Este libro esta dedicado a
Mi esposa Ruth y al tesoro de mi hija Sarah.

Los nuevos bucaneros del planeta

Programación Avanzada en Harbour & Fivewin

By Rafa Carmona (Thefull)

1Parte

Índice.

Traducción a Español de la GNU Free Document License 1.1 (GFDL)

Los autores de esta traducción son:

- Igor Támara ikks@bigfoot.com
- Pablo Reyes reyes_pablo@hotmail.com

Esta es una traducción no oficial de la GNU Free Document License (GFDL), versión 1.1 a Español. No ha sido publicada por la Free Software Foundation, y no establece legalmente los términos de distribución para trabajos que usen la GFDL -- sólo el texto de la versión original en Inglés de la GFDL lo hace. Sin embargo, esperamos que esta traducción ayudaré a hablantes de español a entender mejor la GFDL.

This is an unofficial translation of the GNU Free Document License (GFDL) into Spanish. It was not published by the Free Software Foundation, and does not legally state the distribution terms for works that uses the GFDL --only the original English text of the GFDL does that. However, we hope that this translation will help Spanish speakers understand the GFDL better.

La versión original de la GFDL esta disponible en: http://www.gnu.org/copyleft/fdl.html

Tabla de Contenidos

Versión 1.1, Marzo de 2000

0. PREÁMBULO

El propósito de esta licencia es permitir que un manual, libro de texto, u otro documento escrito sea "libre" en el sentido de libertad: asegurar a todo el mundo la libertad efectiva de copiarlo y redistribuirlo, con o sin modificaciones, de manera comercial o no. En segundo término, esta licencia preserva para el autor o para quien publica una manera de obtener reconocimiento por su trabajo, al tiempo que no se considera responsable de las modificaciones realizadas por terceros, lo cual significa que los trabajos derivados del documento deben a su vez ser libres en el mismo sentido. Complementa la Licencia Pública General GNU, que es una licencia de copyleft diseñada para el software libre.

Hemos diseñado esta Licencia para usarla en manuales de software libre, ya que el software libre necesita documentación libre: Un programa libre debe venir con los manuales que ofrezcan las mismas libertades que da el software. Pero esta licencia no se

limita a manuales de software; puede ser usada para cualquier trabajo textual, sin tener en cuenta su temática o si se publica como libro impreso. Recomendamos esta licencia principalmente para trabajos cuyo fin sea instructivo o de referencia.

1. APLICABILIDAD Y DEFINICIONES

Esta Licencia se aplica a cualquier manual u otro documento que contenga una nota del propietario de los derechos que indique que puede ser distribuido bajo los términos de la Licencia. El "Documento", en adelante, se refiere a culquiera de dichos manuales o trabajos. Cualquier miembro del público es un como "Usted".

Una "Versión Modificada" del Documento significa cualquier trabajo que contenga el Documento o una porción del mismo, ya sea una copia literal o con modificaciones y/o traducciones a otro idioma.

Una "Sección Secundaria" es un apéndice titulado o una sección preliminar al prólogo del Documento que tiene que ver exclusivamente con la relación de quien publica, o con los autores del Documento, o con el tema general del Documento (o asuntos relacionados) y cuyo contenido no entre directamente en tal tema general. (Por ejemplo, si el Documento es en parte un texto de matemáticas, una Sección Secundaria puede no explicar matemáticas). La relación puede ser un asunto de conexión histórica, o de posición egal, comercial, filosófica, ética o política con el tema o la materia del texto.

Las "Secciones Invariantes" son ciertas Secciones Secundarias cuyos títulos son denominados como Secciones Invariantes en la nota que indica que el documento es liberado bajo esta licencia.

Los "Textos de Cubierta" son ciertos pasajes cortos de texto que se listan, como Textos de Título o Textos al respaldo de la página de tí, en la nota que indica que el documento es liberado bajo esta Licencia.

Una copia "Transparente" del Documento, significa una copia para lectura en máquina, representada en un formato cuya especificación está disponible al público general, cuyos contenidos pueden ser vistos y editados directamente con editores de texto genéricos o (para imágenes compuestas por pixeles) con programas genéricos de dibujo o (para dibujos) con algún editor gráfico ampliamente disponible, y que sea adecuado para exportar a formateadores de texto o para traducción automática a una variedad de formatos adecuados para ingresar a formateadores de texto. Una copia hecha en un formato de un archivo que no sea Transparente, cuyo formato ha sido diseñado para impedir o dificultar subsecuentes modificaciones posteriores por parte de los lectores no es Transparente. Una copia que no es "Transparente" es llamada "Opaca".

Como ejemplos de formatos adecuados para copias Transparentes están el ASCII plano sin formato, formato de Texinfo, formato de LaTeX, SGML o XML usando un DTD disponible publicamente, y HTML simple que siga los estándares, diseñado para modificaciones humanas. Los formatos Opacos incluyen PostScript, PDF, formatos propietarios que pueden ser leídos y editados unicamente en procesadores de palabras propietarios, SGML o XML para los cuáles los DTD y/o herramientas de procesamiento no están disponibles generalmente, y el HTML generado por máquinas, producto de algún procesador de palabras solo para propósitos de salida.

La "Pagina de Título" en un libro impreso significa, la pagina de titulo misma, más las páginas siguientes que sean necesaria para mantener, legiblemente, el material que esta Licencia requiere en la página de titulo. Para trabajos en formatos que no tienen página de título como tal, "Página de Título" significa el texto cercano a la aparición más prominente del título del trabajo, precediendo el comienzo del cuerpo del trabajo.

2. COPIA LITERAL

Puede copiar y distribuir el Documento en cualquier medio, sea en forma comercial o no, siempre y cuando esta Licencia, las notas de derecho de autor, y la nota de licencia que indica que esta Licencia se aplica al Documento se reproduzca en todas las copias, y que usted no adicione ninguna otra condición a las expuestas en en esta Licencia. Usted no puede usar medidas técnicas para obstruir o controlar la lectura o copia posterior de las copias que usted haga o distribuya. Sin embargo, usted puede aceptar compensación a cambio de las copias. Si distribuye un número suficientemente grande de copias también deberá seguir las condiciones de la sección 3.

También puede prestar copias, bajo las mismas condiciones establecidas anteriormente, y puede exhibir copias públicamente.

3. COPIADO EN CANTIDAD

Si publica copias impresas del Documento que sobrepasen las 100, y la nota de Licencia del Documento exige Textos de Cubierta, debe incluir las copias con cubiertas que lleven en forma clara y legible, todos esos Textos de Cubierta: Textos de título en la portada, y Textos al respaldo de la página de tílo. Ambas cubiertas deben identificarlo a Usted clara y legiblemente como quien publica tales copias. La portada debe mostrar el título completo con todas las palabras igualmente prominentes y visibles. Además puede adicionar otro material en la cubierta. Las copias con cambios limitados en las cubiertas, siempre que preserven el título del Documento y satisfagan estas condiciones, pueden considerarse como copias literales.

Si los textos requeridos para la cubierta son muy voluminosos para que ajusten legiblemente, debe colocar los primeros (tantos como sea razonable colocar) en la cubierta real y continuar el resto en páginas adyacentes.

Si publica o distribuye copias Opacas del Documento cuya cantidad exceda las 100, debe incluir una copia Transparente, que pueda ser leída por una máquina, con cada copia Opaca o entregar en o con cada copia Opaca una dirección en una red de computadores accesible publicamente que contenga una copia completa Transparente del Documento, sin material adicional, a la cual el público en general de la red pueda acceder a bajar anónimamente sin cargo usando protocolos públicos y estandarizados. Si usted hace uso de la última opción, deberá tomar medidas necesarias, cuando comience la distribución de las copias Opacas en cantidad, para asegurar que esta copia Transparente permanecerá accesible en el sitio por lo menos un año después de su última distribución de copias Opacas (directamente o a través de sus agentes o distribuidores) de esa edición al público.

Se solicita, aunque no es requisito, que contacte a los autores del Documento antes de redistribuir cualquier gran número de copias, para darle la oportunidad de que le provean una versión actualizada del Documento.

4. MODIFICACIONES

Puede copiar y distribuir una Versión Modificada del Documento bajo las condiciones de las seccions 2 y 3 anteriores, siempre que usted libere la Versión Modificada bajo esta misma Licencia, con la Versión Modificada haciendo el rol del Documento, por lo tanto licenciando la distribución y modificación de la Versión Modificada a quienquiera posea una copia de esta. Además, debe hacer lo siguiente en la Versión Modificada:

- **A.** Usar en la Página de Título (y en las cubiertas, si hay alguna) un título distinto al del Documento, y de versiones anteriores (que deberían, si hay alguna, estar listados en la sección de Historia del Documento). Puede usar el mismo título de versiones anteriores al original siempre y cuando quien publicó originalmente otorge permiso.
- **B.** Listar en la Página de Título, como autores, una o más personas o entidades responsables por la autoría o las modificaciones en la Versión Modificada, junto con por lo menos cinco de los autores principales del Documento (Todos sus autores principales, si hay menos de cinco).
- **C.** Incluir en la Página de Título el nombre de quién publica la Versión Modificada, como quien publica.
- **D.** Preservar todas las notas de derechos de autor del Documento.
- **E.** Adicionar una nota de derecho de autor apropiada a sus modificaciones, adyacente a las otras notas de derecho de autor.
- **F.** Incluir, immediatamente después de la nota de derecho de autor, una nota de licencia dando el permiso público para usar la Versión Modificada bajo los términos de esta Licencia, de la forma mostrada en el anexo al final de este documento.
- **G.** Preservar en esa nota de licencia el listado completo de Secciones Invariantes y de los Textos de Cubiertas que sean requeridos como se especifique en la nota de Licencia del Documento
- **H.** Incluir una copia sin modificación de esta Licencia.
- **I.** Preservar la sección llamada "Historia", y su título, y adicionar a esta una sección estableciendo al menos el título, el año,los nuevos autores, y quién publique la Versión Modificada como reza en la Página de Título. Si no hay una sección titulada "Historia" en el Documento, crear una estableciendo el título, el año, los autores y quieng publicó el Documento como reza en la Página de Título, añadiendo además una sección describiendo la Versión Modificada como se estableció en la oración anterior.
- **J.** Preservar la localización en red, si hay , dada en la Documentación para acceso público a una copia Transparente del Documento, tanto como las otras direcciones de red dadas en el Documento para versiones anteriores en las cuáles estuviese basado. Estas pueden ubicarse en la sección "Historia". Se puede omitir la ubicación en red para un trabajo que sea publicado por lo menos cuatro años antes que el Documento mismo, o si quien publica originalmente la versión da permiso explícitamente.
- **K.** En cualquier sección titulada "Agradecimientos" o "Dedicatorias", preservar el título de la sección, y preservar en la sección toda la sustancia y el tono de los agradeimientos y/o dedicatorias de cada contribuyente que estén incluídas.

- **L.** Preservar todas las Secciones Invariantes del Documento, sin alterar su texto ni sus títulos. Números de sección o el equivalente no son considerados parte de los títulos de la sección.
- **M.** Borrar cualquier sección titulada "Aprobaciones". Tales secciones no pueden estar incluidas en las Versiones Modificadas.
- **N.** No retitular ninguna sección existente como "Aprobaciones" o conflictuar con el título de alguna Sección Invariante.

Si la Versión Modificada incluye secciones o apendices nuevos o preliminares al prólogo que califiquen como Secciones Secundarias y contienen material no copiado del Documento, puede opcionalmente designar algunas o todas esas secciones como invariantes. Para hacerlo, adicione sus títulos a la lista de Secciones Invariantes en la nota de licencia de la Versión Modificada. Tales títulos deben ser distintos de cualquier otro título de sección.

Puede adicionar una sección titulada "Aprobaciones", siempre que contenga unicamente aprobaciones de su Versión Modificada por varias fuentes--por ejemplo, observaciones de peritos o que el texto ha sido aprobado por una organización como la definición oficial de un estándar.

Puede adicionar un pasaje de hasta cinco palabras como un Texto de Portada, y un pasaje de hasta 25 palabras al respaldo de la página de título al final de la lista de Textos de Cubierta en la Versión Modificada. Solamente un pasaje de Texto de Portada y uno al respaldo de la página de título puede ser adicionado por (o a manera de arreglos hechos por) una entidad. Si el Documento ya incluye un texto de cubierta para la misma portada o al respaldo de la portada, previamente adicionado por usted o por arreglo hecho por la misma entidad, a nombre de la cual usted está actuando, usted no puede adicionar otro; pero puede reemplazar el anterior, con permiso explícito de quien publicó previamente y agregó el texto anterior

El(los) autor(es) y quien(es) publica(n) el Documento no dan con esta Licencia permiso para usar sus nombres para publicidad o para asegurar o implicar aprobación de cualquier Versión Modificada.

5. COMBINANDO DOCUMENTOS

Puede combinar el Documento con otros documentos liberados bajo esta Licencia, bajo los términos definidos en la sección 4 anterior para versiones modificadas, siempre que incluya en la combinación todas las Secciones Invariantes de todos los documentos originales, sin modificar, y listadas todas como Secciones Invariantes del trabajo combinado en su nota de licencia.

El trabajo combinado necesita contener solamente una copia de esta Licencia, y múltiples Seccions Invariantes idénticas pueden ser reemplazadas por una sola copia. Si hay múltiples Secciones Invariantes con el mismo nombre pero con contenidos diferentes, haga el título de cada una de estas secciones único adicionándole al final del mismo, entre paréntesis, el nombre del autor o de quien publicó originalmente esa sección, si es conocido, o si no, un número único. Haga el mismo ajuste a los títulos de sección en la lista de Secciones Invariantes en la nota de licencia del trabajo combinado.

