta ese momento.

Para obtener un informe **en un fichero**, se debe indicar en la creación del informe la cláusula **TO FILE** y el nombre del fichero de destino, en el que se va a grabar el informe:

```
// Creamos un informe a un fichero

REPORT oInforme TO FILE "c:\informe.txt"
.....
.....
END REPORT // Fin de la definición del informe
ACTIVATE REPORT oInforme // Imprimimos el informe
```

Para obtener un informe por la impresora se puede incluir la cláusula **TO PRINTER**, aunque si no se incluye también funcionará al ser el dispositivo de salida por defecto. Así, cualquiera de las dos definiciones siguientes mandaría la información a la impresora.

```
// Por defecto a la impresora
REPORT oInforme
```

```
// Indicamos salida impresa
REPORT oInforme TO PRINTER
```

Cuando mandemos un informe a pantalla o impresora, es posible definir un nombre para el informe. Esto se hace a través de la cláusula **CAPTION**. Este nombre será el título de la ventana de visualización si se manda a pantalla, o la descripción del trabajo dentro del administrador de impresión

```
// Informe con título
REPORT oInforme CAPTION "Informe de gastos mensuales
....
```

6.4 Cabeceras, pies de página y títulos

A la hora de realizar un informe, suele haber un grupo de información que se ha de repetir en todas las páginas, bien al principio o bien al final de la página, como por ejemplo el título del informe, el número de página, la fecha, etc.

El texto que aparece repetido al principio de cada página es una cabecera, y el texto que aparece al final de cada página es un pie.

El título del informe se va a repetir en todas las páginas situado entre la cabecera del informe y el comienzo de las columnas.

La forma de colocar una cabecera, un pie o un título en un informe es indicándolo en la línea de creación del informe:

```
REPORT oReport ;
  TITLE "Este es el título" ;           // Título del informe
  HEADER "Esta es la cabecera"; // Cabecera del informe
  FOOTER "Este es el pie"              // El pie de página
```

Cabeceras, pies y título no están limitados a una única línea de texto. Si deseamos incluir varias líneas, se ha de hacer separando las cadenas de caracteres que componen cada una de las líneas con comas:

```
REPORT oReport ;
  TITLE "Primera línea del titulo" , "Segunda línea del título" ;
  HEADER "Línea 1 de la cabecera" , "Línea 2 de la cabecera" ;
  .....
```

Asimismo, cabeceras, pies y títulos pueden ser colocados centrados en la página, a la izquierda o a la derecha. Para ello deberemos indicar la posición en la que queremos que aparezcan cada uno de ellos dentro de la página mediante las cláusulas CENTER, RIGHT o LEFT :

```
REPORT oReport ;  
  TITLE "El titulo" RIGHT ;      // Título a la derecha  
  HEADER "Esta es" , "la cabecera" CENTER // Cabecera centrada
```

En el caso de que la cabecera, el título o el pie tengan varias líneas, lo que no es posible con éste método situarlos en lugar diferente de la página. Es decir, todas las líneas tendrán que estar a la derecha, a la izquierda o centradas.

Además, y esto es valido PARA CUALQUIER PARTE DE UN INFORME, cabeceras, pies y titulo no han de ser necesariamente un literal en el programa.

Cualquier expresión valida en Clipper que se evalúe a una cadena de caracteres puede ser utilizada. Por ejemplo, si deseáramos colocar la fecha en la cabecera de la página, lo podríamos hacer de la siguiente manera:

```
REPORT oReport ;  
  HEADER "Fecha: "+DTOS( DATE())
```

6.5 Las columnas del informe

El principal elemento de la mayoría de los informes realizados a partir de bases de datos son las columnas, y el aspecto final del informe va a depender en gran medida de la correcta disposición y aspecto de cada una de ellas.

Ya hemos visto que el generador de informes es capaz de manejar automáticamente los datos de las columnas que definimos, pero también nos deja que modifiquemos cualquier aspecto de la impresión de estas, pudiendo cambiar tipos de letra, máscaras de los campos (PICTURE), tamaños, posición, sombreados, etc.

6.5.1 Contenido de las columnas

Cuando se crea una columna, es necesario indicar cual va a ser su contenido, los datos que se van a imprimir. Ya hemos visto algún ejemplo de como hacer esto y ahora vamos a comprobar todas las posibilidades.

Cuando definimos el contenido de una columna utilizamos la cláusula **DATA** y a continuación la expresión que proporcionará el valor que se va a imprimir:

```
COLUMN DATA GENTE->NOMBRE
```

Esto crearía una columna cuyo contenido sería el campo NOMBRE de la base GENTE, imprimiendo una línea para cada registro de la base de datos. Si necesitásemos que la información apareciese dividida en más de un línea (toda no cabe en una sola línea o cualquier otra razón), simplemente se indicará el contenido de cada una de las líneas separados por comas. Así el siguiente código:

```
COLUMN DATA GENTE->APELLIDO1, GENTE->APELLIDO2
```

define una columna en la que los campos APELLIDO1 y APELLIDO2 de la base GENTE se imprimirán en líneas separadas dentro del informe.

6.5.2 El título de las columnas

Toda columna puede tener un título, una cabecera que identifique cual es su contenido. Esto se realiza mediante la cláusula **TITLE** en la definición de la columna.

```
COLUMN DATA GENTE->NOMBRE TITLE "Nombre"
```

El código anterior definiría una columna con una cabecera con el texto "Nombre" que se repetirá en todas las páginas del informe al principio de la misma.

Si el título de la columna ha de contener más de una línea de texto, se indica el texto de cada línea separado por comas:

```
COLUMN DATA GENTE->RENTA TITLE "Renta","Per Capita"
```

Esto define una columna con una línea de datos pero con un título de dos líneas.

6.5.3 Tamaño, aspecto, alineación y posición de las columnas

A la hora de definir una columna se puede indicar el formato que van a presentar los datos, el ancho de la columna, la alineación de los datos y la posición en la página de la columna. Esto se hace a través de una serie de cláusulas en la declaración de la columna.

La cláusula **PICTURE** permite modificar el aspecto que presentan los datos. Su utilización es igual que en los GETS de Clipper.

```
COLUMN TITLE "Sueldo" DATA GENTE->SUELDO PICTURE "99,999,999.99"
COLUMN TITLE "Nombre" DATA GENTE->NOMBRE PICTURE "@!"
```

Si la columna va a ser multilínea, se puede especificar una cláusula picture para cada una de las líneas de la columna

```
COLUMN TITLE "Sueldo" , "Renta Per Capita" ;
DATA GENTE->SUELDO, GENTE->RENTA ;
PICTURE "9.999.999" , "999.999"
```

La cláusula **SIZE** permite indicar el ancho de la columna en caracteres.

```
COLUMN TITLE "Apellidos" ;
DATA GENTE->APELLIDO1 + GENTE->APELLIDO2 ;
SIZE 50 // 50 caracteres de ancho para la columna
```

La posición de la columna dentro de la página se indica mediante la cláusula **AT**. Esto nos permite especificar el carácter a partir del cual comienza la columna (no obstante, los creadores del generador de informes recomiendan no utilizarlo. Sus razones tendrán)

```
COLUMN TITLE "Dirección" DATA GENTE->DIRECCION AT 65
```

La alineación de los datos dentro de la columna se indica mediante las cláusulas **LEFT**, **CENTER** o **CENTERED** y **RIGHT**, que permiten indicar para cada columna si está va a estar alineada a la izquierda, centro o derecha respectivamente. Por defecto, el generador de informes alinea todos los datos a la izquierda, menos los números que van alineados a la derecha.

```
COLUMN TITLE "Sueldo" ;  
DATA GENTE->SUELDO ;  
PICTURE "99,999,999" ;  
SIZE 10 ;  
RIGHT  
COLUMN TITLE "Nombre" ;  
DATA GENTE->NOMBRE ;  
CENTER
```

6.5.4 Tipos de letras en las columnas

Es posible utilizar distintos tipos de letra en las columnas de un informe. Para ello, se deben seguir una serie de pasos:

- Definir un objeto FONT para cada tipo de letra que se vaya a usar
- Definir el informe indicando que tipos de letra va a utilizar
- Indicar para cada columna el tipo que se desea para esa columna

Veamos un ejemplo:

```
#include "Fivewin.ch"  
#include "Report.ch"  
  
// Creamos una función que imprima el informe  
FUNCTION Informe()  
LOCAL oInforme , ; // El objeto informe  
oFont1, ; // Un tipo de letra  
oFont2 // Otro tipo de letra  
  
// Definimos los tipos de letra  
DEFINE FONT oFont1 NAME "Arial" SIZE 0,-10  
DEFINE FONT oFont2 NAME "Arial" SIZE 0,-10 BOLD  
  
// Abrimos una base de datos  
USE GENTE  
  
// Creamos el informe e indicamos las fuentes  
REPORT oInforme ;  
FONT oFont1, oFont2  
  
// Añadimos las columnas indicando que  
// datos contienen, su cabecera y su font
```

```

        COLUMN      TITLE "Nombre";
                      DATA GENTE->NOMBRE ;
                      FONT 1
        COLUMN      TITLE "Apell. 1"          ;
                      DATA GENTE->APELLID01 ;
                      FONT 2
        COLUMN      TITLE "Apell. 2"          ;
                      DATA GENTE->APELLID02 ;
                      FONT 2

END REPORT // Fin de la definición del informe

ACTIVATE REPORT oInforme // Imprimimos el informe

// Cerramos la base y liberamos las fuentes
CLOSE GENTE
oFont1:Release()
oFont2:Release()

RETURN NIL

```

ATENCIÓN: Cuando nosotros le indicamos a una columna que use un tipo de letra, **no especificamos el objeto FONT** que hemos creado, **sino su orden** en la lista de las fuentes que se han indicado en la creación del informe.

No es necesario especificar las fuentes de todas las columnas. Sólo de aquellas que vayan a usar una distinta. Para las columnas en las que no se especifique, se utilizará la fuente por defecto del dispositivo de salida.

6.5.5 Sombras y líneas en las columnas

Para conseguir que una columna aparezca con un fondo gris, en la definición de la columna se ha de especificar la cláusula **SHADOW**.

```
COLUMN TITLE "Sueldo" DATA GENTE->SUELDO SHADOW
```

Para separar las columnas del informe con líneas se emplea la cláusula **GRID**. El empleo de esta cláusula provoca que la columna en la que se utiliza aparezca separada de las otras por dos líneas verticales (una a la derecha y otra a la izquierda).

```
COLUMN TITLE "Sueldo" DATA GENTE->SUELDO GRID
```

Por defecto, las líneas que se dibujan son de color negro, sin trama y con un ancho de 1 punto. Es posible modificar esto mediante la utilización de los objetos **PEN** de FiveWin. Para ello procederemos de la misma manera que con las fuentes de letra:

```

DEFINE PEN oLapiz1 WIDTH 4
DEFINE PEN oLapiz2 WIDTH 10

REPORT oInforme PEN oLapiz1, oLapiz2

// Columna que utiliza el lápiz de 10 puntos
COLUMN TITLE "Nombre" DATA GENTE->NOMBRE GRID 2

// Columna que utiliza el lápiz de 4 puntos

```

```
COLUMN TITLE "Sueldo" DATA GENTE->SUELDO GRID 1
```

6.5.6 Totales en las columnas

En muchas ocasiones nos encontramos con columnas numéricas en un informe que es necesario totalizar. El generador de informes de FiveWin nos permite realizar esto de modo automático, simplemente indicando la cláusula **TOTAL** en la definición de la columna. Por ejemplo:

```
#include "Fivewin.ch"
#include "Report.ch"

// Creamos una función que imprima el informe
FUNCTION Informe()
LOCAL oInforme // El objeto informe

    // Abrimos una base de datos
    USE GENTE

    // Creamos el informe
    REPORT oInforme

    // Añadimos las columnas indicando que
    // datos contienen y su cabecera
    COLUMN TITLE "Nombre"          DATA GENTE->NOMBRE
    COLUMN TITLE "Apell. 1"        DATA GENTE->APELLIDO1
    COLUMN TITLE "Apell. 2"        DATA GENTE->APELLIDO2

    // Columna totalizada
    COLUMN TITLE "Sueldo"          DATA GENTE->SUELDO TOTAL

    END REPORT // Fin de la definición del informe

    ACTIVATE REPORT oInforme // Imprimimos el informe
    CLOSE GENTE
    RETURN NIL
```

También podemos especificar una cláusula **FOR** en el cálculo del total, para indicar que registros han de totalizarse. Por ejemplo, para totalizar solamente los que viven en Valencia hacemos:

```
COLUMN TITLE "Sueldo" ;
DATA GENTE->SUELDO ;
TOTAL FOR GENTE->PROVINCIA == "Valencia"
```

Naturalmente, solo es posible totalizar columnas numéricas.

6.6 Informes agrupados y sumarios

El generador de informes de FiveWin realiza automáticamente informes agrupados con cabeceras, pies y totales (siempre que en alguna de las columnas se esté totalizando) para cada uno de los grupos.

Para realizar un informe agrupado se deben realizar los siguientes pasos:

- Definir el informe
- Definir las columnas
- Definir los grupos
- Imprimir el informe.

La definición del grupo se realiza mediante el comando **GROUP ON** y una expresión que indique que valor es el que determina el cambio de grupo.

Naturalmente, la base de datos ha de estar ordenada por la misma expresión del grupo.

Por ejemplo, para realizar un listado de la base GENTE agrupado por provincias, hacemos lo siguiente:

```
FUNCTION Informe()  
LOCAL oInforme  
  
    // Abrimos la base de datos con un índice  
    // que para nuestro ejemplo indexa la base por la provincia  
  
    USE GENTE INDEX PROVINCIA  
  
    // Creamos el informe  
    REPORT oInforme  
  
        // Creamos las columnas  
  
        COLUMN TITLE "Nombre" ;  
            DATA GENTE->NOMBRE  
  
        COLUMN TITLE "Apellidos" ;  
            DATA GENTE-> APELLIDO1 + GENTE->APELLIDO2  
  
        COLUMN TITLE "Provincia"  
            DATA GENTE->PROVINCIA  
  
        // Definimos el grupo como PROVINCIA  
        GROUP ON GENTE->PROVINCIA  
  
    END REPORT  
  
    // Imprimimos el informe  
    ACTIVATE REPORT oInforme  
  
    CLOSE ALL  
  
    RETURN NIL
```

Para definir las cabeceras y pies para el grupo utilizaremos las cláusulas **HEADER** y **FOOTER**, aunque en el caso de los grupos, éstas solo podrán tener una línea.

Es posible utilizar tipos de letra distintos para la cabecera y el pie de un grupo. Esto se hace con la cláusula **FONT** en la definición del grupo.

```
GROUP ON GENTE->PROVINCIA FONT 2
```

El tipo de fuente se define y utiliza de la misma manera que en las columnas.

También le podemos indicar al generador de informes que cada vez que acabe un grupo y comience otro se realice un salto de página. Esto se hace mediante la cláusula **EJECT** en la definición del grupo.

```
GROUP ON GENTE->PROVINCIA EJECT
```

Cuando en la creación del informe se especifica la cláusula **SUMMARY**, solamente se imprimirá la información relativa a los grupos (sin el contenido de los mismos) y sin separadores entre grupos.

```
REPORT oInforme TITLE "....." SUMMARY
```

Lógicamente, no se puede imprimir un informe de tipo sumario si no existe al menos un grupo.

6.7 Selección de los registros a imprimir.

A la hora de imprimir un informe, es posible indicar que registros van a ser impresos mediante la utilización de las cláusulas **FOR** y **WHILE**. Su utilización es idéntica al resto de los mandatos de Clipper que usan estas cláusulas.

```
ACTIVATE REPORT oInforme ;  
    FOR GENTE->PROVINCIA == "Madrid" ; // Solo los de Madrid  
    WHILE !EOF()
```

El generador de informes realiza internamente un bucle del tipo DO WHILE !EOF() cuando imprime un informe, pero esto es modificado por la inclusión de la cláusula WHILE en la activación del informe, con lo cual el generador no hace ningún tipo de comprobación de fin de base de datos. Por ello es recomendable añadir esta condición dentro de nuestra cláusula WHILE, a menos que por alguna circunstancia no sea realmente necesaria.

6.8 Gestión de sucesos en un Informe

Cuando activamos un informe, al igual que cuando activábamos una ventana, podemos definir una serie de funciones que serán llamadas cada vez que se produzca un determinado evento. Esto se realiza mediante una serie de cláusulas en la orden de activación del informe. Estas cláusulas y su acción son:

ON INIT

Llama a la función asociada una sola vez, al principio del informe, cuando las cabeceras de las columnas ya están impresas y se va a comenzar a imprimir registros.

ON END

Se llama una vez que se ha acabado de realizar todo el trabajo y los totales han sido impresos.

ON STARTPAGE

Se llama al principio de cada página. Es una de las más útiles, utilizándose sobre todo para dibujar elementos de fondo de la página antes de la impresión de la misma (inclusión de bitmaps, líneas, etc.)

ON ENDPAGE

Es llamada cada vez que se ha acabado de imprimir una página.

ON STARTGROUP

Se llama cuando se va a comenzar un grupo, justo antes de imprimir la cabecera del grupo.

ON ENDGROUP

Es llamada cada vez que se finaliza un grupo.

ON STARTLINE

Es llamado cada vez que se comienza una de las líneas del cuerpo del informe.

CUIDADO: Se evalúa para cada línea que se imprime, no para cada registro que se procesa.
--

Es decir, si estamos trabajando con columnas multilínea, se llamará a la función especificada en STARTLINE por cada línea que se imprima. Si para cada registro se necesitan dos líneas de la página, se realizarán dos llamadas, si se requieren tres, tres llamadas etc.

ON ENDLINE

Se llama cada vez que se ha acabado de imprimir una línea y después de haber incrementado el contador de líneas (al que ya veremos como acceder)

ON CHANGE

Es llamada cada vez que se va a realizar un SKIP, pero ANTES de realizarlo.

```
ACTIVATE REPORT oInforme ;
ON STARTLINE      Funcion1() ;
ON ENDPAGE        Funcion2()
```

MUY IMPORTANTE: Cuando se utilizan estas cláusulas, se está trabajando dentro del núcleo del generador de informes. Es necesario tener mucho cuidado con las bases de datos que se utilizan, índices, etc.

6.9 Los objetos del Generador de Informes

Todas las opciones que hemos visto en el generador de informes son o están controladas por una serie de objetos, cada uno con sus propias características, datos y métodos. Así, un informe es un objeto del tipo **TReport**, que a su vez contendrá al menos un objeto del tipo **TRLine** (para el título, las líneas, etc.), las columnas son objetos **TRColumn**, usará tal vez un objeto TRGroup (uno para cada grupo), y un objeto **TPrinter** para imprimir.

6.9.1 El objeto TReport

Es el núcleo del sistema de informes.

6.9.1.1 Sus principales datos son:

oDevice	El dispositivo de salida. Del tipo TPrinter
oTitle	El objeto título del informe. Del tipo TRLine
oHeader	El objeto cabecera del informe. Del tipo TRLine.
oFooter	El objeto Pie de Página. Del tipo TRLine.
oRptWnd	En un objeto DIALOG que contiene la ventana en la que se muestra el proceso de impresión.
oBrush	Es un objeto de tipo BRUSH con el que se realiza la sombra de las columnas.

aGroups	Es un array de objetos TRGroup. Uno elemento por cada grupo definido.
aColumns	Es un array de objetos TRColumn. Un elemento por columna en el informe.
aFonts	Es un array de objetos FONT. Uno por cada tipo a utilizar.
aCols	Array que contiene la posición de comienzo de cada columna dentro de la página.
aText	Array que contiene un elemento por cada columna del informe. Cada uno de estos elementos contiene un array con tantos elementos como líneas de título tiene la columna con más líneas de título. Estos elementos contienen la línea de título correspondiente.
aData	Equivalente al anterior, pero para los datos de las columnas.
aPen	Array de objetos PEN. Un elemento por cada lápiz definido.
bFor	Codeblock del filtro. Se evalúa para cada registro. Solo se imprime si devuelve verdadero.
bWhile	Codeblock de control del informe. Se seguirá imprimiendo mientras el codeblock devuelva verdadero.
blnit	
bEnd	
bStartLine	
bEndLine	
bStartPage	
bEndPage	
bStartGroup	
bEndGroup	
bChange	
bSkip	Codeblock que controla el proceso de paso de un registro al siguiente (SKIP).
bStdFont	Codeblock que devuelve el número de la fuente a utilizar por defecto. Definido inicialmente como { 1 } para utilizar la primera fuente.
bPreview	Es un objeto RPreview utilizado para realizar previsualizaciones del informe.
cTotalChr	Carácter que se utiliza como separador para los totales
cGroupChr	Carácter que se utiliza como separador para los grupos
cTitleUpChr	Carácter con el que se dibuja la línea superior de los títulos
cTitleDnChr	Carácter con el que se dibuja la línea inferior de los títulos
cPageTotal	Carácter separador de los totales de página
cGrandTotal	Carácter separador de los grandes totales (Totales finales)

nWidth	Resolución horizontal del dispositivo de salida
nHeight	Resolución vertical del dispositivo de salida
nMargin	Valor que hay que sumarle al margen izquierdo para que el informe quede horizontalmente centrado
nRow	Posición de la línea actual.
nPage	Contador de páginas
nMaxTitle	Maximo número de líneas de título en la página
nMaxData	Maximo número de líneas de datos en la página.
nSeparator	Separación entre columnas
nLeftMargin	Margen Izquierdo
nRightMargin	Margen derecho
nTopMargin	Margen Superior
nDnMargin	Margen Inferior
nBottomRow	Ultima línea de la página
nStdLineHeight	Altura de una línea con fuente por defecto
nRptWidth	Anchura del informe
nCounter	Contador de líneas
ISpanish	Indica si los mensajes del generador aparecerán en español. Por defecto aparecen en inglés.

Hay otros datos en el objeto, pero dejamos al lector la tarea de ir investigando cuales son.

6.9.1.2 Los principales METODOS del objeto TReport (los que más utilizaremos sin ayuda del preprocesador)

AddColumn(oColumn)

Añade un objeto TRColumn al informe

DelColumn(nColumn)

Elimina una columna del informe

InsColumn(oColumn , nColumn)

Inserta una columna en el informe

AddGroup(oGroup)

Añade un objeto TRGroup en el informe

DelGroup(nGroup)

Elimina un grupo del informe

Skip(n)

Mueve el puntero de la base de datos si no se ha definido bSkip

NeedNewPage()

Indica si es necesario pasar a una nueva página

NewLine()

Avanza una línea

BackLine(nLine)

Retrocede a la línea nLine

Say(nCol, xText, nFont, nPad, nRow)

Escribe un dato (xText) dentro de una determinada columna del informe (nCol) con una fuente del informe (nFont) y un tipo de alineación (nPad, ver report.prg) en la fila indicada (nRow)

SayBitmap(nRow, nCol, cBitmap, nWidth, nHeight, nScale)

Pinta en la página actual un bitmap de un fichero (cBitmap) en una fila (nRow) y columna (nRow) determinada, con un ancho (nWidth) y alto (nHeight) determinado por la escala (nScale) según sea esta de centímetros (nScale==2) o pulgadas (nScale==1)

Box(nRow, nCol, nBottom, nRight, nPen, nScale)

Dibuja una caja en la página actual, en las coordenadas dadas, con un tipo de lápiz (nPen). Las coordenadas dependen de la escala (ver SayBitmap)

Line(nTop, nLeft, nBottom, nRight, nPen, nScale)

Similar al método anterior, pero dibuja una línea.

6.9.2 El objeto TRLine

Los objetos TRLine son los encargados de procesar todos los datos que aparecen en horizontal dentro del informe, cruzando la página, como por ejemplo cabeceras, títulos, pies, etc.

Los objetos TRLine son creados de modo automático por los objetos TReport, y la única manera de acceder a ellos es a través de estos, modificando las variables de instancia que contienen los objetos TRLine (ver los datos de los objetos TReport).

6.9.2.1 Los principales datos de los objetos TRLine son:

oReport	Objeto TReport que contiene la línea
aLine	Array que contiene un elemento por cada línea impresa que va a ocupar el objeto TRLine. Cada uno de estos elementos es un codeblock que devuelve el valor a imprimir al evaluarse.
aFont	Array que contiene las fuentes de letra para cada una de las líneas impresas del objeto TRLine. Cada elemento es un codeblock que se evalúa a un valor numérico que representa un índice dentro de la lista de fuentes del informe.
aRow	Array que contiene las posiciones de cada una de las líneas impresas del objeto TRLine. Cada elemento contiene un número que indica la posición de cada línea correspondiente.
aWidth	Array que contiene el ancho de cada una de las líneas impresas del objeto TRLine.
aPad	Array que contiene la alineación para cada una de las líneas impresas del objeto TRLine.
nHeight	Altura ocupada por las líneas impresas del objeto TRLine
nWidth	Ancho ocupado por el objeto TRLine. Es el valor máximo de los contenidos en aWidth.

Los objetos TRLine tienen algunos métodos, pero son de uso propio del objeto y son de poca utilidad si los manejamos nosotros manualmente, por lo que no los reflejaremos aquí.

6.9.3 El objeto TRColumn

Es el objeto encargado de manejar toda la información que se va a desplegar sobre el informe en modo vertical (a lo largo del informe). Las columnas creadas en los informes pertenecen a este tipo de objeto.

Cuando se crea un informe, los objetos TRColumn son creados al utilizar el comando COLUMN e integrados dentro del objeto TReport, de modo que son modificables mediante la alteración de las variables de instancia del objeto TReport que contienen objetos TRColumn. No obstante, al definir una columna se puede indicar una variable que contendrá una referencia al objeto creado. Para ello utilizaremos la siguiente sintaxis:

```
LOCAL oColumna1, oInforme
.....
REPORT oInforme
    COLUMN oColumna1 DATA ..... TITLE.....
.....
```

A partir de este momento, los datos de la columna (por ejemplo lShadow) serán accesible de cualquiera de los dos modos siguientes:

```
.....
oColumna1:lShadow := .T.
.....

.....
oInforme:aColumns[1]:lShadow := .T.
.....
```

6.9.3.1 Los principales datos de los objetos TRColumn son:

oReport	Es el objeto TReport que contiene la columna
aData	Es el array que contiene los datos que mostrará la columna. Cada elemento es un codeblock para cada una de las líneas impresas para cada registro.
aTitle	Es el array que contiene las líneas de título de la columna. Un elemento por línea de título. Cada uno de los elementos es un codeblock

aPicture	Formato de visualización de los datos de la columna. Un elemento por cada línea de datos de la columna
bDataFont	Fuente de letra para los datos de la columna. Es un codeblock que al evaluarse devuelve un índice a la tabla de fuentes del informe.
bTitleFont	Fuente de letra para el título de la columna. Es un codeblock que al evaluarse devuelve un índice a la tabla de fuentes del informe.
bTotalFont	Fuente de letra para el total de la columna. Es un codeblock que al evaluarse devuelve un índice a la tabla de fuentes del informe.
bTotalExpr	Es un codeblock con la clausula FOR para el cálculo de los totales.
cTotalPict	Es la máscara de visualización (PICTURE) del total de las columnas. Por defecto es el de la primera línea de datos de la columna.
nWidth	Es el ancho de la columna.
nDataHeight	Es la altura de las líneas de datos.
nTitleHeight	Es la altura de las líneas del título.
nTotal	Es el acumulador para el cálculo del total de la columna.
nCol	Es la posición horizontal de la columna en la página.
nSize	Es el ancho en caracteres de la columna.
nPad	Es la alineación de los datos en la columna.
nPen	Es el índice de la tabla de objetos PEN del informe que se va a utilizar para dibujar las líneas a los lados de las columnas.
ITotal	Verdadero si la columna está totalizada
IShadow	Verdadero si la columna está sombreada
IGrid	Verdadero si la columna tiene líneas a los lados.

Al igual que pasaba con los objetos TRLine, los métodos de los objetos TRColumn son de uso interno y de poca utilidad, por lo que no los incluimos aquí.

6.9.4 El objeto TRGroup

Es el encargado de manejar toda la información referida a grupos. Al igual que sucedía con los objetos TRColumn, los objetos TRGroup se crean en el interior del objeto TRReport, pero es posible referenciarlos mediante una variable:

```
.....
GROUP oGrupo1 ON .....
.....
```

```
.....
? oGrupo1:nCounter
.....
```

6.9.4.1 Los principales datos de los objetos TRGroup son:

oReport	Objeto TReport que contiene al objeto TRGroup
aTotal	Array con los valores de los totales del grupo para cada una de las columnas del informe. Un elemento por cada columna.
bGroup	Codeblock que contiene la expresión del grupo.
bHeader	Codeblock de cabecera. Devuelve una cadena de caracteres que se utiliza como cabecera de grupo.
bFooter	Codeblock de pie. Devuelve una cadena de caracteres que se utiliza como pie de grupo.
bHeadFont	Codeblock que indica la fuente para la cabecera del grupo. Devuelve un índice a la tabla de fuentes del informe.
bFootFont	Codeblock que indica la fuente para el pie del grupo. Devuelve un índice a la tabla de fuentes del informe.
cValue	Valor actual del grupo.
nCounter	Número de elementos procesados en el grupo actual
IEject	Indica si se efectuará salto de página al final del grupo.

6.9.4.2 Los METODOS de mayor interés de TRGroup son:

Reset()	Pone a cero el contador del grupo y los totales de las columnas
Check()	Comprueba si se ha producido un cambio de grupo

4

OOP

La programación orientada a objetos merece su apartado propio. Aquí veremos como se programa en OOP, ejemplos e invitación a realizar tus propias clases.

PROGRAMACIÓN ORIENTADA A OBJETOS

Antes de empezar debo indicar que este capítulo tiene poco de cosecha propia y mucho de copia de otros artículos aparecidos en internet, dentro del foro de Fivewin. Me he limitado a realizar una mezcla de varios autores siguiendo criterios muy personales.

Introducción

Actualmente, una de las áreas más candentes en la industria y en el ámbito académico es la orientación a objetos. La orientación a objetos promete mejoras de amplio alcance en la forma de diseño, desarrollo y mantenimiento del software ofreciendo una solución a largo plazo a los problemas y preocupaciones que han existido desde el comienzo en el desarrollo de software: la falta de portabilidad del código y reusabilidad, código que es difícil de modificar, ciclos de desarrollo largos y técnicas de codificación no intuitivas.

Un lenguaje orientado a objetos ataca estos problemas. Tiene tres características básicas: debe estar basado en objetos, basado en clases y capaz de tener herencia de clases. Muchos lenguajes cumplen uno o dos de estos puntos; muchos menos cumplen los tres. La barrera más difícil de sortear es usualmente la herencia.

El concepto de programación orientada a objetos (OOP) no es nuevo, lenguajes clásicos como SmallTalk se basan en ella. Dado que la OOP. se basa en la idea natural de la existencia de un mundo lleno de objetos y que la resolución del problema se realiza en términos de objetos, un lenguaje se dice que está basado en objetos si soporta objetos como una característica fundamental del mismo.

Hay que aclarar que la OOP en inglés ó POO en español, no es un lenguaje sino una técnica de programación, con esto quiero decir que cualquier lenguaje podría utilizar este sistema de programación, de hecho la mayoría de los viejos lenguajes de

Programación han evolucionado en ese sentido, unos de una forma más acertada que otros, creándose así tres grandes grupos a saber:

- Lenguajes de programación POO puros: Son aquellos que desde sus orígenes se pensaron para utilizar esta técnica, quizás los representantes más conocidos sean el SmallTalk y el propio Java (el C++ 'sin punteros').

- Lenguajes mezclados: Son lenguajes en los que se pueden mezclar distintas técnicas, incluidas la POO o la programación estructurada etc. También se pueden incluir en este grupo aquellos que teniendo 'algo de POO' no cumplen con todas las características mínimas que identifican a la POO. Aquí en este grupo estarían la mayoría como el Visual Basic, el C++, xBase++ (por cierto se suele utilizar mucho la nomenclatura de <Lenguaje> ++ para anunciar que contiene características POO). Y también estaría aquí el increíble, el inigualable y grande entre los grandes Clipper 5.x, luego me centraré más en nuestro querido compilador.

- Lenguajes no POO: Son, como es lógico, los que no tienen posibilidades para programar orientado al objeto. de una forma fácil. Aquí estarían los viejos lenguajes secuenciales con su característico GOTO como GWBasic, los procedurales como Ada, APL, Pascal, C y mezclas como las versiones de Clipper anteriores a la 5.x.

El elemento fundamental de la OOP es, como su nombre lo indica, el objeto. Podemos definir un objeto como un conjunto complejo de datos y programas que poseen estructura y forman parte de una organización.

Esta definición especifica varias propiedades importantes de los objetos. En primer lugar, un objeto no es un dato simple, sino que contiene en su interior cierto número de componentes bien estructurados. En segundo lugar, cada objeto no es un ente aislado, sino que forma parte de una organización jerárquica o de otro tipo.

Estructura de un objeto

Los OBJETOS es la parte más importante de la POO, pero antes habría que dar la definición de CLASES. Hay muchas, casi tantas como programadores que usan técnicas POO.

La CLASE es la idea, la definición de un tipo de OBJETOS, por tanto es algo abstracto. Cuando un programador crea una clase lo que está haciendo es un molde del que sacaremos algo concreto, los OBJETOS, de aquí podemos decir como conclusión que un OBJETO es un valor concreto del tipo de la CLASE de la que lo hemos instanciado. Si no lo entiendes no te preocupes ya que con los ejemplos que veremos más adelante no cabrán dudas.

Hay que decir que al igual que existe el tipo carácter, el numérico, el fecha etc. también un OBJETO es un tipo de dato especial, de hecho en POO cada CLASE es un tipo de dato extendido o complejo, de ahí que empresas especializadas en POO como Borland o CA, a los nombres de clases le pongan una 'T' inicial indicando Tipo, por ejemplo TBrowse, TBColumn, TGet en Clipper o TForm, TEdit etc en los lenguajes de Borland, esto mismo lo ha adoptado FiveTech para sus clases, así se puede decir que, por ejemplo, un objeto de nombre oGet es del tipo TGet.

En Objects podemos averiguar el tipo o nombre de la clase a la que pertenece un objeto usando el método predefinido ::ClassName():

```
local oGet := TGet():New() // oGet es tipo "0" de la clase TGet
```

```
? ValType( oGet )
```

```
? "El objeto oGet es del tipo", oGet:ClassName()
```

```
/* Truco 01*/
```

A veces además de saber que el tipo de variable que estamos tratando es un objeto, necesitamos su tipo extendido, o sea de qué clase es, para hacer eso podríamos hacer una función que nos devolviera realmente el tipo tradicional para cualquier dato o el nombre de la clase para los objetos utilizando alguna función interna indocumentada de Clipper:

```
//-----//
```

```
function main()
```

```
    local n := 1000.09
    local d := date()
    local c := "lolo"
    local l := .t.
    local b := { || nil }
    local a := { n, d, c }
    local o := TBrowseNew()
```

```
    cls
```

```
    ? ClsType( n )
    ? ClsType( d )
    ? ClsType( c )
    ? ClsType( l )
    ? ClsType( b )
    ? ClsType( a )
    ? ClsType( o )
```

```
return( nil )
```

```
//-----//
```

```
// Función que devuelve el tipo de dato que le enviamos aunque éste sea un
// objeto. Utilizo la funcion __ClassNam() y el método ::ClassH() de Clipper,
// tanto una como el otro indocumentados.
```

```
function ClsType( x )
```

```
    local cRet := ""
```

```
    cRet := ValType( x )
```

```
    if cRet == "0"
```

```
        cRet := __ClassNam( x:ClassH() )
```

```
    endif
```

```
return( cRet )
```

```
//-----//
```

Un objeto puede considerarse como una especie de cápsula dividida en tres partes:

- 1 - RELACIONES
- 2 - PROPIEDADES
- 3 - METODOS

Cada uno de estos componentes desempeña un papel totalmente independiente:

Las relaciones permiten que el objeto se inserte en la organización y están formadas esencialmente por punteros a otros objetos.

Las propiedades distinguen un objeto determinado de los restantes que forman parte de la misma organización y tiene valores que dependen de la propiedad de que se trate. Las propiedades de un objeto pueden ser heredadas a sus descendientes en la organización.