En la combinación, debe combinar cualquier sección titulada "Historia" de los varios documentos originales, formando una sección titulada "Historia"; de la misma forma combine cualquier seción titulada "Agradecimientos", y cualquier sección titulada "Dedicatorias". Debe borrar todas las secciones tituladas "Aprobaciones."

6. COLECCIONES DE DOCUMENTOS

Puede hacer una colección consistente del Documento y otros documentos liberados bajo esta Licencia, y reemplazar las copias individuales de esta Licencia en los varios documentos con una sola copia que esté incluida en la colección, siempre que siga las reglas de esta Licencia para cada copia literal de cada uno de los documentos en cualquiera de los demás aspectos.

Puede extraer un solo documento de una de tales colecciones, y distribuirlo individualmente bajo esta Licencia, siempre que inserte una copia de esta Licencia en el documento extraido, y siga esta Licencia en todos los otros aspectos concernientes a la copia literal de tal documento.

7. AGREGACIÓN CON TRABAJOS INDEPENDENTES

Una recopilación del Documento o de sus derivados con otros documentos o trabajos separados o independientes, en cualquier tipo de distribución o medio de almacenamiento, no como un todo, cuenta como una Versión Modificada del Documento, siempre que no se aleguen derechos de autor por la recopilación. Tal recopilación es llamada un "agregado", y esta Licencia no aplica a los otros trabajos auto-contenidos así recopilados con el Documento, o a cuenta de haber sido recopilados, si no son ellos mismos trabajos derivados del Documento.

Si el requerimiento de la sección 3 sobre el Texto de Cubierta es aplicable a estas copias del Documento, entonces si el Documento es menor que un cuarto del agregado entero, los Textos de Cubierta del Documento pueden ser colocados en cubiertas que enmarquen solamente el Documento entre el agregado. De otra forma deben aparecer en cubiertas enmarcando todo el agregado.

8. TRADUCCIÓN

La Traducción es considerada como una clase de modificación, Así que puede distribuir traducciones del Documento bajo los términos de la sección 4. Reemplazar las Secciones Invariantes con traducciones requiere permiso especial de los dueños de derecho de autor, pero usted puede incluir traducciones de algunas o todas las Secciones Invariantes adicionalmente a las versiones originales de tales Secciones Invariantes. Puede incluir una traducción de esta Licencia siempre que incluya también la versión en Inglés de esta Licencia. En caso de un desacuerdo entre la traducción y la versión original en Inglés de esta Licencia, la versión original en Inglés prevalecerá.

9. TERMINACIÓN

No se puede copiar, modificar, sublicenciar, o distribuir el Documento excepto por lo permitido expresamente bajo esta Licencia. Cualquier otro intento de copia, modificación, sublicenciamiento o distribución del Documento es nulo, y serán automáticamente

terminados sus derechos bajo esa licencia. Sin embargo, los terceros que hayan recibido copias, o derechos, de su parte bajo esta Licencia no tendrán por terminadas sus licencias siempre que tales personas o entidades se encuentren en total conformidad con la licencia original.

10.REVISIONES FUTURAS DE ESTA LICENCIA

La Free Software Foundation puede publicar versiones nuevas y revisadas de la Licencia de Documentación Libre GNU de tiempo en tiempo. Tales nuevas versiones serán similares en espíritu a la presente versión, pero pueden diferir en detalles para solucionar nuevos problemas o intereses. Vea http://www.gnu.org/copyleft/.

Cada versión de la Licencia tiene un número de versión que la distingue. Si el Documento especifica que se aplica una versión numerada en particular de esta licencia o "cualquier versión posterior", usted tiene la opción de seguir los términos y condiciones de la versión especificada o cualquiera posterior que haya sido publicada (no como un borrador) por la Free Software Foundation. Si el Documento no especifica un número de versión de esta Licencia, puede escoger cualquier versión que haya sido publicada(no como un borrador) por la Free Software Foundation.

Para usar esta licencia en un documento que usted haya escrito, incluya una copia de la Licencia en el documento y ponga el siguiente derecho de autor y nota de licencia justo después de la página de título:

Derecho de Autor (c) 2004 Rafael Carmona Grande
Se otorga permiso para copiar, distribuir y/o modificar este documento bajo los términos de la Licencia de Documentación Libre GNU, Versión 1.1 o cualquier otra versión posterior publicada por la Free Software Foundation; con las Secciones Invariantes siendo **TODOS LOS TÍTULOS.**
Una copia de la licencia es incluida en la sección titulada "Licencia de Documentación Libre GNU".

Si no tiene Secciones Invariantes, escriba "Sin Secciones Invariantes" en vez de decir cuáles son invariantes. Si no tiene Texto de Portada, escriba "Sin Texto de Portada" en vez de "con los Textos de Portada siendo LISTELO"; e igualmente para los Textos al respaldo de la página de título.

Si su documento contiene ejemplos de código de programa no triviales, recomendamos liberar estos ejemplos en paralelo bajo su elección de licencia de software libre, tal como la Licencia Pública General GNU (**GNU General Public License**), para permitir su uso en software libre.

1. N. del T. *copyleft* es un nuevo término acuñado por GNU. Nace de un juego de palabras en Inglés, en vez de "copyright" usan "copyleft", indicando que no se restringe la copia, sino por el contrario se permite mientras que el derecho a seguir copiando no se restrinja. http://www.gn\u.org/copyleft/copyleft.es.html

2. N. del T. *licenciatario*: A quien se le otorga la licencia.

1. Prologo

Una dura batalla se esta surcando por las sucias aguas del planeta ' Lenguaje ', donde la caída del antiguo Imperio 'HAL-1' a sido aprovechado por los secuaces del nuevo Imperio $CELDA$.

Las tropas del Imperio $CELDA$ esta expandiendo sus dominios sobre las cálidas aguas del inmenso océano 'InterWater' arrastrándose toda cosa viviente, después de su sonado fracaso de crear océanos por sí misma.
No hay compasión.
Primero invaden con pequeñas embarcaciones los mares del Sur.
Después continúan con su afán de conquista aliándose con el mismísimo ‘Emperador AsU’.
Para poder atacar desde todos los frentes... La amenaza esta al acecho...

En otra parte del planeta, localizados en 680.123.45.3 , unos pequeños estados con poder , conocidos como la alianza "SIHIC" continúa evolucionando sus 'maquinarias' para impedir dicho contraataque del Imperio $CELDA$.

Conocen a su enemigo, sus tácticas, todos ellos las sufrieron o las siguen sufriendo, y están preparados para la 'Guerra del Lenguaje'.

En un punto de la Antártica, desconocido por ahora, pero que seguro están localizados en 127.0.0.1, aunque dicho localización no puede ser rastreada, se encuentran las tropas rebeldes 'G-FREE.
Comandadas por Ricardo 'Corazón de Pingüino' y por su lugarteniente 'LiX-T', esta contraatacando continuamente.
Es una lucha que no piensan perder. Ellos saben que el Imperio $CELDA$ esta copiando toda su base tecnológica. No les importa. Ellos están seguros que ganaran la guerra, y los resultados parecen ser mucho antes de lo previsto, puestos que están viendo como la Alianza "SIHIC" esta virando hacia la Antártica en busca de captar aliados para la batalla final.

Por ahora, un buen refugio parece ser los incomprendidos del planeta 'M.A.C', donde su tecnología lleva ya años por delante de cualquier cosa conocida, en parte por su todo su 'misterio' que rodea a dicho planeta.

Y es que los viajes galácticos cuestan una pasta...

Mientras tanto, el Imperio $CELDA$ esta terminando y en fase de prueba las nuevas tropas, un conjunto de Fragatas VC, Portaaviones V-S, Aviones V6 y submarinos C-#0.

Pero, todavía quedan marineros, donde para pescar con redes no les suponen ningún problema. Ellos tienen el poder de decidir como pescar, ver las hierbas crecer debajo del mar sobre sus aguas limpias de 'algas-rítmicas property'.

Han renunciado en parte a 'ver' lo que unos quieren que vean con los ojos de otro, no con los propios.

No les importa demasiado las maniobras del Imperio ni de la Alianza.

Solamente simpatizan con las tropas 'G-FREE', puesto que gracias en parte a su filosofía de navegación, pueden ponerse al mando de un nuevo barco, conocido como:

- 'HARBOUR. El ultimo XBASE'.

Otros han añadido el Motor '5-v2.3' para estar preparados para contraatacar al Imperio $CELDA$ si fuese necesario. Pelearan hasta quedar exhaustos.

Quizás por ello, sean los 'nuevos bucaneros del planeta', o él ultima foco de resistencia, pero eso es otra historia.

INTRODUCCION

Este libro sobre Harbour es un esfuerzo personal sobre el uso de este maravilloso compilador Open Source.

El motivo de este libro de Harbour es que no me gusta NADA lo que he encontrado. Todo es muy técnico o nada claro o simplemente no existe.

Este libro estará orientado para niveles un poco avanzados, sobretodo para los programadores de Fivewin.

Yo no soy un 'experto' en todos los temas, pues el tiempo requerido seria demasiado alto, y como yo no me gano la vida programando, no dispongo de mucho tiempo para investigación.

El tiempo libre generalmente lo dedico para investigar temas muy concretos que me afectan o me dedico a mi familia o sencillamente me divierte averiguar un tema en concreto.

Por que este titulo? Por que parece que siempre seremos unos nostálgicos de los lenguajes XBASE, ahora que lo 'visual' esta de moda.

Quiero también desde aquí hacer una mención especial a los amigos:

Joaquim Ferrer
Mi buen amigo de penas y alegrías sobre Fivewin. Gracias compañero!!!.

Rene Flores.
Mi 'Maestro' en San Sebastian y un amigo de corazón.

Manu Exposito y Eduardo Riscart
En mi memoria os tendré siempre por enseñarme Sevilla tipo 'Japonés'.
Dos bellísimas personas.

Felix Pablo Grande.
Este tío tiene un 'gran corazón'. Gracias Felix!.

A la gente de GO2000, por darme la oportunidad de conocer a mas gente:
Manu Calero, Jose Luis Sanchez, Jose Alfonso, etc... a todos ellos gracias.

A la gente del Foro de Spanish de Fivetech.
A Hernán, Ricardo, Luis Krause, Manuel Mercado, y otros tantos, a todos vosotros gracias y en especial a mis amigos de Argentina.

Angel Martin.
La parte de las clases Timprime / TutilPrn esta dedicado a la memoria de
Una gran persona que nos tuvo que dejar.
Gracias Ángel, estés donde estés.

Antonio Linares.
Por crear las Five. Aunque a veces me ignoras, te lo perdono por esta vez.
Gracias Antonio por hacer posible que me divierta programando. ;)

Pido perdón si me olvido de alguien, a todos :

MUCHAS GRACIAS AMIGOS!!!.

De Clipper a Harbour

Un nuevo comienzo

Como dice el refrán, " A Rey Muerto, Rey Puesto ".
Esto le viene como anillo al dedo al lenguaje Harbour.

Descartando al C, creo sin exagerar, que el lenguaje Clipper es el que a tenido más programadores a lo ancho y largo del planeta que ningún otro lenguaje. Miles y miles de librerías de terceros, miles de aplicaciones escritas con Clipper, pero no sé porque, CA 'mato' a Clipper.

Clipper murió porque no supieron dotarle para realizar aplicaciones para Windows. CA vino con un Visual Objects, pero esto no era lo que los programadores de Clipper esperaban.

' Crónica de una muerte anunciada ' defino yo la muerte de Clipper.
Ahora bien, Clipper como lenguaje en si no había muerto. Lo que verdaderamente murió fue la interfaz con el usuario.

Productos como Clip4Win y más concretamente Fivewin, nos brindaron la oportunidad de seguir con nuestro querido Clipper pero realizando ejecutables para WINDOWS!!!.

¿ Tanto les costaba a CA haber hecho lo que esta gente realizó?
¿ Cómo es posible que una casa de software reconocida en el ámbito mundial por su capacidad de creación, engendró a VO y no supieron o no querían hacer una cosa tan simple como Clip4Win o Fivewin?

Ahora, podíamos seguir con Clipper y productos de terceros como Clip4Win o Fivewin para realizar nuestras aplicaciones corriendo bajo Windows.

Pero esto ya no será posible. Las nuevas tecnologías, la programación en 32 bits, que ya están empezando a recaer para dar paso a las de 64bits, hacen que programar con Clipper & Clip4Win || Fivewin, sea el final de una etapa, la de los 16 bits.

Una vez más, lo programadores de Clipper estamos contra la espada y la pared.

O aprendemos un nuevo lenguaje, muchos programadores de Clipper se pasaron o bien a Visual Basic o a Delphi, o nos dedicamos al cultivo de los caracoles.

Pero como dice el refrán ' **No hay mal que dure cien años** ', tenemos un nuevo lenguaje compatible con Clipper: **HARBOUR.**

Pero Harbour va más allá de lo que Clipper debería haber sido.
Implementación Multi-Plataforma, Programación Orientado al Objeto, código en C en LINEA, productos de terceras partes como Fivewin, ADS, etc.

Mucha gente esta colaborando para hacer de Harbour de ese Clipper que todos nosotros hubiésemos querido.

Es el resurgir de un nuevo lenguaje en que, si venimos del mundo Clipper, no deberemos de aprender prácticamente nada.
Años y años de formación te seguirán siendo validos con Harbour.

Las principales características de Harbour son:

- Multiplataforma. Dices de un mismo lenguaje para que funcione en diferentes sistemas operativos. Así es como yo definiría lo que es la multiplataforma. Tenemos a Harbour corriendo en:

  - Windows (Para una mayor rapidez se aconseja en uso de Fivewin)
  - D.O.S
  - Linux (Harbour ya piensa el futuro)
    Decir que hay ya un proyecto para usar GTK para Harbour.