Los métodos son las operaciones que pueden realizarse sobre el objeto, que normalmente estarán incorporados en forma de programas (código) que el objeto es capaz de ejecutar y que también pone a disposición de sus descendientes a través de la herencia.

Los Métodos son como las funciones o procedimientos que se ejecutan como respuesta a un mensaje (MESSAGE). Como regla general los Métodos y los mensajes suelen tener el mismo nombre, aunque con Objects no siempre tiene porqué ser así.

Para enviar un MESSAGE a un objeto se utiliza el operador de envío SEND representado, en Clipper, por ':' como por ejemplo: oObj:End()

Hay unos métodos especiales que conviene conocer y tener en cuenta ya que existen en casi todos los Sistemas Orientados al Objeto:

- los **CONSTRUCTORES** para hacer una reserva de memoria, en ellos se les dan valores por defecto a las DATAS, Variables de Instancia, Atributos ó Propiedades, como se quieran llamar, cada clase debería tener al menos uno. Normalmente se le da el nombre New, Init, Start, etc... siempre devuelven Self o sea al propio objeto.

- los **DESTRUCTORES** para liberar la memoria, en Clipper + objects no hace falta definirlo aunque no estaría de más a acostumbrarnos a hacerlo y llamarlo End, Destroy etc como en otros sistemas. En Clipper el ámbito del objeto hace que se libere la memoria en un determinado momento, por ejemplo si lo definimos como local, sólo estará vigente mientras estemos en la función. En cualquier caso podríamos hacer oObj := nil y también funciona ya que el área de memoria que usaba queda marcada como disponible y el sistema de recolección de basura de Clipper la recuperará en cuanto le sea posible.

Al igual que cualquier variable de cualquier tipo, las DATAS y los Métodos de un objeto se les puede asignar un ámbito al definirlos. Estos pueden ser:

- **EXPORT** ó **PUBLIC**: Cuando podemos manipularlos directamente desde fuera del objeto.

- **PROTECTED** ó **READONLY**: Cuando la única manipulación que admite es la lectura, la asignación la controla el Objeto internamente.

- **LOCAL**, **HIDDEN** ó **HIDE**: Cuando el control absoluto lo tiene el propio objeto, si intentamos acceder a ellos el sistema nos da un error.

Encapsulamiento

Como hemos visto, cada objeto es una estructura compleja en cuyo interior hay datos y programas, todos ellos relacionados entre sí, como si estuvieran encerrados conjuntamente en una cápsula. Esta propiedad (encapsulamiento), es una de las características fundamentales en la OOP.

Los objetos son inaccesibles, e impiden que otros objetos, los usuarios, o incluso los programadores conozcan cómo está distribuída la información o qué información hay disponible. Esta propiedad de los objetos se denomina ocultación de la información.

Esto no quiere decir, sin embargo, que sea imposible conocer lo necesario respecto a un objeto y a lo que contiene. Si así fuera no se podría hacer gran cosa con él. Lo que sucede es que las peticiones de información a un objeto. deben realizarse a través de “mensajes” dirigidos a él, con la orden de realizar la operación pertinente. La respuesta a estas ordenes será la información requerida, siempre que el objeto considere que quien envía el mensaje está autorizado para obtenerla.

El hecho de que cada objeto sea una cápsula facilita enormemente que un objeto determinado pueda ser transportado a otro punto de la organización, o incluso a otra organización totalmente diferente que precise de él. Si el objeto ha sido bien construido, sus métodos seguirán funcionando en el nuevo entorno sin problemas. Esta cualidad hace que la OOP sea muy apta para la reutilización de programas.

Herencia

Otra de las cosas que nos aporta la POO es la posibilidad de derivar nuevas clases de otras ya existentes (en POO eso se llama REUSABILIDAD). La nueva clase derivada se llama HIJA y hereda todas las características de la MADRE o Clase Superior o SuperClase. Desde la MADRE y de las nuevas HIJAS se pueden derivar otras clases, con lo que crearíamos lo que se llama GERARQUIA DE CLASES, por ejemplo en FiveWin casi todas las clase están derivadas de TWindows.

La clase HIJA hereda todas las características y comportamientos, o sea DATAS y Métodos de la MADRE, sólo tenemos que modificar (sobreescribir) o crear las nuevas propiedades o comportamientos que nos interese.

Por ejemplo:

```
//-----//
#include "Objects.ch"
//-----//

CLASS TMsg FROM TBox // TMsg hereda automáticamente todos las data y métos de
//la clase TBox, podemos ahora crear nuevas datas y métodos
// ó sobreescribir los que ha heredado de TBox como es el
```

```

// caso del método Show() ó New() el método constructor

DATA cMessage AS CHARACTER

METHOD New( cMessage, nTop, nLeft, nBottom, ;
            nRight, cTitle, cBox, cColor, cClrTit ) CONSTRUCTOR
METHOD Show()

ENDCLASS

```

TMsg está derivada de **TBox** por lo que tiene las misma DATAs y Métodos, tan sólo nos preocuparemos de lo nuevo del cMessage y de darle nuevas posibilidades a Show().

La Herencia puede ser SIMPLE cuando la clase sólo hereda de una Clase Madre ó MÚLTIPLE cuando la clase hereda de más de una Clase Madre.

Ante la herencia múltiple existe lo que se llama CLASES CONTENEDORAS ó PORTADORAS y Objetos contenidos.

A veces es necesario que nuestra clase tome características o comportamiento de más de una clase, es aquí donde tenemos que decidirnos CLASE CONTENEDORA ó HERENCIA MÚLTIPLE esa es la cuestión.

Objects admite las dos posibilidades. Yo recomiendo utilizar objetos contenidos, ya que será mucho más fácil de entender el código fuente de las clases y desaparecen las colisiones que pueda haber entre las Superclases cuando tienen métodos o datas con el mismo nombre.

Mira este ejemplo:

```

/-- SuperClase 01 -----//

CLASS SClas01

    DATA cNombre AS STRING
    DATAS cArticulo

    METHOD New()
    METHOD Aceptar()
    METHOD Ver()

ENDCLASS

/-- SuperClase 02 -----//

CLASS SClas02

    DATA cNombre AS STRING
    DATAS nValor

    METHOD New()
    METHOD Aceptar()
    METHOD Acumular()

ENDCLASS

/-- Clase derivada -----//

CLASS Clas03 FROM SClas01, SClas02

    DATA nTotal

```

```
METHOD New() CONSTRUCTOR
METHOD Restar()
```

```
ENDCLASS
```

La clase **Clas03** hereda las datas y los métodos de las Superclases **SClas01** y **SClas02**, pero existen colisiones ya que hay métodos y datas con el mismo nombre en las clases **SClas01** y **SClas02**. Objects lo resuelve por preferencias, cuando no se dice nada Objects piensa que el método o la data es el de la primera. Para dar preferencia a un método de la SClas02 tendríamos que hacer este lío:

```
//-- Clase derivada -----//
CLASS Clas03 FROM  SClas01, SClas02
DATA nTotal
METHOD New() CONSTRUCTOR
METHOD Restar()
    // para que el mensaje Acptar() de la SClas02 tenga preferencia
    DELEGATE MESSAGE Aceptar, _Aceptar TO _SClas02
ENDCLASS
```

Si no lo hacemos dentro de la clase heredada Clas03 ::Aceptar() estaremos accediendo al método aceptar de la Superclase SClas01.

Por eso yo recomiendo el uso de clases contenedoras de objetos, el ejemplo anterior quedaría así:

```
CLASS Clas03 FROM  SClas01
DATA nTotal
DATA oSClas02
AS OBJECTS // En New por ejemplo creamos un objeto SClas02
METHOD New() CONSTRUCTOR
METHOD Restar()
ENDCLASS
```

De esta manera cuando queramos acceder al Aceptar sólo tenemos que hacer:

```
::oSClas02:Aceptar() // De la clase SClas02
::Aceptar()         // De la Clase SClas01
```

¿ Cual es más fácil ?

La compatibilidad está garantizada con las clases contenedoras...

Polimorfismo

Esto quiere decir que mensajes con el mismo nombre enviados a objetos de diferente tipo de clase tienen respuestas diferentes.

En la vida real si le enviamos el mensaje "patada" a un objeto "balón" responde saliendo disparado hacia una portería de futbol por ejemplo, pero si enviamos ese mismo mensaje "patada" a un objeto "vaso" este responderá posiblemente partiéndose en mil pedazos.

En POO es lo mismo, podemos definir métodos con el mismo nombre pero el comportamiento será diferente en objetos de clases distintas.

Por ejemplo el método CLOSE() de TWindows y de TDbf tienen comportamientos diferentes.

```
oWnd:Close()  
oDb:Close()
```

Por la experiencia que tengo os recomiendo que pongais nombres iguales para acciones idénticas aunque el comportamiento por la clase del objeto sea diferente, como en el ejemplo anterior de oWnd y oDb y :Close(), con esto se consigue parametrizar muchísimo los programas.

Por ejemplo, podríamos tener en nuestro programa una función genérica a la que se le enviara un array de objetos o un sólo objeto y esta se encargara de cerrarlo sea del tipo de clase que sea:

```
function Cierra( aObj )  
  
    local nElementos := len( aObj )  
    local i := 0  
  
    FOR i := 1 TO nElementos  
        aObj[ i ]:Close()  
    NEXT  
  
return( nil )
```

Como es lógico esta función se podría refinar mucho más e incluso adaptarla a otros métodos. Con FiveWin tenemos una función que hace esto de una forma genérica, se llama aSend().

OBJECT: POO en Fivewin

En éste capítulo vamos a estudiar la comandos propuestos por Objects el motor de POO de FiveWin.

Aunque es posible definir más de una clase por PRG siguiendo una serie de pautas con Objects pueden surgir problemas, así que por este motivo y por la modularidad y claridad en nuestros fuentes mejor definir sólo una clase por PRG.

El prototipo de definición de la clase se hace entre las sentencias CLASS y ENDCLASS y luego se desarrollan los métodos. La sintaxis general de Objects quedaría así:

```
CLASS <ClassNombre> [ FROM | INHERIT <SuperClass> ]

    DATA <DataNombre1> [, <DataNombreN>] [ AS <Tipo> ] [ INIT <uValor> ]
    CLASSDATA <DataNombre1> [, <DataNombreN>] [ AS <Tipo> ] [ INIT <uValor> ]

    METHOD <MethodNombre>( [<params,...>] ) [ CONSTRUCTOR ]
    METHOD <MethodNombre>( [<params,...>] ) BLOCK <CodeBlock>
    METHOD <MethodNombre>( [<params,...>] ) EXTERN <FuncNombre>( [<params,...>] )
    METHOD <MethodNombre>( [<params,...>] ) INLINE <Code,...>
    METHOD <MethodNombre>( [<params,...>] ) VIRTUAL
    METHOD <MethodNombre>( [<params,...>] ) SETGET
    METHOD <MethodNombre>( [<param>] ) OPERATOR <op>

    MESSAGE <MessageNombre> METHOD <MethodNombre>( [<params,...>] )
    MESSAGE <MessageNombre>() METHOD <MethodNombre>( [<params,...>] )

    ERROR HANDLER <MethodNombre>( [<params,...>] )
    ON ERROR <MethodNombre>( [<params,...>] )

ENDCLASS | END CLASS

//----- Para la definición de métodos:

METHOD <MethodNombre>( [<params,...>] ) CLASS <ClassNombre>
```

Siendo:

ClassNombre: El nombre de la clase que se está definiendo.

SuperClass: El nombre de la Clase Madre.

DataNombre...: El nombre de las Datas.

Tipo: El tipo de las Datas puede ser CHARACTER o STRING, NUMERIC, DATE, BOOL o LOGICAL, OBJECTS, BLOCK y ALL cuando no se prototipa una data se considera ALL.

uValor: Valor inicial por defecto de las datas

MethodNombre: Nombre de los métodos.

Params: Parámetros pasados a los Métodos ó a los mensajes.

CodeBlock: El CodeBlock para los Métodos de tipo BLOCK.

FuncNombre: Nombre de la función en C ó en Ensamblador a la que llamará el Método Extern.

Code: Código en Clipper para los Métodos INLINE, puede haber más de una sentencia separada por comas, la última será la que definirá el valor de retorno del método.

Op: Puede ser cualquier operador matemático de los existentes en Clipper.

MessageNombre: Nombre del Mensaje.

Además con Objects se puede utilizar:

```
[CREATE] CLASS <ClassNombre> [INHERIT FROM | INHERIT | FROM | OF] <SuperClass>
[, <MasSuper> ]
// Se puede poner CREATE CLASS y OF
// Objects emula la herencia múltiple, puede haber una <MasSuper>

    VAR <DataNombre1>, <DataNombreN,...> [ AS <Tipo,...> ][AMBITO] [INIT |
DEFAULT]
    INSVAR <DataNombre1>, <DataNombreN,...> [ AS <Tipo,...> ][AMBITO] [INIT |
DEFAULT]
    // INSVAR y VAR son sinónimo de DATA
    // El AMBITO puede ser: PUBLIC, EXPORT, READONLY, PROTECTED, LOCAL ó HIDDEN
    // Además de INIT se puede usar DEFAULT

    CLASSVAR <DataNombre1>, <DataNombreN,...> [ AS <Tipo,...> ][AMBITO] [INIT |
DEFAULT]
    // CLASSVAR es sinónimo de CLASSDATA

// Tanto a las DATAS como a las CLASSVAR se le puede INSTANTIATE en vez de INIT

    METHOD <MethodNombre1> [, <MethodNombreN> ] [AMBITO]
    // Se pueden definir más de un método por línea separados por ','
    // El Ambito puede ser: PUBLIC, EXPORT, LOCAL ó HIDDEN
    // en vez de EXTERN se puede usar también CFUNC ó CMETHOD
    // MESSAGE es sinónimo de METHOD, ya explicaré cuando se debe usar.

    DELEGATE MESSAGE <MethodNombre>
    // Para Mensajes delegados de otras clases.

// Los métodos se pueden definir:
    METHOD [FUNCTION | PROCEDURE] <MethodNombre>
```

Ejemplo de gerarquia

Vamos a hacer una gerarquía de clases de gestión de Cajas en modo texto que ha sido probada con el motor de objetos de FiveWin (Objects) y con Harbour, por tanto, funciona en entornos MS-DOS y en cualquier otro donde lo haga el compilador Harbour.

Iremos explicando paso a paso lo que se hace:

```

//-- BOF TBox.prg --//
//-----//
// Para la definición de clase hay que incluir este
#include "Objects.ch" // fichero de cabecera, o FiveWin.ch si vas usar alguna
// clase de FiveWin ya existente.
//-----//

CLASS TBox // La definición de la clase comienza con el nombre de ésta

// Ahora seguimos con las datas. Normalmente se agrupan por tipos y además
// se pueden inicializar con algún valor:
DATA nTop, nLeft, ;
    nBottom, nRight AS NUMERIC INIT 0
DATA cTitle, cBox, ;
    cColor, cClrTit, ;
    cImage AS STRING INIT ""
// Tanto a las datas como a los métodos les podemos declarar el ámbito de
// visibilidad, esta data, por ejemplo, es de sólo lectura

DATA lVisible AS LOGICAL INIT .t. READONLY

// Ahora es el turno de los métodos:
// Normalmente se comienza por el método constructor y se continúa con los
// demás.
METHOD New( nTop, nLeft, nBottom, nRight, ;
    cTitle, cBox, cColor, cClrTit ) CONSTRUCTOR
METHOD Show()
METHOD Hide()

ENDCLASS // Fin de la definición de la clase

//-----//
// Aquí fuera de la definición de la clase se desarrollan los métodos:
//-- Asignación de valores iniciales
METHOD New( nTop, nLeft, nBottom, nRight, ;
    cTitle, cBox, cColor, cClrTit ) CLASS TBox

// Con Objects podemos usar esta definición para asignar los valores
// pasados en el método o asignarles un valor por defecto:
BYNAME nTop DEFAULT 0 // Equivale a ::nTop := if( nTop == NIL, 0, nTop )
BYNAME nLeft DEFAULT 0
BYNAME nBottom DEFAULT MaxRow()
BYNAME nRight DEFAULT MaxCol()
BYNAME cTitle DEFAULT ""
BYNAME cBox DEFAULT "__+|.-+| "

BYNAME cColor DEFAULT "W+/RB"
BYNAME cClrTit DEFAULT "N/W*"

// El método constructor lo que hace es dar valores por defecto a las
// datas

::cImage := ""
::lVisible := .f.

return( Self ) // El constructor siempre devuelve una referencia al propio
// objeto

//-----//
// Acuerdate de asociar el método a una clase, aunque en objects no es
// necesario en otros sistemas como en Harbour o VO si lo es:
//-- Enseña la caja

```

```

METHOD Show() CLASS TBox

    local nY := int( ( ::nLeft + ::nRight - len( ::cTitle ) + 1 ) / 2 )

    if !::lVisible
        ::cImage := SaveScreen( ::nTop, ::nLeft, ::nBottom + 1, ::nRight + 2 )
        DispBegin()
        DBGShadow( ::nTop, ::nLeft, ::nBottom, ::nRight )
        DispBox( ::nTop, ::nLeft, ::nBottom, ::nRight, ::cBox, ::cColor )
        @ ::nTop, nY SAY ::cTitle COLOR ::cClrTit
        DispEnd()
        ::lVisible := .t.
    endif

return( Self ) // Excepto para los métodos que tengan que devolver algún valor
                // específico yo recomiendo que devuelvan una referencia a
                // ellos mismos para luego poder encadenar metodos así:
                // local oBox := TBox():New():Show()

//-----//
//-- Oculta la caja
METHOD Hide() CLASS TBox

    if ::lVisible
        RestScreen( ::nTop, ::nLeft, ::nBottom + 1, ::nRight + 2, ::cImage )
        ::lVisible := .f.
    endif

return( Self )

//-----//
//-- EOF TBox.prg --//

```

Ahora vamos a aplicar la herencia creando otra clase

```

//-- BOF TBoxMsg.prg --//
//-----//

#include "Objects.ch"

//-----//

CLASS TBoxMsg INHERIT TBox // Esta clase está heredada de TBOX

    DATA aItem          AS ARRAY INIT {}
    DATA nTemp, nMaxLen, ;
        nItem, nKey      AS NUMERIC INIT 0
    DATA lSound          AS LOGICAL INIT .f.

    METHOD New( aItem, nTop, nLeft, nTemp, ;
        cTitle, cBox, cColor, cClrTit, lSound ) CONSTRUCTOR

    METHOD Show() // Se pueden sobrescribir los métodos definidos
                // la superclase para ampliar o dar un nuevo
                // comportamiento al método

    METHOD Exec()

ENDCLASS

//-----//

```

```

METHOD New( aItem, nTop, nLeft, nTemp, ;
           cTitle, cBox, cColor, cClrTit, lSound ) CLASS TboxMsg

    // Con Super: podemos acceder a los método de la superclase
    // Aqui lo hacemos para iniciar las datas de la superclase que son también
    // de la clase que estamos definiendo, esto es de uso normal:

    Super:New( nTop, nLeft,,, cTitle, cBox, cColor, cClrTit )

    BYNAME aItem      DEFAULT {}
    BYNAME nTemp      DEFAULT 0
    BYNAME lSound     DEFAULT .t.
    BYNAME cTitle     DEFAULT " Atención "
    BYNAME cColor     DEFAULT "w+/R"
    BYNAME cClrTit    DEFAULT "N/W"

    if ::nTemp == 0
        AAdd( ::aItem, " " )
        AAdd( ::aItem, "Pulse una tecla" )
    endif

    ::nItem := len( ::aItem )

    aEval( ::aItem, { |Ele, nTmp| ;
        ::nMaxLen := if( ( nTmp := len( Ele ) ) > ::nMaxLen, nTmp, ::nMaxLen ) } )

    ::nMaxLen := ::nMaxLen + 2

    if ( ::nMaxLen - 4 ) < len( ::cTitle )
        ::nMaxLen := len( ::cTitle ) + 4
    endif

    BYNAME nTop      DEFAULT int( ( MaxRow() - ( ::nItem + 2 ) ) / 2 )
    BYNAME nLeft     DEFAULT int( ( MaxCol() - ::nMaxLen ) / 2 )

    ::nBottom := ::nTop + ::nItem + 1
    ::nRight  := ::nLeft + 1 + ::nMaxLen

    return( Self )

//-----//

METHOD Show() CLASS TboxMsg

    local nTp  := ::nTop
    local nLft := ::nLeft + 1

    Super:Show() // Asi se puede acceder al metodo de la Superclase cuando
                // sobreescribimos a uno de sus métodos

    aEval( ::aItem, { |Ele| DevPos( ++nTp, nLft ), ;
        DevOut( PadC( Ele, ::nMaxLen, " " ), ::cColor ) } )

    return( Self )

//-----//

METHOD Exec() CLASS TboxMsg

    local nCursor := SetCursor( 0 )

    ::Show()
    if ::lSound

```

```

        Tone( 600, 0.25 )
        Tone( 300, 0.5 )
        Tone( 900, 1 )
    endif

    CLEAR TYPEAHEAD

    ::nKey := InKey( ::nTemp )

    ::Hide()

    SetCursor( nCursor )

return( Self )

//-----//

//-- EOF TBoxMsg.prg --//

```

/* Truco*/

Para los que vengan de otros lenguajes y estén familiarizados con las estructuras pueden utilizar una clase como un almacén de variables de diferente tipo al estilo de las estructuras de C o los registros de Pascal:

```

//-----//
//

#include "Objects.ch"

function Main()

local oEstruct := TStruct()
oEstruct:cMiembro := "Hola, soy un miembro de tipo caracter"
oEstruct:nMiembro := 123
    ? oEstruct:cMiembro
    ? oEstruct:nMiembro
    ? oEstruct:lMiembro
    ? oEstruct:dMiembro
oEstruct:lMiembro := .t.

if oEstruct:lMiembro
    oEstruct:nMiembro += 1501

// Es igual que hacer:
// oEstruct:nMiembro := oEstruct:nMiembro + 1501

oEstruct:dMiembro += 10

endif

    ? oEstruct:cMiembro
    ? oEstruct:nMiembro
    ? oEstruct:lMiembro
    ? oEstruct:dMiembro

return( nil )

// Definicion de la "Clase/Struct"

```

```
// Sería fácil crear el include para hacer algo así como:
// STRUCT TStruct
// MEMBER cMiembro AS CHARACTER
//      ...
// END STRUCT
// Observa que solo tiene DATAS
```

```
CLASS TStruct
DATA cMiembro AS CHARACTER
DATA nMiembro AS NUMBER
DATA lMiembro AS LOGICAL
DATA dMiembro AS DATE INIT Date()
ENDCLASS
```

```
//-----
//
```

/* Truco*/

En OBJECTS, tanto los METHOD como las DATAS son métodos. Las DATAS son métodos especiales de tipo SETGET. Podemos aprovecharnos de esto haciendo que cuando asignemos u obtengamos un valor de una DATA se ejecute alguna acción a la vez, para ello tenemos que definir dos métodos uno de ellos con el mismo nombre pero precedido del guión bajo. Vamos a hacer un ejemplo:

```
//-----
```

```
#include "Objects.ch"
```

```
//-----
```

```
function main()
```

```
    local oObj := TObj():New()
```

```
    cls
```

```
    // Asignación SET
```

```
    oObj:cValor := "Esto es una prueba"
```

```
    // Obtención GET
```

```
    ? oObj:cValor
```

```
return( nil )
```

```
//-----
```

```
CLASS TObj
```

```
    DATA cVal AS CHARACTER
```

```
    METHOD New() CONSTRUCTOR
```

```
    METHOD cValor()
```

```
    METHOD _cValor()
```

```
ENDCLASS
```

```
//-----
```

```
METHOD New() CLASS TObj
```

```
        ::cVal := space( 10 )

return( Self )

//-----//
// GET
METHOD cValor() CLASS TObj

    Alert( "Obtengo el valor;aquí se puede meter alguna acción" )
    Alert( ::cVal )

return( ::cVal )

//-----//
// SET
METHOD _cValor( cStr ) CLASS TObj

    Alert( "Doy el valor " + cStr + ";aquí se puede meter alguna acción" )
    ::cVal := cStr

return( Self )

//-----//
```

Puedes compilar el ejemplo para ver como funciona. Si te das cuentas se usa cValor como una DATA y cVal como una especie de buffer intermedio.

Consideraciones finales

Beneficios con OOP

Día a día los costos del Hardware decrecen. Así surgen nuevas áreas de aplicación cotidianamente: procesamiento de imágenes y sonido, bases de datos multimediales, automatización de oficinas, ambientes de ingeniería de software, etc. Aún en las aplicaciones *tradicionales* encontramos que definir interfases hombre-máquina "a-la-Windows" suele ser bastante conveniente.

Lamentablemente, los costos de producción de software siguen aumentando; el mantenimiento y la modificación de sistemas complejos suele ser una tarea trabajosa; cada aplicación, (aunque tenga aspectos similares a otra) suele encararse como un proyecto nuevo, etc.

Todos estos problemas aún no han sido solucionados en forma completa. Pero como los objetos son portables (teóricamente) mientras que la herencia permite la reusabilidad del código orientado a objetos, es más sencillo modificar código existente porque los objetos no interaccionan excepto a través de mensajes; en consecuencia un cambio en la codificación de un objeto no afectará la operación con otro objeto siempre que los métodos respectivos permanezcan intactos. La introducción de tecnología de

objetos como una herramienta conceptual para analizar, diseñar e implementar aplicaciones permite obtener aplicaciones más modificables, fácilmente extendibles y a partir de componentes reusables. Esta reusabilidad del código disminuye el tiempo que se utiliza en el desarrollo y hace que el desarrollo del software sea más intuitivo porque la gente piensa naturalmente en términos de objetos más que en términos de algoritmos de software.

Problemas con OOP

Un sistema orientado a objetos, por lo visto, puede parecer un paraíso virtual. El problema sin embargo surge en la *implementación* de tal sistema. Muchas compañías oyen acerca de los beneficios de un sistema orientado a objetos e invierten gran cantidad de recursos luego comienzan a darse cuenta que han impuesto una nueva cultura que es ajena a los programadores actuales. Específicamente los siguientes temas suelen aparecer repetidamente:

Curvas de aprendizaje largas. Un sistema orientado a objetos ve al mundo en una forma única. Involucra la conceptualización de todos los elementos de un programa, desde subsistemas a los datos, en la forma de objetos. Toda la comunicación entre los objetos debe realizarse en la forma de mensajes. Esta no es la forma en que están escritos los programas orientados a objetos actualmente; al hacer la transición a un sistema orientado a objetos la mayoría de los programadores deben capacitarse nuevamente antes de poder usarlo.

Dependencia del lenguaje. A pesar de la portabilidad conceptual de los objetos en un sistema orientado a objetos, en la práctica existen muchas dependencias. Muchos lenguajes orientados a objetos están compitiendo actualmente para dominar el mercado. Cambiar el lenguaje de implementación de un sistema orientado a objetos no es una tarea sencilla; por ejemplo C++ soporta el concepto de herencia múltiple mientras que SmallTalk no lo soporta; en consecuencia la elección de un lenguaje tiene ramificaciones de diseño muy importantes.

Determinación de las clases. Una clase es un molde que se utiliza para crear nuevos objetos. En consecuencia es importante crear el conjunto de clases adecuado para un proyecto. Desafortunadamente la definición de las clases es más un arte que una ciencia. Si bien hay muchas jerarquías de clase predefinidas usualmente se deben crear clases específicas para la aplicación que se este desarrollando. Luego, en 6 meses ó 1 año se da cuenta que las clases que se establecieron no son posibles; en ese caso será necesario reestructurar la jerarquía de clases devastando totalmente la planificación original.

Performance. En un sistema donde todo es un objeto y toda interacción es a través de mensajes, el tráfico de mensajes afecta la performance. A medida que la tecnología avanza y la velocidad de microprocesamiento, potencia y tamaño de la memoria aumentan, la situación mejorará; pero en la situación actual, un diseño de una aplicación orientada a objetos que no tiene en cuenta la performance no será viable comercialmente.

Idealmente, habría una forma de atacar estos problemas eficientemente al mismo tiempo que se obtienen los beneficios del desarrollo de una estrategia orientada a objetos.

Debería existir una metodología fácil de aprender e independiente del lenguaje, y fácil de reestructurar que no drene la performance del sistema .

CLASES

Fivewin es una librería de clases. Estas clases, en formato OBJ pasan a nuestros ejecutables por medio del enlazador. Sin las clases de la POO, no sería posible enlazar con el API de windows. Esta es una relación, tomada de la propia ayuda que viene en el CD y de otros artículos de terceros, de las clases contenidas en la librería de Fivewin. La mayoría esta si traducir, puesto que se tomó directamente de la pantalla de ayuda.

BIPMAPS**Help**

Bitmaps Objects.

Inherits from TControl

DATOS

nX	Relative origin coordinates of the picture.
nY	Relative origin coordinates of the picture
nOldX, nOldY	Last relative origin of the picture
hBmpPal	Handle and palette of the Bitmap
cBmpFile	Name of the BMP file to display
cResName	Name of the BMP resource to display
ISretch	True if image has to be adjusted to all visible area
IScroll	True if image may be scrolled into its own visible area
ITransparent	If the bitmap is transparent. Pixel 0,0 (Top-left) indicates the mask color
aHotAreas	An array with some hot-areas on the bitmap
nVStep	Units to scroll on vertical scroll
nHStep	Units to scroll on horizontal scroll
nZoom	Units to zoom the picture

METODOS

New	Constructor from source code
Define	Constructor for resources
ReDefine	Constructor for resources
End	Ends control and make it disappear
AdjControls	Adjust controls contained in the bitmap object

Center	Centers the bitmap in the container window
Command	Processes WM_COMMAND WinApi messages
Default	Default initialization of the Bitmap
Destroy	Destroy the bitmap object.
Display	Displays the bitmap. Use :Refresh() instead to force it.
GotFocus	Action to be performed when getting the focus
HScroll	Horizontal ScrollBar messages dispatcher
Initiate	Generic Control initialization method
Inspect	Interactive load of Bitmap
LButtonDown	Action to be performed when clicking the mouse.
LoadImage	Load the bitmap from resources
LoadBmp	Load bitmap from disk.
nHeight	Retrieves height of the image.
nWidth	Retrieves width of the image.
nXExtra	Retrieves the size of image which it is not visible
nYExtra	The same for the other axis
Paint	Draws the bitmap
PageUp	Performs a PageUp of the bitmap if IScroll is .t.
PageDown	Performs a PageDown of the bitmap if IScroll is .t.
PageLeft	Performs a PageLeft of the bitmap if IScroll is .t.
PageRight	Performs a PageRight of the bitmap if IScroll is .t.
Reload	Load again a bitmap
Resize	Changes the dimension of the bitmap
ScrollAdjust	Updates the values on the scrollbars
ScrollDown	Scrolls the bitmap DOWN if IScroll is .t.
ScrollLeft	Scrolls the bitmap LEFT if IScroll is .t.
ScrollRight	Scrolls the bitmap RIGHT if IScroll is .t.
ScrollUp	Scrolls the bitmap UP if IScroll is .t.
SetBmp	Load bitmap from resource.
SetFore	Set this bitmap palette as the main active palette
VScroll	Vertical ScrollBar messages dispatcher
HScroll	Horizontal ScrollBar messages dispatcher
Zoom	Zoom the picture.

Note: The ScrollXXX Methods assume that IScroll is TRUE AND that the BitMap is larger than the visible area

BROWSE**Help****ListBox controls Class**

Inherits from **TControl**

DATOS

altems	Array of Items of the ListBox
aBitmaps	Array of related bitmaps handles for ownerdraw ListBox.
IOwnerDraw	If it is a ownerdraw ListBox.
nBmpHeigth	Height of bitmaps if ListBox is OwnerDraw
nBmpWidth	Width of Bitmaps if Listbox is OwnerDraw
cFileSpec	FileSpec to display a fields directory items.
bDrawItem	Optional codeblock to select what bitmap to draw on a ownerdraw

METODOS

New	Constructor from source code
ReDefine	Constructor for resources
Add	Add an Item to the list.
aSelections	Array with selected items on a MultiSelect ListBox.
cGenPrn	Generates the xBase source code of the control
Change	Changed event handler.
cToChar	Generates a data template to build a DialogBox
Default	Default Listbox initialization
Del	Deletes an Item.
Destroy	Destroys the control. Automatically called.
DrawItem	DrawItem event handler.
FillMeasure	FillMeasure event handler.

GetItem	Retrieves the text of any item.
GetPos	Retrieves the index of the currently selected item.
GetSel	Index number of the selected item. Zero if no item selected
GetSelCount	Retrieves the number of selected items on a multiselect ListBox
GetSelText	Retrieves the current select item as text
GetSelItems	Retrieves all the current selected items (Array of numbers)
GoBottom	Go to the last item in the listbox
GoTop	Go to the first item in the listbox
Initiate	Generic control initialization
Insert	Insert an Item.
IsMultipleSel	Check if the ListBox is multiple selection style
Len	Retrieves the length of the list.
LostFocus	Behavior when losing the focus
LButonUp	Behavior when unclicking the mouse left button
Modify	Modify the text of an Item.
MouseMove	Behavior when moving the mouse over the control
Reset	Empties the list.
Select	Select an item by its order
Set	Select an item by its text
SetBitmaps	Sets the ownerdraw related bitmaps
SetItems	Changes the items of the listbox
SetSel	Selects a different item
SetSelItems	Select a group of items on a multiselection ListBox
SwapDown	Toggles an item with its following one
SwapUp	Toggles an item with its preceding one
VScroll	Vertical scroll events handler

BRUSH**Help**

Brushes objects

DATOS

hBrush	Handle of the brush.
hBitmap	Handle of Bitmap if the brush is an image.
nCount	Internal counter for life control
cStyle	Basic style
nRGBColor	Color
cBmpFile	Bitmap file
cBmpRes	Bitmap resource

METODOS

New	Constructor from source code.
cGenPRG	Returns a string with the source code of the Brush.
End/Release	Destroy the Brush image.

BTNBMP

Help

Button Bitmaps Object.

Inherits from TControl.

DATOS

bAction	CodeBlock for the click event.
cAction	A string description of the action
cMsg	Message bar Text.
IPressed	Is button is pressed.
ICaptured	Is mouse is captured by the button.
IGroup	Beginning of new button group
IWorking	If .t. button is pressed
IBtnUp	if .t. Button is not pressed
IBtnDown	if .t. Button is pressed
IAdjust	* Obsolete If the bitmap has to be resized to cover all its surface
IProcessing	If .t. bAction is been evaluated at this moment
IBorder	If .t. the button has a border
hBmpPal1 pressed	The bitmap handle of the main image of button when not pressed
hBmpPal2 button.	The bitmap handle of the image of button pressed for animated button.
hBmpPal3	The bitmap handle of the image of button when disabled
cBmpPal1	The main image of button when not pressed
cBmpPal2	The image of button pressed for animated button.
cBmpPal3	The image of button when disabled
oPopup	Popup menu to show when the button is pressed

METODOS

New	Constructor from source code
NewBar	Constructor from source code for Button Bar
ReDefine	Creates a BtnBmp from resources
Click	Mouse left button event handler
FreeBitMaps	Releases all the bitmaps associated with this control
cGenPRG	Generates the source code of this control
Display	Displays the bitmap button
Disable	Disables the button
Enable	Enables the button
GoUp	Forces the button to go Unpressed
GotFocus	Action to be performed when getting the focus
HandleEvent	Generic events handler
Initiate	Generic Control initialization method
KeyChar	Action to be performed when pressing a key
LostFocus	Action to be performed when losing the focus.
LbuttonDown	Behavior when pressing the mouse
LbuttonUp	Behavior when releasing the mouse click
LoadBitmaps	Load the bitmap/s from resources
Paint	Draws the Button
MouseMove	Behavior when moving the mouse over it
Release	Destroy the Button image
SetFile	Changes to a different BMP file
Toggle	Changes the state of the button

Note: The data **IAdjust** is not supported because now the bitmaps background color is automatically adjusted to his window or control container. The pixel 0,0 (Left-Top) indicates the mask color to use for the adjustment. It is important that the color in that pixel is not used on any other part of the bitmap except where the mask should be applied. This kind of color adjustment is not possible with bitmap stretching

BUTTON**Help**

Command buttons Controls.

Inherits from TControl

DATOS

bAction	CodeBlock for the click event.
IDefault	The button has default style.
IProcessing	if .t. then bAction is been evaluated at this time

METODOS

New	Constructor from source code.
ReDefine	Constructor for resources.
cGetPRG	Generates the source code of the button.
Click	Mouse left button event handler.
Colors	Overrides control Color() management.
cToChar	Generates a data template to build a Dialog.
GotFocus	Action to be performed when getting the focus.
Initiate	Action to be performed on init.
KeyDown	Action to be performed when a key is pressed down.
LostFocus	Action to be performed when losing the focus.
MouseMove	Action to be performed when moving the mouse over.

BUTTONBAR

Help

Clase TBar

Button Bar Objects

Inherits from Tcontrol

DATOS

aButtons	Array of TBtnBmp objects
nGroups	Number of groups
nMode	The position of the BtnBar on the window
nBtnWidth	The height and width of its buttons. By default is 28x28
nBtnHeight	The height of the Bar is automatically adjusted to the height of the buttons.
l3D	If the BtnBar has to be shown 3Dlook

METODOS

New	Constructor from source code
NewAt	Constructor from source code at a certain location.
Add	Add a BtnBmp to the bar.
AddGroup	Adds a new logical group to the bar.
Adjust	Adjust the Button bar to the dimensions of its container window.
AdjMode	Changes the BtnBar position to floating
Del	Deletes a button from the bar.
DelGroup	Deletes a logical group from the bar.
Display	Displays the button bar
cGenPRG	Returns a string with the source code of the button bar
Float	Turns the buttonbar into floating mode.
FloatClick	Processes MouseClicks on the float mode buttonbar.
GetBtnTop	Returns the appropriate nTop when adding a new button
GetBtnLeft	Returns the appropriate nLeft when adding a new button
GoDown	Turns the buttonbar into down mode.
GoLeft	Turns the buttonbar into left mode.
GoRight	Turns the buttonbar into right mode.
GoTop	Turns the buttonbar into top mode.
LeftMode	Changes the BtnBar position to the left of the window
MouseMove	Activates the default message on window owner MsgBar

Paint	Displays the ButtonBar. Use :Refresh() instead this.
RButtonDown	Processes mouse right clicks on the buttonbar.
RightMode	Changes the BtnBar position to the Right the window
TopMode	Changes the BtnBar position to the Top of the window
Click	Mouse left button event handler
SaveIni	Saves the bar to an INI file
GetBtnCol	Computes the column of the new Button
SetMsg	Set the text of the message bar of its container window

CHECKBOX**Help**

Clase Tcheckbox

Logical CheckBox Control

Inherits from TControl

DATOS

None

METODOS

New	Constructor from source code.
ReDefine	Constructor for resources.
Click	Mouse left button event handler.
Default	Default initialization.
cToChar	Generates a data template to build a Dialog.
Initiate	Generic initialization.
MouseMove	Action to be performed when moving the mouse over.
Refresh	Repaints the control.
cGenPRG	Generates the source code of the control.
Reset	Resets the checkbox

CLIPBOARD

Help

COMBOBOX

Help

CURSOR

Help

Clase **TCursor**.

Mouse cursors managed as objects.

DATOS

hCursor	Handle of the cursor.
IPredef	If it is a Windows system predefined cursor.

METODOS

New	Constructor from source code
End	Destroys the cursor object

DATABASE

Help

Clase **Tdatabase**

Databases management as objects. DataBase OOPS management

DATOS

cAlias	The workarea alias where the DataBase object has been created
cFile	The name of the DBF file associated
cDriver	The name of the driver associated
IShared	If the workarea is shared
IReadOnly	If the workarea is readonly
bBoF	CodeBlock to execute when BoF()
bEoF	CodeBlock to execute when EoF()
bNetError	CodeBlock to execute when Net error
nArea	The workarea order used (i.e.: 1,2,3, ...)
lBuffer	To use or not a buffer. It is a must for NetWorking!
aBuffer	An editing buffer holding fields temporary data

The big benefit of using a TDataBase object is the editing where we may drastically reduced our source code size. aBuffer keeps an array with the same number of elements as fields are on the workarea. This buffer may be loaded from the record or saved to the record. So there is no need to work with local variables (normally used with this same purpose, to work as a field buffer), and just one TDataBase object will be enough and perfect to hold all temporary fields values

IOemAnsi	Oem to Ansi translation
ITenChars	if fields have ten chars, used internally
aFldNames	Array with all the field names

.

METODOS

New	Constructor
Skip	Skips like DbSkip()
Activate	Select its workarea as the one currently active
AddIndex	Adds a new index
Append	Adds a new record
AnsiToOem	Ansi to Oem translation
Blank	Empties the editing buffer
Bof	Checks if beginning of file
Close	Closes the workarea
CloseIndex	Closes an index
Commit	Flushes pending file buffers
Create	Creates a new database
CreateIndex	Creates a new Index
ClearRelation	Clear active relations
DeActivate	Closes the database. It acts like DbCloseArea()
DbCreate	Creates the DBF file from its structure in a array
Delete	Mark a record for deletion
Deleted	Return the deleted status of the current record
DeleteIndex	Remove a specified order from an order bag
Eof	Determine when End of file is encountered
Eval	Evaluates a codeblock for records matching a scope/condition
FCount	Return the number of fields in the current DBF file
FieldGet	Retrieve the value of a field variable
FieldName(nField)	Name of the field at nField position
FieldPos(cField)	Position of the field cField
FieldPut	Set the value of a field variable
Found	Determine if the previous search operation succeeded
GoTo	Move to the record having the specified record number
GoTop	Move to the first logical record
GoBottom	Move to the last logical record
IndexKey	Return the key expression of a specified index
IndexName	Return the name of an order in the Order list
IndexBagName	Return the Order bag name of a specified Order
LastRec	Determine the number of records in the current DBF file

IndexOrder	Return the position of an Order in the current Order List
Lock	Lock an open and shared database file
Load	Loads the record contents into the editing buffer
ReCall	Restore records marked for deletion
Modified	if .t. the database buffer hasa been modified
OemToAnsi	Oem to Ansi translation
Pack	Removes all record marked for deletion
RecCount	Determine the number of records in the current DBF file
RecLock	Lock the current record
RecNo	Return the current record number
Save	Saves the editing buffer contents into the DBF file
SetBuffer	Activates/Deactivates the editing buffer

Syntax:

<oWrk>:SetBuffer(<IOnOff>) \\ <IPrevious>

Parameters:

DIALOG

Help

Clase **Tdialog**

Heredada de **TWindows**

DATOS

cResName	Resource Name of the Dialog
cResData	Resource Data of the Dialog
hResources	Handle of the Resource where to find the Dialog
ICentered	Logical value if centering the Dialog when painting
IModal	Logical value if the Dialog Box is modal
IHelpIcon	Windows95 help icon pressed
bStart	Action to be performed automatically just once at start
IModify	Logical value to allow using the Controls -not implemented-

METODOS

New Generic constructor method

Syntax:

```
TDialog():New( <nTop>, <nLeft>, <nBottom>, <nRight>; <cCaption>,
<cResName>, <hResources>, <nStyle>; <ncClrText>, <nClrBack>, <oBrush>,
<oWnd>; <lPixels>, <olco>, <oFont> ) Þ <oDialog>
```

Parameters:

<nTop>, <nLeft>	These two determine the top left corner of the dialog box.
<nBottom>	The right bottom corner of the dialog box.
<nRight>	
<cCaption>	The title of the dialog box.

<cResName>	The name of the dialog box structure stored in a resource (either a .DLL or linked to the executable by RC.EXE).
<hResources>	The handle, which points to the resources that are linked to this application. If this parameter is not passed when <cResName> is passed it is assumed that the dialog box structure is stored in either the .EXE file or in the default .DLL.
<nStyle>	Determines the characteristics of the dialog box. These characteristics are defined by the same values that are used for the <nStyle> parameter of the New() method for the TWindow() class (defined in WINAPI.CH) or one of the following constants which can only be used for a Dialog box:
<nClrText>	The foreground color (the color that is used to draw on the dialog box). This can either be a MS-Windows COLOR number or a CA-Clipper color string (ie "R/G") in which case the nClrBack parameter will be ignored.
<nClrBack>	The background color of the dialog box. This has to be a MS-Windows COLOR number.
<oBrush>	The brush used to paint the client area of the dialog box. It is stored in the instance variable ::oBrush. If none is passed, it will be created by the method SetColor().
<oWnd>	The TWindow object of which this dialog box is the child window. If this parameter is not passed it means that this dialog will be linked to the main window of the application.
<lPixels>	Logical which determines whether the coordinates passed, will either be in pixels (a la Windows) or on a character base (a la CA-Clipper)
<olcon>	The icon that will be used whenever this dialog box is minimized. If none is passed the icon is USER.EXE (which means you get a small MS-Windows logo as the icon).
<oFont>	The font that will be used when something is written onto the dialog box or when a control is placed on the dialog box that has no font defined for it

Returns:

<oDialog>	The TDialog object that was created by the New() method.
-----------	--

Comments:

This constructor has a lot of parameters, so it quite easy to make a mistake. It is easier, both for you and for someone who is reading your code, to use the DEFINE DIALOG statement.

Define

Alternative constructor

Syntax:

```
TDialog():Define( <nTop>, <nLeft>, <nBottom>, <nRight>;<cCaption>, <nStyle>,
<lVbx>, <nClrText>;<nClrBack>, <oBrus>h ) Þ <oDialog>
```

Parameters:

The parameters mentioned for this method have exactly the same purpose and definition as the ones that are defined for the New() method.

Returns:

<oDialog> The TDialog object that was created by the New() method

Comments:

The Define() method is another constructor for TDialog() objects. It will also return a TDialog() object however without all the settings that the New() method will handle for you.

Activate Starts interaction with the Dialog Box

Syntax:

```
TDialog():Activate( <bClicked>, <bMoved>, <bPainted>, <lCentered>, <bValid>,
<lModal>, <bInit>, <bRClicked>, <bWhen> ) Þ NIL
```

Parameters:

<bClicked>	This codeblock will be executed whenever there is a click of the left mouse button on the dialog box. The codeblock will be called with three parameters:
<nRow>	The pixel row where the mouse cursor was positioned when the mouse click occurred.
<nCol>	The pixel column where the mouse cursor was positioned when the mouse click occurred.
<nKeyFlags>	Indicates which virtual keys are down and if the right or the middle mouse button was pressed also. It can be a combination of the following values which have the following constant names in the MS-Windows API description: MK_RBUTTON 2 The right mouse button was pressed also. MK_SHIFT 4 The <Shift> key was pressed. MK_CONTROL 8 The <Ctrl> key was pressed. MK_MBUTTON 16 The middle mouse button was pressed.
<bMoved>	Currently not used.

<bPainted>	This code block will be called whenever the dialog box is repainted (ie the dialog box receives a WM_PAINT message from MS-Windows). When <bPainted> is called it receives one parameter and that is a reference to the TDialog() object
<ICentered>	Determine whether or not the dialog box should be centered on the screen when it is displayed. If <ICentered> is .T. the dialog will be centered regardless of the coordinates that are stored in ::nTop, ::nLeft. The width and height of the dialog box however are NOT changed. If <ICentered> is .F. however the dialog box will be placed on the screen as the coordinates ::nTop, ::nLeft dictate this.
<bValid>	If this codeblock is passed it will be called whenever the dialog box should be closed. <bValid> has to return a logical value to indicate whether or not the dialog box is allowed to close.
<IModal>	This parameter determines whether or not this dialog box will be a MODAL or a NON MODAL dialog box. The default value is .T. which means that the current dialog box will be a modal dialog box, ie the program will continue only after the dialog box is closed.
<bInit>	This codeblock will be executed after the dialog box is displayed for the first time. It receives one parameter which is a reference to the TDialog() object itself.
<bRClicked>	This codeblock will be executed whenever there is a click of the right mouse button on the dialog box. The codeblock will be called with three parameters:
<bWhen>	This codeblock will be executed just before the dialog box is shown. The codeblock will be evaluated and called with three parameters: However if the result is .F. then the dialog box will NOT be shown.

Returns: NIL

Comments:

The Activate() method will actually show the dialog box on the screen. Before a call to this method is made the dialog box is not yet visible. Through the parameters Activate() receives, several aspects of the dialog box are controlled, like whether or not the dialog box is MODAL or NON-MODAL (<IModal>).

Close()

Close dialog box

Syntax:

Tdialog():Close(<nResult>) \ \ NIL

Parameters:

<nResult> The value the instance variable ::nResult receives when the dialog box is closed.

Returns:

NIL

Comments:

The Close() method is almost identical to the End() method, with regards to its purpose. This purpose is to close the dialog box and remove it from the screen. There is however one difference, unlike the End() method the Close() method has no default value for nResult. This means that if <nResult> is not passed that ::nResult will become NIL.

Command Handle commands sent to dialog box

Syntax:

TDialog():Command(<nWParam>, <nLParam>) \ Nil

Parameters:

<nWParam> This parameter actually holds two pieces of information which have the following meaning:

<nCode> = nHiWord(nWParam). This value specifies the notification code if the message is from a control. If the message is from an accelerator, this parameter is 1. If the message is from a menu, this parameter is 0. nID = nLoWord(wParam); This part represents the identifier of either an item, a control or an accelerator.

<nLParam> This value contains the handle of the control sending the message if the message is from a control. Otherwise, this parameter is NULL.

Returns: NIL

Comments:

The Command() method is automatically called (by the eventhandler) when processing a WM_COMMAND WinApi message. If a message is processed it should return 0.

cToChar Translate handle to structure item

Syntax:

TDialog():cToChar(<hActiveWnd>) \\ <cDialogStructure>

Parameters:

<hActiveWnd> The handle that should be included in the structure built by cToChar().

Returns:

<cDialogStructure> The dialog structure created by ::cToChar(). This structure is stored in a character string because CA-Clipper has no provisions for making and using structures.

Comments:

The cToChar() method will translate all the necessary information needed to create a dialog box into one big string. This string has exactly the structure needed by the MS-Windows API functions that create a dialog box. The information is stored in to a character string because CA-Clipper has no knowledge of structure's since this is not part of the CA-Clipper language.

DefControl Connect control to dialog box

Syntax:

TDialog():DefControl(<oControl>) \\ Nil

Parameters:

<oControl> The control that has to be connected to the current dialog box.

Returns:

NIL

Comments:

This method is used when a control is added to a dialog box that gets it's structural information from a resource (either linked to the .EXE by RC.EXE or stored in a .DLL file). Before the control is added a check is made to see whether or not the ::nID instance variable of <oControl> is unique for this dialog box. If the value of <oControl>:nID is already in use a warning will be shown and the control will NOT be added to the ::aControls instance variable.

Destroy Destroy dialog box

Syntax:

TDialog():Destroy() \\`<IxResult>`

Returns:

`<IxResult>` The return value depends upon the fact if the dialog was MODAL. If the dialog box was modal Destroy() will return .T., otherwise Destroy() will return NIL.

Comments:

The Destroy() method handles the actual removal of the dialog box. It should NOT be called directly. If you want to end a Dialog box you should either call the End() method or the Close() method.

!seealso wincls.eho:"Destroy" tdialog.eho:"End" tdialog.eho:"Close"

Display Dialog box

Syntax:

Tdialog():Display() \\`.F.`

Comments:

The Display() method paints the dialog onto the screen. After the dialog box is painted, Display() will see if the `::bStartinstance` variable `<> NIL`. If this is the case Display() will evaluate `::bStart`.

End

End dialog box

Syntax:

Tdialog():End(`<nResult>`) \\`.T.`

Parameters:

`<nResult>` The numerical value that will be passed on to the `::nResult` instance variable, when the dialog box is closed.

Comments:

With the End() method the programmer can lose the dialog box and remove it from the screen. However if the `::bValid` instance variable contains a code block, then the code block will be evaluated first. If the result of the evaluation is .T. the dialog box will be closed. If the evaluation of `::bValid` returns .F., then the dialog box will NOT be ended.

EraseBkGnd

Erase background of dialog box

Syntax:

TDialog():EraseBkGnd(<hDC>) \\.T.

Parameters:

<hDC> The DEVICE CONTEXT handle of this dialog box.

Comments:

This method can be used to repaint the background of the dialog box. It is also used by the eventhandler. However the background will be repainted only if the ::oBrush instance variable contains a brush object

FocusNext	Give focus to next control in dialog box
------------------	--

Syntax:

TDialog():FocusNext(<hCtrlFocus>, <IPrevious>) \\.NIL

Parameters:

<hCtrlFocus> It is the hWnd of the control currently focused.

<IPrevious> It is a logical value: .t. for the next and .f. for the previous.

Comments:

With the FocusNext() method the programmer can place focus on the control that is before (<IPrevious> is .F.) or after (<IPrevious> is .T.) the control identified by <hCtrlFocus>. To get a handle of a certain control you could use the GetItem() method to obtain <hctrlFocus> (assuming that you DO know the ID of the control), or just use the value stored in the ::hWnd instance variable for that specific control

GetItem	Get the handle of a certain control
----------------	-------------------------------------

Syntax:

TDialog():GetItem(<nId>) \\.<nHWnd>

Parameters:

<nId> The ID of the control for which the 'window handle' should be returned.

Returns: <nHWnd> The handle of the control that is identified by <nId>. If a control can be found on the current dialog box, <nHWnd>

will be the handle to that control. If no control can be found with id <nId> the return value will be 0.

Comments:

The purpose of this method is to retrieve a WINDOW HANDLE of a specific control, so you can use an API function to do a certain task. However use this with care, because using the wrong API function with a handle can cause serious problems for your application.

GotFocus	Called when dialog receives focus
-----------------	-----------------------------------

Syntax:

TDialog():GotFocus() \ \ <xValue>

Returns:

<xValue> If the ::bGotFocus contains a codeblock the return value will be the value that was the result of the evaluation of ::bGotFocus. Otherwise <xValue> will be NIL.

Comments:

This method is called just after the dialog box has received focus. This allows you to have some work done whenever the dialog receives focus. Whenever a dialog box receives focus the eventhandler will call the GotFocus() method, which in turn evaluates the ::bGotFocs instance variable.

!seealso tdialog.eho:"Lostfocus"

HandleEvent	Event handler for dialog boxes
--------------------	--------------------------------

Syntax:

TDialog():HandleEvent(<nMsg>, <nWParam>, <nLParam>) P

Comments:

This method handles all the events for a particular TDialog object. It directly calls a C-function (WndHandleEvent()). This is done to increase speed. For complete documentation on each and every value of <nMsg> and its associated values of <nWParam> and <nLParam> please check out a book (or help file) on MS-Windows messages and how they are handled.

Initiate()	Handle initialization of dialog box
-------------------	-------------------------------------

Syntax:

TDialog():Initiate() \\ .T.

Comments:

This method handles the initialization of the dialog box. If the dialog box contains one or more VBX controls the necessary initialization is performed. Also for all controls that are connected to the dialog box their individual Initiate() method is called. Just before the Initiate() method of the dialog ends it evaluates the ::bInit instance variable.

LostFocus	Handles losing focus by dialog box
------------------	------------------------------------

Syntax:

TDialog():LostFocus() \\ <xValue>

Returns:

<xValue> If the ::bLostFocus contains a codeblock the return value will be the value that was the result of the evaluation of ::bLostFocus.

Otherwise <xValue> will be NIL.

Comments:

The LostFocus() method is called by the eventhandler of the dialog box whenever the dialog box loses focus. When LostFocus() is called it in turn evaluates the ::bLostFocus codeblock

QueryEndSession	Check if application can be closed
------------------------	------------------------------------

Syntax:

TDialog():QueryEndSession() \\ <INoEnd>

Returns:

<INoEnd> is a logical which represents whether or not the Windows session may be ended. If <INoEdn> is .T. the Windows session may NOT be ended. If <INoEnd> is .F. the Windows session may be ended.

Comments:

The QueryEndSession() method is called by the eventhandler whenever Windows sends a WM_QUEYENDSESSION to a dialog box. This message is sent to a dialog box if the user wants to end the MS-Windows session. The QueryEndSession() will return .T. if the windows session is NOT allowed to end. Otherwise QueryEndSession() returns .F..

VbxFireEvent

Handling of VBX events by dialog box

Syntax:

TDialog():VbxFireEvent(<pEventInfo>) \\\

Parameters:

<pEventInfo> The <pEventInfo> is a 'pointer' to an event structure used for VBX controls in Borland's BIVBX10.DLL.

Comments:

This method is called by the eventhandler whenever a VBX control generates a WM_VBXFIREEVENT message. The method receives the required information via a pointer to a VBX structure (<pEventInfo>).

TFolder

Help

Folder Objects

Inherits from TControl

Variables

aPrompts	Array of Prompts
aDialogs	Array of NonModal DialogBox objects
nOption	Current selected page
nTabSize	Size of the tabs
oFont	<pre>DEFINE FONT fntArial NAME "Arial" SIZE 8, 12 oFld:oFont= fntArial</pre>



NOTA:

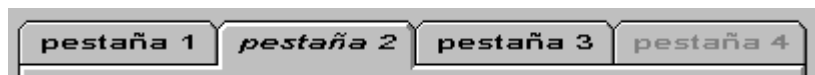
Si oFld:IWin95Look := .f., oFld:oFont hace referencia a las pestañas no seleccionadas.

Si oFld:IWin95Look := .t., oFld:oFont hace referencia a todas las pestañas.

oFont2	Tipo de letra de la pestaña seleccionada.
--------	---

SOLO SI IWin95Look := .f.

```
DEFINE FONT fntArial NAME "Arial" SIZE 8, 12
DEFINE FONT fntArial2 NAME "Arial" SIZE 8, 12 ITALIC
oFld:oFont = fntArial
Fld:oFont2= fntArial2
```



aEnable	Array de valores lógicos que nos define qué pestañas del folder están activas y a cuales no se puede acceder.
---------	---

oFld:aEnable :={.t.,.t.,.t.,.f.} //desactiva la cuarta pestaña



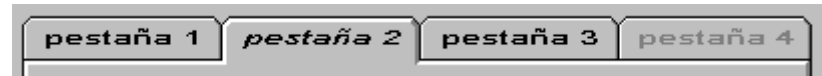
IAIWidth	Make all the tabs the same width
----------	----------------------------------

IWin95Look

Modifica la apariencia de las pestañas.
 Por defecto está en .f. oFld:IWin95Look=.f.



oFld:IWin95Look=.t.



Métodos

New	Constructor from source code
ReDefine	Constructor from Resource
AddItem	Adds an Item to the Folder.
Default	Default control initialization
DelItem	Deletes an Item from the Folder.
Destroy	Destroy the Folder object.
Display	Displays the control. Use :Refresh() to force repaint
GetHotPos	Returns the page index of a specified hotkey
GotFocus	Event response code when receiving focus
Initiate	Generic Control initialization method
LButtonDown	Action to be performed when clicking the mouse
Paint	Draws the control. This process occurs internally from Display.
Refresh	Repaints the folder on the screen
ReSize	Event response code when resizing the folder
SetOption	Changes the current active folder page
Update	Sends an Update message to all the contained Dialog objects.

GET (no memo)**Help**

Gets objects Class

Inherits from TControl

DATOS

oGet	Standard MsDos Get related object to use its buffer contents
nClrFocusText	Color for the get text when receiving focus
nClrFocusPane	Color for the get back when receiving focus
nPos	Position of the cursor (caret)
bMin	Minimum value for the spinner
bMax	Maximum value for the spinner
nClrDef	Default color
lReadOnly	if .t. get is readonly
lPassword	If .t. then a echo of asterisk is shown instead of the chars
hHeap	Handle of memory

METODOS

New	Constructor from source code
ReDefine	Constructor for resources
Assign	Assigns the buffer value to the GET related variable
cGenPrg	Generates the xBase source code of this control
Copy	Copy the contents of the buffer in the Clipboard
Create	Creates a GET control. Internally used
cText	Get/Set the text of the TGet Object.
cToChar	Generates a data template to build a Dialog
Cut	Cut selected text and copies it at the clipboard.
Default	Default control initialization
DispText	Displays the buffer contents
EditUpdate	Updates the variable and the display
GetDelSel	Returns the text selected and deleted
GetSel	Retrieves the selected text from the TGet editing buffer.
GetSelPos	Returns the selected text positions
GotFocus	Action to be performed when receiving the focus
HandleEvent	Generic events handler
HideSel	Hides the selection
Initiate	Generic control initialization
KeyChar	Action to be performed when pressing a key
KeyDown	Action to be performed when pressing a control key.

KeyUp	Action to be performed when releasing a keystroke.
LButtonDown	Action to be performed when clicking the mouse left button.
LButtonUp	Action to be performed when releasing the mouse left button
LostFocus	Action to be performed when losing the focus
IValid	Validates the get
MouseMove	Action to be performed when moving the mouse over
Move	Action to be performed when moving the mouse over the control
Paste	Paste the Clipboard contents into the editing buffer
RButtonDown	Action to be performed when clicking the right button of mouse.
Refresh	Repaints the control
Save	Updates the variable value from the buffer
SelectAll	Selects all the text.
SetPos	Changes the cursor (caret) position.
SetSel	Set a certain text as selected.
SelfFile	Selects a file and puts his contents on the get control
Spinner	Initializes the spinner
Inc (++)	Increments the get value (only for numeric and date values)
Dec (--)	Decrements the get value (only for numeric and date values)
Value	Return the value of the Get
Undo	Undoes the last modification.

GET multilinea (memos)

Help

GROUPS**Help**

Clase Tgroup

Group Dialog control Class

Inherits from TControl

DATOS

No data is added

METODOS

New	Constructor from source code.
ReDefine	Constructor from resource.
cToChar	Control Data to template convertor for creating DialogBoxes.
HandleEvent	Generic control Handle Event method.
Initiate	Generic Control initialization method.

ICONS

Help

Clase **Tico**

Icons class

Inherits from TControl

DATOS

cIcoFile	*.ICO filename if load it from disk.
cResName	Resource name if load it from a resource (DLL, RES, RC)
hIcon	Windows handle of the icon.

METODOS

New	Constructor from source code.
ReDefine	Constructor from resource.
Destroy	Destroy the object.
Display	Displays the control. Use :Refresh() to force repaint.
End	Ends control and make it disappear.
Init	Generic Control initialization method
Paint	Draws the control. This process occurs inside Display.
SetFile	Name of the icon file.
SetName	Set a new name for the icon.

IMAGEN

Help

Clase **TImage**

Image object from any of the following formats: JPG, JIF, GIF, BMP, DIB, RLE, TGA, PCX.

Inherits from TBitmap

DATOS

nProgress If 1 a load progress info is shown .

METODOS

New Constructor from source code

Define Constructor from resources

LoadImage(cResname, cBmpFile) Load image from resources or file

Progress(IMode) If IMode is TRUE the a load progress info is shown

[Note: Check TBitmap class for more info about his DATAs & METHODS](#)

TListBox

Clase para el control de **ListBox**.

Heredada de **TControl**.

VARIABLES DE INSTANCIA

altems	Array conteniendo todos los elementos en el ListBox
aBitmaps	Array conteniendo la lista de bitmaps a mostrar en el ListBox.
lOwnerDraw	Variable lógica indicando si el ListBox posee el estilo LBS_OWNERDRAWFIXED.
nBmpHeigth	Altura asignada a todos los bitmaps.
nBmpWidth	Anchura asignada a todos los bitmaps.
cFileSpec	Mascara conteniendo los archivos a mostrar y opcionalmente la ruta de los mismos.
bDrawItem	Codeblock a ser evaluado cada vez que refrescamos el Listbox o realizamos un scroll. Este codeblock retorna un número, correspondiente a la posición dentro del array aBitmaps del elemento a mostrar. Como parámetro bDrawItem cuenta con la variable nItem , correspondiente a la posición del array altems que estamos evaluando

METODOS

New()	Constructor from source code
ReDefine()	Constructor for resources
Add(cItem,nPos)	Añade el nuevo elemento cItem a la lista en la posición nPos+1 . Para añadir en la primera posición, nPos debe ser 0. So no se indica nPos, se añade al final. Si no es MultiSel, el elemento añadido, pasará a ser el seleccionado. Si existen bitmap, estos no son insertados mientras que los textos si. Esto puede provocar un desfase entre dibujos y textos. No confundir con Insert()
aSelections()	En un MultiSel, retorna un array con los textos de los elementos seleccionados. No funciona bien si hay bitmaps.
CGenPrn()	Generates the xBase source code of the control
Change	Changed event handler.

cToChar	Generates a data template to build a DialogBox
Default	Default Listbox initialization
Del(nPos)	Borra el elemento indicado en nPos . Si no se indica nPos, se borrará el elemento seleccionado. En MultiSel, borrará el último seleccionado.
Destroy()	Destruye el objeto y los bitmaps asociados
DrawItem	DrawItem event handler.
FillMeasure	FillMeasure event handler.
GetItem(nItem)	Retorna el contenido del elemento indicado en nItem . Si tiene bitmaps asociados, no funciona bien.
GetPos()	Retorna el índice dentro de altems correspondiente al elemento seleccionado. En un ListBox MultiSel, retorna el del elemento seleccionado en último lugar.
GetSel	Index number of the selected item. Zero if no item selected
GetSelCount()	Retorna el número de elementos seleccionados en un MultiSel.
GetSelText()	Retorna el índice dentro de altems correspondiente al elemento seleccionado. No con MultiSel.
GetSelItems()	Retorna un array con los elementos actualmente seleccionados en un MultiSel.
GoBottom()	Selecciona el último elemento de la lista. No con MultiSel.
GoTop()	Selecciona el primer elemento de la lista. No con MultiSel.
Initiate	Generic control initialization
Insert(cItem,nPos)	Similar al método Add(), pero en este caso insertando el dato, esto es, oLbx:Insert("edf") situará el nuevo elemento en la primera posición de la lista, o sobre el elemento seleccionado en caso de que el ListBox no sea MultiSel. En este caso, al contrario que con Add(), el nuevo elemento no pasa a ser el seleccionado. Si hay bitmaps no quedarán bien situados.
IsMultipleSel()	Retorna una variable lógica si ListBox es de tipo MultiSel o no.
Len()	Retorna el número de elementos totales del ListBox.
LostFocus	Behavior when losing the focus
LButonUp	Behavior when unclicking the mouse left button
Modify(cItem,nPos)	Modifica el texto antiguo con el nuevo cItem del elemento nPos .
MouseMove	Behavior when moving the mouse over the control
Reset()	Borra todos los elementos, dando como resultado un ListBox vacío.
Select(nItem)	Selecciona el elementos situado en nItem . No con MultiSel.
Set(cItem)	Selecciona el elemento con el texto igual a cItem . No distingue entre mayúsculas y minúsculas. No funciona con MultiSel

- SetBitmaps(acBitmaps)** Reemplaza el actual array de bitmaps por el nuevo **acBitmaps**. Para que tenga efecto hay que hacer un **oLbx:refresh()**.
- SetItems(altems)** Reemplaza los elementos actuales, por los nuevos indicados en **altems**.
- SetSel(nItem, lOnOff)** En un ListBox MultiSel, activa o desactiva la selección del elemento situado en la posición **nItem**. **lOnOff** es una variable lógica que nos indica si el elemento será marcado (.t.) o desmarcado (.f.). Por defecto **lOnOff = .t.**
- SetSelItems(anItems)** Selecciona los elementos contenidos en el array **anItem** en un MultiSel. (p.e. {1,3,7,9})
- SwapDown()** Intercambia el elemento seleccionado con el inmediatamente siguiente.
- SwapUp()** Intercambia el elemento seleccionado con el inmediatamente anterior.
- VScroll Vertical scroll events handler

TMenu**Help**

Menu Class

DATOS

hMenu	The Windows handle of the Menu
altems	An array of MenuItem
oWnd	A reference to the container Window of the Menu
ISysMenu	If it is a System menu
IPopup	If it is a popup menu
nHelpId	The help topic identifier of this Menu
cVarName	A variable name to identifier this object (visual editor)

METODOS

New	Creating a new Menu
NewSys	Redefining a SysMenu
Redefine	Creates a menu from resource
Activate	PopupMenu activating method
Add	Adding a new menuitem to the Menu
AddEdit	Adds standard editing menu options
AddHelp	Adds standard help menu options
AddMdi	Adds standard MDI menu options
AddFile	Adds standard File menu options
Command	Processes the user selection command
Delltems	Invoked when destroying the menu. Destroys each menuitem
Destroy	Destroys the menu
Disable	Disables the entire menu
End	Ends the menu
GetMenuItem	Retrieves a reference to a MenuItem Object given its Id

GetPopup	Retrieves a reference to a contained Popup given its position
GetSubMenu(hPopUp)	Retrieves a reference to a submenu
HelpTopic	Invoked when pressing enter over this menu
Hilite	Hilites a specific menuitem
Initiate	Called everytime the menu is going to be shown
Insert	Inserts a new menuitem at a specified position
LastItem	A reference to last processed MenuItem Object
Load	Changes some menu properties based on a binary info
Refresh	Repaints the menu
Release	Destroy the Menu Object
Reset	Reset to the original System Menu of a Window
Save	Generates a binary representation of the menu
UnHilite	DeHilites a specific menuitem
Disable	Disables de menu
Enable	Enables de menu

TMenuItem**Help**

TMenuItem class

DATOS

cPrompt	The label of the MenuItem
cMsg	The text to be shown on the MsgBar of the main Window
nId	A numeric identifier for this menuitem
nHelpId	A numeric help topic identifier used to select invoked help
bAction	The action to be performed when selecting the MenuItem
bWhen	A codeblock to be evaluated to enable/disable the menuitem
lChecked	Logical value if the menuitem is checked
lActive	Logical value if the menuitem is enabled
lHelp	If this is a menuitem help type (right placed)
oMenu	A reference of its container Menu object
hBitmap	A handle to a Bitmap if defined
lBreak	Separator
cCaption	Menu item caption

METODOS

New	Constructor from source code
ReDefine	Constructor from resource
Destroy	Called when the menuitem is destroyed.
Disable	Disable the item.
Enable	Enable the item.
End	Ends control and make it disappear.
EveCount	Number of total events for this class (used by the Object Inspector).
Event	Retrieve the event from the event array.

Load	Set the properties of the menuitem based on a supplied binary information.
PropCount	Number of total property.
Property	Retrieves the property from the property array.
Save	Generates a binary representation of this object.
SetCheck	Enables / Disables check.
SetPrompt	Changes the associated menuitem prompt.
HelpTopic	Method invoked when pressing F1 when this menuitem is selected.

TMsgBar

Help

Inherits from TWindow

DATOS

cMsgDef	The default message to show
ICentered	If the message should look centered at screen
lInset	If the message bar should look inset
oTimer	Timer for MsgItems repaint
altem	Array of MsgItems
oKeyNum	MsgItem for Num Lock
oKeyCaps	MsgItem for Caps Lock
oKeyIns	MsgItem for Ins state
oClock	MsgItem for Clock
oDate	Msgitem for Date
ICheckRes	if .t. then memory and resources consumption is show in the message bar

METODOS

New	Constructor from source code
AddItem(oltem)	Adds an Item to the MsgBar.
InsItem(oltem, nAt)	Inserts an Item to the MsgBar at a specific location.
Adjust	Adjust the MsgBar dimensions to its container window
ClockOff	Turn off the clock.
ClockOn	Turn on the clock.
DateOff	Turn off the date.
DateOn	Turn on the date.
Default	Default initialization
Delltem	Deletes an Item from the MsgBar.

Destroy	Destroys the MsgBox. Automatically called.
GetItem	Retrieves the number of items of the MsgBox
Height	Retrieves the MsgBox Height.
KeybOff	Turn the keyboard off.
KeybOn	Turn the Keyboard on.
LButtonDown	Action to be performed when clicking the mouse
LDbIClick	Action to be performed when double-clicking the mouse
Paint	Draws the control. This process occurs inside Display.
Refresh	Repaints the control
SetMsg	Set the text of the MsgBox

TMetaFile

Help

Windows WMF files management

Inherits from TControl

DATOS

hMeta	Handle of the MetaFile.
oPen	Reference to a pen object.
nWidth	Dimensions
nHeight	Dimensions
nXorig	Origin in Por axis.
nYorig	Origin in Y axis.
lZoom	if Zoom is enabled.
lShadow	If Shadow

METODOS

New	Constructor from source code
ReDefine	Constructor from resource
Display	Displays the control. Use :Refresh() to force repaint
End	Ends control and make it disappear
Hide	Hides the control.
Paint	Draws the control. This process occurs inside Display.
SetFile	Name of the metafile.
SetOrg	Set the X and Y origin.
Shadow	Activate / Deactivate Shadows.
ZoomIn	Zoom in the image.
ZoomOut	Zoom out the image.