Esto se consigue porque realmente Harbour genera PCODE.
Me acuerdo que la gente de otros lenguajes me decían o decían que Clipper era una birria, ya que no generaba auténticos EXE, si no que generaba un código que era interpretado por la Virtual Machine.

Yo ahora me rió de esa gente que dice que Java es uno de los mejores lenguajes por ser Multiplataforma, etc. , pues bien, Java hace los mismo que hacia Clipper hace ya mas de 15 años.

¿ Dónde esta ahora esa gente que criticaba el PCODE?
- POO. Programación orientado al Objeto.

Hoy en DIA es inconcebible un lenguaje que no tenga soporte POO. La POO es el pilar de la nueva forma de programar.
Con ella te darás cuenta realmente que la simplificación de código es abismal y la reutilización de código ahora si que podemos decir que es una realidad palpable.

- Código C en línea.
Si señores, esto es una maravillosa idea.
¿ Os acordáis como en C podemos usar ensamblador directamente, haciendo uso de la cláusula ASM { ..código asm... } ?
Eso mismo podemos hacer nosotros dentro de un código fuente en Harbour.

```
/* Código Harbour */
Function Main()

  ? Hola()

RETURN NIL

/* Código C */

#pragma BEGINDUMP

#include <windows.h>
#include "hbapi.h"

HB_FUNC( HOLA )
{
 hb_retc( "Hola Mundo" );
}

#pragma ENDDUMP
```

¿ Y como funciona el invento? La explicación es muy simple.
Recordad que Harbour NO genera OBJ , si no que te genera código C, el cual después a partir de un compilador de C, nos crea el ejecutable.

Entonces, a la hora de preprocesar el código fuente PRG, las directivas #pragma BEGINDUMP... ENDDUMP, le estamos diciendo a Harbour que todo lo que este dentro de estar par de directivas no lo debe de tocar.
A ser la ultima fase, la creación del EXE, un trabajo del compilador de C, y como nosotros hemos escrito en lenguaje C, obtendremos el resultado esperado: Código C dentro de un PRG.

ES UNA IDEA GENIAL!!
Más adelante veremos como funciona esto.

Todo esto y más cosas que veremos más adelante en HARBOUR.

Harbour es el futuro para los desarrolladores de XBASE. Quizás la única pega es que por ahora, aunque muy en pañales, la programación para Windows o GTK no siga los mismos pasos en el desarrollo como la versión para consola.

Para la versión Windows dispones de Fivewin, donde crearas auténticos ejecutables de 32 bits bajo Windows!.

Para Linux a través de GTK todavía hay que esperar para verlo realizado.

Para Windows hay gente que ya esta haciendo el trabajo que en su DIA hizo Fivetech con Fivewin. Si quieres esperar que este operativo Harbour bajo Windows nativamente o bien optas por comprar las librerías Fivewin.

Mi humilde consejo es que Fivewin ya es un producto bastante probado, aunque con algunos fallos, Fivetech intenta y lo van logrando en corregir los posibles bugs que puedan contener, si quieres realmente a empezar a programar bajo Windows, hazte con estas librerías.

Hay otro producto, Clip4Win, pero desconozco si hay soporte para Harbour. Creo que John Sketon también ha hecho compatible Clip4Win para Harbour, pero no te lo puedo asegurar.

No quiero desde aquí entrar en polémicas. ;) Cada cual escoja la opción que más le plazca.

2.1.- Conexión Con C

En este apartado veremos la parte de una de las cosas más interesantes que posee Harbour que es su conexión directa en el mismo modulo .prg con el lenguaje C.

Primeramente debemos saber porque se hace uso de C y no del C++. Hemos dicho del lenguaje C independientemente del fabricante, puesto que Harbour esta construido sobre la base de las normas ANSI que determinan la compatibilidad del lenguaje. Así cada fabricante tiene sus propias funciones, que pueden ser incompatibles a nivel de función o de sistema operativo, y la portabilidad seria prácticamente nula.

Al seguir el standard del consorcio que define el lenguaje C, Harbour es ya independiente del un lenguaje C en particular.

Funciones Standard definidas por ANSI (ISO/IEC 9899:1990)

Debemos saber en primer lugar como funciona el compilador y porque podemos realizar dicho 'milagro' a los ojos de un pagano.

Cuando nosotros compilamos, podemos indicar el tipo de salida que generara el compilador.

Normalmente, todos los compiladores generan un fichero objeto, .OBJ, (no voy a explicar ahora a estas alturas que es un fichero .OBJ), cosa que de momento Harbour no realiza.

En vez de ello, nos genera un fichero .C, es decir, el compilador de Harbour podríamos decir que funciona como un 'traductor' del lenguaje Harbour a C.

Si vemos el código del fichero .C y tienes algo de idea del lenguaje C, veras que es prácticamente ilegibles. La razón de ello, es que lo que estas viendo son las tablas de símbolos, y los PCODES de ejecución, que lanzados a la maquina virtual, realizan lo que tu has escrito en el lenguaje Harbour.

Por ejemplo, la típica ventana 'Hola Mundo' codificada en Harbour, quedaría así:

```
Function Main()

  ? "Hello World"

Return Nil
```

Si compiláramos dicho código en Harbour, obtendremos lo siguiente :

```
/*
 * Harbour Compiler, Apha build 38.1 (Flex) ---> Version Actual
 * Generated C source code
 */

#include "hbvmpub.h"
#include "hbinit.h"


HB_FUNC( MAIN );
extern HB_FUNC( QOUT );


HB_INIT_SYMBOLS_BEGIN( hb_vm_SymbolInit_HELLO )
{ "MAIN", HB_FS_PUBLIC | HB_FS_FIRST, HB_FUNCNAME( MAIN ), NULL },
{ "QOUT", HB_FS_PUBLIC, HB_FUNCNAME( QOUT ), NULL }
HB_INIT_SYMBOLS_END( hb_vm_SymbolInit_HELLO )

#if defined(_MSC_VER)
   #if _MSC_VER >= 1010
     #pragma data_seg( ".CRT$XIY" )
     #pragma comment( linker, "/Merge:.CRT=.data" )
   #else
     #pragma data_seg( "XIY" )
   #endif
   static HB_$INITSYM hb_vm_auto_SymbolInit_HELLO = hb_vm_SymbolInit_HELLO;
   #pragma data_seg()
#elif ! defined(__GNUC__)
   #pragma startup hb_vm_SymbolInit_HELLO
```

```
#endif

HB_FUNC( MAIN )
{
 static const BYTE pcode[] =
 {
   36, 9, 0, 108, 1, 100, 106, 12, 72, 101, 108, 108, 111, 32,
  119, 111, 114, 108, 100, 33, 20, 1, 36, 11, 0, 100, 110, 7
 };

 hb_vmExecute( pcode, symbols );
}
```

Como podéis observar , al final de todo , se le pasa a la maquina virtual de Harbour , el PCODE y la tabla de símbolos y esta es ejecutada en memoria.

Esto nos da , al contrario de lo que podáis pensar, una flexibilidad y potencia que con los .OBJ no lo tendríamos de inmediato, si no que habría que realizar un paso previo después de esto. Si la salida fuese un .OBJ directamente, para poder hacer nuestras rutinas en C, lo deberíamos de hacer en otro fichero aparte y después a través del linkado , crear el ejecutable.

De esta manera si nosotros escribimos en C , dentro del mismo modulo .prg, el fichero resultante sera:

El codigo en harbour .prg + codigo en C dentro del .prg := ANSI C listo para compilar.

Cual de ellos elegir ? Por supuesto que más elegante es la primera opción, pues es menos lioso y además el resultado es el mismo.

Ahora bien para poder estableces funciones en C para ser vistas desde Harbour, la primera condición viene establecida por la cláusula
#pragma BEGINDUMP

#pragma BEGINDUMP, ENDDUMP
Significa que le estamos diciendo al compilar de Harbour, que lo que esta dentro de estas directivas no lo tiene que preprocesar y lo deje tal cual.
De esta manera , podemos escribir código C puro y duro para ser usado por Harbour.

Ahora bien, hay que tener en cuenta que para manejar tipos de datos entre Harbour y C no lo podemos hacer libremente.

Para ello, existe un API llamado "hbapi.h" en el cual usaremos para el intercambio de información y para que Harbour pueda procesarlo.
Ello no implica que podemos usar código C estándar dentro de una función en C que será llamada desde Harbour.
Es decir, podemos hacer una llamada a una función en C nuestra que no tenga nada que ver con el API de Harbour.

¿ Y yo pregunto, porque necesariamente debo de usar C si lo puedo realizar directamente en Harbour ?

Bien, usted esta en lo cierto. Lo único importante de hacerlo en C con respecto en Harbour es simplemente y llanamente por cuestión de velocidad.

El lenguaje C es conocido por ser capaz de hacer código compacto y muy rápido. Nunca llegara al nivel del Ensamblador, pero es que el nivel de programación en Ensamblador es tan complejo para cualquier cosa, que si no es por una cuestión CRITICA, no lo va hacer ninguna rutina en ensamblador.
Si necesitas hacer algo en ensamblador por puro cuestión critica, le aconsejo que cambie de compilador y programes en C directamente.

Al contrario del código Harbour que debe de ser ejecutado por la maquina virtual, el código C que use en su programa, a excepción de la llamadas a HB_FUNC(<mi_funcion>), no será ejecutadas por la maquina virtual.
Al no intervenir la maquina virtual, la funciones en C son MUCHO mas rápidas que con la codificación anterior.

Pero eso no debe de preocuparse. Es más anecdótico, pues con la potencia actual de los micros, 2Ghz, no debemos de preocuparnos demasiado.

¿ Entonces para que lo hago servir ?
Pues para , por ejemplo, acceder a otras API de otros fabricantes.
En el siguiente capitulo veremos como acceder a una API de gráficos a través de Harbour en C.

Bueno, habíamos quedado que para hacer una función entre Harbour y C, debíamos de hacerlo a través de un API que dispone Harbour.

Imaginemos que queremos enseñar un alert hecho por nosotros bajo Windows, sin hacer uso de las librerías de Fivewin.

Lo primero que tenemos que hacer es nuestro programa eh Harbour:

```
Function Main()

  MiAlertaEnC()

return Nil
```

A continuación debemos de poner la directiva #pragma

```
#pragma BEGINDUMP
#include <windows.h>  --> Includes del Lenguaje C para Win32
#include "hbapi.h"       --> Nuestros includes o los de harbour
```

La diferencia entre <> y "" es que si lo ponemos <> ira a buscar los ficheros de cabecera en el directorio donde le hayamos especificado dentro del BCC32.CFG del directorio BIN del Lenguaje C(en el caso de Borland).
Mientras que "" ira al directorio donde nosotros le especificaremos en la línea de comandos.

Si hacemos uso de Front-End que diseñe para ayuda a la compilación de Harbour, esto es pan comido ;)

Ahora nos queda definir nuestra función en C: (Ver Func1.prg)

```
HB_FUNC( MIALERTAENC )
{
MessageBox( GetActiveWindow(),  "ESTA ES MI ALERTA" ,"Mialerta",
            MB_ICONINFORMATION ) ;
}
```

PRESTAR MUCHA ATENCION!!!

Debemos de poner en mayúsculas SIEMPRE la función HB_FUNC( ) y dejando los espacios correspondientes, tal y como lo veis arriba. Si no lo hacéis, os dará errores en la compilación.

En este caso nos saldría un MessageBox sin tener que hacer uso de las librerías de Fivewin. Estamos accediendo directamente al API de Windows desde un programa hecho en Harbour puro y duro.

Quizás este ejemplo no tenga mucho sentido , pero es para que comprendas de un modo simple el funcionamiento.

Claro que ahora, cuando nos adentramos en el C, encima de tenernos que pelear con el Harbour, con Fivewin y ahora con el C, el lío mental puede provocarnos una derrame en el cerebelo superior izquierdo ;)

La rutina de arriba la podemos mejorar pasándole parámetros desde Harbour y recogiéndolo desde el C. ¿ Como lo haremos ? Muy simple.

Imaginemos ahora el ejemplo de arriba. Estaba muy limitado pues simplemente lo que hacia era enseñarnos un dialogo con un texto ya definido y punto. Ahora veremos como podemos mejorarlo, pasándole parámetros desde Harbour y los trataremos desde el propio C.

Siguiendo el mismo patrón del ejemplo anterior, modificaremos las líneas pertinentes;

```
HB_FUNC( MIALERTAENC )
{
  MessageBox( GetActiveWindow(),  hb_parc( 1 ), hb_parc( 2 ), hb_parni( 3 ) ) ;
}
```

Ahora , nuestra función en C esta esperando desde Harbour 3 nuevos parámetros:

```
#define MB_ICONINFORMATION   0x00000040 // 64 decimal

FUNCTION Main()

   MiAlertaEnC("Alentando al personal","Titulo Alerta", MB_ICONINFORMATION )

RETURN NIL
```

Lo podéis comprobar si compiláis el ejemplo func2.prg

Como podéis apreciar , Harbour te permite definir números en formato Hexadecimal. Esto va de fábula, sobre todo a la hora de trabajar con el C, ya que todas las constantes casi siempre están codificadas en este formato, de lo contrario, como cuando lo hacíamos en Clipper, deberemos de transformarlas en decimal, para poder hacer uso de ellas.
Podéis comentar la cadena 0x0000040 y cambiarla por 64. Veréis que el comportamiento es exactamente lo mismo.