TMeter

Help

Inherits TControl

DATOS

nTotal	Total amount to be represented by the Meter
nClrBar	RGB color to use for the filled background of the meter
nClrBText	RGB color to use for the filled text of the meter
cText	The text to show on the meter
lPercentage	If the % sign is to be painted

METODOS

New	Constructor from source code.
ReDefine	Constructor from resources.
Default	Default meter initialization.
HandleEvent	Events dispatcher manager.
Initiate	Control initialization.
Paint	Paints the Meter.
Set	Changes the value represented and the value of the variable.
SetTotal	Sets the maximum value of the meter

TMsgItem

Help

Bar message item objects

DATOS

oMsgBar	Reference to its MsgBar container object.
cMsg	shown text.
nWidth	width of the MsgItem.
nClrText	Text color as a RGB number.
nClrPane	Pane color as a RGB number.
nClrDisabled	Disabled color as a RGB number.
bMsg	Optional codeblock to evaluate to generate the cMsg text.
bAction	Action to be preformed when clicking on it.
oFont	Font object used.
lActive	.T. if the msgitem is active
hBmpPal1	Handle of bitmap for active msgitem
hBmpPal2	Handle of bitmap for non-active msgitem
cToolTip	ToolTip

METODOS

New	Constructor from source code
Click	Action to be performed when clicking the mouse.
Refresh	Repaints the item
SetFont	Set the font for the MsgItem.
SetText	Set the message for the MsgItem.
Show	Makes the item visible
IsOver(nRow, nCol)	.t. if the coors are over the MsgItem
nLeft	Left coordinate of the msgitem

TPages

Help

Multiple DialogBox Pages management

DATOS

nOption	Current selected page
aDialogs	Array of NonModal DialogBox objects

METODOS

New	Method Constructor from source code
Redefine	Method Constructor from resources
Initiate	Generic control initialization
Default	Control default initialization. Called from Init
Destroy	Generic destructor method
SetOption	Sets a different selected page
GoPrev	Selects the previous page to the current one selected
GoNext	Selects the next page to the current one selected

TPen**Help**

Pen management as objects

DATOS

hPen	The handle of the pen
nStyle	The Pen style
nColor	Specifies the color of the Pen as an RGB value
nWidth	Specifies the width, in logical units, of the Pen

METODOS

New	Constructor Method
Release	Destroy Method
End	Destroy Method. Use this method to destroy a Pen.

TRadio**Help**

Inherits from TControl

DATOS

lChecked	If the radio control is currently selected
oRadMenu	Its related container TRadMenu Object
nPos	Relative position within respect to the other Menu Radios buttons

METODOS

New	Constructor from source code.
ReDefine	Constructor from resource.
Click	Mouse left button event handler.
cToChar	Generates a data template to build a Dialog
Default	Default control initialization.
Initiate	Generic control initialization.
KeyDown	Action to be performed when pressing a key.
lIsChecked	Retrieves its current logical value
MouseMove	Behavior when moving the mouse over it.
Refresh	Repaints the control
SetCheck	Changes its select/deselected state.

SAY

Help

TPrinter

Help

DATOS

HDC	Handle of the Device Context of the Printer
hDCOut	Handle of the current metafile in use if file output specified
aMeta	Arrays of metafiles names generated in disk
cDevice	The name of the output device
cDocument	The name of the spooler document
cDriver	The name of the printer driver in use
cModel	Alternative printer to use
cPort	The name of the port used
cDir	The path for the metafiles storage. By default TEMP
nPage	The page currently printed
nXOffset	Coordinates of the virtual page origin
nYOffset	Coordinates of the virtual page origin
nPad	Default pad for all the Say instructions: <pre>#define PAD_LEFT 0 #define PAD_RIGHT 1 #define PAD_CENTER 2</pre>
IModified	If the printer device handle should be initialized again
IMeta	If the output has to go to metafiles in disk
IStarted	If the printing already started
IPrevModal	If the preview window should be executed in modal mode
oFont	Default font to use for printing

METODOS

New	Constructor from source code
------------	------------------------------

Syntax:

TPrinter():New(<cDocument>, <IUser>, <IMeta>, <cModel>, <IModal>)

Parameters:

<cDocument>	The name to assign to cDocument DATA, which it is the name of the spooler document to generate
<IUser>	To let the user interactively select the printer to use.
<IMeta>	To PREVIEW the printout work.

<cModel> To select a different printer from the default one. It is the name of any printer installed in the system, (FiveWin PrintBegin() function generates this name from a partial name or from an index name). See Printer xBase Commands for a sample.

<lModal> If the preview window should be executed in modal mode

End	Destroys the current printer object
------------	-------------------------------------

Syntax: **<oPrn>:End()**

This method should be used to release the current printer object and all its associated handles. It is the latest operation to perform with a printer object.

Warning: The printer object does not destroy any related font you may have created. You must destroy the fonts you create for the printer:

DEFINE FONT oFont NAME ... SIZE ... OF oPrn oFont:End() // Destroy the font

EndPage	Ends the actual printed page.
----------------	-------------------------------

Syntax: **<oPrn>:EndPage()**

This method completes the current page. Once you do it, there is no way to perform any further printing on this same page.

FillRect	Fills a rectangle with a specified brush object.
-----------------	--

Syntax: **<oPrn>:FillRect(<aRect>, <oBrush>)**

Parameter:

<aRect> It is an array with the rectangle dimensions: { nTop, nLeft, nBottom, nRight }

<oBrush> It is the brush object to use for filling the rectangle area.
To create the brush use:
DEFINE BRUSH oBrush ...

Box	Draws a box.
------------	--------------

Syntax: **<oPrn>:Box(<nTop>, <nLeft>, <nBottom>, <nRight>, <oPen>)**

Parameters:

<nTop> The coordinates of the box to draw. By default they ...

<nBottom> ...are expressed in pixels, unless a different ...

<nLeft> ...SetMapMode() mode is selected.

<nRight>

<oPen> It is an optional pen object to use for painting the box. To create it use:

DEFINE PEN oPen STYLE ... COLOR ... WIDTH ...

Returns: A logical value indicating if the operation was successfully performed.

Cmtr2Pix

Converts a centimeters coordinates into pixels coordinates

Syntax: **<oPrn>:Cmtr2Pix(<@nRow>, <@nCol>) || <aPoint>**

Parameters:

<nRow>

The centimeters coordinates as numeric values.:

<nCol>

Returns: <aPoint>

An array with the new point in pixel coordinates: { nTop, nLeft }

DraftMode

Select the draft mode for speed up the printing

Syntax: **<oPrn>:DraftMode(<lOnOff>) || <nRetCode>**

Parameters: <lOnOff>

A logical value to turn on or off the draft mode.

Returns:

A numeric value indicating if the operation was successfully performed:

Positive

The operation was successful

Negative

It was not performed

GetOrientation

Retrieves the printer orientation.

Syntax: **<oPrn>:GetOrientation() || <nOrientation>**

Parameters:

None

Returns:

A numeric value indicating the orientation type:

1 Portrait

2 LandScape

GetPhySize

Retrieves the paper physical dimensions

Syntax: **<oPrn>:GetPhySize() || <aSize>**

Parameters:

None

Returns:

An array with the dimensions of the paper: {nHeight ,Widt}

GetTextHeight

Retrieves text Height for a specific font object

Syntax: **<oPrn>:GetTextHeight(<cText>, <oFont>) || <nHeight>**

Parameters: <cText>

The text to retrieve its height.

<oFont>

A font object. By default it uses the default font oFont of the printer object.

To create it use: **DEFINE FONT oFont NAME ... SIZE**

...

Returns:

The height for that specified font. In fact, this height it is not depending on the text, but just on the font object.

GetTextWidth

Retrieves text Width for a specific font object

Syntax: **oPrn>:GetTextWidth(<cText>, <oFont>) || <nWidth>**

Parameters:

<cText>

The text to retrieve its width.

<oFont>

A font object. By default it uses the default font oFont of the printer object.

To create it use: **DEFINE FONT oFont NAME ... SIZE ...**

Returns:

he width for that specified font.

ImportWMF

Imports a Windows metafile to be used as template

Syntax:

<oPrn>:ImportWMF(<cFile>, <IPlaceable>)

Parameters:

<cFile>

The name of the Windows metafile WMF file to use as a template for printing.

<IPlaceable>

If it is a placeable metafile. By default is .t..

Comments:

This new powerful FiveWin feature lets you use i.e.: Corel Draw to design a form, later you save it as a Windows metafile WMF file and use it from the FiveWin printer object. See SAMPLES\Corel.prg All the printed pages will show the metafile as their background, letting you place on it certain information. It is a very quick and easy way to design forms.

ImportRaw

Imports a RAW file

Syntax:

<oPrn>:ImportRAW(<cFile>)

Parameters: <cFile>

The name of the RAW file to import

Inch2Pix

Converts a inches coordinates into pixels coordinates

Syntax:

<oPrn>:Inch2Pix(<nRow>, <nCol>) || <aPoint>

Parameters:

<nRow>

The inches coordinates as numeric values.

<nCol>

Returns:

<aPoint>

An array with the new point in pixel coordinates: { nTop, nLeft }

Line

Draws a line.

Syntax:

<oPrn>:Line(<nTop>, <nLeft>, <nBottom>, <nRight>, <oPen>) || <ISuccess>

Parameters:

<nTop>

The coordinates of the line to draw. By default they...

<nLeft>

...are expressed in pixels, unless a different...

<nBottom>

...SetMapMode() mode is selected

<nRight>

<oPen>

It is an optional pen object to use for painting the box

To create it use:

DEFINE PEN oPen STYLE ... COLOR ... WIDTH ...

Returns:

A logical value indicating if the operation was successfully performed.

nHorzRes

Retrieves the horizontal resolution.

Syntax: **<oPrn>:nHorzRes() || <nHorzRes>**

Parameters: None

Returns: A numeric value indicating the horizontal resolution of the printer expressed in pixels.

nHorzSize

Retrieves the width of the physical dimensions

Syntax: **<oPrn>:nHorzSize() || <nHorzSize>**

Parameters: None

Returns: A numeric value indicating the physical dimensions of the printer page expressed in millimeters.

nLogPixelX

Retrieves the number of pixels per logical inch

Syntax: **<oPrn>:nLogPixelX() || <nPixels>**

Parameters: None

Returns: A numeric value indicating the number of pixels per logical inch along the display width.

nLogPixelY

Retrieves the number of pixels per logical inch

Syntax: **<oPrn>:nLogPixelY() || <nPixels>**

Parameters: None

Returns: A numeric value indicating the number of pixels per logical inch along the display height.

nVertRes

Retrieves the height of the printer page in raster lines

Syntax: **<oPrn>:nVertRes() || <nVertRes>**

Parameters: None

Returns: A numeric value indicating the vertical resolution of the printer in raster lines.

nVertSize

Retrieves the vertical dimensions in millimeters

Syntax: **<oPrn>:nVertSize() || <nVertSize>**

Parameters: None

Returns: A numeric value indicating the vertical dimensions of the printer page in millimeters.

Pix2Inch

Converts pixels coordinates into inches coordinates

Syntax: **<oPrn>:Pix2Inch(<nRow>, <nCol>) || <aPoint>**
 Parameters: **<nRow>** The pixels coordinates as numeric values.
<nCol>
 Returns: **<aPoint>** An array with the new point in inches coordinates: {nTop, nLeft}

Pix2Mmtr	Change from pixels to millimeters coordinates
-----------------	---

Syntax: **<oPrn>:Pix2Mmtr(<nRow>, <nCol>)**
 Parameters: **<nRow>** The pixels coordinates as numeric values.
<nCol>
 Returns: **<aPoint>** An array with the new point in millimeters coordinates: {nTop,nLeft}

Preview	Make a preview of the print work
----------------	----------------------------------

Syntax: **<oPrn>:Preview()**
 Parameters: None
 Returns: NIL
 Comments: This method invokes the preview mode, which displays a window with a buttonbar, a visual page/s, to interactively let the user preview the print work and optionally to select what pages to finally send to the printer.

Rebuild	Rebuilds all the associated handles and files of the printer
----------------	--

Syntax: **<oPrn>:Rebuild()**
 Parameters: None
 Returns: Nothing
 Comments: Some printer methods require to rebuild the printer object to properly generate new handle values and also to reset the associated preview metafiles files array. FiveWin Printer object automatically invokes this method when it is required.

ResetDC	Updates the associated device handle of the printer object
----------------	--

Syntax: **<oPrn>:ResetDC()**
 Parameters: None
 Returns: The handle of the original device context with updated values.
 Comments: FiveWin Printer object automatically uses this method when an update of its device handle is required.

Say	Prints a text at a certain coordinates.
------------	---

Syntax: **<oPrn>:Say(nRow,nCol,cText,oFont,nWidth,nClrText,nBkMode,nPad)**

Parameters:

<nRow>,<nCol>	The coordinates where to display a text. They are graphical coordinates, and may be pixels, inches, millimeters, twips, etc... accordingly to the current SetMapMode() active on the device. Pixels are used by default.
<cText>	The text to display.
<oFont>	An optional font object to use for printing.
<nWidth>	The total width of the text. By default it is the width of the text.
<nClrText>	An optional color to use for printing the text.
<nBkMode>	A numeric value indicating how to mix the text with the background: 1 Transparent 2 Opaque
<nPad>	A numeric value indicating how align the text to print: (PAD_LEFT is used by default) #define PAD_LEFT 0 #define PAD_RIGHT 1 #define PAD_CENTER 2

CmSay	Exactly as Say method but the coordinates are in centimetres
--------------	--

InchSay	Exactly as Say method but the coordinates are in inches
----------------	---

SayBitmap	Prints a Bitmap.
------------------	------------------

Syntax: **<oPrn>:SayBitmap(nRow,nCol,xBitmap,nWidth,nHeight,nRaster) || nil**

Parameters:

<nRow> <nCol>	The start coordinates for the bitmap. By default they are expressed in pixels, unless a different SetMapMode() is used.
<xBitmap>	A string with the name of the BMP file to print, if it exists, or the name of the bitmap resource or a numeric value with the resource identifier if the bitmap should be loaded from resources.
<nHeight> <nWidth>	The dimensions of the image to print original dimensions of the bitmap are used.. By default the operation to perform. Some of the possible values are: #define MERGEPAINT 12255782 // 0xBB0226 #define SRCAND 8913094
<nRaster>	An optional numeric value indicating the kind of raster(to review all possible operations with bitmaps we strongly recommend a Graphics management Windows book)

SayImage	Prints any kind of Image.
-----------------	---------------------------

Syntax: **<oPrn>:SayImage(nRow,nCol,oImage,nWidth,nHeight,nRaster) || nil**

Parameters:

<code><nRow> <nCol></code>	The start coordinates for the bitmap. By default they are expressed in pixels, unless a different <code>SetMapMode()</code> is used.
<code><xBitmap></code>	A string with the name of the BMP file to print, if it exists, or the name of the bitmap resource or a numeric value with the resource identifier if the bitmap should be loaded from resources.
<code><nHeight> <nWidth></code>	The dimensions of the image to print original dimensions of the bitmap are used.. By default the operation to perform. Some of the possible values are: <pre>#define MERGEPAINT 12255782 // 0xBB0226 #define SRCAND 8913094</pre>
<code><nRaster></code>	An optional numeric value indicating the kind of raster(to review all possible operations with bitmaps we strongly recommend a Graphics management Windows book)

SetAnisotropicMode	Set the anisotropic mode.
---------------------------	---------------------------

Syntax: **<oPrn>:SetAnisotropicMode() || <nPrevMapMode>**

Parameters: None

Returns: A numeric value indicating the previous map mode if this operation was successful.

Comments:

Logical units are converted to arbitrary units with arbitrarily scaled axes. Setting the mapping mode to `MM_ANISOTROPIC` does not change the current window or viewport settings. To change the units, orientation, and scaling, an application should use the **SetWindowExt()** and **SetViewportExt()** functions.

SetCopies	Set the number of copies to print.
------------------	------------------------------------

Syntax: **<oPrn>:SetCopies()**

Parameters: None

Returns: Nothing

SetHiInchMode	Set the high inch mode.
----------------------	-------------------------

Syntax: **<oPrn>:SetHiInchMode() || <nPrevMapMode>**

Parameters: None

Returns: A numeric value indicating the previous map mode if this operation was successful.

Comments:

Each logical unit is converted to 0.001 inch. Positive x is to the right; positive y is up.

SetHiMetricMode Set the high metric mode.
--

Syntax: **<oPrn>:SetHiMetricMode() \ \ <nPrevMapMode>**

Parameters: None

Returns: A numeric value indicating the previous map mode if this operation was successful.

Comments:

Each logical unit is converted to 0.01 millimeter. Positive x is to the right; positive y is up.

SetIsotropicMode Set the Isotropic mode.

Syntax: **<oPrn>:SetIsotropicMode() \ \ <nPrevMapMode>**

Parameters: None

Returns: A numeric value indicating the previous map mode if this operation was successful.

Comments:

Logical units are converted to arbitrary units with equally scaled axes; that is, one unit along the x-axis is equal to one unit along the y-axis. The SetWindowExt and SetViewportExt functions must be used to specify the desired units and the orientation of the axes. GDI makes adjustments as necessary to ensure that the x and y units remain the same size.

SetLandscape Set the printer orientation to Landscape
--

Syntax: **<oPrn>:SetLandscape()**

Parameters: None

Returns: Nothing

SetLoInchMode Set the low inch mode.

Syntax: **<oPrn>:SetLoInchMode() \ \ <nPrevMapMode>**

Parameters: None

Returns: A numeric value indicating the previous map mode if this operation was successful.

Comments:

Each logical unit is converted to 0.01 inch. Positive x is to the right; positive y is up.

SetLoMetricMode Set the low Metric mode.

Syntax: **<oPrn>:SetLoMetricMode() \ \ <nPrevMapMode>**

Parameters: None

Returns: A numeric value indicating the previous map mode if this operation was successful.

Comments:

Each logical unit is converted to 0.1 millimeter. Positive x is to the right; positive y is up.

SetPage

Specifies the size of the paper to print on.

Syntax: **<oPrn>:SetPage(<nPageType>) || nil**

Parameters:

<nPageType>

Specifies the size of the paper to print on. This method is superseded by SetSize(<nWidth>, <nHeight>) method.

- 1 Letter 8 1/2 x 11 in
- 2 Letter Small 8 1/2 x 11 in
- 3 Tabloid 11 x 17 in
- 4 Ledger 17 x 11 in
- 5 Legal 8 1/2 x 14 in
- 6 Statement 5 1/2 x 8 1/2 in
- 7 Executive 7 1/4 x 10 1/2 in
- 8 A3 297 x 420 mm
- 9 A4 210 x 297 mm
- 10 A4 Small 210 x 297 mm
- 11 A5 148 x 210 mm
- 12 B4 250 x 354
- 13 B5 182 x 257 mm
- 14 Folio 8 1/2 x 13 in
- 15 Quarto 215 x 275
- 16 10x14 in
- 17 11x17 in
- 18 Note 8 1/2 x 11 in
- 19 Envelope #9 3 7/8 x 8 7/8
- 20 Envelope #10 4 1/8 x 9 1/2
- 21 Envelope #11 4 1/2 x 10 3/8
- 22 Envelope #12 4 1/2 x 10 3/8
- 23 Envelope #14 5 x 11 1/2
- 24 C size sheet
- 25 D size sheet
- 26 E size sheet
- 27 Envelope DL 110 x 220mm
- 28 Envelope C5 162 x 229 mm
- 29 Envelope C3 324 x 458 mm
- 30 Envelope C4 229 x 324 mm
- 31 Envelope C6 114 x 162 mm
- 32 Envelope C65 114 x 229 mm
- 33 Envelope B4 250 x 353 mm
- 34 Envelope B5 176 x 250 mm
- 35 Envelope B6 176 x 125 mm
- 36 Envelope 110 x 230 mm
- 37 Envelope Monarch 3.875 x 7.5 in

38	6 3/4 Envelope 3 5/8 por 6 1/2 in
39	US Std Fanfold 14 7/8 x 11 in
40	German Std Fanfold 8 1/2 x 12 in
41	German Legal Fanfold 8 1/2 x 13 in

```
#define DMPAPER_LAST MPAPER_FANFOLD_LGL_GERMAN
#define DMPAPER_USER 256
```

Returns: NIL.

SetPixelMode	Set the Pixel mode.
---------------------	---------------------

Syntax: **<oPrn>:SetPixelMode() || <nPrevMapMode>**

Parameters: None

Returns: A numeric value indicating the previous map mode if this operation was successful

Comments:

Each logical unit is converted to one device pixel. Positive x is to the right; positive y is down.

SetPortrait	Set the printer orientation to Portrait
--------------------	---

Syntax: **<oPrn>:SetPortrait()**

Parameters: None

Returns: Nothing

SetPos	Set a new position on the page for next printing
---------------	--

Syntax: **<oPrn>:SetPos(<nRow>, <nCol>) || <nLPrevPos>**

Parameters:

<nRow> <nCol> The coordinates of the new position. By default they are expressed in pixels, unless another mode (**SetMapMode()**) is being used.

Returns: A numeric long value containing the previous device selected position:

```
<nPrevRow> = nLoWord( <nLPrevPos> )
<nPrevCol> = nHiWord( <nLPrevPos> )
```

SetSize	Selects a different page dimensions
----------------	-------------------------------------

Syntax: **<oPrn>:SetSize(<nWidth>, <nHeight>) || nil**

Parameters:

<nWidth> <nHeight> The new dimensions for the printer page in tenths of millimeters.

Returns: Nothing

SetTwipsMode	Each logical unit is converted to 1/20 of a point
---------------------	---

Syntax: **<oPrn>:SetTwipsMode() || <nPrevMapMode>**

Parameters: None

Returns: A numeric value indicating the previous map mode if this operation was successful

Comments:

Each logical unit is converted to 1/20 of a point. (Because a point is 1/72 inch, a twip is 1/1440 inch). Positive x is to the right; positive y is up.

SetFont	Sets and retrieves the default printer font object
----------------	--

Syntax: **<oPrn>:SetFont(<oNewFont>) || <oFont>**

Parameters: **<oNewFont>** Is the new font object to set as the printer default one.
To create it use: **DEFINE FONT oFont NAME ... SIZE ...**

Returns: **<oFont>** The default font object associated to the printer.

Setup	Standard Setup DialogBox for printing
--------------	---------------------------------------

Syntax: **<oPrn>:Setup() || nil**

Parameters: None

Returns: Nothing

SetViewportExt	Sets the X and Y dimensions of the viewport
-----------------------	---

Syntax: **<oPrn>:SetViewportExt(<nXExtent>,<nYExtent>) ||--> <nPrevViewportExt>**

Parameters:

<nXExtent> ...
<nYExtent>

Specifies the new X-extent and Y-extent, in device units of the viewport.

Returns: The return value is the previous viewport extents, in device units, if the function is successful. The low-order word contains the previous x-extent; the high-order word contains the previous y-extent. Otherwise, the return value is zero:

<nPrevXExtent> = nLoWord(<nPrevViewportExt>)
<nPrevYExtent> = nHiWord(<nPrevViewportExt>)

SetWindowExt	Sets the X and Y extents of the printer device
---------------------	--

Syntax: **<oPrn>:SetWindowExt(<nWidth>, <nHeight>) ||--> <nPrevWindowExt>**

Parameters: Specifies the new X-extent and Y-extent, in logical units, of the device context of the printer.

Returns: The return value is the previous device extension, in logical units, if the function is successful. The low-order word contains the previous x-extent; the high-order word contains the previous y-extent. Otherwise, the return value is zero:

`<nPrevXExtent> = nLoWord( <nPrevWindowExt> )`

`<nPrevYExtent> = nHiWord( <nPrevWindowExt> )`

StartPage	Begins a new page
------------------	-------------------

Syntax: `<oPrn>:StartPage()`

StartPage starts a new page. From that moment on you may perform any printing action on the printer object, like :Say(), :Box(), :Line(), etc... You may print at any coordinate of the page. Once you have completed the page, an :EndPage() should be done.

SetBin	Changes default bin printer
---------------	-----------------------------

Syntax: `<oPrn>:SetBin(nBin)`

```
#define DMBIN_FIRST           0
#define DMBIN_UPPER           1
#define DMBIN_ONLYONE         1
#define DMBIN_LOWER           2
#define DMBIN_MIDDLE           3
#define DMBIN_MANUAL           4
#define DMBIN_ENVELOPE         5
#define DMBIN_ENVMANUAL        6
#define DMBIN_AUTO             7
#define DMBIN_TRACTOR           8
#define DMBIN_SMALLFMT         9
#define DMBIN_LARGEFORMAT      10
#define DMBIN_LARGECAPACITY    11
#define DMBIN_CASSETTE          14
#define DMBIN_LAST              DMBIN_CASSETTE
```

GetModel	Retrieves the printers model name
-----------------	-----------------------------------

CharSay	Simulates MSDOS printing
----------------	--------------------------

Syntax: `<oPrn>:CharSay(nRow, nCol, cText)`

CharWith	Retrieves MSDOS printing char width
-----------------	-------------------------------------

CharHeight

Retrieves MSDOS printing char height

TScrollBar

Help

Inherits from TControl

DATOS

IVertical	If it is a vertical ScrollBar.
lReDraw	If the scrollbar must update automatically its position
lIsChild	If belongs to a Window with ScrollBars style
nMin, nMax	The numeric range to be represented by the Scrollbar
nPgStep	The increment of value when PgUp or PgDn on the ScrollBar
bGoUp	A codeblock to eval when going up
bGoDown	A codeblock to eval when going down
bGoTop	A codeblock to eval when top is selected
GoBottom	A codeblock to eval when bottom is selected
bPageUp	A codeblock to eval when PageUp is selected
bPageDown	A codeblock to eval when PageDown is selected
bPos	A codeblock to eval when the Thumb is moved to a new position
bTrack	A codeblock to eval when the thumb track is moved

METODOS

New	Constructor from source code.
WinNew	Constructor for non-true scrollbar.
ReDefine	Constructor from resource.
cToChar	Generates a data template to build a Dialog.
GetPos	Retrieves the current value represented by the Scrollbar.
GetRange	Returns an array of two elements with the Min and the Max values
GoBottom	Evaluates the bGoBottom block and updates thumb position
GoDown	Evaluates the bGoDown block and updates thumb position

GoTop	Evaluates the bGoTop block and updates thumb position
GoUp	Evaluates the bGoUp block and updates thumb position
HandleEvent	Redefined events manager
Initiate	Generic initialization
MouseMove	Specific MouseMove behavior
PageDown	Evaluates the bPageDown block and updates thumb posit
PageUp	Evaluates the bPageUp block and updates thumb posit
SetPos	Changes the position of the thumb
SetRange	Changes the represented range of the ScrollBar
ThumbPos	Changes the position of the Thumb
ThumbTrack	Processes the Thumb track movement

Tabs**Help**

Tab objects

DATOS

aPrompts	Labels of tab.
aSizes	Sizes of tab.
bAction	CodeBlock to evaluate when changing the tab label
nOption	Current tab label

METODOS

New	Constructor from source code
ReDefine	Constructor from resources
AddItem	Add an item to the folder.
Default	Default initialization
DelItem	Delete an item from the folder.
Display	Displays the control. Use :Refresh() to force repaint
GetHotPos	Returns the tabs option that matches a specified HotKey char
Init	Generic Control initialization method
LButtonDown	Action to be performed when clicking the mouse
Paint	Draws the control. Internally used by Display method
SetOption	Select the active tab option.
SetTabs	Set the array of tabs and option. Default "One", "Two" and "Three".
SysCommand	Behavior when pressing Alt+... combinations
EditPrompts	Edits the tab labels

TIMER

Help

UNTIL**Help**

Windows timers

DATOS

bAction	The action to be performed on each interval
lActive	If the timer is currently active
nId	The API identifier of the timer
nInterval	The interval of each action in 1/1000 secs
hWndOwner	The handle of the window container of the timer

METODOS

New	Constructor method TTimer.New(nInterval, bAction, oWnd)
Activate	Activates the timer TTimer.Activate() \ \ nil
DeActivate	Deactivates the timer TTimer.DeActivate() \ \ nil
Release	Destroys the timer TTimer.Release() \ \ nil

TVbControl

Help

CLASS TVbControl FROM TControl

DATOS

hCtl	The handle of the VBX control (internal API value)
aEvents	An array with all the defined events and its related actions.

METODOS

New Creates a new VBX control from source code
 New(nRow, nCol, nWidth, nHeight, oWnd, cVbxFile, cVbxClass,; aEvents)
 CONSTRUCTOR

ReDefine Redefine a VBX to be used from resources
 ReDefine(nId, oWnd, nClrFore, nClrBack, aEvents) CONSTRUCTOR

Initiate Generic initialization of a VBX
 Init(hDlg)

GetPropName Retrieves the name of a VBX property by its order
 GetPropName(nProp) INLINE VBXGetPrpName(::hCtl, nProp - 1)
 aGetProps Returns an array with all the properties names
 aGetProps()

Default Default values for the initialization of the VBX
 Default()

SetProp Changes a VBX property by its order
 SetProp(nProp, uValue) INLINE VBXSetProp(::hCtl, nProp - 1, uValue)

GetName Retrieves the class name of the VBX control (API value)

GetName() INLINE GetClassName(::hWnd)

nGetProps Retrieves the number of properties of the VBX

nGetProps() INLINE VBXGetNumProps(::hCtl)

nGetEvents Retrieves the number of events of the VBX

nGetEvents() INLINE VBXGetNumEvents(::hCtl)

GetEvent Retrieves the name of a VBX event by its name

GetEvent(nEvent) INLINE VBXGetEveName(::hCtl, nEvent - 1)

GetEnums Retrieves an array with a VBX enum all possible values

GetEnums(cnProp) INLINE VBXGetEnums(::hCtl, cnProp)

Get Retrieves the value of a property by its name

Get(cPropName) INLINE VBXGetPrpByName(::hCtl, cPropName)

Set Changes a VBX property the value of a property by its name

Set(cPropName, uValue) INLINE; VBXSetPrpByName(::hCtl, cPropName, uValue)

GetProp Retrieves the value of a property by its order

GetProp(nProp) INLINE VBXGetProp(::hCtl, nProp - 1)

GetPropType Retrieves the type of a VBX property

GetPropType(nProp) INLINE VBXGetPrpType(::hCtl, nProp - 1)

HandleEventVBX specifically HandleEvent (inherits from TControl)

HandleEvent(nMsg, nWParam, nLParam)

VbxEvent Internal VBX events management function support

VbxEvent(hCtl, bEvent, cEvent) EXTERN __VBXEvent()

MouseMove Virtual MouseMove method (to request default behavior)

MouseMove(nRow, nCol, nFlags) VIRTUAL

VbxInherit Internal automatic inherit process for 5W VBX system
VbxInherit()

TWindows

Help

DATOS

hWnd	window handle hWnd
nOldProc	Old window procedure when SubClassing
bInit	Code Block to execute on initialization
bMoved	Code Block to execute after moving the window
bLClicked	Code Block to execute when left clicking with the mouse
bKeyDown	Code Block to execute when pressing a key
bPainted	Code Block to execute when painting the Window
bRClicked	Code Block to execute when right clicking with the mouse
bResized	Code Block to execute after resizing a Window
bLDbIClick	Code Block to execute when left double clicking the mouse
bWhen	Code Block for WHEN clause
bValid	Code Block for VALID clause
bKeyChar	Code Block to execute when pressing a no-control key
bMMoved	Code Block to execute when moving the mouse
bGotFocus	Code Block to execute when receiving the focus
bLostFocus	Code Block to execute when losing the focus
cCaption	The caption of the Window
cPS	Paint structure meanwhile painting
nPaintCount	Nº of times painting procedure has been invoked
cMsg	The text for the default message on the MsgBar
Cargo	Standard Clipper general purpose user defined DATA
hDC	window Device Context handle while painting
nId	Numeric identifier of the Window
lActive	If the window is active
lVisible	If the window is visible
lFocused	If the window has the focus
nStyle	Numeric windows style value
nChrHeight	Standard height for characters of the window
nChrWidth	Standard width for characters of the window
nLastKey	Last key pressed on the window
nTop	window coordinates
nLeft	window coordinates
nBottom	window coordinates
nRight	window coordinates
nClrPane	Color of the background of the window
nClrText	Color of the foreground of the window
nResult	Numerical value after last activation
lValidating	If the window is in a process of validating a control
nHelpId	The help topic identifier for this window
hCtlFocus	Handle of the child control which has the focus

aControls	An array holding all the controls of the window
oBar	Bar Object attached to the window
oBrush	Brush Object attached to the window
oCursor	Cursor Object attached to the window
oFont	Font Object attached to the window
oIcon	Icon Object attached to the window
oMenu	Menu Object attached to the window
oSysMenu	SysMenu Object attached to the window
oPopup	Temporary popup menu acting over the window
oMsgBar	MsgBar Object attached to the window
oWnd	Window Object container of the window
oVScroll	Vertical scrollbar Object attached to the window
oHScroll	Horizontal scrollbar Object attached to the window

METODOS

New

Constructor of window class

Syntax:

TWindow():New(nTop, nLeft, nBottom, nRight, cTitle, nStyle, oMenu,; oBrush, oIcon, oWnd, IVScroll, IHScroll, nClrFore, nClrBack,; oCursor, cBorder, ISysMenu, lCaption, lMin, lMax) --> oTWindow

Parameters:

nTop ,nLeft	These two determine the top left corner of the window
nBottom, nRight	The right bottom corner of the window.
cTitle	The title of the window.
nStyle	Determines the characteristics of the window. These characteristics are define by the following constants (defined in WINAPI.CH):

WS_BORDER Window has a border.

WS_CAPTION Window has a title bar (implies the WS_BORDER style). This style cannot be used with the WS_DLGFRAME style.

WS_CHILD A child window. Cannot be used with the WS_POPUP style.

WS_CHILDWINDOW Same as the WS_CHILD style.

WS_CLIPCHILDREN Excludes the area occupied by child windows when drawing within the parent window. Used when creating the parent window.

WS_CLIPSIBLINGS Clips child windows relative to each other; that is, when a particular child window receives a paint message, the WS_CLIPSIBLINGS style clips all other overlapped child windows out of the region of the child window to be updated. (If WS_CLIPSIBLINGS is notspecified and child windows overlap, it is possible, when drawing within the client area of a child window, to draw within the client area of a neighboring child window.) For use with the WS_CHILD style only.

WS_DISABLED Window will be initially disabled.

WS_DLGFRAME Defines a window with a double border but no title.

WS_GROUP Specifies the first control of a group of controls in which the user can move from one control to the next by using the arrow keys. All controls defined with the WS_GROUP style after the first control belong to the same group. The next control with the WS_GROUP style ends the style group and starts the next group (that is, one group ends where the next begins). Only dialog boxes use this style.

WS_HSCROLL Window has a horizontal scroll bar.

WS_MAXIMIZE Window should be of maximum size.

WS_MAXIMIZEBOX Creates a window that has a Maximize button.

WS_MINIMIZE Define window that is initially minimized. For use with the WS_OVERLAPPED style only.

WS_MINIMIZEBOX Creates a window that has a Minimize button.

WS_OVERLAPPED Defines an overlapped window. An overlapped window has a title and a border.

WS_OVERLAPPEDWINDOW Creates an overlapped window having the WS_OVERLAPPED, WS_CAPTION, WS_SYSMENU, WS_THICKFRAME, WS_MINIMIZEBOX, and WS_MAXIMIZEBOX styles.

WS_POPUP Creates a pop-up window. Cannot be used with the WS_CHILD style.

WS_POPUPWINDOW Creates a pop-up window that has the WS_BORDER, WS_POPUP, and WS_SYSMENU styles. The WS_CAPTION style must be combined with the WS_POPUPWINDOW style to make the System menu visible.

WS_SYSMENU Creates a window that has a System-menu box in its title bar. Used only for windows with title bars.

WS_TABSTOP Specifies one of any number of controls through which the user can move by using the TAB key. The TAB key moves the user to the next control specified by the WS_TABSTOP style. Only dialog boxes use this style.