Ahora bien , nos queda ver como desde C podemos devolver valores a Harbour. Si el ejemplo anterior lo definimos como:

```
FUNCTION Main()

 // Muestra valor de retorno de la funcion MiAlertaEnC
 Alert( MiAlertaEnC(" Al personal","Titulo Alerta", MB_ICONINFORMATION ) )

RETURN NIL
```

Y la función en C la definimos así :

```
HB_FUNC( MIALERTAENC )
{
 hb_retni( MessageBox( GetActiveWindow(),  hb_parc( 1 ), hb_parc( 2 ),
                       hb_parni( 3 ) ) ) ;
}
```

Estamos indicando que devuelva a Harbour un valor numérico devuelto por la función MessageBox.

Compilar o ejecutar FunC3.prg para ver el funcionamiento.

Todo esto que he explicado funciona tanto como si haces uso de las librerías *Fivewin for Harbour* , como en Harbour puro y duro.
Ver como sale un MessageBox en un dialogo y después una caja de alert, queda horrible ;), pero es un pelín curioso.

Para pasar desde Harbour a C parámetros existe un API de Harbour para tratar de recibir correctamente lo que se espera.

Si la cadena a enviar es un texto, usaremos para recibir el valor la función **hb_parc(Numero)**, donde Numero es la posición que ocupa nuestra valor pasado desde Harbour.
Imaginemos que vamos a pasar un valor solamente, pues lógicamente siempre será Numero = 1, esto es, hb_parc(1).
Ahora vamos a pasar dos, "cadena" y 12345. Lógicamente el primero es el que hará uso de **hb_parc(1)**. En cambio si el orden es 12345 y "cadena", el uso de **hb_parc(2)** es el que contiene la cadena.

Tenemos que tener especial atención como nos llegan desde el Harbour los parámetros , para así nosotros desde el función en C actuar en consecuencia.
Si desde Harbour ,
MiAlertaEnC(MB_ICONINFORMATION,"titulo","string a mostrar"),
lo codificamos de esta manera, en la funcion HB_FUNC( ) deberemos de retocar el paso de parámetros, quedando así:

```
MessageBox( GetActiveWindow(),  hb_parni( 1 ), hb_parc( 3 ), hb_parc( 2 ) )
```

Otra cosa que os puede llegar a pasar es que el compilador os eche warnings por el tipo de dato pasado cuando esperaba otro.

Esto es muy simple de solucionar.
Imaginemos que vamos a tratar de enviarle desde Harbour un handle de una ventana a C. Dicho handle debe de ser tratado como HDC, pero nosotros no tenemos ese tipo 'especial' pues simplemente existe para windows.

Ahora bien, podemos hacer que la función que requiere un **HDC**, nosotros lo manejaremos como un parámetro pasado como **LONG, hb_parnl( ).**

Si cogiéramos la función de arriba, la podemos convertir a:

```
MessageBox(( HDC )hb_parnl( 4 ),  hb_parni( 1 ), hb_parc( 3 ), hb_parc( 2 ) )
```

Entonces le estamos diciendo al compilar que le estamos pasando el tipo que el espera, y no nos dará más el coñazo.

Entonces la nueva función espera un nuevo parámetro que será el handle de la ventana o dialogo.

Ahora veamos la funciones de recogía de parámetros desde Harbour.

- **hb_parc()** El valor pasado es una cadena.
- **hb_parni()** El valor pasado es un numero.
- **hb_parnl()** El valor pasado en un LONG.
- **hb_parnd()** El valor pasado es un DOUBLE.

Los posibles retornos de C para harbour son:

- **hb_ret()** Retorna un valor NIL.
- **hb_retni()** Retorna un valor entero.
- **hb_retc()** Retorna una cadena.
- **hb_retnl()** Retorna un valor LONG
- **hb_retnd()** Retorna un valor DOUBLE.

Hay varios tipos más , pero no me extenderé a ello por ser poco comunes.

Hablando de tipos numéricos, que es lo que puede llegar a confundirnos. Todo esto es así porque , a diferencia del C, en Harbour trataremos los datos de una manera distinta. Para nosotros solamente existen si un valor es numérico o no. El tipo de numérico no nos importa, por la sencilla razón que a Harbour le importa un pimiento.

Pero en C, si hay que saberlo para seguir que tipo de dato vamos a hacer uso. Esto es por la sencilla razón de que el compilador reservara más o menos espacio en memoria dependiendo del tipo de dato.
Por ejemplo , si queremos desde Harbour enviar al lenguaje C el numero 12.459, no podemos tratarlo como un entero, esto, **hb_parni(**), si no como un DOUBLE, **hb_parnd(**). Y al igual , el paso desde C a Harbour, no podemos hacer, hb_retni(12.45), pues solamente cogeremos 12.

Leeros el manual de C para el tipo de datos existentes y actuar en consecuencia a la hora de pasar y/o recibir los datos.

Es muy común cometer errores aquí, así que si algún no funciona , puede deberse a como estamos tratando el tipo de dato recibido o/y enviado.

USANDO UNA DLL CUALQUIERA.

Otra cosa muy importante y que veremos en el siguiente capitulo es el uso de una DLL cualquiera a través del lenguaje C y hacer las llamadas desde Harbour a la DLL como si fueran nuestras propias funciones.

Dentro de Borland C++ 5.5 ,tenemos disponibles una utilidad llamada **IMPLIB**.

Esta utilidad nos es de especial interés pues nos creara una librería a partir de la DLL que queremos usar, para que nuestro compilador resuelva las referencias a funciones externas a la aplicación.
Dicho de otra manera. Si dentro de nuestra DLL tenemos una función que se llama , por ejemplo, LoadPNG(), una de las maneras elegantes de hacerlas servir, es hacerlo con IMPLIB.

Si , ya se que desde Fivewin podemos hacer llamadas a una DLL sin tener que hacer un wrapper en C, pero la velocidad de ejecución es muchísimo más lenta que si lo hacemos de esta manera.

Bueno, seguimos. Cogemos la DLL, por ejemplo FREEIMAGE.DLL y ejecutamos el comando IMPLIB:

C:\> IMPLIB FREEIMAGE.LIB FREEIMAGE.DLL

como resultado obtenemos una librería que la usaremos para compilar para cuando hagamos una llamada a una función que contenga la DLL, el compilador no falle por no encontrar la función.

Ahora bien, esta librería **NO ELIMINA EL USO DE LA DLL**. Es decir, necesitas la DLL por narices.

En el siguiente capitulo construiremos una clase a traves de lo que hemos explicado hasta ahora.

2.2.- FreeImage. Ejemplo de conexión con C

En este capitulo veremos un ejemplo practico de la potencia de lo que en el anterior capitulo leímos.

En este caso , vamos a hacer uso de las librerías Fivewin, las cual nos ahorran mucho trabajo ya realizado.

Empecemos pues.

¿ Que es FREEIMAGE ?
FreeImage es una DLL de gestión de formatos gráficos.
Tiene una licencia GPL, la cual nos permite usar dicha librería para fines comerciales si ese es vuestro deseo.
Una de las cosas 'malas' o 'regulares' que tiene Fivewin es la clase Image.
Como supongo que sabréis la clase TImage hace uso de las nViewLib, la cual no es muy estable que digamos. Pero para salir de paso esta bien.

La ventaja de FreeImage vs NViewLin, es que tienes disponible el código fuente de la FreeImage y al ser un proyecto Open Source y GPL, la continuidad es mucho más estable que cualquier otra cosa.

Pero bueno, a lo que íbamos. Vamos a construir un nuevo Control para Fivewin, que nos brinde más posibilidades que la Clase TImage.

¿ Y como empezamos ? Esa es la pregunta mágica.
Yo siempre digo que lo primero es empezar que después todo caerá por su propio peso.

Yo, supongo que más gente, me encontré en la necesidad de poder grabar JPEGS. (Nota: Esto no todavía no lo he conseguido ;( ).
Así que buscando , preguntado, en los CHATs, por todo INTERNET , estaba en busca de ALGO por me pudiera seguir, pues empezar a empollar los formatos gráficos no me pasaba ni por la cabeza, pues el tiempo requerido seria demasiado y no seria ni viable y no estoy seguro que lo consiguiera.

Lo cuestión es que di con esta librería y me gusto. GPL y Open Source que más se puede pedir ;)

Lo primero que haremos será plantearnos que es lo que esperamos que haga y como lo haremos.

Siguiendo el patrón de Fivewin, me dije :

Si Fivewin para ver Jpegs ha hecho una clase , TImage, que hace uso de una DLL externa, pues yo haré lo mismo con la FreeIMage.DLL.

Al principio, cuando empecé a experimentar con Harbour y C, use esta DLL para mis propios experimentos , sin clases ni POO, simplemente llamadas a funciones de la DLL.

Como vi que la conexión entre Harbour , C y la DLL era correcta , fue cuando aborde el tema de la clase TFreeImage.

Lo primero que hice fue 'heredar' de la clase BITMAP.
Alguien se preguntara porque. La respuesta es bien sencilla. Si a través de la herencia obtengo casi el 90% del trabajo, ¿ para que voy a escribirlo si ya lo tengo ?

Así la Clase TFreeImage hereda de la clase Bitmap y simplemente dispone de 5 métodos propios muy sencillos.

```
METHOD New( ) CONSTRUCTOR
METHOD Load( cBmpFile )        INLINE ::ReLoad( AllTrim( cBmpFile ) )
METHOD LoadBmp( cBmpFile ) INLINE ::ReLoad( AllTrim( cBmpFile ) )
METHOD LoadImage( )
METHOD SaveImage( cFormat )
METHOD ReLoad( cBmpFile )
```

El segundo paso que tenemos que hacer es modificar lo métodos de carga/save de la clase TFreeImage, donde es aquí donde haremos las llamadas al C.
Al disponer de la ayuda de la DLL, nuestra trabajo es bastante simple.
Simplemente debemos de pasar desde Harbour a C lo que nos interesa, procesarlo en Lenguaje C que hará las llamadas a la DLL y devolver unos valores para que la clase TFreeImage lo procese. Ya esta.

Miremos los métodos de la clase y veréis que la simplicidad roza el extremo de lo absurdo:

METHOD New(..parametros...) CLASS TFREEIMAGE

Super:New(..parametros...)

RETURN SELF

¿Ya esta ? SI YA ESTA!!! El constructor de la clase lo uso para llamar al constructor de la clase TBitmap.
Por ello os comentaba la importancia que es usar la herencia, pues ya tengo el trabajo hecho, ¿ para que voy a realizar un nuevo y casi exactamente igual al constructor de la clase TBitmap ?

Ahora pasemos a ver los métodos más interesante, el método **Load()**.

METHOD Load(cBmpFile) INLINE ::ReLoad(AllTrim(cBmpFile))

Ahora bien, hay que dejar clara una cosa.
¿ Porque existen los métodos **LoadBmp** , **ReLoad** si ya los tengo en la clase TBitmap ?
La razón es que en vez de pasarle dos parámetros , solamente le vamos a pasar uno, ya que nosotros no haremos uso de ficheros en recurso.
Si no , habría que pasarle dos parámetros y el primero como " ", para que funcionara la clase TFreeImage.
Así lo tenia al principio, pero no me gustaba eso de tener por el código parámetros que no voy hacer uso de ellos.
Pues rescribo el método que me interesa y asunto concluido. ;)
Otro motivo muy importante es que el método **Paint** de la Clase Tbitmap ,hace una llamada al método **Reload**, es por ello que debe de existir dicho método para que la clase Timage funcione como debería de funcionar.

Ahora veremos el método **LoadImage** para cargar una imagen, y nos quedaremos con esta línea :

::hBitmap = FreeLoad(AllTrim(cBmpFile) , ::hDC)

hBitmap contendrá el handle al fichero de imagen que nos devuelve **FreeLoad().**

Del resto prácticamente se encarga la clase TBitmap ;)

Mirando la función **FreeLoad**, veremos como lo anteriormente explicado no tiene ningún misterio, la conexión con C.
Imaginemos que le pasamos que la extensión es un fichero .bmp.
(Para abreviar el código)

```
*------------------------------------------------------------------------
#pragma BEGINDUMP

#include <windows.h>

#include "FreeImage.h"  --> Include de FreeImage
#include "hbapi.h"      --> Extend Harbour

HB_FUNC( FREELOAD )   // cFile, extension, hDc
{

 FIBITMAP *dib   ;
 HBITMAP  bitmap ;

 switch( FreeImage_GetFileType( hb_parc( 1 ), 16 ) ) {
       case FIF_BMP :
           dib = FreeImage_LoadBMP( hb_parc( 1 ), 0 );
           break;
```

hb_parc(1) es el parámetro pasado desde Harbour que contiene el nombre del fichero que vamos a abrir.

Si continuamos con el código , haremos uso de las funciones que nos provee la DLL FreeImage para hacer compatible el Dib con un bitmap.
Esta variable, bitmap, es el que le pasaremos a la Clase TFreeImage.
El **hb_parnl(2)** es el contexto de dispositivo en cual vamos a escribir

```
bitmap = CreateDIBitmap( ( HDC ) hb_parnl( 2 ),
                  FreeImage_GetInfoHeader(dib),
                  CBM_INIT,
                  FreeImage_GetBits(dib),
                  FreeImage_GetInfo(dib),
                  DIB_RGB_COLORS );
```

El Dib ya no nos hace falta , pues lo eliminamos.

```
if (dib != NULL) {
   FreeImage_Free( dib );
  }


hb_retnl( (LONG) bitmap );

#pragma ENDDUMP
```

Bueno, pues el resultado de todo ello , es que devolvemos el handle del bitmap listo para ser usado a través de **hb_retnl((LONG) bitmap)**

Como podéis ver, no es tan enrevesado como podíamos suponer al principio.

2.3.- Front-End. Compilando...

Introducción al Front-End para Harbour

Este proyecto es lanzado por la imperiosa necesidad de dotar a Harbour y en concreto a Fivewin de un sistema de compilación desde un gestor Front-End.