WS_THICKFRAME Creates a window with a thick frame that can be used to size the window.

WS_VISIBLE Creates a window that is initially visible. This applies to overlapped, child, and pop-up windows. For overlapped windows, the y parameter is used as a ShowWindow function parameter.

WS_VSCROLL Creates a window that has a vertical scroll bar.

Note: If this parameter is used it will override the values of the following parameters:

cBorder
lCaption
lSysmenu
lmin
lMax
lVScroll
lHScroll

oMenu	Menu for Window. This parameter should pass on a Tmenu object.
oBrush	The brush used to paint the client area of the window. It is stored in the instance variable::oBrush. If none is passed, it will be created by the method SetColor().
olcon	The icon that will be used whenever this window is minimized. If none is passed the icon is extracted from USER.EXE (which means you get a small MS-Windows emblem as icon).
oWnd	The TWindow object of which this window is the child window. If this parameter is not passed it means that this window will stand on its own.
lVScroll	Determines whether the window will have a vertical scrollbar. The default value is .F.

IHScroll	Determines whether the window will have a horizontal scrollbar. The default value is .F.
nClrFore	The foreground color (the color that is used to draw on the window). This can either be a MS-Windows COLOR number or a CA-CLipper color string (ie "R/G") in which case the nClrBack parameter will be ignored.
nClrBack	The background color of the window. This has to be a MS-Windows COLOR number.
oCursor	Determines which cursor will be shown when the mouse moves of the surface of this window.
cBorder	This parameter controls the type of border that the window has. It can be one of the following character strings: NONE The window will be a popup window SINGLE (Default) The window will have a thick frame
ISysMenu	Create window with a system Menu button on the top/left corner of the window. The system menu will only be shown if the window has a title bar (see ICaption)
ICaption	Determine if the window will have a title bar. Default is .T.
IMin	Determine whether or not the window will have a Minimize box. This box will only be shown if the window has a title bar (see ICaption)
IMax	Determines whether there will be a Maximize box for this window. This box will only be shown if the window has a title bar (see ICaption)

Purpose:

The New() Method creates a TWindow object and initializes a number of instance variables of the TWindow object.

Note: Do not use the parameters indiscriminately because several parameters have effect upon each other. For example the behaviour determined by the IMin and the IMax parameters will have NO effect if the nStyle parameter is passed.

Activate	Activates a Window
-----------------	--------------------

Syntax:

TWindow():Activate(cShow, bLClicked, bRClicked, bMoved, bResized, bPainted,; bKeyDown, bInit, bUp, bDown, bPgUp, bPgDown,; bLeft, bRight, bPgLeft, bPgRight, bValid, bDropFiles) \\ Self

Parameters:

<cShow>	This character string determines the 'visibility status' of the window when it is displayed. It can be one of the following strings: "NORMAL" Show the window 'normal' ie with the size as defined by ::nTop, ::nLeft, ::nBottom, ::nRight. This is the default value. "ICONIZED" The window will be shown iconized on the desktop.
---------	---

"MAXIMIZED" Show the window in it largest possible size ie it will cover the entire screen.													
<bLClicked>	This codeblock will be executed whenever there is a click of the left mouse button on the window. The codeblock will be called with three parameters:												
<nRow>	The pixel row where the mouse cursor was positioned when the mouse click occurred.												
<nCol>	The pixel column where the mouse cursor was positioned when the mouse click occurred.												
<nKeyFlags>	Indicates which virtual keys are down and if the right or the middle mouse button was pressed also. It can be a combination of the following values which have the following constant names in the MS-Windows API description:: <table><tr><td>MK_RBUTTON</td><td>2</td><td>The right mouse button was pressed also.</td></tr><tr><td>MK_SHIFT</td><td>4</td><td>The <Shift> key was pressed.</td></tr><tr><td>MK_CONTROL</td><td>8</td><td>The <ctrl> key was pressed.</td></tr><tr><td>MK_MBUTTON</td><td>16</td><td>The middle mouse button was pressed.</td></tr></table>	MK_RBUTTON	2	The right mouse button was pressed also.	MK_SHIFT	4	The <Shift> key was pressed.	MK_CONTROL	8	The <ctrl> key was pressed.	MK_MBUTTON	16	The middle mouse button was pressed.
MK_RBUTTON	2	The right mouse button was pressed also.											
MK_SHIFT	4	The <Shift> key was pressed.											
MK_CONTROL	8	The <ctrl> key was pressed.											
MK_MBUTTON	16	The middle mouse button was pressed.											
<bRClicked>	This codeblock will be executed whenever there is a click of the right mouse button on the window. The codeblock will be called with the same parameters as bLClicked												
<bMoved>	Currently not used.												
<bResized>	This code block will be called whenever a window is resized. When <bresized> is called it will be passed two parameters:												
<nWidth>	The new width of the window in pixels.												
<nHeight>	The new height of the window in pixels.												
<bPainted>	This code block will be called whenever the window is repainted (ie the window receives a WM_PAINT message from MS-Windows). When <bPainted> is called it receives one parameter and that is a reference to the TWindow() object												
<bKeyDown>	Whenever the window has focus and a NON-SYSTEM key is pressed (ie the <Alt> key was not pressed in conjunction with this key), this codeblock will be called. When <bKeyDown> is called it receives two parameters:												
<nKey>	The virtual key code of the key.												
<nFlags>	Specifies several internal values like repeat code and scan code. For more information about these values please read a MS-Windows API description about the WM_KEYDOWN message.												
<bInit>	This codeblock will be executed after the window is displayed for the first time. It receives one parameter which is a reference to the tWindow object itself.												
<bUp>	Whenever the VSCROLL keyword is used when a TWindow() object is defined, this codeblock will be called when the 'up' arrow of the vertical scrollbar is clicked upon. It receives no parameters.												
<bDown>	Whenever the VSCROLL keyword is used when a TWindow() object is defined, this codeblock will be called when the 'down' arrow of the vertical scrollbar is clicked upon. It receives no parameters.												

<bPgUp>	Whenever the VSCROLL keyword is used when a TWindow() object is defined, this codeblock will be called when the vertical scrollbar receives a 'pageup' message (ie the shaft of the scrollbar was clicked above the scrollbar). It receives no parameters.
<bPgDown>	Whenever the VSCROLL keyword is used when a TWindow() object is defined, this codeblock will be called when the vertical scrollbar receives a 'pagedown' message (ie the shaft of the scrollbar was clicked below the scrollbar). It receives no parameters.
<bLeft>	Whenever the HSCROLL keyword is used when a TWindow() object is defined, this codeblock will be called when the 'left' arrow of the horizontal scrollbar is clicked upon. It receives no parameters.
<bRight>	Whenever the HSCROLL keyword is used when a TWindow() object is defined, this codeblock will be called when the 'right' arrow of the horizontal scrollbar is clicked upon. It receives no parameters.
<bPgLeft>	Whenever the HSCROLL keyword is used when a TWindow() object is defined, this codeblock will be called when the horizontal scrollbar receives a 'pageleft' message (ie the shaft of the scrollbar was clicked left of the scrollbar). It receives no parameters.
<bPgRight>	Whenever the HSCROLL keyword is used when a TWindow() object is defined, this codeblock will be called when the horizontal scrollbar receives a 'pageright' message (ie the shaft of the scrollbar was clicked to the right of the scrollbar). It receives no parameters.
<bValid>	If this codeblock is passed it will be called whenever the window should be closed. <bValid> has to return a logical value to indicate whether or not the window is allowed to close.
<bDropFiles>	This codeblock is called whenever files are dropped on the window. When <bDropFiles> is called it receives three parameters:
<nRow>	The row (in pixels) where the file(s) were dropped.
<nCol>	The column (in pixels) where the file(s) were dropped.
<aFileNames>	This array contains all the file names of the files that were dropped on the window.

Purpose:

Shows the window and start interacting with it. The Activate method starts the main application event loop if not already working.

AddControl	Adds a new child control
-------------------	--------------------------

Syntax: TWindow:AddControl(<oControl>) --> nil

<oControl> Is a reference to the child control to be added.

This method adds a new child control to the window :aControls array, only if the window is currently visible.

If the window is not visible, TWindow:DefControl() is used instead.

BeginPaint	It is automatically called before painting
-------------------	--

Syntax: TWindow:BeginPaint() \ cPS

cPS Is the PaintStructure (Windows API) associated to the painting process. The DATA :cPS also keeps that value.

Purpose:

This method is automatically called when a painting process is starting. You may not call this method directly. Instead use TWindow:Refresh() to force a window painting.

Box	Draws a Box on the Window
------------	---------------------------

Syntax: TWindow:Box(nTop, nLeft, nBottom, nRight) \ nil

Parameters:

nTop , nLeft These two determine the top left corner of the window
nBottom, nRight The right bottom corner of the window.

Purpose:

Draws a box on the window. The coordinates are expressed in pixels. Note that ::Box() draws a box on the window for which it was called. So the coordinates are relative to the top left of the window.

Capture	Capture all the Mouse activity for the Window
----------------	---

Syntax:

TWindow:Capture() \ hWndPrevious

Returns the previous hWnd which had the mouse captured.

Center	Centers the window on the screen
---------------	----------------------------------

Syntax:

TWindow:Center(<oWnd>) --> Nil

Parameters:

<oWnd> The window relative to which this window will be centered. If it is omitted the window will be centered using the dimensions of the desktop.

Purpose:

Centers the window on the screen relative to another window or to the desktop.

Command	Process a generic command
----------------	---------------------------

Syntax:

TWindow:Command(<nWParam>, <nLParam>)

Parameters:

<nWParam> This parameter actually holds two pieces of information which have the following meaning:

<nCode> = nHiWord(nWParam). This value specifies the notification code if the message is from a control. If the message is from an accelerator, this parameter is 1. If the message is from a menu, this parameter is 0.

nID = nLoWord(wParam); This part represents the identifier of either an item, a control or an accelerator.

<nLParam> This value contains the handle of the control sending the message if the message is from a control. Otherwise, this parameter is NULL.

Purpose:

This method is automatically called when processing a WM_COMMAND WinApi message. If a message is processed it should return 0.

Circle	Draws a circle
---------------	----------------

Syntax:

TWindow:Circle(<nRow>, <nCol>, <nWidth>) --> nil

Parameters:

<nRow> <nCol> These coordinates represent the center of the circle. They are relative to the top left of the window or which this method is called

<nWidth> The radius of the circle.

Purpose:

This method can be used to draw a circle on the client area of the window.

CoorsUpdate	Updates the coordinates of the window
--------------------	---------------------------------------

TWindow:CoorsUpdate() \nil

The window object has nTop, nLeft, nBottom, nRight DATA, but they are not automatically updated. Use this method to refresh those values and have the latest values.

Copy	Places a copy of the window image at the clipboard
-------------	--

TWindow:Copy(<IAI>) \nil

<IAI> Is a logical value meaning if all the surface has to be copied or just the client area of the window.

Create	Generic new Window Class creation
---------------	-----------------------------------

TWindow:Create(cClsName) \nil

Windows API creation process. You should not call this method directly.

To create a window, use instead TWindow:New().

cClsName The name of the window class. The class name can be any name registered with the RegisterClass() function or any of the predefined control-class names

CtlColor	Set color of control.
-----------------	-----------------------

Syntax:

TWindow:CtlColor(<hWndChild>, <hDCChild>) \

Parameters:

<hWndChild> The handle of the child window

<hDCChild> The handle of the DC

Purpose:

cTitle	The caption of the Window
---------------	---------------------------

TWindow:cTitle(<cNewTitle>) \ cTitle

This is a SETGET method which assigns or retrieve the window caption.

You may use it as if it were a DATA (instance variable) of the TWindow object, but it is in fact a method.

<cNewTitle> The new value for the title.

DdeInitiate	Initiate DDE session from the window
--------------------	--------------------------------------

Syntax:

TWindow:DdeInitiate(<hWndClient>, <nAppName>, <nTopicName>) \\\

Parameters:

<hWndClient> The handle of the client window for which the DDE session has to be initiated.

<nAppName> The handle to a global atom (created by GetGlobalAtom()) which contains the name of the DDE server.

<nTopicName> The handle to a global atom which contains the name of the topic for with which the DDE session is to be initiated.

Purpose:

The DdeInitiate() method is called by the eventhandler of a DDE server(normally the main window of an application) when a DDE client wants to start a session. The method also receives the name of the application that want a connection and the topic the client program wants a DDE link for.

DdeAck	Acknowledge DDE.
---------------	------------------

Syntax:

TWindow:DdeAck(<hWndSender>, <nLParam>) \\\

Parameters:

<hWndServer> It is the handle of the window which it is acting as a server in a DDE conversation.

<nExtraData> Some extra data sometimes required.

Purpose:

This method is called by the eventhandler of the DDE client window whenever it receives an acknowledgement from the DDE server.

DdeExecute	Execute DDE command
-------------------	---------------------

Syntax:

```
TWindow:DdeExecute( <hWndSender>, <nCommand> ) \\\
```

Parameters:

<hWndSender> The handle off the DDE client window who initiated the command.

<nCommand> The command sent by the DDE client window. It is actually a handle to a GLOBAL ATOM.

Purpose:

Whenever the DDE server window receives a DDE command from a client the eventhandler calls this method with information about the application that requested the command an the actual command.

DdeTerminate	Terminate DDE session
---------------------	-----------------------

Syntax:

```
TWindow:DdeTerminate( <hWndSender> ) \\\
```

Parameters:

<hWndSender> The DDE window that wants to terminate the DDE session.

Purpose:

Whenever a DDE window receives a signal that the link is to be terminated the eventhandler calls this method.

Destroy()	Destroy a window
------------------	------------------

```
TWindow:Destroy( ) --> NIL
```

Function:

This method handles the actual removal of a window. It should NOT be called directly. If you want to end a window's existence you should

call the End() method. The Destroy() method also destroys (ie releases) the following objects if they were associated with the window:

- ::oBrush
- ::oCursor
- ::oIcon
- ::oFont
- ::oMenu
- ::oSysMenu
- ::oVScroll
- ::oHScroll

and the handling of Drag-and-Drop if it was installed. If the window was the main window off the application it will also close the main window and proceed with the code behind the activation of the window.

Disable

Disables the Window

```
TWindow:Disable() \nil
```

This method disables all the input to this window. Meanwhile a window is disabled, it can not be selected with the mouse and no keyboard strokes are sent to it.

DrawItem

Draw item

Syntax:

```
TWindow:DrawItem( <nIdCtl>, <pItemStruct> ) \l
```

Parameters:

<nIdCtl> Control Item

<pItemStruct> handle of Item structure

DropFiles

Handle files dropped on this window

Syntax:

```
TWindow:DropFiles( <hDrop> ) \l
```

Parameters:

<hDrop> The handle of the MS-Windows DROP structure.

Purpose:

If a window can accept files that are dropped from the File Manager this method is called by the eventhandler whenever files are dropped on the window. The handle (<hDrop>) points to a MS-Windows DROP structure, from which both the file names of the files dropped on this window can be extracted and the point on the window where the files were dropped on.

Enable

Enable Window activity

```
TWindow:Enable() \nil
```

Enables a previous disabled window. When a window is enabled it may be selected using the mouse and the keyboard events are sent to it.

End

End Activity

`TWindow:End() \\Success`**Purpose:**

This method should be called if the window has to be closes. If the VALID clause has been defined, the clause has to return .t. in order to let the window end.

EndPoint

Automatically called when ending painting

`TWindow:EndPoint() \\ nil`

This method ends the painting process of a window. It is called automatically in response to a WinApi WM_PAINT message. Don't call it directly.

To display a window use `TWindow:Refresh()`.

EraseBkGnd

Erase background of window

Syntax:`TWindow:EraseBkGnd( <hDC> ) \\`**Parameters:**

<hDC> The DEVICE CONTEXT handle of this window.

Purpose:

The purpose of this method is to erase the background of the window.

FloodFill

Fill area from startpoint with color

Syntax:`TWindow:FloodFill( <nRow>, <nCol>, <nRGBColor> ) \\`**Parameters:**

<nRow> The pixel row that is the top row from which the clientarea that will be filled with color <nRGBColor>.

<nCol> The pixel column that is the left most pixel row from which the client area of the window is to be filled with the color <nRGBColor>.

<nRGBColor> A MSWindow color number which represents the color that is to be used.

Purpose:

This method can be used to color a part of the client area of the window. The area is defined by the area enclosed in the rectangle defined by <nRow>,<nCol>, which represent the top left corner of the rectangle and the right bottom of the client area.

FocusNext	Changes the child focus to the next or to the previous
------------------	--

TWindow:FocusNext(<hCtrl>, <IPrevious>) \ \ nil

<hCtrl> It is the hWnd of the control currently focused.

<IPrevious> It is a logical value: .t. for the next and .f. for the previous.

This method sets the focused child control of the window to the next one or to the previous according to IPrevious value.

cGenPrg	Generates the source code of the window
----------------	---

Syntax:

TWindow:cGenPRG(<cFileName>) \ \

Parameters:

<cFileName> The file where the generated source code should be written to. The file will be overwritten if it already exists.

Purpose:

This can be a very powerful method in a sense that you can design a window with controls on it and the you can make a call to the cGenPrg() method to generate the source code to generate the window you just designed.

nGetChrHeight	Updates the nChrHeight DATA of the window
----------------------	---

TWindow:nGetChrHeight() \ \ nil

This method calculates the character height of the window based on the current active font and places the result in the :nChrHeight DATA of the window.

GetCliRect	Returns the client window area
-------------------	--------------------------------

Return TRect() object with client rectangle of window.

TWindow:GetCliRect() \ TRect()

Function:

The GetCliRect() method will return a TRect() objects which contains the coordinates describing the client rectangle of the window. Since client rectangle coordinates are relative to the window they will start in the upper left corner of the window with 0,0. The <nBottom>,nRight>pair will give the height and the width of the client rectangle. All coordinates are in pixels.

GetFont

Get font from window

Syntax:

TWindow:GetFont() \

Purpose:

This method retrieves the current font object for this window. if ::oFont was not defined for this window GetFont() will retrieve the current font settings for this window, create a font object, place it in ::oFont and returns the information. If on the other hand, ::oFont was defined, GetFont() just returns the Font object in ::oFont.

GetRect

Returns the whole area

TWindow:GetRect() \TRect()

Function:

The GetRect() method will return a TRect() objects which contains the coordinates describing the bounding rectangle of the window. The coordinates returned are relative to the top-left corner of the screen. They include the border, title bar and scrollbars if they are present. The coordinates are in pixels.

GetDC

Returns the current device context. if any or request a new one.
--

TWindow:GetDC() \

Purpose:

The GetDC() method returns the current DEVICE context for this window. If ::hDC already contains a handle to the window's DEVICE context GetDC() returns that. Otherwise it will retrieve the current DEVICE CONTEXT for this window, store it in ::hDC and return the handle to this DEVICE CONTEXT.

GetDlgCode

Return the kind of Dialog information a window requests

TWindow:GetDlgCode() \ NIL

This is a VIRTUAL method for this class ie for the TWindow class this method performs no action at all.

GetWidth

Get width of string written with certain font

Syntax:

```
TWindow:GetWidth( <cText>, <oFont> ) \\\
```

Parameters:

<cText> The text string for which the length is to be determined.

<oFont> The font that should be taken into account when determining the length of the string. If <oFont> is not supplied GetWidth will use the font stored in ::oFont.

Purpose:

This method allows you to determine what the exact length is of a string, when it is written to this window. This can be useful if you want to determine beforehand if a string can be completely written to this window without being 'clipped'. GetWidth() takes into account the font information supplied by <oFont> (if passed). If <oFont> is not supplied the default font for the window is used.

GoNextCtrl

Goto next control in window

Syntax:

```
TWindow:GoNextCtrl( <hCtrl> ) \\\
```

Parameters:

<hCtrl> The handle of the control from where the search for the next control on this window should start.

Purpose:

This method will set focus to the next control on this window. GoNextCtrl() will start searching from the control identified by <hCtrl>.

GoPrevCtrl

Goto previous control in window

Syntax:

```
TWindow:GoPrevCtrl( <hCtrl> ) \\\
```

Parameters:

<hCtrl> The handle of the control from where the search for the previous control on this window should start.

Purpose:

This method will set focus to the previous control on this window. GoNextCtrl() will start searching from the control identified by <hCtrl>.

GotFocus

Called when window got focus

Syntax:

TWindow:GotFocus() \

Purpose:

This method is called by the eventhandler of this window whenever the window has received focus. In turn GotFocus() will call the ::bGotFocus codeblock (if specified). This will allow you to perform certain actions whenever the window has received focus.

GoTop

Bring window to top

Syntax:

TWindow:GoTop() \

Purpose:

The GoTop() method brings the given pop-up or child window (including an MDI child window) to the top of a stack of overlapping windows. In addition, it activates pop-up, top-level, and MDI child windows. The GoTop() method should be used to uncover any window that is partially or completely obscured by any overlapping windows.

HandleEvent

Generic handle event management

TWindow:HandleEvent(<nMsg>, <nWParam>, <nLParam>) \ nVal

This method handles all the events for a particular TWindow object. It directly calls a C-function (WndHandleEvent()). This is done to increase speed. For complete documentation on each and every value of <nMsg> and its associated values of <nWParam> and <nLParam> please check out a book (or help file) on MS-Windows messages and how they are handled.

HardCopy()

Makes a printout of the Window.

TWindow:HardCopy(<nScale>, <lUser>) \ Nil

<nScale> The scale that will be used when printing the contents of the window. It has to be an integer which is greater or equal to 1. Default is 1.

<lUser> Determines whether the user of the application will have to choose a printer or if the printing will be done on the default printer. If <lUser> is >T> the user can choose a printer. Default is .T..

Hide

Hide window

Syntax:

```
TWindow:Hide() \
```

Purpose:

Hides the window, and brings another one to the top.

HScroll

Horizontal scroll method dispatcher

```
TWindow:HScroll( <nWParam>, <nLParam> ) \ 0
```

Processes the WM_HSCROLL messages that are sent to this window object. This method handles both the messages for both a scrollbar that is connected to a window or for a particular horizontal scrollbar control.

Parameters:

<nLParam> The high word of this parameter contains the handle of the scrollbar.

<nWParam> This parameter contains a numerical value that specifies the action that should be taken. The following messages are processed (these constants are defined in WINAPI.CH):

```
SB_LINEUP
SB_LINEDOWN
SB_PAGEUP
SB_PAGEDOWN
SB_THUMBPOSITION
SB_ENDSCROLL (only if the method is called for a scrollbar that is connected to a
window).
```

Iconize()

Iconize window

Syntax:

```
TWindow:Iconize() \
```

Purpose:

This method iconizes (minimizes) the current window. It does NOT destroy the window. For that you should call the End() method.

KeyDown

Key holding down method dispatcher

```
TWindow:KeyDown( <nKey>, <nFlags> ) \\\
```

Function:

This method handles all the WM_KEYDOWN messages Windows sends to this particular object. These messages are sent when a window has keyboard focus and when a NON-SYSTEM key (the <Alt> key was NOT pressed) is pressed. This method receives both the virtual keycode and a set of flags which can be used to determine more about the non-system key that was pressed.

Parameters:

<nKey> The value of the non-system key that was pressed.

<nFlags> Specifies the repeat count, scan code, extended-key flag, context code, previous key-state flag, and transition-state flag, For more information on these flags please consult the MS-Windows API documentation.

KeyChar

Key pressed method dispatcher

```
TWindow:KeyChar( <nKey>, <nFlags> ) \\\
```

Function:

This method handles all the WM_CHAR messages Windows sends to this particular object. This method receives both the keycode and a set of flags which can be used to determine more about the keystroke.

Parameters:

<nKey> The value of the key pressed

<nFlags> Specifies the repeat count, scan code, extended-key flag, context code, previous key-state flag, and transition-state flag, For more information on these flags please consult the MS-Windows API documentation.

KillFocus

Called when window loses focus

Syntax:

```
TWindow:( <hWndFocus> ) \\\
```

Parameters:

<hWndFocus>

LButtonDown	Left Mouse button down dispatcher
--------------------	-----------------------------------

TWindow::LButtonDown(nRow, nCol, nKeyFlags)

Function:

This method is called whenever the left mouse button is pushed down. LButtonDown() will check to see if the bLClicked instance variable contains a code block. If that is the case the codeblock will be called with the parameters LButtonDown() has received.

Parameters:

<nRow> The pixel row where the mouse cursor was positioned when the mouse click occurred.

<nCol> The pixel column where the mouse cursor was positioned when the mouse click occurred.

<nKeyFlags> Indicates which virtual keys are down and if the right or the middle mouse button was pressed also. It can be a combination of the following values which have the following constant names in the MS-Windows API description::

MK_RBUTTON	2	The right mouse button was pressed also.
MK_SHIFT	4	The <Shift> key was pressed.
MK_CONTROL	8	The <ctrl> key was pressed.
MK_MBUTTON	16	The middle mouse button was pressed.

LDblClick	Called when left double click occurs
------------------	--------------------------------------

Syntax:

TWindow::LDblClick(<nRow>, <nCol>, <nKeyFlags>) \\\

Parameters:

<nRow> Row where the mouse was pressed

<nCol> Column where the mouse was pressed

<nKeyFlags> Indicates which virtual keys are down and if the right or the middle mouse button was pressed also. It can be a combination of the following values which have the following constant names in the MS-Windows API description::

MK_RBUTTON	2	The right mouse button was pressed also.
MK_SHIFT	4	The <Shift> key was pressed.
MK_CONTROL	8	The <ctrl> key was pressed.

MK_MBUTTON 16 The middle mouse button was pressed.

Line	Draws a line
-------------	--------------

TWindow:Line(<nTop>, <nLeft>, <nBottom>, <nRight>) \\\

Function:

The method Line() will draw a line on th window associated with this TWindow() object. The coordinates that are passed are relative to the top left corner of this window. All coordinates should be in pixels.

Parameters:

<nTop>, <nLeft> These two parameters determine the starting point of the line.

<nBottom>, <nRight> These two parameters determine the ending point of the line.

Link	Window to OOPS conversion initialization
-------------	--

TWindow:Link(<ISubClass>) \\\

Function:

Window to OOPS conversion initialization

LostFocus	Called when window loses focus
------------------	--------------------------------

Syntax:

TWindow:LostFocus() \\\

IWhen	Evaluate when condition
--------------	-------------------------

Syntax: TWindow:IWhen() \\\

Purpose:

The purpose of this method is to make sure that when the window is created the bWhen instance variable will be evaluated. After this theGotFocus method will take care of handling the bWhen instance variable.

Maximize	Maximize window
-----------------	-----------------

Syntax:

TWindow:Maximize() \\\

Purpose:

This method allows the programmer to maximize the window during program execution from the code instead of having the user click on the maximize button

MenuSelect	Called when menuitem is selected
-------------------	----------------------------------

Syntax:

TWindow:MenuSelect(<nIdItem>) \\\

Parameters:

<nIdItem> Item identifier

MeasureItem	Measure size of control
--------------------	-------------------------

Syntax:

TWindow:MeasureItem(<nIdCtl>, <pMitStruct>) \\\

Parameters:

<nIdCtl> Control Id

<pMitStruct> Handle to the item structure

Minimize	Minimizes the Window
-----------------	----------------------

TWindow:Minimize() \\\

Function:

This function will minimize the current window and gives focus to the top-level window in the system list.

MouseMove	Called when mouse moves over window
------------------	-------------------------------------

Syntax:

TWindow:MouseMove(<nRow>, <nCol>, <nKeyFlags>) \\\

Parameters:

<nRow> Row where the mouse was pressed

<nCol> Column where the mouse was pressed

<nKeyFlags> Indicates which virtual keys are down and if the right or the middle mouse button was pressed also. It can be a combination of the following values which have the following constant names in the MS-Windows API description::

MK_RBUTTON	2	The right mouse button was pressed also.
MK_SHIFT	4	The <Shift> key was pressed.
MK_CONTROL	8	The <ctrl> key was pressed.
MK_MBUTTON	16	The middle mouse button was pressed.

Move	Moves the Window
-------------	------------------

TWindow:Move(<nTop>, <nLeft>, <nBottom>, <nRight>, <IRepaint>) \ \ NIL

Parameters:

<nTop>, <nLeft> These two determine the top left corner of the window. The coordinates should be given in pixels.

<nBottom>, <nRight> The right bottom corner of the window. The coordinates should be given in pixels.

<IRepaint> The <IRepaint> parameter controls whether or not a repaint of the window should occur as soon as the window is moved. If <IRepaint> is .T. the window will be immediately repainted.

Function:

The Move() method will move the current window to a new position and change its size. If the window is a top level window, the coordinates are relative to the top left of the screen. If the window is a child window the coordinates are relative to the top left corner of the parent window.

Note:

If <nRight> is 0 (ie zero), the current width and height of the window are used.

NcActivate	Syntax:
-------------------	---------

TWindow:NcActivate(<IOnOff>) \ \

Parameters:

<IOnOff>

Normal	Give window normal size
---------------	-------------------------

Syntax:

TWindow:Normal() \\\

Paint

Generic painting method

TWindow:Paint() \\\ NIL

Function:

The method Paint() controls the actual painting of this object. It should NOT be called directly because it should be called between the BeginPaint() and EndPaint() methods. If you want to repaint a window you should call the Refresh() method.

PaletteChanged**Syntax:**

TWindow:PaletteChanged(<hWndPalChg>) \\\

Parameters:

<hWndPalChg>

PostMsg

Post a Windows message

TWindow:PostMsg(<nMsg>, <nWParam>, <nLParam>) \\\nRetVal

Parameters:

<nMsg> This parameter should contain one of the MS-Windows API 'messages'. The values are defined in WINAPI.CH.

<nWParam> The value of this parameter is directly related to the value of <nMsg>. This should be a 16 bit integer.

<nLParam> The value of this parameter is directly related to the value of <nMsg>. This should be a 32 bit integer.

Purpose:

The PostMsg() method places a API message in the message queue of the window and does not wait for the window to process the message. It can be used to influence the behavior of the window. However the sending of messages can be very dangerous for your application if it is not done properly. Before you use the PostMsg() method be sure of what you are doing, which message you are sending and what the values should be of the related parameters.

Print

Prints the Window at the Printer

```
TWindow::Print( <oPrint>, <nRow>, <nCol>, <nZoom> ) \\\
```

Parameters:

<oPrint> A printer object created by either DEFINE PRINTER or by calling the New() method of the TPrinter() class.

<nRow> The pixel row on the paper where the printing of the contents should start.

<nCol> The pixel column on the paper where the printing of the contents should start.

<nZoom> The enlargement factor that will be used when printing. It must be an integer value (>= 1).

Function:

This method will send the contents of the window to a printer object so that it will be printed. The printer is represented by the printer object <oPrint> which should be a valid printer object. The scaling of the picture on the paper can be controlled by <nZoom>.

QueryDragIcon

Icon for dragging

Syntax:

```
TWindow:QueryDragIcon() \\\
```

QueryEndSession

End of query

Syntax:

```
TWindow:QueryEndSession() \\\
```

QueryNewPalette**Syntax:**

```
TWindow:QueryNewPalette() \\\
```

RButtonDown

Right mouse button down dispatcher

```
TWindow::RButtonDown( nRow, nCol, nKeyFlags )
```

Function:

This method is called whenever the right mouse button is pushed down. RButtonDown() will check to see if the bRClicked instance variable contains a code block. If that is the case the codeblock will be called with the parameters RButtonDown() has received.

Parameters:

<nRow> Row where the mouse was pressed

<nCol> Column where the mouse was pressed

<nKeyFlags> Indicates which virtual keys are down and if the right or the middle mouse button was pressed also. It can be a combination of the following values which have the following constant names in the MS-Windows API description::

MK_RBUTTON	2	The right mouse button was pressed also.
MK_SHIFT	4	The <Shift> key was pressed.
MK_CONTROL	8	The <ctrl> key was pressed.
MK_MBUTTON	16	The middle mouse button was pressed.

ReleaseDC	Release the previous device context requested
------------------	---

TWindow:ReleaseDC() \\\

Function:

ReleaseDC() will release the Device Context (DC) that was requested by GetDC() as soon as the instance variable ::nPaintCount, which is decremented first, becomes zero. In this way the DC can be used by other applications. Be sure that if you call ReleaseDC that you have already made a call to the GetDC() method. These calls must ALWAYS come in pairs.

Refresh	Forces a Window to be repainted
----------------	---------------------------------

TWindow:Refresh(<IErase>) \\\

Parameters:

<IErase>The <IErase> parameter determines whether or not the background of client area should be erased when the repainting of the window begins. If <IErase> is .T. the background will be erased. Default is .T..

Purpose:

The Refresh() method will send a message to Windows that the client area of the window should be repainted. This can become necessary if some part of the client area has been overwritten or if there are some changes to the background of the client area. If no parameters are passed to the Refresh() method, the background of the client area will be erased before the window is painted again.

Register

Register a new Window Class

TWindow:Register(<nClsStyle>) \\\

Parameters:

<nClsStyle> The <nClsStyle> parameter determines the style of the window class when it is registered. This in turn determines a great part of the behavior of the window. For more (complete) documentation about this method please read the MS-Windows API on the RegisterClass() function and the WNDCLASS structure member style.

Purpose:

Register the TWindow() class for use by MS-Windows. Before using this method, please be sure that you are comfortable with CreateWindow() MS-Windows API function and its parameters.

ReSize

Changes the size of a Window

TWindow:ReSize(<nSizeType>, <nWidth>, <nHeight>) \\\ 0

Parameters:

<nSizeType> Determines the type of resize operation that is executed. Specifies the type of resizing requested. This parameter can be one of the following values:

SIZE_MAXIMIZED Window has been maximized.
 SIZE_MINIMIZED Window has been minimized.
 SIZE_RESTORED Window has been resized, but neither the
 SIZE_MINIMIZED nor SIZE_MAXIMIZED value applies.
 SIZE_MAXHIDE Message is sent to all pop-up windows when some other
 window is maximized.
 SIZE_MAXSHOW Message is sent to all pop-up windows when some other
 window has been restored to its former size.

```
#define SIZE_RESTORED    0
#define SIZE_MINIMIZED   1
#define SIZE_MAXIMIZED   2
#define SIZE_MAXSHOW     3
#define SIZE_MAXHIDE     4
```

<nWidth> The new width of the window in pixels.

<nHeight> The new height of the window in pixels.

Purpose:

This method is called whenever the window is resized. It receives information about the new size and how the resizing came to be (ie the window was maximized, minimized etc.). In turn the Resize() method will call the adjust methods of

both Buttonbar and the messagebar (if available) to adjust their size also. After that a call is made to the bResized instance variable (if available) with both the width and the height passes as parameters to the bResized codeblock.

Say

Writes text on a Window

```
TWindow:Say( <nRow>, <nCol>, <cText>, <ncClrFore>, <nClrBack>,
<oFont>,<IPixel> ) \ Nil
```

Parameters:

<nRow> The row where <cText> will be written. This row is relative to the top of the window. Whether or not this parameter represents pixels or 'characters' depends on <IPixel>.

<nCol> The column where <cText> will be written. This column is relative to the left side of the window. Whether or not this parameter represents pixels or 'characters' depends on <IPixel>.

<cText> The text that has to written in the window.

<ncClrFore> This parameter can be either a character string(1) or a numerical(2).

1. If it is a character string it should be a CA-Clipper color-pair, which determines the foreground and the background color used to write <cText> on the window. If it is a CA-Clipper color string <nClrBack> will be ignored. For example when ncClrFore is "R/G" the text will be written on the window in red letters on a green background.

2. When <ncClrFore> is a numerical value it is assumed that this is a valid COLOR value that represents a color that can be displayed. If <ncClrFore> is omitted the value is used that was stored in the instance variable ::nClrText of the TWindow() object.

<nClrBack> This parameter is used whenever a numerical value is passed for <ncClrFore>. It then represents the background color that should be used when displaying the text on the window. If used it should be a Windows COLOR numerical.If omitted, and <ncClrFore> is a numerical, then the value of the instance variable ::nClrPane is used.

<oFont> This font object determines the font that will be used when <cText> is displayed on the window. It should be a valid TFont() object. If it is omitted the ::oFont instance variable will be used.

<IPixel> The <IPixel> parameter determines whether the coordinates that were passed on to the Say() method (<nRow> and <nCol>) are to be used as pixel coordinates or character based coordinates. If <IPixel> is .T. then the coordinates are pixel based. The default value is .F. ie the coordinates are character based.

Purpose:

This method allows the programmer to write something directly to the client area of a window. It gives him/her control over the position, the color and the font used. The Say() method either accepts character based or pixel based coordinates and can handle either

CA-Clipper color strings or MS windows numerical COLOR pairs. Note however that the text is 'painted' directly on the window, ie as soon as the window is redrawn the text will disappear. If you want to place text on a window that stays there you should use the @ x,y SAY command or the related TSay() method

seealso: TSay() TFont() @ SAY

SayRect

Writes text with a box around

```
TWindow:SayRect( <nRow>, <nCol>, <cText>, <nClrFore>, <nClrBack>, <nWidth>
) \Nil
```

Parameters:

<nRow> The row from where <cText> will be written. This row is relative to the top of the window. The row is character based.

<nCol> The column from where <cText> will be written. This column is relative to the left side of the window.

<cText> The text that has to be written in the window.

<nClrFore> The foreground color that should be used when <cText> is written. This should be a MS-Windows COLOR numerical. If it is omitted CLR_BLACK (a constant defined in WINAPI.CH) is assumed.

<nClrBack> The background color that should be used when <cText> is written. This should be a MS-Windows COLOR numerical. If it is omitted CLR_WHITE (a constant defined in WINAPI.CH) is assumed.

<nWidth> The maximum width of the <cText> that should be written on the window. The width is assumed to be on a pixel bases ie if <nWidth> is 16, only 16 pixel columns of the text will be written on the window (which is about two characters for a 'normal font')

Purpose:

The SayRect() method allows the programmer to determine the exact length of a string that is to be written to the client area of the window.

seealso: TWindow:Say()

SendMsg

Send a Windows message

```
TWindow:SendMsg( <nMsg>, <nWParam>, <nLParam> ) \nSucces
```

Parameters:

<nMsg> This parameter should contain one of the MS-Windows API 'messages'. The values are defined in WINAPI.CH.

<nWParam> The value of this parameter is directly related to the value of <nMsg>. This should be a 16 bit integer.

<nLParam> The value of this parameter is directly related to the value of <nMsg>. This should be a 32 bit integer.

Purpose:

The SendMsg() method places a API message in the message queue of the window and waits for the window to process the message. It can be used to influence the behavior of the window. However the sending of messages can be very dangerous for your application if it is not done properly. Before you use the PostMsg() method be sure of what you are doing, which message you are sending and what the values should be of the related parameters.

SelColor

Choose foreground or background color for window

Syntax:

TWindow:SelColor(<IFore>) \\\

SetBrush

Changes the brush of a Window

TWindow:SetBrush(<oBrush>) \\\ Nil

Parameters:

<oBrush> The TBrush() object that should be used from now on to paint the background of the window.

Purpose:

With this method you can change the brush that is used to paint the background of this window. This may be used in various ways to indicate a change for the user or just for sheer artistic pleasure <g>.

SetCoors

Changes the coordinates of the Window

TWindow:SetCoors(<oRect>) \\\ Nil

Parameters:

<oRect> A TRect() object with the new coordinate information for the window. The information passed within the TRect() object should be on a pixel basis.

Purpose:

The SetCoors() method can be used to change the coordinates of the window. If the window is the application's main window the coordinates are interpreted as relative to the

top-left of the screen. If the window is a client window the coordinates are relative to the top-left of the parent window.

seealso: Resize()

SetColor

Set foreground/background color and brush

Syntax:

TWindow:SetColor(<ncClrFore>, <nClrBack>, <oBrush>) \

Parameters:

<ncClrFore> Determines the foreground color that will be used when something is written to the window. The <ncClrFore> parameter can either be a CA-Clipper color string (eg "R/G"), or it can be a MS-Windows COLOR number. If <ncClrFore> IS a character string <nClrBack> will be ignored.

<nClrBack> Determines the background color that will be used When something is written to the window. This must be a MS-Windows color number.

<oBrush>The TBrush() object that will be used to paint the background of the window before something is written on the window. If this parameter is omitted a TBrush() object will be created which will have the background color.

Purpose:

This method allows you to change the color and brush that are used when a window is repainted. The SetColor() method accepts both CA-Clipper color strings and MS-Windows COLOR numbers, so for a CA-Clipper programmer there is no need to lose her/his favorite colorstrings.

SetFocus

Gives the focus to a Window

TWindow:SetFocus() \ Nil

Purpose:

The SetFocus() method will transfer the keyboard focus to this window, However before the transfer of focus actually takes place first a call is made to the :IWhen() method. If and only if the :IWhen() method returns .T. the current window gets the keyboard focus.

SetFont

Select font for window

Syntax:

TWindow:SetFont() \

Purpose:

The SetFont() method will display the standard MS-windows 'Select Font' dialog box to the user. After the user has chosen a font this is set as the default font for this window, and the window is repainted using the selected font where applicable.

SetFont

Changes the font of a Window

```
TWindow:SetFont( <oFont> ) \\\
```

Parameters:

<oFont> The new font object that should be associated with this window. The <oFont> parameter should be a valid TFont() object.

Purpose:

The purpose of this method is to associate a default font with a window so that it can be used when a control is displayed on the window. It can be used either before or after the Activate() method of the window is called. When SetFont() is called it will first call the End() method of the previous default font (if available) and then it replaces the default font(stored in the ::oFont instance variable) with <oFont>.

SetMenu

Changes the Menu of a Window

```
TWindow:SetMenu( <oMenu> ) \\\ Self
```

Parameters:

<oMenu> The new menu object that will be associated with this menu. This should be a valid MENU object, created via the MENU keywords (the easy way) TMENU() and TMENUITEM() (the less easy way).

Purpose:

This method allows the programmer to change the menu during the execution of the program. This can be necessary if at first the user only has a few options like 'Opening a file'. After the file is open a more extensive menu might be necessary.

SetMsg

Changes the message of the bottom of the Window

Syntax:

```
TWindow:SetMsg( <cText> ) \\\ Nil
```

Parameters:

<cText> This text will be displayed in the message bar of the window (ie at the bottom of the window).

Purpose:

The method SetMsg() allows you to display a different text at the message bar at the bottom of the window.

SetPixel

Draws a pixel on the Window

Syntax:

```
TWindow:SetPixel( <nX>, <nY>, <nColor> ) \ Nil
```

Parameters:

<nX> The pixel row where the pixel should be turned on. This row is relative to the top of the window.

<nY> The pixel column where the pixel should be turned on. This column is relative to the left side of the window.

<nColor> This determines the color of the pixel. The color should be a MS-Windows COLOR integer.

Purpose:

The SetPixel() method will enable you to set individual pixels on or off. Since you can control the color of the pixel you can easily turn a pixel 'off' by setting it to the background color of the window (ie stored in the instance variable ::nClrPane)

SetText

Changes the caption of the Window

Syntax:

```
TWindow:SetText( <cText> ) \ Nil
```

Parameters:

<cText> The new title of the window.

Purpose:

This method enables you to change the title of your window. This can be used to indicate a global change to the user.

Show

Show window

Syntax:

```
TWindow:Show() \
```

Purpose:

This method will show the window. After that it becomes the current window.

SysCommand	System Menu commands dispatcher
-------------------	---------------------------------

Syntax:

TWindow:SysCommand(<nWParam>, <nLParam>) \ Nil

Parameters:

<nWParam> This should be a 16 bit integer value. For precise values please check your Windows API reference under the message WM_SYSCOMMAND.

<nLParam> This should be a 32 bit integer value. For precise values please check your Windows API reference under the message WM_SYSCOMMAND.

Purpose:

This method should normally not be called directly in an application. SysCommand() will be called by the internal event handler of the window whenever the window receives a WM_SYSCOMMAND message. For more extensive information please check your MS-Windows API reference under the message WM_SYSCOMMAND.

Timer	Called when timer event occurs for window
--------------	---

Syntax:

TWindow:Timer(<nTimerId>) \ Nil

Parameters:

<nTimerId>

UnLink	Disables Window-OOPS connection
---------------	---------------------------------

Syntax:

TWindow:Unlink() \ Nil

Purpose:

This method handles the administration for the destruction of a window. It removes it from the property list of MS-Windows. This method should not be called by the programmer directly unless she/he has an intimate knowledge of the inter workings of MS-Windows.

Update	Update controls on window
---------------	---------------------------

Syntax:

TWindow:Update() \\\

Purpose:

The Update() method will call the refresh() method of each control that is associated with the window if the instance variable ::IUpdate of the control is .T.. This allows you to update all the controls (for which the ::IUpdate instance variable is .T.) on a particular window with just one call. The ::IUpdate instance variable is normally set whenever you use the keyword UPDATE when defining/redefining a control.

IValid

Check valid condition

Syntax:

TWindow:IValid() \\\

Purpose:

The IValid() method will check to see if the ::bValid instance variable, when evaluated, returns .T., and returns the result. If there is no code block in the ::bValid instance variable IValid() will just return .T.. This method will NOT end the window, it just checks the ::bValid codeblock (if defined). To end a window use the End() method

VScroll

Generic vertical scroll dispatcher

TWindow:VScroll(nWParam, nLParam) \\\ 0

Processes the WM_VSCROLL messages that are sent to this window object. This method handles both the messages for both a scrollbar that is connected to a window or for a particular vertical scrollbar control.

Parameters:

<nLParam> The high word of this parameter contains the handle of the scrollbar.

<nWParam> This parameter contains a numerical value that specifies the action that should be taken. The following messages are processed (these constants are defined in WINAPI.CH):

SB_LINEUP

SB_LINEDOWN

SB_PAGEUP

SB_PAGEDOWN

SB_THUMBPOSITION

SB_ENDSCROLL (only if the method is called for a scrollbar that is connected to a window).

nVertRes()	Vertical resolution of window in pixels
-------------------	---

Syntax:

TWindow:nVertRes() \\\

Purpose:

This method returns the height of the window in pixels.

nHorzRes	Horizontal resolution of window in pixels
-----------------	---

Syntax:

TWindow:nHorzRes() \\\

Purpose:

This method returns the width of the window in pixels.

AEvalWhen()	Evaluate when clauses for controls when starting window
--------------------	---

Syntax:

TWindow:AEvalWhen() \\\

Purpose:

The AEvalWhen() method evaluates the ::bWhen instance variable (only if ::bWhen contains a code block) from each control that is associated with this window. When the evaluation of the ::bWhen codeblock (of a particular control) results in .F. that particular control is disabled (ie it will not receive focus when clicked on, and can not be reached by using the keyboard).

VbxFireEvent	A VBX event was triggered
---------------------	---------------------------

Syntax:

TWindow:VbxFireEvent(<pEventInfo>) \\\

Parameters:

<pEventInfo> The <pEventInfo> is a 'pointer' to an event structure used for VBX controls by BIVBX10.DLL.

Purpose:

This method is called by the eventhandler whenever a VBX control generates a WM_VBXFIREEVENT message. The method receives the required information via a pointer to a VBX structure (<pEventInfo>).

FOROS DE NOTICIAS

Este es un capítulo muy amplio. Es el clásico apartado de “**Varios**” que termina con entidad propia y que se llena de múltiples cosas que no sabemos donde poner. Hay desde preguntas y respuestas (FAQ) aparecidas en los foros de noticias de Internet, que creo interesante, hasta direcciones WEB para poder ampliar conocimientos y/o programas. No confundir con el capítulo dedicado a la programación en Interne

FORO DE NOTICIAS

Estas son algunos de los diálogos que he encontrado en el foro de noticias **local.fivewin.spanish**. No se indica el autor por discrepción, debido a no tener autorización para ello. Si alguien se ve reflejado y quiere que aparezca su nombre, puede decírmelo y lo incluiré en el aparatdo de créditos.

Se ha eliminado el saludo de entrada (Hola foro, Algún colega, etc) y de salida (Saludos, Muchas gracias, etc) para mayor claridad. Si de una pregunta hay varias respuestas y yo personalmente no conozco la respuesta, he incluido solo la que a mi entender esta mas clara, esto no quiere decir que sea la mejor. Si una respuesta está probada, incluyo la indicación **OK**

¿ PUEDE FIVEWIN TRABAJAR CON LOS NOMBRES DE ARCHIVO LARGOS PROPIOS DE WINDOWS 95 ?

Inicialmente, no, porque las funciones que FiveWin utiliza para el manejo de ficheros son las originarias de clipper y éste sólo está preparado para trabajar con el sistema fat, el Vfat.

Pero este pequeño inconveniente se puede arreglar utilizando una librería que parchee las funciones originarias de clipper, como la Dark Black Long FileName.

Dblfn es una librería que situaremos en nuestro fichero .lnk antes de la librería clipper.lib utilizando la librería Search, por ejemplo:

```
file test
search dblfn
lib clipper, extend
```

En cuanto a los programas, no necesitamos realizar ningún cambio, salvo tener en cuenta que la longitud máxima alcanzada por un fichero ya no son 13 caracteres (8 más el punto y la extensión) sino 256.

¿ Que es Harbour ?

Harbour es una iniciativa surgida de un grupo de programadores en Clipper y FiveWin para desarrollar un compilador del lenguaje Xbase compatible con CA-Clipper.

El proyecto Harbour nace tras el anuncio de Computer Associates de dejar de desarrollar nuevas versiones de Clipper, lo cual fuerza a los programadores en este lenguaje a buscar nuevos productos y nuevas formas de programación.

Principales características del proyecto Harbour:

Compilador compatible con CA-Clipper 5.2e

32 bits (Clipper es un compilador de 16 bits)

Multiplataforma (DOS, Windows, Unix, OS/2)

Gratuito y con licencia GLP (se entrega el código fuente)

Como vemos, Harbour es un sustituto de clipper.exe y sus librerías originales.

En realidad, lo podemos considerar como un puente entre los lenguajes Xbase y el C, ya que lo que nos permite es generar código C a partir de un programa escrito en Clipper.

Este hecho dota a Harbour de una gran portabilidad, ya que el código C generado puede ser compilado con cualquier sistema operativo que tenga un compilador C (todos)

Así, necesitaremos un compilador de C para construir nuestros ejecutables. Harbour recomienda utilizar el compilador GNU GCC <http://www.gnu.org/software/gcc> si bien puede usarse cualquier compilador como Borland C, Microsoft Visual C++, Visual Age, etc..

Actualmente Harbour está todavía en fase de proyecto, esto es, aún no están implementadas todas las funciones propias de Clipper, ni se trata de un paquete "cerrado"

Más información en la página oficial del Proyecto Harbour en castellano

<http://www.geocities.com/SiliconValley/Board/5300/>

¿ Se puede cambiar el color de un group box .Lo tengo definido en una dll, segun el workshop es un "Button" pero como le cambio el color ?

```
REDEFINE GROUP oGroup ID 301 OF ::oDlg COLOR <TuColorRGB>
```

y si quieres cambiar la fuente del mismo, es así:

```
oGroup:SetFont( oFont )
```

¿ Alguien podría ayudarme con los menus popups. Los ejemplos de FIVEWIN no funcionan correctamente y no he podido implementar un menu popup ?

Para hacer un menu popup solo tienes que agregar la Clausula POPUP al definir tu Menu.

```
MENU oMenu POPUP
```

```
    MENUITEM "Opcion Uno" ACTION MsgInfo("Hola")
```

```
    MENUITEM "Opcion Dos" ACTION MsgInfo("Bienvenida al Foro")
```

```
ENDMENU
```

y después lo activas

```
ACTIVATE POPUP oMenu AT nRow, nCol OF oWnd
```

Una vez de definido un report, ¿ como se puede hacer que en base a una variable se active o no el preview o se envíe el listado a un fichero de disco.?

Defines tu reporte como lo haces normalmente, y le pones a la definicion del mismo PREVIEW aunque no lo quieras usar, así:

```
REPORT oReport CAPTION "Reporte del Catálogo de Cuentas";
HEADER NOMBRE_EMPRESA, " " CENTER;
TITLE "Reporte del catálogo de cuentas al ";
+CIBDTOSTxt( DATE() )+SPACE(90)+"Página:"+LTRIM(STR(oReport:nPage)), " " LEFT;
PREVIEW FOOTER VERSION LEFT FONT oFont2, oFont1
```

Inmediatamente después y ANTES del activate report haces lo siguiente:

```
oReport:lScreen := IIF(nScreen = 1 ,.T. ,.F.)
oReport:lPrinter := .T.
```

El objeto reporte tiene una variable de instancia **IScreen** que indica si va a la impresora o directo a la impresora, si la pones a .F. va sin escalas directo a la impresora, la variable **lPrinter** indica si va a la impresora, por default lo pones a .T. pero si quieres que vaya a un archivo, lo pones a .F. y asunto arreglado.

¿ Existe alguna función que devuelva la lista de impresoras instaladas ? He leído una conversación reciente donde se hablaba de "GetPrinters()", pero no existe :-)

Ahí va el GetPrinters

```
Function GetPrinters()
```

```
/* -----
```

```
    Crea un array con los nombres de las impresoras del sistema
    Devuelve : aPrinters. El array creado
```

```
-----
```

```
    Última revisión : 10/07/00 20:27
```

Revisado por : Angel

```

*/
Local aPrinters:={}, oIni, cText, cDevice
oIni := tTxtFile():New(GetWinDir()+"\WIN.INI")

If !oIni:Seek("[DEVICES]",0,1)
    oIni:End()
    MsgBox ("No hay impresoras instaladas en el sistema !!")
Else
    oIni:Advance()
    cText := oIni:ReadLine()

    While !empty(cText) .and. !["$cText
        cDevice := StrToken( cText, 1, "=" )
        cText := StrToken( cText, 2, "=" )
        cDevice += "," + StrToken( cText, 1, "," )
        cDevice += "," + StrToken( cText, 2, "," )
        aAdd (aPrinters,cDevice)

        oIni:Advance()
        cText := oIni:ReadLine()
    Enddo
Endif
Return (aPrinters)

```

Se basa en el archivo WIN.INI

¿ Puedo cambiar, sin usar "PrinterSetup()", la impresora por la que deseo imprimir ?

Usando la clausula FROM USER te permite escoger la impresora que tu desees, incluso en ambitos de RED.

PRINT oPrn NAME "Recibos de Nominas" FROM USER

¿ Alguien tiene idea el porque una la barra de herramientas no muestra los iconos (que estan en una DLL) con 56 colores, siempre lo muestra con 16 colores. ?

El workshop NO PUEDE EDITAR BITMAPS DE 256 colores, si tu tienes un bitmap de 256 colores y lo editas con el Workshop entonces destruyes la paleta de colores original del bitmap y por ende su combinacion de colores.

Lo que yo aconsejo para trabajar con bitmaps 256 es utilizar un editor externo (el mismo paint puede servir) y dejar el archivo como bmp; posteriormente incorporarlo al Workshop con ADD TO PROJECT del menu FILE y una vez incorporado olvidarlo, es decir, ni se te ocurra tocarlo, el bitmap conservara su paleta de colores y podrás usarlo con toda la gama de colores, pero si se te ocurre aunque sea por equivocación abrir el editor de bitmaps del workshop con la imagen a 256 colores, entonces habras hechado a perder la imagen.

¿Hay alguna instruccion (Funcion, metodo, etc) para saber la version de FW utilizada en un programa ? O mejor aun, ¿hay alguna forma de saber version de FW se utilizo en un EXE.?

Hay una constante predefinida en FIVEWIN.CH llamada **FWVERSION** que te indique version de FW estas usando, ejemplo: **gInfo(FWVERSION)**

Hola quisiera saber que DLL tengo que instalar en casa de un cliente para una aplicación en FW 2.0. De entrada se: **preview.dll, **infounz.dll**, **bivbx10.dll** (hago servir el chart2fx.vbx) y la wizzip16.dll (hago servir funciones para comprimir y descomprimir archivos zip). Hay alguna más ? Donde hay que colocarlas, en el directorio de la aplicación ó en el %windir%\system ? El programa de instalación es suficiente que las copie en esos directorios o hacer falta hacer algo más como registrarlas ? Como se registran con el InstallShield por ejemplo ?**

Yo concretamente pongo esas que tu dices y **CTL3D.DLL**, **INFOUNZ.DLL** y **WCC.DLL**, esta última renombrada para no tener problemas. Colocalas en el directorio de la aplicación y listo. Con el instalador solo hace falta que las copies al directorio de la aplicación.

Esta inquietud no es necesariamente sobre FW, pero de todas maneras la expongo acá. ¿Como puedo hacer para conectar dos pc's por medio de la linea telefónica?, la idea es poder actualizar bases de datos desde una pc remota. Utilizo Windows 98.

La manera mas fácil, y económica y Super Segura (ya tienes el software para ello), es con El Servidor de Acceso Telefonico a Redes, del Win98. La máquina que contesta la llamada es el Servidor de Acceso Telefonico. La máquina que hace la llamada es el Cliente.

Cuando ambas máquinas esten conectadas (las dos tienen que tener el mismo protocolo y el cliente de redes de Microsoft, y perfectamente bien configurado), solo tienes que en la máquina Cliente correr el Explorer, luego te metes en Herramientas, luego en Conectar a unidad de Red (Map Network Drive), allí tendrás que escribir la ruta de acceso (a la máquina servidora y su recurso compartido (nombre de carpeta compartida)), asignas una letra de Drive, y listo: Ya tienes acceso en la máquina cliente a una letra de drive (que no es mas que la carpeta compartida de la Pc servidora, el cual puedes ya tratarla como otro disco duro). Podrás hacer Copy, XCopy, MD, CD, todo lo que haces con una unidad de red, y/o letra de drive. o tengo experiencia de casi 5 años con este tipo de trabajo y resulta realmente sencillo transmitir datos bidireccionalmente entre sucursales de empresas. Con Win95 desde 1995, hago esto. El servidor de acceso telefonico a redes venía en el paquete Plus! del Win95, en Win98 ya viene en el CD principal.

¿ Como puedo hacer para que un programa solo funcione durante 30 dias y después deje de funcionar ? He creado una funcion en archivo ini que me funciona bien pero claro si el cliente cambia la fecha de sistema el programa vuelve a funcionar .

Esto usualmente resulta infalible: La primera vez que se ejecute tu programa busca en el directorio \windows\system un fichero que se llame: <tunombrefavorito>.DLL

Puedes ponerle el nombre que quieras y la extension que quieras, yo le pongo DLL porque no conozco ningun valiente que se atreva a quitar un DLL del \windows\system nada mas por que le sale de las narices. Este fichero no es en realidad un DLL es un fichero .MEM (de esos que se crean con SAVE TO <tunombrefavorito.DLL> ALL LIKE <nombredevars>).

Vamos a suponer que es la primera vez que se ejecuta tu programa, el programa va y busca en el directorio \windows\system el fichero <tunombrefavorito.DLL>, asi:

```
IF .NOT. FILE("c:\windows\system\<tunombrefavorito>.dll")
```

Vas a hacer lo siguiente:

```
dFechInst := DATE() // guarda la fecha del dia de hoy (instalacion)
dFechVen:= DATE()+<numerodediasdeprueba> // fecha de vencimiento
dFechUsal:= DATE() // fecha de la ultima salida del sistema.
```

Y luego vas a guardar las variables en el fichero MEM disfrazado de DLL:

```
SAVE TO C:\windows\system\<tunombrefavorito.dll> ALL LIKE dFech*
```

Esto generara el fichero MEM disfrazado del DLL en el \windows\system.

Una vez creado el fichero, cada vez que entres al programa recuperas las fechas con un RESTORE FROM c:\windows\system\<tunombrefavorito.dll>, haces la comparacion entre la fecha del dia (DATE()) contra la fecha de vencimiento (dFechVen) y contra la fecha de la ultima salida (dFechUSal) y si no es mayor, le dejas seguir usando el programa.

Cuando el programa se cierre (en el VALID de la ventan principal lo puedes hacer), vas a actualizar la variable dFechUSal, la cual vas a igualar a la fecha del DIA, y vas a actualizar el fichero tunombrefavorito.dll> esta variable guarda la ultima fecha en que salio del programa.

Vamos a suponer que el periodo de evaluación terminó, el usuario entra al programa, se hacen las comparaciones entre la fecha de vencimiento la fecha del dia y si es el último dia se le avisa que su periodo de evaluación ha terminado y que el programa finalizara, en el momento que termina el programa se actualiza nuevamente dFechUsal y <tunombrefavorito.dll>.

Aquí el usuario puede hacer varias cosas:

- 1) Cambiar la fecha del sistema e intentar ejecutar el programa: Solución: tenemos guardada la ultima fecha de salida, y recuerda que la comparación se hace al momento de entrar al programa contra la fecha el dia y la fecha de ultima salida
- 2) Borra todo y vuelve a instalar el Software Solucion: No hay problema porque el fichero <tunombrefarito.dll> no esta en el directorio donde instalaste el software sino dentro del \windows\system,

es importante asegurarse que el sea el programa el que cree el fichero <tunombrefavorito.dll>, no se ocurra ponerlo en el programa de instalacion.

3) Borrar el fichero <tunombrefavorito.dll> pero de entrada tendría que saber como se llama el susodicho fichero y donde esta. y como te comento, no conozco a ningun valiente que se atreva a quitar un fichero DLL del \windows\system.

Si quieres mayor despiste para el usuario, puedes guardar el fichero con fecha diferente, por ejemplo:

```
dDate:=Date()  
SetDate(21,04,1997)      file://fecha al azar para despistar  
SAVE TO &Archi ALL LIKE dFech*  
SetDate(Day(dDate),Month(dDate),Year(dDate))    // fecha actual
```

ENLACES EN INTERNET

<http://www.tafweb.com>

Programas de utilidad sobre la edición y mantenimiento de páginas web. También tiene como programa interesante un editor para un programa de instalación/desinstalación **Inno Setup**.

<http://www.jordanr.cjb.net>

Instalador/desinstalador de aplicaciones. Funciona con un fichero script que puede crearse con la utilidad indicada en el enlace www.tafweb.com.

<http://www.ciber-tec.com/fwgal.htm>

Galeria de imágenes de programas hechos con Fivewin.

<http://www.ciber-tec.com>

Página comercial de la empresa Cibernética y Tecnología, S.A. de México.

APENDICES

El contenido de este capítulo, quizás no tengan mucho que ver con Fivewin, pero me resisto a no ponerlo. Seguro que te puede servir de ayuda en un momento dado y evitarte los calentamientos de abeza que yo tuve que pasar.

A – INDICES CDX

Nota.- Este apéndice corresponde a la transcripción parcial de un documento interno de **ARG Consulting**, que a su vez tiene incluidos total o parcialmente otros documentos de dominio público. Se ha trasladado a este libro con mínimos cambios. Es por este motivo por el que pueden aparecer términos específicos del sistema de desarrollo de esta empresa así como temas no relacionados con Fivewin, pero que quizás puedan ser interesante. Se ruega al lector sea comprensible.

FICHEROS CDX

- ♦ Los drivers RDD de los índices .CDX, en la 5.2 eran de Successware SIX driver mientras que los de la 5.3 son de COMIX, mucho mejores, estables y fiables.
- ♦ No es buena idea mezclar en una misma aplicación, ni incluso en un mismo programador los dos formatos de índices. Se debe decidir que formato usar y una vez definido hacer TODO para ese tipo de índice. El sistema permite mezclar, incluso en el mismo .PRG los dos tipos, pero el rendimiento del programador baja considerablemente. Aquí tomamos la decisión de pasar TODO a .CDX por lo que no se explican en estas notas las instrucciones que permite mezclar los dos índices. Para lograr esto, lo mejor es modificar el fichero **rddsys.prg** como se indica en otro apartado más abajo, y enlazarlo siempre con el .LNK del blinker. De esta forma SIEMPRE pone por defecto el sistema .CDX.
- ♦ Los índices .CDX son compatibles con el sistema FoxPro 2, aunque no en todo.
- ♦ Las posibilidades de estos índices son:
 - ♦ Indices compactos, es decir, están comprimidos (similar a un PKZIP, aunque no con el mismo algoritmo)
 - ♦ Indices compuestos, es decir, puede haber varios índices en un solo fichero, hasta un máximo de 99, si bien el límite práctico esta en 50.

- ◆ Índices condicionales, es decir, solo se incorporan al índice los registros que cumplan una condición.
- ◆ Campos memos más pequeños y mejor tratados que los DBT. De todas formas nosotros utilizamos los .DBV de Flexfile puesto que los ficheros dBase que utilizan los campos memo .FPT, no son visibles con el fox.
- ◆ Ver en la 2ª parte de estos apuntes, las incompatibilidades indirectas en el sistema clipper.
- ◆ Explicaciones y fijado de términos:
- ◆ **Fichero:** es el fichero .CDX, por ejemplo, CLIENTES.CDX. Este fichero contiene los distintos índices. El término en inglés sería BAG o bolsa. Aquí lo denominaremos "ficheros contenedor", o simplemente fichero, en alusión al sistema de ficheros de MS-DOS o WINDOWS.
- ◆ **Índice:** Es el índice propiamente dicho. NO es un fichero. El índice esta contenido dentro del fichero .CDX. No hay que confundir fichero (contenedor) con índice (contenido). Puede haber varios, o muchos, índices en un fichero. El término en inglés es TAG. De lo dicho anteriormente, se deduce, que no es correcta la expresión "fichero índice"
- ◆ A menos que se indique expresamente lo contrario por medio de SET AUTOPEN OFF, al abrir un .DBF, siempre se abre su .CDX en el mismo directorio, aunque no tenga nada que ver, solo con el hecho de llamarse igual. Si no existe, no da error y solo abre el .DBF. Puesto que en el fichero .CDX están todos los índices, al abrir el fichero .DBF se abren TODOS los índices. Igualmente, si se cambia el contenido del registro, se actualizan también TODOS los índices.
- ◆ Unos comandos completos serían:

INDEX ON CODIGO TAG CLIEN1 TO CLIENTES

Crearé un índice (TAG) con la denominación CLIEN1 en un fichero con nombre CLIENTES.CDX.

Supongamos que vamos quitando partes a la instrucción y dejando valores por defecto, por ejemplo, no indicar el fichero:

INDEX ON CODIGO TAG CLIEN1

Crearé un índice con la denominación CLIEN1 en el fichero de índices actualmente abierto. Si no hay ninguno, creará un fichero .CDX con el mismo nombre y directorio que la base de datos actual. Si existiendo fichero .CDX ya tenía índices, añadirá el nuevo (CLIEN1) al fichero. Si ya existía uno con el mismo nombre, lo deja tal como esta (pero inactivo) y crea otro con el mismo nombre. El anterior del mismo nombre **no sirve para nada**, solo para ocupar espacio.

Si por medio de una secuencia de instrucciones o un bucle, creamos varios índices, actuaría de la siguiente forma:

```
//abre el fichero .DBF y supondremos que no existe el .CDX
abre('CLIEN')
index on codigo tag clien1 //crea CLIEN.CDX y pone CLIEN1
index on nombre tag clien2 //añade a CLIEN.CDX el índice CLIEN2
index on NIF tag clien3 //añade a CLIEN.CDX el índice CLIEN3
etc.
```

Si por error después de crear los tres índices continuamos con:

```
index on NIF tag clien3 // deja CLIEN3 anterior y crea otro CLIEN3
```

Ahora supongamos que no indicamos el índice pero si el fichero:

INDEX ON CODIGO TO CLIEN1

Crearé, ni no existe, el fichero CLIEN1.CDX y dentro pondré el índice CLIEN1. Si ya existiese CLIEN1.CDX, pondré el índice CLIEN1 en él. Si a su vez, también existía dentro de CLIEN1.CDX el índice CLIEN1, lo dejara inservible y crearé otro índice CLIEN1. De esta forma, si hacemos el siguiente código:

```
Abre('CLIEN')           // abre la base de datos.  
index on CODIGO to CLIEN1//crea CLIEN1.CDX y pone el indice CLIEN1  
index on NOMBRE to CLIEN2//crea CLIEN2.CDX y pone el indice CLIEN2
```

Esta forma sería similar al anterior sistema NTX, pero se pierde la ventaja de fichero único, apertura automática, etc.

Como regla general rápida, podemos sustituir TO por TAG y el sistema se encarga del resto.

- ♦ Una ventaja de los índices CDX es la despreocupación por parte del programador de estos índices, en el desarrollo normal de la aplicación. Solo hay que tenerlos en cuenta cuando se crean. Por ejemplo, al crear el fichero con dbcreate() o en rutinas espaciales, como fiche.prg, cuando se regeneran, crean, etc.-
- ♦ Puesto que su apertura es automática, siempre se abren los índices correctos y la posibilidad de corrupción es mínima, por ejemplo no existe el "internal error 19".
- ♦ Al existir un fichero para varios índices, se reduce el número de ficheros abiertos, consumiendo menos FILES del CONFIG.SYS. También al ser menos ficheros, se abren las bases de datos más rápidamente.
- ♦ En .CDX se pueden hacer índices condicionales, de modo que se puede emplear la condición FOR. Esta condición se graba junto al índice, de modo que solo se crean los punteros a los registros que cumplan la condición. Por ejemplo:

```
abre('CLIENTES')  
index on CODIGO tag CONDICION for PROVI='MADRID'
```

Esto crea un fichero (si no existía ya) CLIENTES.CDX y pone dentro el índice (TAG) CODICION. En este índice solo se graban las referencias a los registros que el campo PROVI sea MADRID. Si no hay ninguno con esta condición, aunque el fichero sea muy grande, el índice estará vacío.

Como es lógico, en un mismo fichero pueden convivir índices normales e índices condicionales.

- ♦ Hacer un índice nuevo con la instrucción

```
index on ....
```

si el índice (TAG) ya existía con el mismo nombre, NO borra el anterior, de modo que no aprovecha el espacio que ocupaba el viejo. La instrucción REINDEX "empaqueta" el fichero .CDX y borra los índices antiguos.

Ejemplos:

1 Creación de una base de datos con dos índices:

```
Mdbf := {}  
aadd(mDbf, {'CODIGO', 'N', 6, 0})  
aadd(mDbf, {'NOMBRE', 'N', 6, 0})  
aadd(mDbf, {'DIRE', 'N', 6, 0})  
aadd(mDbf, {'POBLA', 'N', 6, 0})  
aadd(mDbf, {'PROVI', 'N', 6, 0})  
dbcreate(mDbf, 'CLIENTES')  
abre('CLIENTES')  
index on CODIGO tag CLIEN1  
index on NOMBRE tag CLIEN2  
cierra('CLIENTES')
```

2 Puesto que ahora los índices están dentro de ficheros, puede haber un índice de clientes denominado CODIGO dentro de CLIENTES.CDX y otro índice de proveedores, también denominado CODIGO dentro de PROVE.CDX:

```
abre('CLIENTES')  
index on CODIGO tag CODIGO // este sera CODIGO dentro de CLIENTES  
index on NOMBRE tag NOMBRE // este sera NOMBRE dentro de CLIENTES  
abre('PROVE')  
index on CODIGO tag CODIGO // este sera CODIGO dentro de PROVE  
index on NOMBRE tag NOMBRE // este sera NOMBRE dentro de PROVE
```

La utilización de nombres claros en los índices, en lugar de CLIEN1, CLIEN2, etc. clarifica los programas, puesto que es más explicativo:

```
select CLIENTES  
set order to NOMBRE
```

Que:

```
select CLIENTES  
set order to 2
```

INSTRUCCIONES PROPIAS DE INDICES

DBCLEARINDEX()	Cierra el índice abierto.
ORDBAGEXT()	Retorna la extensión de los ficheros índices.
ORDBAGNAME()	Retorna el nombre del fichero de índices.
ORDCONDSET()	Pone la condición y alcance de un scope
ORDCREATE()	Crea un nuevo índice dentro de un fichero CDX
ORDDESCEND()	Retorna y opcionalmente cambia la bandera descendente.
ORDDESTROY()	Quita un índice (TAG) dentro de un fichero (BAG)
ORDESUNIQUE()	Retorna el estado de la bandera de único.
ORDFOR()	Retorna la condición FOR de un índice.
ORDKEY()	Retorna la clave de un índice.
ORDKEYADD()	Añade un clave a un índice personalizado.
ORDKEYCOUNT()	Retorna el número de claves dentro de un índice.
ORDKEYDEL()	Borra una clave en un índice personalizado.
ORDKEYGOTO()	Salta a un registro en función de la posición lógica dentro de un índice
ORDKEYNO()	Retorna la posición lógica dentro del índice. (se utiliza para barras de proceso, etc.)
ORDKEYVAL()	Retorna el valor de la clave actual.
ORDLISTADD()	
ORDLISTCLEAR()	
ORDLISTREBUILD()	
ORDNAME()	Retorna el nombre del índice (TAG). No confundir con el BAG.
ORDNUMBER()	Retorna la posición de un índice en la actual lista de índices.
ORDSCOPE()	
ORDSETRELATION()	
ORDSKIPUNIQUE()	
ORDSETFOCUS()	
SET AUTOPEN	Abre automáticamente (o no) el .CDX al abrir el .DBF

SET AUTOPEN ON | OFF permite abrir o no de forma automática los índices CDX relacionados con el fichero DBF. Normalmente, por defecto esta puesto ON de modo que al abrir el DBF también abre el índice correspondiente. Poniéndolo OFF, se abre el fichero de datos pero NO el índice. De esta última forma es como funcionan los índices normales NTX.