La idea se me ocurrió pensando en el M.A.M.E.
El M.A.M.E es un Multi Arcade Machine Emulator, para los que no tengáis idea.
Ahora imagina que Harbour al igual que M.A.M.E funciona sobre la línea de comandos del D.O.S.

Alguien, y digo alguien porque hay varios Front-End para el M.A.M.E, pensó que era un coñazo estar trabajando sobre la línea de comandos, e ideó un sistema para configurar un montón de aspectos del M.A.M.E y lanzarlo desde su propio FRONT-END.
La verdad es que gracias a los Front-End , el M.A.M.E tuvo, bueno, tiene un éxito impresionante.

(Nota: No se cual fue el primer Front-End para hacer cualquier cosa, simplemente pensé en el MAME pues es lo que busque en mis tiempos del D.O.S para no tener que lanzarlos desde la línea de comandos, y es ahí donde por primera vez oí hablar de un Front-End.)

Pues ya es hora de que nosotros tengamos un Front-End como Dios manda.
Esto solamente es el principio, ya que la versión lanzada es la beta 1.0 Release 0.1 y en este momento acabo de perfilar la versión estable 1.0 Release 1.3.

Puedes compilar para Fivewin , como para Harbour sin ningún tipo de problemas.

Nota: Para Harbour puro y duro , quitarle al Link Flags, **-aa**. Esta flag indica al compilador de Borland C que la aplicación será para modo consola.

Ahora seria interesante dotarle de más funcionalidad, pero eso ya depende de vuestras necesidades de trabajo.

Yo, me he limitado a ser lo más practico posible, intentando que este proyecto llegue a más gente posible.

En la medida en que pueda iré desarrollándolo , también se necesita que la gente se vaya implicando, reportando Bugs, mejoras, ideas o implementando directamente en código fuente.

De momento no se que alcance puede llegar a tener ni si la duración de este Front-End será larga.

Lo que si se seguro es que a mi, de momento, se me hace imperiosamente crucial para compilar varios prgs.

Eso depende de que desde el Proyecto Harbour realicen un IDE como dios manda, pero de momento , esto cumple de sobra la misión por el cual lo empecé a realizar:

- Compilar varios ficheros fuentes para obtener un ejecutable.

De momento solamente puedo mantener esta versión para Windows, ya que me gustaría poder implementarla bajo Harbour puro y duro, de esta manera, podríamos tener uno común para diferentes sistemas operativos, aunque sea en modo consola.

Por si alguien se anima, decirle que solamente se debería de tocar lo que corresponde a la clase TIni, ya que es una clase de Fivewin que se basa en llamadas al API de Windows para la SECTION,etc...
Bueno y currarse la parte visual bajo Harbour....eso te lo dejo como ejercicio... ;)

El Front-End es un gestor de compilaciones, donde podemos tener todos nuestros proyectos aglutinados en un simple listbox y con una pulsación de tecla podemos compilar en un instante un proyecto u otro, sin salir para nada desde el Front-End.

Una de las cosas que le he dotado a sido que sea multi-idioma. De esta manera, nadie se me quejará ;)

El Front-End disfruta de una licencia GPL, el cual llevo consigo de que dispones el código fuente completo, sin limitaciones de ningún tipo.

Escogí este tipo de licencia entre otras cosas para que podáis observar como esta montado internamente el Front-End.

2.3.1.- Opciones disponibles

El Front-End lo he dividido en 6 secciones :

Sección 1: Central

Esta parte es la encargada de escoger , crear, compilar un proceso que nosotros hayamos definido. Las opciones disponibles son:

Files List.\INI*.ini: Combobox en el cual tendremos cargado todos los ficheros .ini que encuentre dentro del directorio INI.

Description: Descripción del fichero que tenemos seleccionado en el Combobox.

Después disponemos de los siguientes botones:

NEW: Crea un fichero .INI para realizar una nueva compilación.
Atentos. Cada vez que creemos una nueva compilación, se cargara por Defecto los valores que por defecto hubiéramos grabado.

RELOAD: Recarga otra vez el fichero que tenemos en el Combobox.
Esto nos es útil si queremos cargar otra vez de nuevo el fichero de compilación , descartando los posibles cambios que hayamos efectuado.

SAVE: Guarda el contenido de TODO en el fichero INI.

Generate Make: Nos genera simplemente un fichero con extensión .MAK dentro del directorio INI.

COMPILE: Guarda en el fichero INI, genera .MAK y lanza la ejecución de la compilación a través de un fichero .bat, bMake.bat, donde también es creado en tiempo de ejecución.

About: Sobre el autor, Versión, Release y date de la Release.

Select Language: Selecciona el lenguaje del Front-End.
Por defecto siempre saldrá el primer lenguaje que hayas definido en la primera fila del fichero de configuración Language.ini. También se usará la próxima vez que entremos el ultimo lenguaje usado.

Edit Notepad: Edición del fichero de configuración del lenguaje. Desde aquí podemos cambiar las definiciones de los lenguajes, y poner etiquetas a los controles así como la ayuda de texto.
Si miráis dentro de language.ini podemos ver como esta estructurado para que puedas añadir más lenguajes o modificar un lenguaje a tu dialecto.
Eso pasa mucho entre los Hispano-Parlantes.!! ;)

Nota:
Un ejemplo de ello es que mientras en España 'cogemos' un autobús, los mexicanos, los 'agarran' (joder, que fuertes son XDDD)

Sección 2: Aplication

Esta sección esta dedicada a los aspectos de nuestra aplicación.

Aplication Name: Como se llamará el ejecutable. Atención, no pongas la extensión del fichero, en este caso , .EXE.

Path Prgs: Directorio donde se encuentran los .prg

Path Objs: Directorio donde colocara los ficheros objetos el compilador.
Si no estuviera creado, te preguntara si lo quieres crear, y si no lo creas te avisa que tendrás problemas a la hora de compilar.

Path Includes: Directorio donde están los ficheros de cabecera de la aplicación. Aquí si es posible dar diferentes rutas, muy útil para usar clases / librerías de Terceros. Por ejemplo: *.\MisIncludes;c:\tdbf\include*

Path of EXE: Directorio donde se creara el .EXE.

Por defecto tiene el valor de una macro,**$(APP_PRG_DIR)**, que indica que creara el .EXE donde estén ubicados los .prg.

Path of RC: Ruta donde esta el fichero de recursos si hacemos uso de el.

Use .RC File: Checkbox para informar al Front-End que haremos uso de un fichero de recursos. Dicho fichero debe *OBLIGATORIAMENTE* llamarse igual que ***Aplicattion Name.***

Generate file .MAP Si queremos generar además un fichero MAP.

Aplication Prgs List Lista de ficheros .prgs que forman nuestra aplicación ***OBLIGATORIAMENTE*** debes de declarar primero el fichero fuente que contiene la función *Main()* y esta debe de existir. De lo contrario, aunque el ejecutable se cree, la invocación del mismo no produce ningún efecto.

List My LIbs. Lista de tus propias librerías, que serán puestas las primeras.

El botón **Load All .PRG Files**, nos es útil para cargar todos los prgs ubicados en el directorio de Path Prgs. y las introduce automáticamente, ahorrándonos tiempo en hacerlo nosotros manualmente.

Options Make. Opciones de configuración del Makefile.
Aquí no se va ha explicar el uso de dicha utilidad, remitirse a la ayuda de dicha herramienta. Este dialogo nos permite pasarle parámetros adicionales a dicha utilidad de una manera visual.

El resultado de estas opciones las podéis ver en el bMake.Bat de compilación.

Force Compilation. Esta opción solamente funcionará si previamente hemos compilado alguna vez. Fuerza al compilador a compilar TODOS los ficheros nuevamente.

Silent Messages No informa de los mensajes de error.

Mode Debug Informa detalladamente de todo relacionado con la compilación.

View Date /Times Files Informa de la fecha / hora de los ficheros.

Show ALWAYS Make.Log Cuando compilamos se genera un fichero .log , en cual solamente veremos si se producen errores de compilación en algún modulo. Esto obliga a enseñarlo igualmente.

Ignores any rules Makefile por defecto usa este fichero. Ocasionalmente podemos decirle que no lo tenga en cuenta. BUILTINS.MAK.

Path Ruta donde se encuentra la utilidad MakeFile.

Sección 3: Harbour

Desde esta pestaña del folder podemos configurar los aspectos al compilador Harbour.

Name Nombre del compilador. Por defecto HARBOUR.EXE

Path Includes Ruta de los includes del compilador.

Path Bin Ruta donde se haya el compilador Harbour.

Path Libraries Ruta donde se haya las librerías de Harbour.

Load LIbs. For Carga las librerías que tenemos por defecto para harbour definidas en la sexta pestaña si hubiese alguna , si no, cargará las que hay por defecto definidas en código fuente.

FLAGS or Options to compilation Escribiremos los FLAGS del compilador directamente
Default

Configur Opción visual para escoger cada una de las opciones de Harbour. Esta al 90% terminada, y no esta del todo operativa.

Sección 4: Fivewin

Este es el más fácil. ;)

Path .CH Ruta de los includes de fivewin
Path .LIB Ruta de las librerías de Fivewin.
Defect Libraries Librerías de Fivewin por defecto.

Sección 5: Borland C+

Este apartado esta para definir rutas y flags de compilación relativos a Borland C+

Name Nombre del Compilador. Por defecto *BCC32.EXE*

Path Include Ruta de los includes.

Path Bin Ruta para el compilador , linkador y el compilador de recursos.

Path Lib Ruta de las librerías.

Name Link Nombre del linkador. Por defecto *iLink32.EXE*

Comp.Res. Nombre del Compilador de recursos. Por defecto *BRC32.EXE*

Flags of Compialator. Opciones del compilador.
-I$(HARBOUR_INCLUDE_DIR);$(BORLANDC_INCLUDE_DIR)
La definición de arriba por ejemplo, son constantes que puedes consultar dentro del .MAK generado.
En este caso estamos diciendo que los include (-I) están localizados en la ruta de harbour include(*HARBOUR_INCLUDE_DIR*) y los de BorlandC. En (*BORLAND_INCLUDE_DIR*), de esta manera para cualquier programa usaremos esto mismo, independientemente donde se encuentren los compiladores.

Flags of Linkator Opciones del linkador. Misma explicación que los flags del compilador.

Defect. Lib To Load. Carga librerías por defecto.
Atención. Internamente el Front-End pone un .obj necesario para la creación de Windows Ejecutables. Por lo tanto , aquí solamente se pueden poner ficheros de librerías, sin extensión, excluyendo los ficheros .Obj, puesto el Front-End automáticamente pone la extensión .Lib a la hora de generar el makefile.Mak

Sección 6: Valors Default

Que valores por defecto usaremos en nuestras compilaciones.

Aquí podemos poner que librerías usaremos por defecto.
Tenemos tres listas en la que podemos definir por separador, cual son las que usaremos por defecto, las de Harbour, Fivewin y las de Borland.

El botón de guarda valores por defecto, nos guardará las rutas , y los flags de compilación y linkador tanto para Harbour, Fivewin como para BorlandC.

Así , cada vez que creamos una nueva compilación, podemos venir aquí, presionar cargar valores por defecto, y todas las características que hubiésemos guardado anteriormente las tendremos disponibles para esta nueva compilación.

Las librerías hay que definirlas en cada apartado por separado.
Lo realice de esta manera para tener un control más preciso sobre lo que quiero por defecto.

2.3.3.- Notas Técnicas

¿ Que es lo que hace el Front-End ?

Decir que el Front-End es un esfuerzo personal para la creación de un fichero .MAK, basándose en un trabajo que publico Ignacio Ortiz en el Foro de Spanish.

Dicho fichero no es más que un texto plano que será usado por la utilidad MAKE de Borland C para la creación de nuestros ejecutables.
Lo que pasa es que hacerlo manualmente es un engorro.

Pues bien, el resultado final es que el Front-End te crea:

1.- Primero El .MAK dependiendo de tus opciones escogidas.
2.- Segundo El .Bat para lanzar el MAKE de Borland.

y yastà!!? Si , ya esta. Es absurdo hacer un programa para hacer esto, pero créeme si te digo que era más necesario que hacer una 'meada' en un atasco de trafico ;)

El Front-End esta compuesto por los siguientes ficheros fuentes (.\source):

- FRONTEND.PRG
Modulo Principal y Visual. Esto modulo se encarga de la puesta en escena ;)
Dialogos, Folders, Gets,etc...

- SUPPORT.PRG
Modulo de soporte de funciones para el Front-End.
Selección del Idioma y montaje de las etiquetas en el idioma escogido, selección del cursor para la aplicación, etc...

- TMAK.PRG
Clase TMak. Esta clase es la clase principal que usamos para todo el sistema de configuración, creación y ejecución de un fichero .MAK
A su vez esta nos creara los diferentes objetos de configuración para su manejo desde el FrontEnd.prg

- TAPPMAK.PRG
Esta clase es la encargada de controlar las opciones de la configuración de la aplicación , como el nombre, si haremos uso de un fichero .RC, nuestras propias librerías, etc...

- TBAT.PRG
Esta clase es la encargada de crear nuestro bMake.bat.
De esta forma, podemos seleccionar diferentes opciones desde el Front-End, para crear diferentes compilaciones.
Podemos tener de esta manera , un .BAT diferente por cada compilación si así lo deseamos.

- TBORLMAK.PRG
Esta clase es la encargada de las configuraciones de BORLAND C.

- TFIVEMAK.PRG
Esta clase es la encargada de las configuraciones de Fivewin.

- THRBMAK.PRG
Esta clase es la encargada de las configuraciones de Harbour.

- THRBOPT.PRG
Esta clase es la encargada de las configuraciones de las opciones del compilador de Harbour visualmente.

- TDEFAULT.PRG
Esta clase es la encargada para opciones por defecto de configuración para diferentes objetos.