SET SCOPE

SET SCOPEBOTTOM

SET SCOPETOP

SEEK

Notas traducidas del original de **Jo W. French**(copiado de Oasis)

INDICES

Pueden ser creados dos tipos de índices compuestos:

- Índices estructurales o compuestos.
- Índices independientes

INDICES ESTRUCTURALES .- Estos índices tienen el mismo nombre que el fichero .DBF. Normalmente son creados de forma automática por un comando que incluye la frase TAG <nombre del índice> y NO incluye la frase TO <nombre del fichero>. P.e.

INDEX ON <expresión> TAG <nombre_del_índice>

Puede incluir **TO <nombre_del_fichero>** si este es el mismo que el .DBF. P.E.

INDEX ON <expresión> TAG <nombre_del_índice> TO <nombre_fichero_dbf>

Si se pone:

INDEX ON <expresión> TO <nombre_fichero_dbf>, hay que tener cuidado, puesto que esta forma borra cualquier .CDX con el mismo nombre, cosa que no pasa en los dos casos anteriores donde se incluye la parte **TAG <nombre_del_fichero>**.

Si se incluye la frase TAG y existe uno con el mismo nombre, dentro del fichero, el actual TAG se anula, pero ocupando el mismo sitio, y se crea y activa otro nuevo, que ocupa un nuevo orden dentro de la estructura de los índices, es decir, si había dos índices (TAG's) en el fichero y se crea un TAG con el mismo nombre del primero, el nuevo estará en segundo orden, y el que antes era segundo ahora será el primero. Si se activan los índices por nombre, no importa, pero si se hacen por su posición física si tiene importancia. Naturalmente esto no importa si el viejo TAG ya era el último o era el único. Este sistema de hacer índices poniendo TAG ya existentes, aumenta el tamaño del fichero .CDX.

Si al abrir un fichero .DBF existe un índice estructural .CDX con el mismo nombre, el fichero se abre automáticamente a menos que hagamos algo por impedirlo, pero por defecto siempre se abrirá el índice y el orden por defecto será el número de registro. Esto puede causar algunos problemas

Clipper nos brinda funciones para cambiar el automatismo de la apertura:

SET AUTOPEN ON | OFF

SET AUTOPEN (IVar)

Set(_SET_AUTOPEN, .T., .F.)

Donde ON es el método por defecto. Nota: si AUTOPEN esta OFF, dbClearIndex() deberá cerrar el índice que tenga el mismo nombre que el .dbf.

SET AUTORDER TO nOrdNumber o

Set(_SET_AUTORDER,nOrdNumber)

Donde 0 es el orden natural de registros (por defecto). Nota: En ficheros estructurales abiertos con SET INDEX TO por defecto siempre es el primer índice, a menos que se haya modificado con AUTOPEN o AUTORDER. Recordar que al abrir un fichero DBF con índice asociado, el puntero de registros se posiciona en el **primer registro físico**, no el el primer registro según el índice. Después de abrir hay que hacer un **go top** si se quiere empezar por el primero.

Re-crear un fichero estructural (lo que sería un REINDEX en NTX), puede hacerse de varias formas:

Si AUTOPEN esta ON y el fichero .DBF asociado esta abierto, (lo más normal)

- A) Cerrar el .DBF, borrar el .CDX y abrir de nuevo el DBF
- B) OrdDestroy() todos los TAG existentes de la siguiente forma:
 DO WHILE !EMPTY(cTag := ordName(1))
 OrdDestroy(cTag)
 ENDDO

A y B: Entonces hacer los apropiados INDEX ON....

Si AUTOPEN esta OFF y el fichero .DBF asociado esta abierto,

Cerrar todos los índices usando dbClearIndex()

Entonces hacer los apropiados INDEX ON....

Si un índice estructural esta abierto, la instrucción REINDEX puede también usarse para re-crear el índice.. Esto también “empaqueta” el fichero .CDX y recupera espacio oculado por TAG inactivos o borrados.

MISCELANEAS:

Un byte en la cabecera del fichero .DBF es puesto a CHR(1) cuando un índice estructural es automáticamente creado. Es puesto a CHR(0) cuando el índice es borrado anulando todos los TAG's en el fichero. NOTA: Cliepper 5.3 falla al poner/quitar este byte.

FoxPro ignora los ficheros índices con el mismo nombre, a menos que el indicador de la cabecera haya sido puesto. Por esto, si FoxPro es usado para acceder a fichero .CDX hechos en clipper, es necesario manipular este byte por programación utilizando funciones de bajo nivel.

INDICES INDEPENDIENTES .- Un índice independiente tiene un nombre distinto al fichero .DBF. Siempre es creado manualmente con intrucciones apropiadas que incluyen la frase **TO <nombre_de_fichero_distinto_al_DBF>**. Si la frase TAG es omitida, se usa un TAG con el mismo nombre del fichero índice. Este tipo de índice no se abren automáticamente al no tener el mismo nombre del fichero .DBF.

No es aconsejable el sistema de índices independientes en formato CDX puesto que se pierden algunas de las ventajas de estos índices.

OPTIMIZACION del sistema CDX.

El RDD _DBFCDX utiliza automáticamente los índices, para determinar que registros cumplen una condición sin necesidad de acceder a la base de datos. El RDD, también provee Optimización Lineal que aumenta considerablemente el rendimiento, por ejemplo, no es necesario añadir a la clave de los índices STR(RECNO())

Otro método de optimización es que en las búsquedas, etc. utiliza el índice siempre que pueda en lugar de recurrir a la base de datos, que normalmente será mas grande que el índice. Por ejemplo, si existe un índice por EDAD, al hacer una búsqueda, selección, etc. de EDAD=30, no recurre a la base de datos, solo al índice, optimizando la operación.

Igualmente ocurre en las condiciones FOR cuando se utilizan índices condicionales que han incluido la condición (excepto si se usa WHILE, USECURRENT o NOOPTIMEZE).

La instrucción SET FILTER TO funciona igual que el sistema standard, pero mucho más rápido. Recordar que las variables dentro de la instrucción SET FILTER no pueden ser local ni static puesto que no han podido ser incluidas en el índice cuando se crearon estos. Para evitarlo, puede hacer:

a) poner las variables private o public

b) usar NOOPTIMEZE desde el comienzo.

c) convertir el valor de la variable en una constante de carácter equivalente que pueda ser concatenada como parte de un codeblock

Ejemplos:

```
local nAge := 30
index on age tag age
dbSetFilter(  &( "{|| age >= " + Str( nAge, 2 ) + "}" ), ;
              "age >= " + Str( nAge, 2 ) )

local dDate := DATE()
index on date tag date
dbSetFilter(  &( "{|| date < ctod('" + dtoc( dDate ) + "')} " ), ;
              "date < ctod('" + dtoc( dDate ) + "')" )
```

Nota: ver el apéndice sobre la función data2str().

```
dbsetfilter( &( "{|| date < " + data2str( dDate ) + "}" ), ;
              "date < " + data2str( dDate ) )
```

El sistema CDX, nunca utiliza optimización cuando en el índice se incluye una condición FOR.

Si la expresión de búsqueda, etc. coincide exactamente con la expresión de uno de los índices, se utiliza optimización. El uso de paréntesis, espacios, etc. produce una expresión equivalente.

Ejemplos:

```
index on edad tag edad
dbSetFilter( {||edad = 30 }, "edad = 30")           // Optimizable

index on apellido tag apellido
dbSetFilter( {|| apellido = 'J'}, "apellido = 'J'") // Optimizable

index on apellido+nombre+provincia tag anp
dbSetFilter( {||(apellido + nombre) + provincia = 'C'}, ;
            "(apellido + nombre) + provincia = 'C'" ) // Optimizable

index on iif(married, 'M' + last, 'S' + last) tag ifMarried
dbSetFilter( {||iif(married, 'M'+last, 'S'+last) = 'M'}, ;
            "iif(married, 'M'+last, 'S'+last) = 'M'" ) // Optimizable
```

NOMBRES DE FUNCIONES ABREVIADOS:

El RDD _DBFCDX automáticamente utiliza el nombre abreviado de las funciones. P. e., SUBSTR() es sinónimo de SUBS() y SUBST(); DELETED() es sinónimo de DELE(), DELET() y DELETE(); etc.

Ejemplos:

```
index on substr(apellido,4,3) tag apellido
dbSetFilter( {||subs(apellido,4,3) = 'abc'}, ;
            "subs(apellido,4,3) = 'abc'" )           // Optimizable
```

Leading Character Expressions

Given an index on A+B+C (where A, B, and C are all character strings), The _DBFCDX RDD will be able to optimize a query expression using A, A+B, or A+B+C. In other words, it can optimize any leading part of the key.

The _DBFCDX RDD cannot, however, optimize an embedded (non-leading) part of the key. In the example of the key A+B+C, an expression on B or B+C would not be optimizable.

Ejemplos

```

index on last+first tag lastfirst
dbSetFilter({|| last = 'J' }, "last = 'J'")           // Optimizable
dbSetFilter( {|| last+first = 'Johnson      T'}, ;
              "last+first = 'Johnson      T'" )      // Optimizable

dbSetFilter({|| first = 'J'}, "first = 'J'")           // NOT optimizable
                                                         // (first is not
                                                         // leading part of
                                                         // key)

```

EMPTY()

El _DBFCDX automáticamente optimiza cualquier función EMPTY() producida en la expresión de un índice.

Ejemplos:

```

index on age tag age
dbSetFilter({|| empty(age)}, "empty(age)")             // age is optimizable,
                                                         // so empty(age) is.

```

DTOS()

The _DBFCDX RDD can use an index on DTOS(dDate) to optimize an expression involving dDate. Conversely, the _DBFCDX RDD can use an index on dDate to optimize an expression involving DTOS(dDate).

Ejemplos:

```

index on dtos(lastcall) tag date
dbSetFilter({|| dtos(lastcall) = '1989'}, ;
              "dtos(lastcall) = '1989'" )             // Optimizable

dbSetFilter({|| lastcall = ctod('01/01/89') }, ;
              "lastcall = ctod('01/01/89')")          // Optimizable

index on lastcall tag lastcall
dbSetFilter({|| dtos(lastcall) = '1989'}, ;
              "dtos(lastcall) = '1989'" )             // Optimizable

```

REGISTROS BORRADOS

Por razones de funcionamiento, dbSetFilter(), usado con _DBFCDX siempre asume que SET DELETED esta OFF. En otras palabras, la instrucción SET DELETED.

Los registros que se borran o que están borrados al hacer el índice, se incluyen en éste. Es el sistema quien desechara los registros borrados una vez leídos de la base de datos.

Esto puede conducir a resultados inesperados. Por ejemplo, consideremos el siguiente caso:

```
use demo
index on apellido tag apellido
index on nombre tag nombre
dbSetfilter({|| apellido = 'García' .and. nombre = 'José' }, ;
            "apellido = 'García' .and. first = 'José'")
? OrdKeyCount()      // retorna 29 (incluye los registros borrados)
```

En estas condiciones se hace un Tbrowse y los registros borrados que son leídos son eliminados de la lista_de_registros, por ello OrdKeyCount() devuelve ahora 28.

Para evitar esto, incluir explícitamente “.and. !deleted()” en la condición.

```
dbSetfilter( {|| <condition> .and. !deleted() }, ;
            "<condition> .and. !deleted()")
```

APERTURA DE FICHEROS DBF

El modo en que los ficheros son abiertos pueden ser controlado mediante la instrucción AUTOSHARE

```
SET AUTOSHARE TO nModo
Set( _SET_AUTOSHARE, nModo )
```

Donde 0 es el valor por defecto.

Si nModo es 0, no se hace el control automáticamente. Esta es la forma normal en los ficheros NTX. De esta forma se permite al programador controlar el modo de apertura.

Si nModo es 1, _DBFCDX automáticamente determina en tiempo de ejecución, si la aplicación esta en una red local. Si NO esta funcionando en red, los ficheros serán abiertos automáticamente en modo EXCLUSIVE. Esto permite escribir todo el programa como si fuese para una red y en el momento de ejecución se comportará como monopuesto, sacando provecho de esta circunstancia.

Si nModo es 2, entonces las aperturas se hacen siempre en modo EXCLUSIVE, tanto si esta trabajando en red, como en monopuesto. Esto permite hacer todo el programa como si fuese en red, pero con el cambio de esta simple instrucción, solo funcionará monopuesto, consiguiendo un control simple para demos, etc.

LECTURA ESTRUCTURA DE REGISTROS

La lectura de registros por el _DBFCDX durante la indexación, puede controlarse con la instrucción STRICTREAD

```
SET STRICTREAD ON | OFF
SET STRICTREAD ( IStrictRead )
Set( _SET_STRICTREAD, IStrictRead )
```

IStrictRead es una variable lógica. Si es .F. (por defecto), el RDD, deberá leer los registros directamente del disco al crear los índices. Si es .T., los registros son leídos desde el _DBFCDX.

Esto es necesario, solamente si se utilizan ficheros encriptados (p.e., RDDKit o encriptación NetLib).

Si se utilizan CDX en ficheros encriptados, poner STRICTREAD ON para forzar la lectura de los registros a través del _DBFCDX .

APENDICES:

```
/* Función para Poner/Quitar el byte de la cabecera del dbf. */
FUNCTION IndexByte( cFileName, lSet )
    LOCAL nHandle
    IF( VALTYPE( lSet ) != "L", lSet := .F. , )
    IF !( ( nHandle := FOPEN( cFileName, 2 ) ) == -1 )
        FSEEK( nHandle, 28 )
        FWRITE( nHandle, IIF( lSet, CHR(1), CHR(0) ) )
        FCLOSE( nHandle )
        RETURN .T.
    ENDIF
    RETURN .F.
```

```
/* Función para convertir datos a carácter. */
FUNCTION Data2Str( xVar )
    LOCAL cRet := "", cType := VALTYPE( xVar )
    DO CASE
        CASE ( cType == 'C' ) .OR. ( cType == 'M' )
            cRet := '"' + STRTRAN( xVar, '"', '""' ) + '"'
        CASE ( cType == 'D' )
            cRet := "CTOD('" + DTOC( xVar ) + "')"
        CASE ( cType == 'L' )
            cRet := IIF( xVar, ".T.", ".F." )
        CASE ( cType == 'N' )
            cRet := LTRIM( STR( xVar ) )
    ENDCASE
    RETURN cRet
```

ANOMALIAS:

En Clipper 5.3 la implantación del sistema CDX tiene las siguientes anomalías:

- a) OrdKeyGoto() no esta implementado..
- b) Un error 8006 es generado si se trata de escribir en el registro fantasma (lastrec()+1).
- c) Poniendo SCOPE en un índice en orden natural o cuando este no esta activado, puede causar el error VM Integrity o R6001 Nul Pointer.
- d) El byte de la cabecera .dbf no es puesto/quitado por el driver.
- e) OrdKeyCount() y OrdKeyNo() no trabajan si OPTIMIZE esta OFF o si la variable del filtro es un elemento de un array.
- f) Variables local o static no pueden ser utilizadas en expresiones de filtro si OPTIMIZE esta ON, que es lo más normal.
- g) En DBSEEK(<expKey>, [<ISoftSeek>], [<ILast>]) --> IFound la posibilidad de poner ILast no trabaja. Si se hace una asignación a este parámetro, ISoftSeek también falla.
- h) Un bloqueo permanece activo en el último registro, lastrec(), en un fichero compartido, después de un comando APPEND FROM.

Fin del artículo de Jo W. French

B - FICHEROS DE AYUDA

Nota.- Este apartado esta copiado casi integramente de la documentación de dominio público aparecida en Internet, denominada **Introducción a la Programación Windows** de los autores Francisco García, Jesús Morán y Fernando Ballesteros. Me he limitado a formatear un poco el texto y modificar algunas frases. He preferido mantener las referencias a los párrafos, por ejemplo, **7.1 EL Creación de ficheros....**

7.1 Creación de ficheros de ayuda para Windows.

La creación de un fichero de ayuda para una aplicación Windows debería ser una parte del desarrollo tan importante como la programación. ¿.. y por qué ? Bueno, existen una serie de razones por las que desarrollar una ayuda para Windows puede ser interesante:

- Es una forma de proporcionar a los usuarios una serie de información en línea sensible al contexto (o sea, tener la información que se necesita en el momento en que se necesita). Cuanto mayor sea la información de que dispone el usuario, menor es la probabilidad de que tenga que llamar al servicio de apoyo y mayor es su satisfacción.
- Permite crear sistemas de información mucho más atractivos gracias al soporte de tipos de letras y gráficos incorporables a los ficheros.
- Los ficheros de ayuda pueden ser aumentados o corregidos de forma realmente simple. Además, su coste (comparado con el de los clásicos manuales impresos) es mínimo, y se pueden incluir todos los ejemplos, guías de referencia, tutoriales, etc. que normalmente no es posible incluir en la documentación impresa.

- Es fácil, eficiente, rápido, productivo y, por si fuera poco, las herramientas necesarias para que los usuarios puedan manejar los ficheros de ayuda SON GRATIS (el visualizador es el programa WINHELP.EXE que se incluye con todas las copias de Windows).

Por simple que sea un programa, siempre se debería proporcionar a los usuarios una forma de poder obtener información sobre lo que están haciendo, o recibir ayuda sobre el mejor modo de realizar una determinada acción. Desgraciadamente no es mucha la gente que ve las cosas así, y suministra programas sin ayudas o, como mucho, con un fichero separado (preparado mediante algún procesador de texto, generalmente el Write) que contiene un manual de la aplicación.

Esperamos que este capítulo sirva para enmendar de alguna forma esta práctica tan corriente. Para ello vamos a ir explicando el proceso más cómodo (o a nosotros nos lo parece) para desarrollar un fichero de ayuda, y lo vamos a hacer creado una ayuda para el comando ATTRIB del MS-DOS. Realmente se trata de un proyecto muy pequeño, pero vamos a usar todas las técnicas necesarias para el desarrollo de un gran proyecto.

7.2 Requisitos necesarios para desarrollar un fichero de ayuda

Para el desarrollo de una ayuda para Windows es necesario contar con una serie de herramientas:

- Un compilador de ayudas para Windows. El más corriente es el HC31.EXE. Este se distribuye junto con casi cualquier paquete de desarrollo que sea capaz de generar una aplicación Windows.
- Un procesador de texto con capacidad para generar ficheros en formato RTF (Rich Text Format). Para los ejemplos de este libro utilizaremos el Word para Windows de Microsoft (nos parece el más simple, el más practico para el desarrollo de ayudas y, ademas, es el único que teníamos instalado a la hora de escribir esto). Se utilizará para escribir el contenido de las páginas del fichero de ayuda.
- Un editor de texto que genere ficheros de texto ASCII, sin ningún tipo de código de formato o atributos. Se utilizará para escribir el fichero de proyecto. Solo es necesario en el caso de que la ayuda se encuentre dividida en más de un fichero.

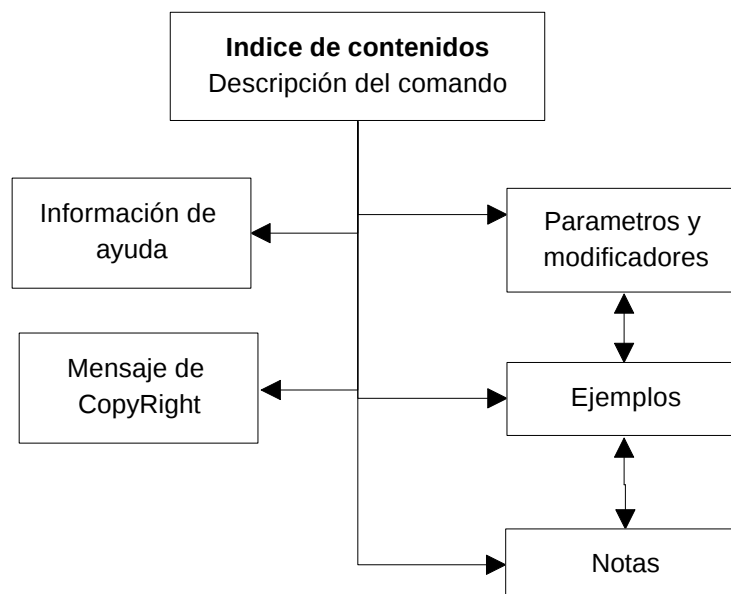
7.3 Diseñar el fichero de ayuda

Seguro que a nadie (bueno, casi nadie) se le ocurriría ponerse a construir una casa sin antes contar con un plano de lo que desea hacer. Pues bueno, con los ficheros de ayuda de Windows ocurre lo mismo. El paso más importante en el desarrollo de una

ayuda es sentarse ante un papel en blanco y diseñar cual va a ser la apariencia final de la ayuda tal y como el usuario va a encontrársela, teniendo en cuenta una serie de reglas:

- Las pantallas de ayuda ha de ser clara, precisa y ordenada.
- No se ha de abusar de los tipos de letra. Es aconsejable utilizar una sola fuente de letra con variaciones en el tamaño, pero sin pasarse. De nada sirve un texto en el que las letras no caben en la pantalla por ser muy grande, u otro que no puede leerse por ser muy pequeño.
- El desplazamiento a través del fichero de ayuda ha de ser rápido y eficaz, sin ambigüedades, de forma que el usuario tenga que utilizar el ratón el menor número de veces posible para llegar a donde quiera.
- Hay que contemplar la búsqueda de cualquier información mediante todos los sinónimos, alias o sobrenombres posibles.
- Se debe permitir un cambio rápido desde temas generales a aspectos específicos de un tema.
- Hay que documentar todas las características del programa.

Una vez que tenemos una idea clara de lo que deseamos hacer, comenzamos el proceso creando un diagrama de nuestro fichero de ayuda (recordar que como ejemplo estamos documentando el comando ATTRIB)



Una vez decidida la estructura de nuestra ayuda vamos a crear una serie de ficheros que contengan la información. Esta podría haber estado repartida en más de un fichero, pero dado lo simple del ejemplo, solo utilizaremos uno al que llamaremos ATTRIB.DOC. El proceso que se seguiría si hubiésemos utilizados varios ficheros sería el mismo.

7.4 Escribiendo en fichero de ayuda

El primer paso es colocar en el fichero las cabeceras de cada una de las pantallas de ayuda de nuestro programa.

Editamos ATTRIB.DOC y escribimos lo siguiente dándole el tipo de letra y colores que consideremos oportunos.

```
ATTRIB.COM
Parámetros y modificadores
Ejemplos
Notas
Acerca de Ayuda ATTRIB.COM
Autores
```

Una vez hecho esto introduciremos un salto de página después de cada una de las líneas que hemos escrito y antes de la primera línea del documento. Estos saltos de página indican al compilador de la ayuda donde empieza y terminan cada una de las pantallas de ayuda. Tras esto el fichero queda así:

```
ATTRIB.COM
-----
Parámetros y modificadores
-----
Ejemplos
-----
Notas
-----
Acerca de Ayuda ATTRIB.COM
-----
Autores
-----
```

El espacio comprendido entre dos saltos de página se denomina tema. Un tema es lo que el WINHELP.EXE presenta en una ventana y lo que el usuario ve.

7.5 Las cadenas de contexto

Una vez que hemos creado las cabeceras de los temas, es hora de crear las Cadenas de Contexto. Una cadena de contexto es un identificador para un tema, de forma que si queremos saltar de un tema a otro podamos indicar a donde queremos ir. Estas cadenas son texto normal SIN ESPACIOS EN BLANCO que el compilador de ayuda utiliza para crear Hiperenlaces (conexiones de un lugar a otro del fichero).

Para crear una cadena de contexto nos situamos inmediatamente después del salto de página que delimita el comienzo de un tema y antes de la cabecera, e insertamos una nota a pie de página, indicando como marca para la nota al pie el carácter # y tecleamos como nota el texto de la cadena que queremos crear SIN ESPACIOS EN BLANCO.

Las cadenas de contexto deben ser únicas dentro del fichero de ayuda, es decir, no puede haber dos temas con la misma cadena de contexto.

En nuestro ejemplo hemos asignado las siguientes cadenas:

```
attrib_indice
attrib_parametros
attrib_ejemplos
attrib_notas
attrib_acerca_de
attrib_autores
```

Aunque el texto de la cadena es arbitrario, se debe procurar escoger etiquetas sencillas, fáciles de recordar y que hagan referencia al contenido del tema.

7.6 Creación de hiperenlaces

Una vez que hemos creado las cadenas de contexto, es hora de establecer los primeros hiperenlaces. Un hiperenlace es el punto de un fichero de ayuda desde el que se puede efectuar un salto a otro punto del fichero. Por ejemplo, en nuestro fichero de ayuda tendremos una pantalla principal que nos informa de para que sirve el comando ATTRIB, y en esta pantalla aparecerá un texto resaltado de modo que si pulsamos sobre el con el ratón pasaremos a ver la pantalla de parámetros del comando.

El primer paso para crear un hiperenlace es teclear el texto que aparecerá resaltado en la pantalla de la ayuda. Este texto puede ser cualquier cosa, no tiene por que tener nada que ver con el contenido del tema al que se va a saltar (aunque si fuera así no serviría de mucho, ¿no?) y puede contener espacios en blanco. Por ejemplo, en nuestro fichero de ayuda, dentro de la primera página vamos a colocar texto para poder enlazar con otras pantallas:

ATTRIB.COM

Parámetros y Modificadores

Ejemplos

Notas

Parámetros y modificadores

Ejemplos

Notas

Acerca de Ayuda ATTRIB.COM

Autores

Una vez que hemos escrito el texto que servirá de enlace, hemos de indicar que ese texto va a ser realmente un enlace. Esto se hace aplicando a ese texto un formato de doble subrayado o de subrayado.

Si se da formato de doble subrayado, al pulsar sobre el enlace el visualizador de ayuda quitará la información del tema que estábamos viendo y mostrará la información del tema al que saltamos.

Si se da formato de subrayado, aparecerá una ventana popup con la información del tema enlazado.

Por regla general, si el tema con el que estamos enlazando contiene poca información (un párrafo o dos breves) utilizaremos las ventanas popup (subrayado simple). En otro caso utilizaremos subrayado doble.

Una vez que hemos formateado el texto según el tipo de enlace que deseamos, hemos de indicar con que es con lo que deseamos enlazar. Para ello colocaremos el cursor justo a continuación del texto de enlace, sin dejar ningún espacio en blanco, y escribiremos la cadena de contexto del tema con el que queremos enlazar, exactamente como la escribimos cuando la definimos, sin dejar ningún espacio en blanco:

ATTRIB.COM

Parámetros y Modificadoresattrib_parametros

Ejemplosattrib_ejemplos

Notasattrib_notas

-----

Una vez hecho esto, para indicarle al compilador que el texto que acabamos de escribir es una cadena de contexto de un enlace, le daremos a la cadena de contexto que acabamos de escribir el atributo de oculto.

Con esto hemos creado el hiperenlace, y el visualizador de ficheros de ayuda lo mostrará en pantalla en el texto visible con color verde y un subrayado simple de línea continua (si nosotros lo hemos subrayado doble) o discontinua (si hemos subrayado simple).

Una vez creados los hiperenlaces, rellenaremos los temas con el texto necesario de la ayuda. Si dentro de un párrafo es necesario crear un hiperenlace, lo haremos de la misma manera.

Cuando hemos acabado de escribir nuestro fichero de ayuda, aparte de haberlo guardado con el formato del procesador que estemos utilizando, deberemos generar un fichero con formato RTF. Esto es necesario porque el compilador de ayudas no entiende otros formatos. Así, si el fichero que hemos creado se llama ATTRIB.DOC, tendremos también un fichero ATTRIB.RTF

7.7 Hiperenlaces entre ficheros

Es posible realizar un enlace entre dos ficheros .HLP. Para ello, después de la cadena de contexto, y todavía con texto oculto, dejamos un espacio en blanco e incluimos una carácter arroba (@) y el nombre del fichero.

El resultado es que en lugar de saltar al tema con la cadena de contexto especificada, dentro del fichero actual, saltaremos a otro fichero .HLP, al tema que tenga la cadena de contexto indicada.

7.8 El fichero de proyecto

Animo, que ya casi acabamos. Una vez que hemos creado nuestro fichero (o nuestros ficheros, según como lo hayamos organizado), y antes de poder compilar la ayuda, es necesario crear un fichero de proyecto. Este fichero es el encargado de indicarle al compilador (entre otras cosas) que ficheros son los que se van a utilizar para generar la ayuda, cual es el tema principal, y cual va a ser el título para la pantalla.

El fichero de proyecto ha de ser un fichero ASCII (sin códigos de control ni formatos de caracteres), debe tener extensión HPJ (para nuestro ejemplo se llama ATTRIB.HPJ y lo hemos creado mediante el editor de windows), y al igual que los ficheros INI de windows se encuentra dividido en secciones, cada una de las cuales tiene una serie de apartados. De momento solo vamos a ver dos secciones.

La primera es la sección [OPTIONS]. En ella le indicaremos al compilador cual es el tema principal y cual es el título de la ventana de ayuda. Esto se hace mediante las cláusulas CONTENTS y TITLE respectivamente

La siguiente sección es la de [FILES]. En ella se indican los ficheros que se van a incluir en la compilación del fichero. Así, nuestro fichero ATTRIB.HPJ queda de la siguiente manera:

```
[OPTIONS]
CONTENTS=attrib_indice
TITLE=Fichero de ayuda ATTRIB.COM
[FILES]
ATTRIB.RTF
```

Si tuviéramos la ayuda dividida en más ficheros, se colocaría cada uno en una línea a partir de la cabecera [FILES]

Si los ficheros que contienen los datos de la ayuda no se encuentran en el directorio en el que estemos trabajando, o bien indicamos el camino al fichero dentro de la sección files:

```
[FILES]
C:\WINDOWS\AYUDAS\ATTRIB.RTF
```

o bien incluimos en la sección [OPTIONS] la cláusula ROOT que indicará al compilador donde se pueden encontrar los ficheros. Si están en más de un directorio, cada uno de los caminos se separa de los demás con una coma:

```
[OPTIONS]
CONTENTS=attrib_indice
TITLE=Fichero de ayuda ATTRIB.COM
ROOT=C:\WINDOWS\AYUDAS , C:\RTF
```

7.9 Compilación y visualización de la ayuda

Bueno, ya lo tenemos todo. El siguiente paso es compilar nuestra ayuda. Para ello llamaremos al compilador (HC31) con el nombre de nuestro fichero de proyecto como parámetro de la línea de comandos. Por ejemplo:

```
C:\> HC31 ATTRIB.HPJ
```

El compilador comenzará a trabajar e irá mostrando una serie de puntos (uno por cada tema que va encontrando) y si hemos cometido algún error nos mostrará un mensaje de advertencia.

Ya lo único que queda es ejecutar nuestro fichero de ayuda. Para ello utilizaremos la opción de Archivo-Ejecutar del administrador de programas, indicando nuestro fichero de ayuda (ATTRIB.HLP, y sin olvidar el camino de búsqueda para llegar hasta él) y podremos contemplar el fichero que hemos creado.

7.10 Inclusión de gráficos en un fichero de ayuda

Una vez que hemos probado nuestro nuevo fichero de ayuda, y hemos visto que podemos utilizar las fuentes, tamaños y colores de Windows a la hora de escribir el texto (y si no lo habeis probado, ya estáis tardando) diremos: "si, pero si en Windows todo son iconos y gráficos, ¿no se pueden poner en el fichero de ayuda?". La respuesta es si, y ademas de más de una manera.

Podemos incluir un gráfico en un tema usando las opciones de importación o inserción de gráficos de que disponga el procesador de textos con el que estemos desarrollando la ayuda. Este gráfico aparecerá después en el fichero de ayuda de la misma manera en que aparece en nuestro procesador. Esto tiene la ventaja de que mientras estamos desarrollando el fichero, nos podemos hacer una idea bastante aproximada del aspecto final de la ayuda. Por contra, esto tiene una desventaja. El tamaño máximo que puede tener un párrafo en un fichero de ayuda es de 64KB. Cada vez que nosotros incluimos un gráfico en un párrafo, todos los bytes del gráfico son incluidos dentro de ese párrafo, y si el gráfico es muy grande, o si estamos incluyendo bastantes gráficos pequeños, el tamaño puede ser excesivo.

El otro método para incluir gráficos en un fichero de ayuda es utilizar directamente directivas RTF de inclusión de gráficos. Esta directivas no suman bytes al párrafo, pero tienen la desventaja de que no podemos observar en pantalla el resultado final hasta que no hemos compilado el fichero de ayuda.

Esta directivas son:

bmc

Muestra un fichero bitmap (.BMP) en la línea actual de texto. Su posición sería la equivalente al siguiente carácter de la línea, apoyado sobre la línea base y con las propiedades del párrafo en curso.

bml

Muestra un bitmap en el margen izquierdo de la ventana de ayuda. La primera línea de texto tras el bitmap se escribe a partir de la esquina superior derecha del bitmap, y fluye a lo largo del borde derecho del mismo.

bmr

Muestra un bitmap en el margen derecho de la ventana de ayuda. La primera línea de texto tras el bitmap se escribe a la altura de la esquina superior izquierda del bitmap, y fluye a lo largo del borde izquierdo del mismo.

La utilización de estas directivas es realmente sencilla. Todas se utilizan igual, se sitúa el cursor en el lugar (bmc) o al principio del párrafo (bmr y bml) en el que debe ir el gráfico y teclearemos la directiva y el nombre del fichero que queremos incluir, todo ello encerrado entre llaves. Por ejemplo:

```
{bmr f1.hlp} Esta tecla se utiliza para .....
```

```
Pulse la tecla {bmc escape.bmp} para cancelar el proceso
```

Al igual que ocurría con los ficheros de datos, si los bitmaps no se encuentran en el directorio en el que estamos trabajando, deberemos indicar donde están. O bien dentro de la misma directiva

```
{bmc c:\windows\malla.bmp}
```

o dentro de la sección [BITMAPS] del fichero de proyecto

```
[BITMAPS]
```

```
C:\WINDOWS\MALLA.BMP
```

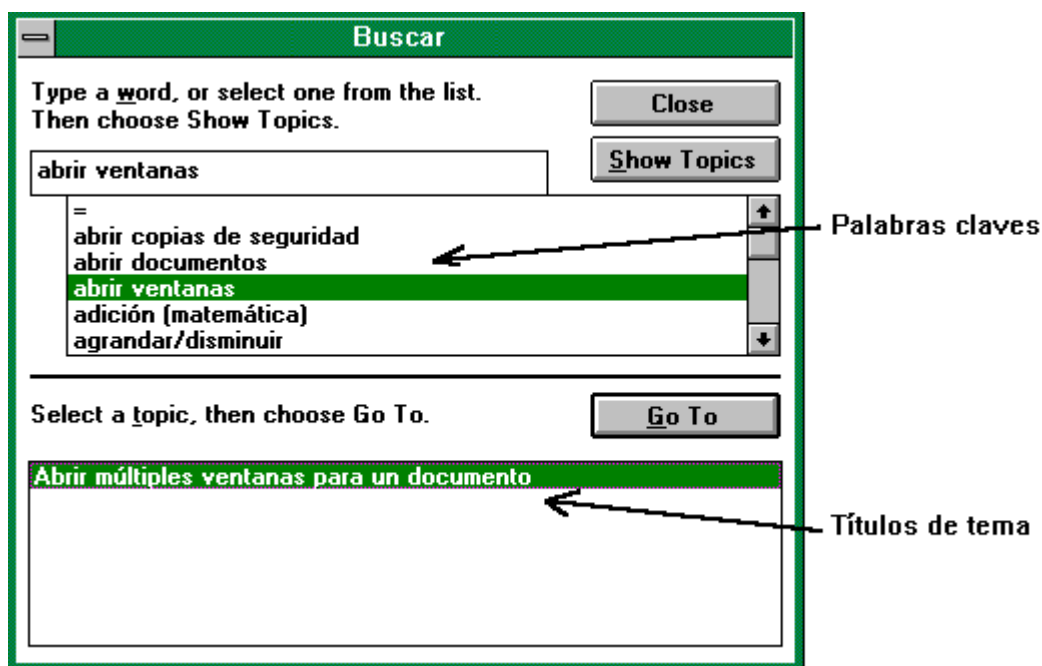
o indicando en la sección [OPTIONS] la directiva BMROOT para indicar los directorios en los que se pueden encontrar los ficheros de gráficos. Se siguen las mismas reglas que para ROOT

```
[OPTIONS]
```

```
BMROOT=C:\WINDOWS , C:\WINDOWS\AYUDA
```

7.11 El botón de buscar que no funciona

Cualquiera que halla trabajado mínimamente con una ventana de ayuda Windows, conoce el botón de buscar. Este botón permite localizar un tema determinado a partir de diversas palabras claves que identifican al tema. Una vez seleccionada una palabra clave, nos muestra una lista de los títulos de los temas asociados a esa clave, y nos deja elegir cual queremos visualizar. Esto se hace en una pantalla parecida a la siguiente:



Pues bien, en nuestro fichero de ayuda el botón de buscar aparece desactivado. ¿Por que? Por que no le hemos dicho al compilador cuales son las claves que se pueden buscar ni cuales son los títulos de los temas. Veamos como se hace esto.