- Fichero FRONT32.RC
Fichero de recurso.

Como podéis observar, casi todo son clases.
Lo diseñe de esta forma porque realmente es mucho más sencillo su mantenimiento.

El Front-End es una versión 'inacabada'.

¿ Que quiero decir por inacabada ? Quiero decir que todavía le falta mucha mas funcionalidad, como por ejemplo:

Selección de todos las opciones para el compilador de Harbour.
Esto no costaría demasiado pues simplemente deberíamos de tocar THrbOpt.prg.
De momento , como cuando seleccionamos las opciones, automáticamente las transforma en una línea de texto , nosotros podemos añadir las opciones que nos hagan falta en esa línea y ya esta.

Selección de las opciones del compilador de Borland C visualmente.
Igual que las opciones del compilador de Harbour, pero aquí se hace necesario la creación de una nueva clase.
Pero lo dicho anteriormente, de momento podemos escribir las opciones a mano. Por defecto están las que necesitamos y no deberías de tocar nada.

Lo mismo para el enlazador, iLink32.

Selección de las opciones de la utilidad MAKE.
Lo mismo que lo anterior de las opciones del compilador de Harbour.
Faltaría añadir el resto de las posibles opciones en TBAT.PRG.

Creación de librerías en vez de ejecutables.

Estas son algunas de las ideas y/o posibles mejoras que se deberían hacer en el Front-End.

¿ Cuando estará previsto que soporten estas mejoras ? Pues no lo se.
Sinceramente, libere también el Front-End como GPL para que la gente se involucrara en el desarrollo de nuevas funcionalidades.

Pero , aunque se que el Front-End goza de popularidad entre la comunidad de desarrolladores de Fivewin, no se han animado nadie a mejorarlo.

Pero también he de decir que gracias a la gente se han solucionado bastantes problemas y portado a diferentes idiomas. Gracias amigos.

¿ Quiere decir esto que el Front-End funciona a MEDIAS ?
No. El Front-End es 100% operativo.
Lo que pasa es que en vez de seleccionar opciones en modo visual, deberemos de recurrir a teclearlas nosotros mismos si fuese necesario.

Pero eso solamente lo tendremos que hacer para cosas muy puntuales, pues como esta ahora las opciones por defecto, muy pocos cambios deberá de realizar.
Si por ejemplo, necesitara declarar para el compilador de BC++ un DEFINE, simplemente en la línea de texto de las opciones de compilación añada -Dmi_define, y esa cadena será guardada la próxima vez que guarde la compilación o compile directamente.
Cuando entre otra vez al Front-End , no deberá especificarlo pues ya estará disponible para esa compilación.

Si lo que quiere es mantenerlo siempre, grábelo como valores por defecto y cada vez que cree una nueva compilación, dispondrá automáticamente de esos valores.

3.0-Programación Orientado al Objeto Como y Porque

Hace un tiempo dedique mi tiempo a hacer un manual de POO para Fivewin. Aunque parecido, el motor de objetos de Harbour es mucho más potente.

Hay varias cosas que han cambiado, para mejor, pero que nos pueden liar si venidos de programar con Objects de Fivetech.

Prácticamente la sintaxis para la creación de las clases es idéntica, pero hay matices que no podemos pasar por alto, y se requiere de otro manual completo.

Intentaré lo humanamente posible explicar este concepto que es la POO.
Empecemos pues que es esto de la programación orientado al objeto.

La POO es simplemente una forma de programar, donde las variables y las funciones van juntas de la mano y son indivisibles.

'Coñe, esto suena a chino' ;) Me explico.

Generalmente, casi toda la literatura que he leído sobre POO, empiezan a explicar lo que es la POO, con la típica clase TAutomovil, TPelota, TVaso,etc..

LO ODIO!!!! Es absurdo hacerle entender a un tío que lleva media vida programando con sus PRIVATES y PUBLIC , meter esos conceptos.
Si, suenan muy bonito, pero es tan abstracto.

Lo que yo os lo voy a explicar de una forma mucho más simple, comparando código Clipper / Harbour contra la POO para los viejos 'clipperos' y la demás clases para la nueva gente ;)

Imaginemos que tenemos una función que nos actualiza un numero cada vez que la llamamos y nos devuelve dicho resultado.

Un código fuente típico podría ser:

```
Static nNumeric := 0

Function Main()

  ? Numero()

Return NIL

Function Numero()
return ( ++nNumeric )
```

Ahora bien en POO, para similar dicho funcionamiento, podemos optar por varias vías, aquí cada cual es libre de la implementación que quiera hacer:

```
Function Main()

Local oNumero := TNumeric():New()

? oNumero:Numero()

Return NIL

CLASS TNumeric
    DATA nNumeric INIT 0
    METHOD New()    INLINE Self
    METHOD Numero() INLINE ++::nNumeric
ENDCLASS
```

Obviamente aquí tenemos que estamos escribiendo más de la cuenta.
Y así es!. Pero , como ya he mencionado anteriormente, unas de la potencia de la POO es la reusabilidad de código.

Ahora , imagina que yo te digo que quiero que la variable en el código Harbour/Clipper *nNumeric* , cuando sea llamada otra vez , no quiero coger el ultimo numero sumado, si no que quiero tener otro contador independiente.

Hay dos posibles soluciones:

1ºSolución.Creamos una nueva variable *nNumeric2,* y otra función Numero2()

```
Static nNumeric  := 0
Static nNumeric2 := 0

Function Main()

  ? Numero(), Numero2()

Return NIL

Function Numero()
return ( ++nNumeric )

Function Numero2()
return ( ++nNumeric2 )
```

2º Solución creamos una nueva variable y modificamos la función Numero()

```
Static nNumeric  := 0
Static nNumeric2 := 0

Function Main()

  ? Numero( nNumeric ), Numero( nNumeric2 )

Return NIL

Function Numero( nNum )
return ( ++nNum )
```

En los dos casos , esto es un engorro. Estamos necesitando crear variables y modificando la lógica de la función Numero.
Ahora bien , el **MISMO EJEMPLO** de la clase *TNumeric*, **NO TENEMOS QUE TOCAR NADA!.**

```
Function Main()

  Local oNumero  := TNumeric():New()
  Local oNumero2 := TNumeric():New()

  ? oNumero:Numero(), oNumero2:Numero()
```

```
Return Nil

CLASS TNumeric
    DATA nNumeric INIT 0
    METHOD New()    INLINE Self
    METHOD Numero() INLINE ++::nNumeric
ENDCLASS
```

VES !???! Sin tocar nada, obtengo el mismo resultado.
El mismo código que tenia ya me esta sirviendo para lo que intento realizar.

Hay a veces que la gente dice que no hace falta hacer una clase para realizar pequeñas cosas. Bien , yo estoy de acuerdo con ellos, en este caso en concreto es absurdo realizar una clase.
Pero si vemos la importancia en las pequeñas cosas del día a día, acabareis dándome la razón de que con la POO, el día de mañana cualquier modificación, será muchísimo más simple que retocando funciones.

Ahora , siguiendo con el mismo ejemplo de arriba, alguien puede pensar que lo interesante es que la creación del objeto *oNumero2* utilice el contador del objeto *oNumero.*

Nada más simple:

```
DATA nNumeric INIT 0
```

lo convertimos a :

```
CLASSDATA nNumeric INIT 0
```

Ya esta! Estamos diciendo que la variable de Instancia nNumeric será como una variable static para todas las instancias de ese objeto.

Como puedes ir observando , la POO te abre un mundo impresionante ante tus ojos. Mas tarde entraremos dentro de toda esta jerga / sintaxis para una mayor compresión, y si, aquí también veremos las típicas clases TVentana ;)

La Sintaxis para escribir clases en HARBOUR será la siguiente:

Atención, Harbour te permite usar diferentes estilos de declaración para crear una clase , métodos, etc...
Esto es debido a que la gente de Harbour se ha currado la parte del Preprocesado para los diferentes dialectos XBASE para la definición de las clases.
Actualmente soporta la sintaxis de :
Class(y), TopClass ,Visual Object y FiveWin-Objects

Ahora bien, tocaremos un poco de teoría de la Programación Orientada al Objeto. Podemos decir que un lenguaje es Orientado al objeto cuando cumple:

- **HERENCIA**
- **POLIMORFISMO**
- **ENCAPSULACION.**

Dentro de todo esto , todavía hay más diferencias entre lenguajes, pero eso a nosotros no nos debe de preocupar.

Clipper por si mismo no era POO , pues no permitía ni la creación de nuevas clases ni la herencia.

Visual Basic, hasta la versión 6, tampoco era POO, pues no permitía ni la herencia ni la delegación de mensajes. (Impresionante!)

Harbour dispone desde el comienzo POO. Pero veamos que quiere decir las siglas antes mencionadas.

- HERENCIA.
Unos de los pilares de la POO es la herencia. Sin ella, no tiene ni sentido el platearse tan siquiera que dicho lenguaje tiene o no POO.
Ahora bien, que podemos entender como HERENCIA.
La herencia la usaremos para HEREDAR TODAS LAS CARACTERISTICAS de una clase superior o superiores, es decir, que si tengo la clase base TVehiculo, y quiero crear una nueva clase TMoto, a través de la herencia, ya obtengo todas y cada una de las características de TVehiculo.
¿ Para que escribir otra vez toda unas definiciones para una clase , si puedo heredar automáticamente de otra ?

La manera de hacerlo en Harbour es:

```
CLASS TMoto FROM TVehiculo
```

Ahora imaginemos que tenemos la clase TVehiculo que tiene las siguientes caracteristicas:

```
CLASS TVehiculo
   DATA nColor
   DATA nRuedas
   DATA nPuertas

   METHOD New()
   METHOD Acelera()
   METHOD Frena()
ENDCLASS
```

Escribiendo simplemente esto :

```
CLASS TMoto FROM TVehiculo
   METHOD New() INLINE Super:New(),Self
ENDCLASS
```

Ya tenemos a nuestra disposición todas las variables de instancia y métodos de la clase TVehiculo como si fueran propios de la clase TMoto.
De lo contrario, deberíamos de escribir TODO el rato lo mismo.

Cuando empecemos a dominar esto de la POO, veremos que hacer uso de la herencia nos simplificará mucho el trabajo , en todos los aspectos.

Ahora bien, hay que tener muy claro lo que debemos realizar de antemano, ya que así no perderemos horas y horas. No es fácil al principio, pero realmente es impresionante la cantidad de horas que te llegara a quitar.

-POLIMORFISMO
Coñe que palabreja! ;)
Esto tan raro es muy simple de explicar.
Nos permite enviar el mismo mensaje para diferentes objetos, es decir, que si partimos de que tenemos dos objetos , por ejemplo, Pelota y Vaso.

Yo puedo "lanzar" una pelota o "lanzar" un vaso.
Independientemente del resultado, el mensaje "lanzar" es enviado al objeto:

*Pelota**:Lanzar()***
*Vaso**:Lanzar()***

Obviamente, el comportamiento será distinto, ya que deberemos de programar el método Lanzar() para ambas clases.
En este caso , la Pelota puede rebotar, y el vaso romperse.

-ENCAPSULACION
Simplemente significa que los Datos y sus métodos son indivisibles, todo el conjunto forma una unidad, es decir, que si tenemos la clase Coche, es absurdo pensar que los DATOS, variables de instancias, y los métodos, las acciones que realiza el objeto, pueden ser separadas, es decir , las propiedades del coche por un lado, y lo que este hace, como acelerar, frenar, etc..., por otro.

Todo , *DATOS* + *METODOS* = **OBJETO.** Uno e indivisible.

Como puedes observar esto realmente no es complicado.
El 'gran' problema que se encuentran la gran mayoría de gente que no usa POO, es que puede perfectamente vivir sin ella.
Todo a base de programación estructural, con funciones, etc..., es posible hacerlo.

Ahora bien, mantener la lógica de un programa en la cabeza con implementación de objetos es MUCHISIMO más fácil.

Lo único malo es la pereza del programador en aprender un nuevo estilo de programación, la POO, siendo esta mucho más acorde con los tiempos modernos, y más sobre sistemas operativos como Windows o Linux (GTK) donde la programación visual todavía es mucho más compleja.

De ahí la importancia de la POO. Imaginemos un simple botón en Fivewin :

(Para esta explicación es necesario tener el producto Fivewin para saber realmente de lo que estamos hablando.)

```
@1,1  BUTTON oBtn PROMPT "Accion" ACTION MsgInfo("A") OF oDlg
```

Esta sencillez SOLAMENTE se puede conseguir a través de la POO.
Las llamadas que deberían de realizarse con el estilo estructural, seria del tipo Lenguaje C, hay que escribir mucho para tan poco resultado ;(

Pero si analizamos detenidamente en profundidad lo que realiza este simple objeto, oBtn, nos daremos cuenta del gran beneficio que tiene la POO.

Si yo te dijera: Ey! Que ese botón es lo mismo que la pantalla o dialogo en el que esta contenido, quizás pensarías que se me ha ido la 'olla' ;)

Pero, analicemos por un momento de donde viene el objeto oBtn.
Si miramos el código fuente, veremos que :

CLASS TBUTTON FROM TCONTROL -> y esta a su vez ,
CLASS TCONTROL FROM TWINDOW.

Eh!!! Estamos viendo que la clase Button 'hereda' de TControl y que esta a su vez 'hereda' de TWindow.
Por lo tanto, podemos afirmar que la clase TButton es una 'ventana' un tanto especial. Después de todo , no estoy loco. ;)

Os sugiero que le echéis un vistazo a las clases, y es aquí donde realmente veréis, que la POO es imprescindible para los tiempos que corren y más si lo nuestro es realizar programas bajo entornos gráficos, tipo Windows.

El mejor consejo que te puedo dar es que practiques mucho sobre objetos reales, como pelota, vehículo, coche, fruta, etc..., donde la comprensión del mundo real con el mundo de la programación es mucho más sencilla.

Cuando domines esto, estarás en condiciones de realizar clases que no tengan nada que ver con la vida real, donde aquí si que entran los líos mentales, y atención, que pueden ser un verdadero dolor de cabeza. ;)

Pero eso simplemente al principio. Al final del camino descubrirás que:
YA NO PODRAS VIVIR SIN POO!!!! y te estarás preguntando :
- Dios mio!!! Como he podido programar sin la POO!.

En la medida de lo posible , yo usare la sintaxis de Fivewin-Objects.
Empecemos pues a pelearnos con las Clases.

3.1-Declaración de una nueva Clase
CLASS ... ENDCLASS

Para crear una nueva clase deberemos de empezar a usar las palabras mágicas **CLASS ... ENDCLASS**

La síntesis completa de CLASS es :

CLASS <ClassName> [**METACLASS** <metaClass>] ;
[**FROM, INHERIT** <SuperClass1>,<SuperClassN>] [**STATIC**]

CLASS <ClassName>
Nombre de la clase. Generalmente en POO , los nombres de las clases empiezan por T, TWindow, TImage, TVentana, etc..
No es obligatorio, pero queda más chulo ;)
Ejemplo : CLASS TVentana

METACLASS <MetaClass>
Opcional. Realmente no se como funciona esto.(Manu Please)

FROM,INHERIT <SuperClass1>,<SuperClassN>
Opcional. Indicamos de que clases heredaremos.
Harbour permite MultiHerencia, aunque para no marearte te aconsejo que no lo uses , y en su lugar uses una variable de instancias que contenga dicho objeto. Todo porque el lió mental puede provocarte un derrame en el coco ;)

STATIC
Indicamos que esta clase solamente será vista en el modulo actual.
Digamos que trabaja como una **Static Function** para que lo entiendas.
Por lo tanto, esta clase solamente existirá en el modulo que la hayas creado.

3.2-Declaración de variables de Instancias
DATA y CLASSDATA

La declaración de las variables de instancias las realizaremos dentro de las cláusulas CLASS...ENDCLASS.

La diferencia entre la declaración de una variable de instancia como DATA y una CLASSDATA radica que mientras la DATA es una variable de instancia al objeto, la CLASSDATA es una variable de instancia a la clase. Eh ? ;). Viendo un ejemplo lo entenderéis a la primera:

```
CLASS TVariable
    DATA      uData      INIT 0
    CLASSDATA uClassData INIT 0

    METHOD New() INLINE SELF
    METHOD Incrementa()
ENDCLASS

METHOD Incrementa() CLASS TVariable

   ::uData++
   ::uClassData++

RETURN Self
```

Esta clase la usaremos para incrementar unos valores en el método New.
Ahora , para hacer uso de la clase TVariable, la inicializamos :

```
Local oVariable      := TVariable():New():Incrementa()
Local oOtraVariable2 := TVariable():New():Incrementa()


? oVariable:uData      // Es igual a 1
? oVariable:uClassData  // Es igual a 1

? oOtraVariable:uData      // Es igual a 1
? oOtraVariable:uClassData  // Es igual a 2
```

Como es posible que la variable de instancia uClassData del objeto oOtraVariable sea igual a 2 ? No debería ser igual a 1 ? **NO.**

Hemos dicho que las variables de instancias CLASSDATA son compartidas por la clase , no por el objeto.
De esta manera, cualquier cambio que sufra la variable de instancia uClassData, de cualquiera de los dos objetos arriba indicados, automáticamente, el otro objeto también vera el cambio.
Si , manualmente nosotros decimos:

oVariable:uClassData := 5050.35

y a continuación, preguntamos sobre la CLASSDATA del otro objeto:

? oOtraVariable:uClassData

el resultado obtenido será: **5050.35.**

Espero que esto os haya quitado la confusión sobre la diferencia entre la declaración de una variable de instancia como DATA o como CLASSDATA ;)

Ahora bien, podemos indicar el comportamiento que queremos que tengan tanto las DATA o las CLASSDATA dentro del objeto o clase.
Para ello, en la definición de las variables de instancias , tenemos a nuestra disposición una serie de mecanismos donde indicaremos como se debe de comportar dichas variables de instancias cuando se actúan contra ellas, como la asignación, la lectura, etc...

La forma que tenemos de hacerlo es la siguiente:

CLASS
<...>

DATA <DataNames,...> [AS <type>]
[INIT <uValue>]
[EXPORTED, VISIBLE]
[PROTECTED]

[HIDDEN] [READONLY, RO]
[PERSISTENT, PROPERTY]

<...>
ENDCLASS

DATA <DataNames>
Nombre o Nombres de la variable de instancia.

AS <type>
Tipo de valor que puede contener la variable de instancia.
Puede ser: NUMERIC, CHARACTER, DATE, CODEBLOCK, LOGICAL, NIL
Podemos llamar a esto , prototipado de las variables de instancia.
Por defecto, es soportado todo los tipos si no hacemos uso de AS

INIT <uValue>
Valor inicial que contendrá la variable de instancia.

EXPORTED,VISIBLE
Valor por defecto. Indicamos que las variables de instancias serán visibles desde fuera del objeto.

PROTECTED
Estamos indicando que la variable que estamos definiendo estará protegida fuera de la clase ante cualquier cambio que desde fuera del objeto que se quiera hacer, como una asignación.

HIDDEN
La variable de instancia será declarado como si no existiera fuera del ámbito de acceso de la propia clase.
Incluyendo la herencia. Solamente será posible su modificación a través de la propia clase.

READONLY, RO
Esta variable será solamente para lectura, siendo su modificación imposible desde fuera de la clase.

PERSISTENT, PROPERTY
Persistencia de objetos.
Nos permite guardar y restaurar valores de la variable de instancia aquí declarada. Esto requiere una explicación aparte por la que tendremos que abordar la Clase HBPersistent.

La declaración de las variables de instancias CLASSDATA sigue el mismo patrón que las DATA.

Ahora veamos unos cuantos ejemplos y su comportamiento de las variables de instancias. Así dependiendo de lo que hagamos o hagan otros sobre nuestra clase en particular, evitaremos bastantes dolores de cabeza:

DATA cData AS CHARACTER
Variable de instancia que solamente aceptará valores de cadena.
Cualquier intento de asignarle un valor diferente a una cadena, petará el programa.

DATA uNumChar AS CHARACTER,NUMERIC
Usando esta forma podemos definir que la variable de instancia soportará los tipos indicados.

DATA uChar INIT "Hola Don Pepito"
Inicializamos directamente la data uChar con el valor "Hola Don Pepito"

DATA nNumeric INIT 1 READONLY.
Inicializa la variable de instancia a 1 y que solamente sea posible preguntar por su valor fuera de la clase, dando error si intentamos asignar algún valor a nNumeric.

DATA lHidden HIDDEN
Esta data solamente será visible dentro de la clase actual, siendo invisible también aunque usemos la herencia.

Podría estar poniendo más ejemplos, pero creo que con esto os iréis haciendo un idea del funcionamiento.

3.3.- Declaracion de los metodos : METHOD

Los metodos de un objeto son como las funciones, pero con muchas más posibilidades.
Podriamos decir que un metodo es la accion que ejecuta un objeto al envio de un mensaje concreto.

METHOD <NombreMétodo>([<params,...>]) [CONSTRUCTOR]
METHOD <NombreMÚtodo>([<params,...>]) INLINE <Code,...>
METHOD <NombreMÚtodo>([<params,...>]) BLOCK <CodeBlock>
METHOD <NombreMÚtodo>([<params,...>]) EXTERN <FuncName>([<args,...>])
METHOD <NombreMÚtodo>([<params,...>]) SETGET
METHOD <NombreMÚtodo>([<params,...>]) VIRTUAL
METHOD <NombreMÚtodo>([<param>]) OPERATOR <op>

METHOD <NombreMÚtodo>([<params,...>]) CLASS <ClassName>

3.4.- Persistencia de los objetos

Que es la persistencia de objetos ?
La persistencia de objetos es la posibilidad de que un propio objeto puede ser grabado a si mismo y ser recuperado más tarde.

Esto trae consigo una gran alivio, en parte, para los programadores. Cuantas veces hemos tenido la oportunidad de grabar unos datos, que posteriormente los tenemos que recuperar.
Todo ello basandose , particularmente, sobre ficheros .INI, o tipo campos MEMO.

Imaginemos una ventana. Tiene un titulo, un determinado color, etc... Si seguimos podemos pensar que seria interesante que el propio usuario decida ciertas caracteristicas y que estas las tenga disponibles

la proxima vez que entre a la misma ventana.

Como anteriormente dije, casi todo el mundo utiliza un fichero externo, tipo .INI o un campo MEMO para esta necesidad.

Pero ello implica tener que hacer un esfuerzo doble, ya que ahora, a parte de la aplicacion en si, tambien deberemos de programar otro apartado para soportar dicha caracteristica de poder grabar / Recuperar unos datos.

Tambien, cada cosa que queramos grabar, deberemos de reprogramar o volver a programar otras funciones y/o clases para que soporten , por ejemplo, otros casos como la ventana que estamos describiendo.

Ahora bien, a traves de la persistencia de objeto, ya no tendremos que volver a preocuparnos.

Y como puedo hacerlo funcionar ? Muy simple.

```
CLASS Tu_Clase FROM HBPersistent
```

o mejor aun :

```
CLASS Tu_Clase FROM Tu_Herencia, HBPersistent
```

Con esto ya tenemos la mitad del camino recorrido. Acojonante, no ? ;)

Ahora como deberemos de indicar a la clase que es lo que tiene que grabar ?
Si leeistes el capitulo sobre las variables de instancias, recordad que teniamos una propiedad llamada PROPERTY o PERSISTENT.
Pues bien usando dicha propiedad, indicaremos que ese valor de esa variable de instancia , será guardada o cargada.

Imaginemos que tenemos una clase TCaja.

```
CLASS TCaja FROM HBPersistent
```

```
    DATA nTop    INIT 1 PERSISTENT
    DATA nLeft   INIT 1 PERSISTENT
    DATA nBottom INIT 5 PERSISTENT
    DATA nRight  INIT 5 PERSISTENT
    DATA lCorrec INIT .T.          * Esta no sera guardada.

    METHOD New() INLINE Self

ENDCLASS
```

Ahora , cuando tratemos dicho objeto, veremos como hacer funcionar la persistencia de objetos.

```
Local oPersis := TCaja():New()

* Guardar objeto
oPersis:SaveToFile("objeto.pst")

? oPersis:nTop := 12
? oPersis:lCorrec := .F.    // No sera guardado.

* Ahora cargaremos de nuevo las valores anteriormente guardados.
oPersis:LoadFromFile( "objeto.pst")

? oPersis:nTop      // Aqui el valor será = 1
? oPersis:lCorrec   // Aqui el valor sigue siendo .F.
```

Como podeis comprobar , esto nos es de gran utilidad a la hora de guardar ciertos datos.

Eh, y los Codeblocks se guardan ? Ah, amigo mio, los codeblocks no son posibles de guardar. Recuerda que un codeblock no es más que un puntero a una direccion de memoria en particular y no tiene que apuntar siempre a la misma direccion de memoria.

Pero el resto de tipos de datos, pueder ser guardados, incluidos objetos. Por ejemplo, un objeto que contiene a su vez otro objeto y los dos derivan de la clase HBPersistent.

Para saber más mirar el ejemplo de Harbour, testpers.prg y

el codigo fuente de la clase HPersistent.prg

4.5.- MCI. El API Multimedia

MCI. ¿ Que es ?

El significado, Interfaz de Control de Medios, aunque ambiguo, esconde una potencia inusual para el programador.

El MCI realmente es como un pequeño lenguaje orientado a la gestión de dispositivos, que pueden ser de videos, audio, scanner, etc...

La manera de funcionar es similar a todo como funciona los dispositivos en Windows. Nosotros enviamos un mensaje a través del MCI para , por ejemplo, hacer sonar un fichero de sonido en forma de ondas, en concreto, *.WAV.

El MCI interpreta la orden, el cual se lo pasa a los controlares de bajo nivel, y estos serán los que harán funcionar el dispositivo físico.

Algo similar como funciona la impresión en Windows. Esto trae unas connotaciones impresionantes:

Simplemente haciendo OPEN CDAUDIO , PLAY CDAUDIO, estamos reproduciendo el CD!!.

Como puedes observar, el trabajo para el programador que le quita el MCI, es sencillamente una cantidad de código impresionante.

Ahora bien, como es de suponer, el controlador MCI debe de estar preparado para realizar las operaciones que les enviemos, y como puede ser, el controlador quizás no pueda realizar la operación encomendada.

Hay muchas cosas que no será posible hacer con el MCI, como controlar un teclado MIDI conectado al ordenador, etc... Aquí no nos quedará más remedio que pelearnos a bajo nivel con el controlador correspondiente.

Pero esto, será para cosas muy puntuales, para un control mas generalizado, el MCI nos ira de perlas.

4.5.1-Controladores por Defecto

Para poder controlar cualquier controlador, es necesario que dicho controlador nos de soporte para el MCI. De no ser así, no podremos utilizarlo.

Windows por defecto nos provee de unos cuantos controladores, como pueden ser para controlar WAV, MIDI y AVI.
Realmente esto es muy poco cosa con la cantidad de nuevos y cada vez mejores formatos tanto de video como de sonido, los ya famosos MPEG y MP3 y el ultimo en estar de moda, el Divx;), o Mpeg4.

Desde aquí descubriremos como podremos desde Fivewin, a través del MCI, poder controlar dichos formatos así como muchísimos más.