El primer paso consiste en darle un título a los temas en los que queramos definir palabras claves. La razón: No puedo definir una palabra clave para un tema que no tenga título.

NO es necesario definir título para todos los temas. Solo para aquellos a los que queramos que se pueda acceder a través del botón de búsqueda. Para ello, nos situaremos al principio del tema, justo después del carácter de nota a pie que define la cadena de contexto (#), e insertaremos otra nota a pie de página, pero en esta ocasión el carácter que se ha de utilizar para la nota no es la almohadilla, sino el símbolo del dólar \$. El texto que tecleemos dentro de la nota será el título del tema, y se utilizará en el botón de búsqueda y en el botón de historia (lista los títulos de los temas que hemos estado consultando).

Una vez que le hemos dado título a nuestros temas, pasamos a asignar a cada tema una serie de palabras de búsqueda. ¿Alguien se imagina como? Si. Con notas a pies de página.

Para definir palabras clave para un tema, situaremos el cursor después del carácter \$ de la nota a pie que asigna el título, e insertaremos una nota a pie de página, pero la personalizaremos mediante el carácter K. Dentro de la nota escribiremos todas las palabras claves bajo las que se puede localizar el tema, separando cada una de la siguiente con un punto y coma (;).

NOTA. Que se denominen palabras claves no quiere decir que solo hayan de contener una palabra. Pueden ser varias palabras y contener espacios en blanco, etc. Por ejemplo, esta sería la apariencia de una nota a pie con una serie de palabras claves:

KÍndice general;Tabla de contenidos;Contenidos

CUIDADO. Si la primera letra de la primera palabra dentro de la nota es una K, habrá que separarla de la K que identifica la nota con un espacio.

7.12 Uso de bitmaps en hiperenlaces

Cuando creamos un fichero de ayuda, y establecemos un hiperenlace, aparece un texto en el que se puede pinchar con el ratón y que nos lleva a otro tema del fichero de ayuda. ¿Se puede hacer lo mismo con los bitmaps? Si. Se puede definir un bitmap de forma que cuando se pinche sobre el, salte a otro tema o muestre una ventana popup. Para ello deberemos utilizar una utilidad denominada "HotSpot Editor" (Editor de puntos calientes). Esta utilidad aparece con casi todos los paquetes de desarrollo para Windows, y lo que permite es indicar dentro de un bitmap la cadena de contexto a la que se saltará cuando se pulse sobre el gráfico.

Dentro del fichero de ayuda no es necesario indicar nada. Solo incluir el bitmap. El resto de la información necesaria (a donde se salta) se ha de incluir con el HotSpot Editor.

7.13 Creación de secuencias.

Cuando creamos un fichero de ayuda puede presentárenos el caso de necesitar incluir varios temas que por lógica deberían ser leídos uno detrás de otro. Esto se puede implementar mediante la definición de una serie de hiperenlaces entra los temas, pero existe un método más adecuado: La creación de secuencias.

Una secuencia es una serie de temas en los que se ha definido un cierto orden, de modo que podemos pasar de un tema al siguiente o al anterior mediante dos botones que aparecerán en la barra de la ventana de ayuda. Estos botones tienen los símbolos "<<" (anterior) y ">>" (siguiente) y nos permiten movernos dentro de la secuencia.

Para crear una secuencia, es necesario indicar que temas forman la secuencia. ¿Adivinan como? Si. Más notas a pie de página.

Situando el cursor después de la última de las notas que hemos definido (cadena de contexto, título de tema y claves de búsqueda), insertamos una nota a pie de página y la personalizamos con el carácter + (un signo de sumar). Dentro de la nota indicaremos el nombre de la secuencia y el número que hace este tema dentro de la secuencia separado por dos puntos. Por ejemplo:

`+tutor_parrafos:01`

repitiendo este proceso (cambiando el número) para cada uno de los temas de la secuencia.

Es importante recordar que todos los identificadores de una secuencia (el nombre más el número) deben tener la misma longitud. Así, si nuestra secuencia tiene solo 7 temas podremos denominarlos

```
+tutor_parrafos:1
```

```
.....
```

```
+tutor_parrafos:7
```

pero si tenemos más de 9 temas deberemos colocar ceros a la izquierda del número:

```
+tutor_parrafos:01
```

```
.....
```

```
+tutor_parrafos:24
```

Naturalmente dentro de un fichero de ayuda podemos tener tantas secuencias como queramos, teniendo cuidado de que su identificador de secuencia sea distinto (si es igual no produce error, simplemente se mezclan las dos)

Una vez que hemos indicado los temas que pertenecen a una secuencia, es necesario indicarle al compilador que la vamos a usar, y que active los botones de secuencia. Para ello, en el fichero de proyecto, crearemos la sección [CONFIG] e incluiremos una llamada a un macro del visor de ayuda (ya veremos lo que son los macros y cuales tenemos, pero por hagamos un acto de fe y creamos al escritor) BrowseButtons()

```
[CONFIG]
```

```
BrowseButtons()
```

Ya solo recompilar y probar.

7.14 Las etiquetas de construcción (build tags)

Queda un único tipo de nota a pie de página por ver dentro de los ficheros de ayuda. Son las etiquetas de construcción. Se construyen mediante notas personalizadas mediante el carácter * (asterisco). Se indican justo a continuación de todas las demás notas del tema, y el texto contenido en la nota puede ser cualquier cadena SIN ESPACIOS EN BLANCO. Pueden definirse tantas etiquetas de construcción como se deseen dentro de cada tema.

¿ Y para que valen ?. Pues bien, a la hora de compilar el fichero de ayuda es posible indicar en el fichero de proyecto una serie de etiquetas de construcción, y solo aparecerán en el fichero compilado aquellos temas que contengan esas etiquetas. Para ello, aparte de definir las build tags dentro del fichero, es necesario indicar los siguientes datos en el fichero de proyecto:

Hay que definir una sección [BUILDTAGS] que incluirá la lista de las etiquetas de los temas que se van a incluir dentro del fichero de ayuda. Por ejemplo:

```
[BUILDTAGS]
```

```
Capitulo1
```

```
Capitulo2
```

```
Capitulo5
```

```
.....
```

Esto incluiría en el fichero de ayuda todos aquellos temas que tuvieran las etiquetas Capitulo1, Capitulo2 o Capitulo5. Se pueden incluir hasta 30 etiquetas dentro de la sección [BUILDTAGS]

NOTA: Es importante recordar que los temas que NO tienen etiquetas de construcción SE INCLUYEN SIEMPRE.

Ademas de poder indicar que temas se van a incluir listando sus etiquetas de construcción, es posible realizar operaciones lógicas que limiten el grupo de temas que se van a compilar. Esto se realiza mediante la directiva BUILD de la sección [OPTIONS]. Esta directiva tiene el siguiente formato:

```
[OPTIONS]
```

```
BUILD=expresion
```

donde *expresión* es una combinación de etiquetas y los siguientes operadores:

~

Operador NOT. Se aplica a una etiqueta. Solo compila los temas que no tengan incluida la etiqueta indicada.

```
BUILD=~Capitulo1
```

Compila los temas en los que no está definida la etiqueta Capitulo1

&

Operador AND. Se aplica a dos etiquetas. Solo compila aquellos temas que contienen las dos etiquetas

BUILD=Capitulo1 & USUARIO_FINAL

Compila los temas que tienen definidas las dos etiquetas

|

Operador OR. Se aplica a dos etiquetas. Compila los temas que contienen al menos una de las dos etiquetas especificadas.

BUILD=Capitulo1 Capitulo2

Compila los temas que contienen la etiqueta Capitulo1 y los que contienen la etiqueta Capitulo2

Se puede realizar cualquier tipo de combinación de estos operadores, recordando que el operador ~ tiene preferencia sobre el operador &, y que este tiene preferencia sobre el |

BUILD=(Capitulo1 Capitulo2) & USUARIO_FINAL

Compila los temas en los que están definidas las etiquetas Capitulo1 o Capitulo2, y que además contengan la etiqueta Usuario_final

Solo se puede indicar una directiva BUILD en cada fichero de proyecto

7.15 Los ficheros de proyecto

Una vez llegados aquí, y antes de meternos en cosas mas enrevesadas (e interesantes), vamos a dar un vistazo a las secciones y directivas de los ficheros HPJ

Sección [OPTIONS]

Esta sección controla la forma en que se va a construir el fichero de ayuda, y la información que va a proporcionar el compilador. Es una sección optativa, y en el caso de incluirse en el proyecto, debe ser la primera sección del fichero.

La sección OPTIONS esta compuesta por directivas con el formato:

DIRECTIVA = VALOR

Estas directivas son:

BMROOT

Indica el directorio en el que se encuentran los ficheros de gráficos incluidos en el fichero de ayuda. Su formato es:

BMROOT= <i>directorio</i> , [<i>directorio</i> ,...]

Si la directiva BMROOT no se incluye, los ficheros de gráficos se buscarán en los directorios especificados mediante la directiva ROOT. Si la directiva ROOT no se incluye, o no indica los directorios de los gráficos, deberá indicarse donde se encuentra cada uno de los gráficos dentro de la sección [BITMAPS]

BUILD

Especifica una condición que limita los temas que se han de compilar, usando las etiquetas de construcción. Si se especifica una directiva BUILD, obligatoriamente se ha de especificar una sección [BUILDTAGS] que indique que etiquetas se van a utilizar.

Su formato es:

BUILD= <i>expresión</i>

COMPRESS

Indica el grado de compresión que se va a aplicar al fichero de la ayuda al compilarlo. Su formato es:

COMPRESS= <i>nivel-de-compresión</i>

Donde nivel de compresión puede ser 0,NO o FALSE para indicar sin compresión (0%), 1,YES o TRUE para indicar máxima compresión (50%), y MEDIUM para indicar compresión mediana (40%)

Dependiendo del grado de compresión solicitado, el compilador utilizará compresión por bloque o compresión por bloque y compresión de frases. La compresión por bloque comprime los temas en unidades predefinidas conocidas como bloques. La compresión por frases combina cadenas repetidas encontradas a lo largo de los ficheros de datos. En el proceso, el compilador crea un fichero de frases con extensión .PH en el caso de que no encuentre uno. En el caso de que encuentre un fichero con extensión .PH lo utilizará para el proceso de compilación. Debido a que el fichero .PH acelera el proceso de compilación cuando solo pequeños fragmentos de texto han cambiado desde la última compilación, es conveniente no eliminarlo del directorio si se está compilando repetidamente con

compresión. Sin embargo, se obtendrá mayor porcentaje de compresión eliminando el fichero antes de cada proceso.

CONTENTS

Especifica la cadena de contexto del tema que se utilizará como página de contenidos (el índice, el primer tema que aparece al abrir el fichero de ayuda). Su formato es:

CONTENTS= <i>cadena-de-contexto</i>

Si no se especifica una directiva CONTENTS, se toma como página de contenidos el primer tema del primer fichero indicado en la sección [FILES]

COPYRIGHT

Introduce un texto que se mostrará dentro de la ventana de "Ayuda-Acerca de" justo debajo del Copyright de Windows. Su formato es:

COPYRIGHT= <i>texto</i>

El tamaño del texto no puede exceder los 50 caracteres.

ERRORLOG

Escribe los errores del proceso de compilación dentro de un fichero además de mostrarlos en pantalla. Su formato es:

ERRORLOG= <i>fichero</i>

FORCEFONT

Obliga al compilador a utilizar una fuente de letra determinada para todo el fichero de ayuda. Su formato es:

FORCEFONT= <i>Nombre-de-fuente</i>

El nombre de la fuente ha de ser de una fuente disponible, con un máximo de 20 caracteres y escrita tal y como aparece en la opción de fuentes del panel de control. Si la fuente no es válida se utilizará por defecto MS Sans Serif.

ICON

Indica el nombre del fichero de icono que se mostrará cuando la ventana de ayuda se minimize. Su formato es:

ICON= <i>fichero-de-icono</i>

Fichero-de-icono ha de ser el nombre de un fichero de icono en formato standard windows (.ICO).

MAPFONTSIZE

Cambia el tamaño de las fuentes en los temas a otro determinado cuando se muestren en pantalla. Su formato es:

MAPFONTSIZE= <i>original:nuevo</i>

Mostrará las fuentes que tuvieran el tamaño *original* con el tamaño *nuevo*

OLDKEYPHRASE

Indica al compilador si ha de utilizar el fichero de frases existente en el proceso de compilación y compresión, o ha de crear uno nuevo cada vez. Su formato es:

OLDKEYPHRASE= <i>valor</i>

Donde valor puede ser 0, FALSE, NO u OFF para que lo vuelva a crear, o 1, TRUE o YES para que no lo cree.

OPTCDROM

Optimiza la distribución de los temas dentro del fichero de ayuda para mejorar la velocidad de proceso en el caso de que el fichero vaya a estar grabado en CD-ROM. Su formato es:

OPTCDROM= <i>valor</i>

donde *valor* puede ser ON, 1, YES o TRUE.

REPORT

Indica al compilador que nos informe del proceso que está llevando a cabo. Su formato es

`REPORT=ON`

Cuando se activa, el compilador nos irá indicando cuando compila, resuelve saltos o comprueba secuencias.

ROOT

Indica el directorio en el que se van a encontrar los ficheros fuente para la compilación. Su formato es:

`ROOT=directorio [, directorio, ...]`

TITLE

Especifica el título de la ventana en que se mostrará el fichero de ayuda. Su formato es:

`TITLE=título`

WARNING

Indica el nivel de errores que mostrará el compilador. Su formato es:

`WARNING=nivel`

donde *nivel* puede ser 1 (solo muestra los errores más graves), 2 (muestra los errores intermedios) o 3 (muestra los errores y los avisos)

Sección [FILES]

Indica los ficheros (.RTF) que se van a utilizar para obtener el fichero de ayuda final. En esta sección no hay directivas, y cada una de las líneas contenidas en esta sección indica el nombre de un fichero.

`[FILES]`

Primero.RTF Segundo.RTF

Si no se especifica un directorio con el nombre del fichero, se buscarán dentro de los directorios indicados en la directiva ROOT de la sección [OPTIONS]

Sección [BUILDTAGS]

Indica una serie de etiquetas de construcción. Solo se incluirán en el fichero de ayuda aquellos temas cuyas etiquetas de construcción aparezcan en esta sección.

Al igual que ocurría con la sección [FILES], ésta tampoco tiene directivas, y cada línea indica una etiqueta de construcción.

IMPORTANTE: Si no se indica esta sección, todos los temas incluidos en los fichero fuente serán incluidos en el fichero final. Además, los temas para los que no se han definido etiquetas de construcción no se ven afectados por esta sección, y son incluido siempre.

Sección [CONFIG]

Contiene una serie de macros de ayuda windows que se ejecutarán al cargar el fichero de ayuda (por ejemplo habilitar los botones de secuencias). Así mismo, permite registrar funciones externas residentes en librerías de enlace dinámico (.DLL). Su formato y utilización se explican más adelante.

Sección [BITMAPS]

El equivalente a la sección [ROOT] pero aplicado a los fichero de gráficos en lugar de a los de texto. Indica los fichero y la localización de los gráficos que se han de incluir en el fichero de ayuda.

Sección [MAP]

Se utiliza para asignar números a las cadenas de contexto de los temas. Esto es útil para desarrollar ayudas en línea para los programa, ya que el visualizador de ficheros de ayuda permite recibir un número como parámetro indicando el tema al que queremos ir. Este número debe estar asociado a una cadena de contexto dentro de la sección [MAP]. Su formato básico es:

```
[MAP]
cadena-de-contextovalor- numerico
```

También acepta el formato

```
[MAP]
#define cadena-de-contexto    valor- numerico
```

Y es posible incluir un fichero externo que contenga declaraciones del tipo anterior (#define). Esto se hace de la siguiente manera:

```
[MAP]
#include "fichero-de-definiciones"
```

pudiendo indicarse el nombre del fichero entre comillas dobles ("") o entre paréntesis angulares (<>).

Si se intenta generar un mapping para una cadena de contexto inexistente, el compilador generará un mensaje de aviso.

Sección [ALIAS]

Permite que todas las referencias a una cadena de contexto sean pasadas a otra distinta. Su formato es

```
[ALIAS]
cadena-de-contexto=alias
.....
```

Pueden definirse tantos alias como sea necesario. Estos se rigen por las mismas normas que las cadena de contexto (solo letras de la A a la Z, números del 0 al 9, el punto y el subrayado).

Se puede utilizar para:

Referencias a temas obsoletos sean redirigidas a nuevos temas sin necesidad de ir buscando todas las referencias a los temas antiguos.

Si existe mas de un identificador de contexto para un tema (suele pasar en ayudas en linea), acceder al tema correcto.

etc....

Los alias definidos en esta sección pueden ser utilizados en la sección [MAP] siempre y cuando la sección [ALIAS] se haya definido primero en el fichero de proyecto.

Sección [WINDOWS]

Permite especificar características para la ventana principal de la ayuda y para las ventanas secundarias (ah!, ¿que no lo sabíais? Pues si, un fichero de ayuda se puede presentar en más de una ventana. Luego contamos como).

El formato es:

[WINDOWS]

tipo = "Título" , (x , y , Ancho , Alto) , tamaño , (cliente-RGB) , (nonscrollRGB) , (ltop)

Donde los parámetros tienen los siguientes significados:

tipo

Es el nombre de la ventana. Para la ventana principal es **main**. Para las ventanas secundarias puede ser cualquier nombre único de hasta 8 caracteres. Cuando en un tema, un hiperenlace realiza un salto a una ventana secundaria, utiliza este nombre como parte de la cadena de contexto. Así, si queremos que el texto del hiperenlace se muestre en la ventana secundaria, deberemos añadir, justo después de la cadena de contexto (y también con formato de texto oculto) un carácter de mayor que (>) y el nombre de la ventana en la que se van a mostrar los datos. Si el enlace se realiza con otro fichero .HLP, el > y el nombre de la ventana se han de situar antes del carácter @.

Titulo

Es el texto que aparecerá en la barra de titulo de la ventana. Solo se aplica a las ventanas secundarias. Para fijar el título de la ventana primaria es necesario utilizar la directiva TITLE de la sección [OPTIONS]

x

Es la coordenada x de la esquina superior izquierda de la ventana en unidades de ayuda. Windows asume una ventana de 1024x1024 independientemente de la resolución. Así, un valor de 512 haría que el margen izquierdo de la ventana de ayuda quedara en el centro de la pantalla.

y

Es la coordenada y de la esquina superior izquierda de la ventana en unidades de ayuda. Se rige por las mismas reglas que la coordenada x

Ancho

Especifica el ancho en unidades de ayuda de la ventana

Alto

Especifica el alto en unidades de ayuda de la ventana

Tamaño

Especifica el tamaño relativo de la ventana secundaria cuando windows abre la ventana por primera vez. Sus valores pueden ser: 0 para usar el tamaño definido por x,y,ancho y alto, o 1 para que arranque maximizada.

cliente-RBG

Es el color de fondo de la pantalla de ayuda. Es un valor de color RGB, compuesto por tres números de 0 a 256, encerrados entre paréntesis y separados por comas. Si no se especifica, se utilizará el color definido por defecto.

nonscroll-RGB

Es el color de fondo de la zona de NO-Scroll situada al principio de los temas. Al igual que en el caso anterior es un valor RGB.

Una zona de No-scroll se crea aplicando a los párrafos que queremos que permanezcan siempre en pantalla (ojo, tienen que ser los primeros párrafos del tema) el atributo de "Conservar con el siguiente".

ltop

Indica si la ventana secundaria ha de mostrarse encima de todas las demás ventanas. Si el valor es 1, la ventana aparecerá por encima de todas las otras ventanas que no tengan definido también este atributo.

[WINDOWS]

main=, (, , ,), 0, (, ,), (128, 0, 128)

pictures = "Samples", (123,123,256,256), 0, (0,255,255), (255,0,0)

Sección [BAGGAGE]

Lista una serie de ficheros que se incluirán dentro del fichero de ayuda. Estos ficheros son accedidos desde el resto de la ayuda igual que si estuvieran en el directorio, pero de manera más eficiente. Pueden incluirse hasta 1000 ficheros.

Su formato es igual al de las secciones [ROOT] o [BITMAP]

[BAGGAGE]

tada.wav

carillon.wav

Dibujo.wmf

7.16 Utilización de macros en ficheros de proyectos

Los macros se añaden dentro de la sección [CONFIG] del fichero de proyectos. Cuando Windows abre el fichero de ayuda, automáticamente ejecuta los macros que se encuentra en esta sección en el mismo orden en el que se los va encontrando.

Así, es posible modificar los menús de la ventana de ayuda, añadir botones, etc.

7.16.1 Utilización de macros en temas

Si dentro de un tema se define una nota a pie de página, personalizada con el carácter !, el compilador de ayuda espera encontrar dentro de la nota una llamada a un macro de ayuda. Este macro es ejecutado cada vez que se visualiza el tema.

7.16.2 Utilización de macros en hiperenlaces

Y ahora lo mejor de todo. Es posible definir un hiperenlace de modo que al pulsar sobre él no saltamos a otro tema del fichero de ayuda, sino que se ejecuta una acción (por ejemplo tocar un fichero de sonido). Esto se consigue asignando al hiperenlace un macro o una llamada a una función registrada. Para ello, en lugar de indicar una cadena de contexto como texto oculto dentro del hiperenlace, se indica un carácter exclamación (!) y el macro o la serie de macros (separados por punto y coma) que se desean ejecutar.

7.16.3 Macros Disponibles en ficheros de ayuda

About()

Muestra la ventana de ayuda "Acerca de"

AddAccelerator(tecla, estado, "macro...")

Asigna un macro a una pulsación de teclas. Los parámetros son:

Tecla

Un valor de tecla virtual de windows.

Estado

Una valor de tres bits en el que el bit 0 representa las mayúsculas, el 1 el Control y el 2 el Alt. Si la tecla está pulsada, el bit está activado, si no está desactivado

Macro

Es un macro o serie de macros a ejecutar encerrados entre comillas dobles. Si hay más de un macro, es necesario separarlos por puntos y comas (;)

Puede ser abreviado como AA

Annotate()

Muestra el dialogo de anotaciones.

AppendItem("menu-id", "item-id", "item-name", "macro")

Añade un item a un menú creado mediante el macro InsertMenu. Sus parámetros son:

menu-id

Nombre utilizado en el macro InsertMenu para crear un menú. Debe aparecer entre comillas dobles.

item-id

Especifica el nombre (identificador) que el sistema de ayuda usa internamente para identificar el item del menú

item-name

Es el texto que aparece dentro del menú. Hay que escribirlo entre comillas. Dentro de las comillas se puede utilizar el carácter & para indicar el acelerador utilizado por el macro.

macro

Es el macro a ejecutar al seleccionar el item. Debe aparecer entrecomillado. Múltiples macros se separa por puntos y coma.

Si se aplica en una ventana secundaria es ignorado.

Si el acelerador entra en conflicto con otra tecla en los menús, el sistema de ayuda da un error y no añade el ítem.

Back()

Vuelve a mostrar el tema anterior dentro de la lista de temas vistos.

BookmarkDefine()

Muestra el diálogo del definir marcadores del sistema de ayuda. Si se coloca en un popup, el marcador se asocia con el tema desde el que se llamó al pop-up.

BookmarkMore()

Muestra el cuadro de diálogo de "más marcadores" que aparece cuando hay más de nueve marcadores definidos.

BrowseButtons()

Coloca los botones de secuencias dentro de la barra de botones de la ayuda.

Si este macro es invocado desde una ventana secundaria es ignorado.

ChangeButtonBinding("button-id", "button-macro")

Cambia la acción de un botón definido mediante el macro CreateButton.

Los parámetros son:

button-id

Especifica el identificador del botón para botones creados mediante CreateButton, o en el caso de botones standard de windows, uno de los siguientes identificadores:

BTN_CONTENTS	Botón de contenidos
BTN_SEARCH	Botón de búsqueda
BTN_BACK	Botón de Atrás
BTN_HISTORY	Botón de historia
BTN_PREVIOUS	Botón de anterior (secuencia)
BTN_NEXT	Botón de siguiente (secuencia)

button-macro

Macro que se asigna al botón

Si el macro se ejecuta desde una ventana secundaria es ignorado.

El macro ChangeButtonBinding puede ser abreviado como CBB.

ChangeItemBinding("item-id", "item-macro")

Cambia el macro asociado a un ítem de menú. Sus parámetros son:

item-id

Identifica el ítem creado mediante el macro AppendItem

item-macro

El macro a ejecutar cuando se selecciona el ítem.

Si es ejecutado desde una ventana secundaria no tiene efecto.

Puede ser abreviado como CIB

CheckItem("item-id")

Pone una marca al lado del ítem de un menú. Los parámetros son:

item-id

Identificador del ítem

Puede ser abreviado como CI

CloseWindow("window-name")

Cierra una ventana secundaria o la ventana principal de la ayuda. Los parámetros son:

window-name

Nombre de la ventana a cerrar. El nombre "main" esta reservado para la ventana principal. Para las ventanas secundarias el nombre está definido en la sección [WINDOWS]

Contents()

Salta a la pantalla de contenidos del fichero de ayuda actual

CopyDialog()

Muestra el cuadro de dialogo de Copiar del menú de Editar

CopyTopic()

Copia todo el texto del tema actual en el portapapeles

CreateButton("button-id", "name", "macro")

Añade un nuevo botón a la barra de botones. Los parámetros son:

button-id

Especifica el nombre que el sistema de ayuda utiliza internamente para identificar el botón

name

Especifica el texto que aparece en el botón. Se puede definir un acelerador mediante el uso del carácter & en el nombre. La longitud máxima es de 29 caracteres.

macro

Es el macro asignado al botón. Macros múltiples deben ser separados mediante puntos y comas.

El sistema de ayuda permite un máximo de 16 botones personalizados, y 22 botones incluyendo los standard.

Es ignorado si se ejecuta desde una ventana secundaria.

Puede ser abreviado como CB

DeleteItem("item-id")

Elimina un item de un menú. Los parámetros son:

item-id

El identificador del item creado mediante AppendItem. Este macro es ignorado si se ejecuta desde una ventana secundaria.

DeleteMark("marker-text")

Elimina un marcador de texto creado mediante el macro SaveMark. Los parámetros son:

marker-text

Es el identificador del marcador

Si el marcador no existe, el sistema de ayuda muestra un error de elemento no encontrado.

`DestroyButton("button-id")`

Elimina un botón creado mediante el macro `CreateButton`. Los parámetros son:

`button-id`

Identificador del botón

Este macro es ignorado si se ejecuta desde una ventana secundaria.

El identificador el botón no puede ser uno de los identificadores de botones standard.

`DisableButton("button-id")`

Deshabilita un botón impidiendo que pueda ser ejecutado. Los parámetros son:

`button-id`

Identificador del botón

Este macro es ignorado si se ejecuta desde una ventana secundaria.

Puede ser abreviado como DB

`DisableItem("item-id")`

Deshabilita un ítem de un menú. Los parámetros son:

`item-id`

Identificador del ítem

Es ignorado si se ejecuta desde una ventana secundaria.

Puede ser abreviado como DI

`EnableButton("button.id")`

Activa un botón de la barra de botones del sistema de ayuda. Los parámetros son:

`button-id`

Identificador del botón

Este macro es ignorado si se ejecuta desde una ventana secundaria.
Puede ser abreviado como EB

EnableItem("item-id")

Activa un ítem de un menú. Los parámetros son:

item-id

Identificador del ítem

Es ignorado si se ejecuta desde una ventana secundaria.
Puede ser abreviado como EI

ExecProgram("command-line", display-state)

Ejecuta un programa externo al visualizador de ayudas. Los parámetros son:

command-line

Especifica la línea de comandos de la aplicación que va a ser ejecutada.

display-state

Indica como se mostrará la aplicación al arrancarse. Los valores son:

- 0 Normal
- 1 Minimized
- 2 Maximized

Puede ser abreviado como EP

Exit()

Salida del sistema de ayudas. Tiene la misma función que la opción Salir del menú de fichero

FileOpen()

Muestra el diálogo del Abrir Fichero del menú de fichero.

FocusWindow("window-name")

Cambia el foco de una ventana a otra. Los parámetros son:

window-name

Nombre de la ventana que va a obtener el foco.

GoToMark("marker-text")

Salta a un marcador generado mediante el macro SaveMark. Los parametros son:

marker-text

El identificador del marcador.

HelpOn()

Abre el fichero de ayuda sobre la ayuda de Windows

HelpOnTop()

Hace que la ventana de ayuda permanezca siempre sobre el resto de las ventanas.

History()

Muestra el cuadro de dialogo de temas recorridos (history = historia)

IfThen(IsMark("marker-text"), "macro")

Ejecuta un macro si un determinado marcador de texto existe. Los parametros son:

marker-text

Identificador del marcador de texto creado mediante SaveMark

macro

Macro que ha de ejecutarse en el caso de que el marcador exista. Macros multiples han de separarse con puntos y coma.

IfThenElse(IsMark("marker-text"), "macro1", "macro2")

Ejecuta un macro si un determinado marcador de texto existe. En el caso de que no exista ejecuta un macro diferente. Los parametros son:

marker-text

Identificador del marcador

macro1

Macro que se ejecuta si el marcador existe. Macros multiples han de separarse con puntos y coma.

macro2

Macro que se ejecuta si el marcador no existe. Macros multiples han de separarse con puntos y coma.

InsertItem("menu-id", "item-id", "item-name", "macro", position)

Inserta un item en un lugar determiado de un menú existente. El menú puede haber sido creado mediante InsertMenu o ser uno de los menús standard de Windows. Los parametros son:

menu-id

Identificador del menu. Puede haber sido creado con InsertMenu o bien ser uno de los siguientes:

MNU_FILE	Menú de fichero
MNU_EDIT	Menú de edición
MNU_BOOKMARK	Menú de marcadores
MNU_HELPON	Menú de ayuda

item-id

Identificador que utiliza el sistema de ayuda para identificar internamente al item.

item-name

Texto que aparece como item del menú. Puede contener un acelerador mediante el uso del caracter &.

macro

Es el macro que se ejecutará al seleccionar el item. Macros multiples han de separarse mediante puntos y coma.

position

Especifica la posición del item dentro del menú. La primera posición es la 0

Si se aplica en una ventana secundaria es ignorado.

Si el acelerador entra en conflicto con otra tecla en los menús, el sistema de ayuda da un error y no añade el item.

InsertMenu("menu-id", "menu-name", menu-position)

Añade un menú al sistema de ayuda. Los parametros son:

menu-id

Identificador interno para el menú

menu-name

Nombre del menú. Permite la utilización de aceleradores mediante el caracter &

menu-position

Indica la posición del menú. El primer menú es el 0

Este macro se ignora si es ejecutado desde una ventana secundaria.

IsMark("marker-text")

Comprueba la existencia de un marcador de texto. Se utiliza en conjunción con los macros IfThen y IfThenElse.

JumpContents("file")

Salta a la tabla de contenidos del fichero de ayuda indicado como paraametro.

JumpContext("filename", context-number)

Salta a un tema de un fichero de ayuda determinado. Los parametros son:

filename

El nombre del fichero de ayuda. Si no se encuentra el fichero, no se ejecuta el salto y se muestra un mensaje de error.

context-number

Especifica un numero de contexto de un tema del fichero de destino. Este numero debe haber sido creado en la sección [MAP] del fichero de destino. Si no se encuentra el identificador se salta a la pantalla de contenidos y se muestra un error.

Puede ser abreviado como JC

JumpHelpOn()

Salta a la página de contenidos del fichero de ayuda sobre la ayuda.

JumpId("filename", "context-string")

Salta a un tema indicado de un fichero de ayuda externo. Los parametros son:

filename

Fichero externo de ayuda. Si no se encuentra no se produce el salto.

context-string

Cadena de contexto del tema al que queremos saltar.

Puede ser abreviado como JI

JumpKeyword("filename", "keyword")

Salta a otro fichero de ayuda, busca una palabra clave y muestra el primer tema que contiene esa palabra. Los parametros son:

filename

Nombre del fichero de ayuda. Si no se encuentra no se efectua el salto y se produce un mensaje de error.

keyword

Palabra clave que buscamos.

Puede ser abreviado como JK

Next()

Muestra el siguiente tema dentro de una secuencia

Not(IsMark("marker-text"))

Invierte el resultado del macro IsMark(...)

PopupContext("filename", context-number)

Muestra en una ventana popup un tema de otro fichero de ayuda, localizandolo por numero de contexto. Los parametros son:

filename

Nombre del fchero de ayuda

context-number

Numero de contexto. Tiene que haber sido definido en la sección [MAP] del fichero al que se salta.

Puede ser abreviado como PC

PopupId("filename", "context-string")

Muestra en una ventana popup un tema de otro fichero de ayuda, localizandolo por cadena de contexto. Los parametros son:

filename

Nombre del fichero de ayuda

context-string

Cadena de contexto.

Puede ser abreviado como PI

PositionWindow(x, y, Ancho, Alto, Tamaño, "nombre")

Cambia el tamaño de una ventana de ayuda. Ver creación de ventanas secundarias para más información.

Prev()

Muestra el tema anterior dentro de una secuencia.

Print()

Imprime el tema actual

PrinterSetup()

Permite configurar la impresora

RegisterRoutine("DLL-name", "function-name", "format-spec")

Registra una rutina externa residente en una DLL. Se utilizan igual que los macros. Los parametros son:

DLL-Name

Nombre del fichero DLL en el que reside la rutina.

function-name

Nombre de la función

format-spec

Es una cadena que especifica el tipo de los parametros pasados a la función. Los caracteres validos son:

u unsigned short (WORD)

U unsigned long (DWORD)

i	short int
l	int
s	near char * (PSTR)
S	far char * (LPSTR)
v	void

Puede ser abreviado como RR

RemoveAccelerator(key, shift-state)

Elimina un acelerador de teclado. Ver AddAccelerator para más información.

SaveMark("marker-text")

Graba un marcador de textos con la información sobre el fichero y el tema actual. Los parametros son:

marker-text

Nombre del marcador que identifica la localización del tema.

Al salir del sistema de ayuda todos los marcadores son borrados.

Search()

Activa el cuadro de dialogo de busqueda de palabras claves

SetContents("filename", context-number)

Designa un determinado tema como página de contenidos dentro de un fichero de ayuda. Los parametros son:

filename

Nombre del fichero de ayuda

context-number

Número de contexto del tema. Debe haber sido creado en la sección [MAP]

SetHelpOnFile("filename")

Fija cual será el fichero que se mostrará cuando se solicite ayuda sobre como usar la ayuda.

UncheckItem("item-id")

Elimina la marca de un item de un menú. Los parametros son:

item-id

Identificador interno para el item.

Puede ser abreviado como UI