Cuantos mas drivers tengas instalado , más dispositivos podrás controlar a partir del MCI.
Por ejemplo, para poder controlar ficheros AVI(Avi For Windows), debes de tener instalado su correspondiente controlador.
Por defecto, a ser un standard de Microsoft(por cierto muy pésimo),ya lo tienes incluido. (Windows 3.1 hay que instalarlo)
Dentro de SYSTEM.INI tenemos,(todo esto es relativo ;)) :

```
[mci]
WaveAudio=speaker.drv
Sequencer=mciseq.drv
CDAudio=mcicda.drv
avivideo=mciavi.drv

[drivers]
timer=timer.drv
midimapper=midimap.drv
Wave=speaker.drv
VIDC.MSVC=msvidc.drv
```

Podemos ver los drivers que tenemos instalados.
Ahora bien, ¿ Como podemos hacerlos servir ?
En el siguiente capitulo veremos como he modificado las clases TMCI y TVIDEO para que podamos disfrutar de nuestros MPEGS preferidos ;)

4.5.2-Nueva Clase MCI

Como anteriormente habíamos hablado , el MCI es como un pequeño lenguaje interpretado, por lo tanto, debemos conocer los comandos que deseamos enviar.

Hay unos cuantos comandos, que a su vez , dicho comando tiene diferentes cláusulas dependiendo de un controlador especifico.

Bien, en esta nueva clase MCI que he realizado no están todos, pero si algunos útiles para realizar , como veréis más adelante cosas muy útiles.

Como bien sabéis , la clase MCI que trae por defecto Fivewin tiene un inconveniente **MUY GRANDE**.

Intentar poder dos videos a la vez en sendas ventanas MDI, os daréis cuenta enseguida del inconveniente : **Es imposible!!.**

Bueno , pues yo he podido solucionarlo en la nueva clase, creando por cada llamada a la clase un alias independientemente para cada uno.

Alguien se preguntara si no hubiese sido mejor usar el envio de mensajes al comando. He de decir que si. El problema radica en que dicha clase puede ser compartida tanto para Fivewin de 16 como de 32 bits.

Si me hubiese metido directamente en Fivewin For Harbour, hubiese tenido que terminar metiéndome con el lenguaje C, y entonces para los usuarios de Fivewin 16bits no podrían disfrutar de dicha clase.

Vamos a explicar algunos métodos interesantes de dicha clase.

DATAs

cStatus	Informa sobre el estado del dispositivo.
cAlias	Usado para enviar mensajes a través del método SendStr()
nId	Usado para enviar comandos MCISendCommand().
lDebug	Muestra cualquier error producido por cualquier comando si tiene un valor a .T.
cErrorMsg	Contiene mensaje del ultimo error producido en forma de texto literal
lAudio	Informa si queremos sonido o no.
nSpeed	Velocidad de reproducción. Un valor de 1000 es por defecto.
nPos	Posición actual del medio
lFullScreen	Si queremos a pantalla comleta o no

METHODs

Pause()	Hacer Pausa
Resume()	Continuar
Play()	Si ha sido pausado, continuar
Repeat()	Loop continuo de una avi
Status()	Pregunta por el estado del dispositivo
End()	Cierra dispositivo
New()	Nuevo paso de parámetros
Audio()	Controlando el Audio del video
Speed()	Velocidad de ejecución un valor de 1000 es por defecto
Length()	Devuelve longitud
Position()	Devuelve posición actual de reproducción del MEDIO
Seek()	Busca el frame y te posicionas
Config()	Ventana de configuración
Error()	Devuelve el error tipo cadena
DebugMode()	Modo interactivo de depuración.
FullScreen()	Valor .T. vemos el video a pantalla completa ;)
Loop(),Repeat	Repite una y otra vez lo mismo ;)
DeviceType()	Tipo de dispositivo abierto.

Después tenemos varios más usados para preguntar sobre las capacidades del dispositivo:

CanEject()	Si puede ser expulsado
CanPlay()	Si puede ejecutarse
CanRecord()	Si el controlador soporta grabación
CanReverse()	Si puede ir al revés
CanSave()	Si puedo grabar los datos
CanStrech()	Si puede encogerlo / engrandarlo en un rectángulo
CanCompound()	Si soporta nombre de fichero
CanHasVideo()	Si soporta video
CanHasAudio()	Si soporta audio
CanUsePalet()	Si usa paleta.
CanFastPlay()	Si soporta alta velocidad.
CanNormPlay()	Si soporta normal velocidad.
CanSlowPlay()	Si soporta baja velocidad.
CanWindows()	El numero de ventanas que soportan simultáneamente el controlador.

Hay un montón más, que esta fuera del alcance de este libro, pues necesitaría otro libro igual de grande para poder explicar uno por uno cada comando con sus respectivas opciones.

4.5.2.1-Que hay de nuevo, viejo

Todo esto que he modificado en la clase TMCI es compartido por cualquier controlador.

Hay que darse cuenta que dependiendo del tipo de controlador podemos lanzar algunos comandos con sus respectivas opciones.
Por ejemplo, el controlador AVIVIDEO no esta preparado para grabar. Eso no significa que no se pueda guardar secuencias AVI, si no que dicho controlador no esta preparado para ello.
Deberemos de proporcionar un controlador que si lo haga, ya sea de un fabricante independiente de Microsoft o de la propia microsoft.

Hay dos maneras de manejar el MCI:

1 .-Modo Mensaje. A través **MCISendCommand**, como por ejemplo:
MCISendCommand(::nId , MCI_PAUSE , MCI_NOTIFY)

2.-**Modo Cadena.** A través de
MCISendStr(Comando, Buffer, Handle_Ventana), como por ejemplo:
MCISendStr("PLAY ", @cBuffer, ::oWnd:hWnd)

Cada uno tiene sus ventajas y sus inconvenientes. A través del modo cadena, obtenemos una maneja sencilla de enviar comandos de forma literal al dispositivo.

El inconveniente es que dicho comando debe de ser traducido al modo Mensaje, por lo tanto, es más lento de ejecución.

En la nueva clase MCI hago uso de los dos métodos, el cual según me convenga, uso uno u otro.

Si hubiese hecho dicha clase para Fivewin For Harbour, me hubiese decantado por el uso del MCISendCommand() escrito en C, pero no quería defraudar a mis amigos de 16 bits, y por ello he tenido que montarlo a través del modo comando, evitando al lenguaje C.

Lo que he implementado, es un nuevo método en la clase TMCI , SendStr().

Dicho método nos permite más flexibilidad que el que trae por defecto en Fivewin.

METHOD SendStr(cMciStr, cFlag , cMethod) CLASS TMCI

cMciStr
Contiene el comando que queremos ejecutar como por ejemplo:
"*PLAY FULLSCREEN*"

cFlag
Puede contener **" ", Wait, Test o Notify**. Por defecto es " "

cMethod
Contiene desde que método de la clase es enviado.
Esto lo usa para saber de donde a venido la cadena a ejecutarse para la depuración ya sea de la propia clase o del dispositivo que estamos controlando. De esta manera, podemos observar , por ejemplo, el método REPEAT,
::SendStr("PLAY "+ ::cAlias + " REPEAT",,"REPEAT"), si hubiese algún problema , se que ha sido desde aqui , desde le method repeat(), donde me viene cuando ejecuto **SendStr()**.

Pero, ¿ que pasa si hago uso del modo mensaje ?. Ningún problema.
Si miramos por ejemplo el método Pause(), veréis como lo monto:

```
METHOD Pause()   INLINE
            ::Error( MCISendCommand( ::nId , MCI_PAUSE  ),,"PAUSE" )
```

Por lo tanto , al método Error, le paso que método,***"PAUSE",*** es de donde viene. Al fin y al cabo, el método SendStr() termina llamando también al método Error().

Lo diseñe de esta manera para poder ir realizando llamadas en tiempo de ejecución a través del debugger que incorpore para ir depurando la clase.

De esta manera tan sencilla me ahorre un montón de horas de compilación y es la base del debugger incorporado a la clase ;)
Quizás una próxima revisión de la clase, la implemente en C para ganar en velocidad de ejecución.

Pero quizás ;), solamente o ya tienes un ejercicio por hacer ;)

4.5.2.2-Depuración en tiempo real

Siguiendo con el capitulo anterior, donde explicamos las bases por la cual trabaja el depurador incorporado en la clase TMCI , seguimos profundizando en el tema.

Una de las cosas interesantes que tiene dicha clase, es que podemos operar a través de comandos MCI directamente, tecleando para ello dicho comando en un dialogo.

Para activar dicha característica simplemente debemos de llamar a un método llamado **DebugMode().**

Si ejecutáis el Test del MCI que tenéis disponible, y miráis la opción *FullScreen,* podéis presionar el botón de ***Debug*** para ver el funcionamiento.

¿Y como funciona ? Es muy simple:

COMANDO A ENVIAR + "," + OPCIONES DEL COMANDO.

Imaginar que queremos hacer que la ventana de visualización se sea fullscreen, pues dentro del modo de depuración enviaremos el comando que hace dicho efecto:

PLAY,FULLSCREEN

Automáticamente, nuestro ventana de visualización se verá a ventana completa.
De igual manera, el resultado de la operación o el código de error nos lo enseñará en el dialogo en un campo MEMO.

Una de las cosas que tenéis que tener en cuenta es que para hacer uso de este Debugger es que tenéis que abrir primeramente el dispositivo, por ejemplo, el fichero que queréis controlar.

Si no , no podéis depurar nada, pues no existe todavía.

4.5.5.-Nueva Clase TVideo

La nueva clase TVideo que he realizado , soporta Multi-Formatos.

Basándome en el driver que trae el Media Player, **MpegVideo**, podemos acceder a través de él, a cualquier formato que te soporte el Media Player.

Y esto tiene **connotaciones impresionantes.** También soporta los **CODECS** que tengas instalado.

Esto quiere decir, como las pruebas que yo he realizado, que si instalas los CODECS para ver Divx;), osease MPEG4, automáticamente y **TRANSPARENTEMENTE** tu clase TVideo accede a ese formato, sin que tu tan si quieras compiles de nuevo tu programa, pues como es el controlador el que tiene realmente el control, a ti como programador, ni te va ni te viene ;)

La verdad es que la clase en si a sufrido bastantes modificaciones respecto a la que trae Fivewin para hacer muchas mas cosas.

Veréis que como el método **New()** nos pone en marcha el dispositivo a traves de la clase TMCI, simplemente el method play , por ejemplo, poniendo **INLINE ::oMci:Play()** no debéis de escribir nada nuevo.

He implementado un par de cosas en el preprocesado :

NOBORDER Sin bordes alrededor de la ventana.
RESIZE Podemos redimensionar la ventana si queremos.
ADJUST Ajusta el soporte a la ventana contenedora.

A continuación veremos que es lo que hace la magia del soporte de los formatos del Media Player.

4.5.5.1-Soporte Multi-Formatos

Para acceder a esta cualidad de las clases , es necesario saber que deberemos tener instalado el controlador MpegVideo, en cual, según pude deducir se basa el Media Player de Microsoft. Por lo tanto debéis de instalaros una versión de dicho software.

Podéis comprobar si tenéis dicho controlador echando una mirada al System.ini tal y como visteis anteriormente con los otros controladores.

Entonces solamente nos queda una cosa por hacer dentro de la clase Tvideo y es inicializar el objeto oMci de la siguiente forma:

```
::oMci      = TMci():New( "MPEGVideo!", cFileName, Self )
```

Nunca una línea escondía **TANTA POTENCIA!!!** ;)

Si , señores, aquí podemos observar como simplemente cambiando el controlador **AviVideo** que trae por defecto Fivewin, por **MpegVideo**, accedemos a un nuevo mundo de formatos que soporta el Media Player.

Desde Divx;), Mpeg2, Mov, etc... hasta los MP3, Waw, MIDI, etc...

Ahora bien, para que tu clase soporte dichos formatos , tu Media Player debe de soportarlos primeramente.
Según las pruebas que he realizado, cojamos mi mismo ejemplo.
Estaba yo intentado ver las películas en formato Divx;) y necesitaba primeramente un reproductor.

El Media Player NO SOPORTA nativamente dicho formato, y aun bajando los CODECS el solito , nada de nada.

Lo primero que hice fue ir a Internet, busco el Playa, un reproductor Freeware para ver Divx;), el cual provee un CODEC para que el Media Player pueda hacer uso de él.

Abro el Media Player , y pude ver la película ;).

Bien, pues como el Media Player no me gustaba, cojo SIN TOCAR NI UNA LÍNEA DE CODIGO de las clases TMCI y Tvideo, lanzo el Test de rigor y VOILA!!! La película la estaba viendo en MI PROPIO REPRODUCTOR.

OOOOh Mi gozo en un pozo ;)

¿ Ya esta ?

SI. Ya esta. Como me dijo mi buen amigo Rene Flores, cuando se lo enseñé, ***IM---PRESIONANTE!!!!! ;)***

Como podéis observar, el paso del controlador , MpegVideo, acaba terminando en **!. Esto es muy importante.**
Debe de terminar de esta manera , pues esta dentro del standard del MCI, que nos permite abrir un fichero directamente con un controlador determinado.

Lo nuevo de este sistema , es que si por ejemplo, tenemos un nuevo controlador, por ejemplo, para manejar un nuevo dispositivo de captura , que se llamase, pongamos por ejemplo, ControlCap, cambiando solamente dicha línea por esta :

```
::oMci      = TMci():New( "ControlCap!", cFileName, Self )
```

Y guardamos dicha clase Tvideo como Tcapture, YA NO DEBERIAS DE TOCAR NADA MÁS, pues el MCI ya lo podrás controlador como si estuvieses controlando un video.

Recordad que dependiendo de tu controlador , podrás realizar alguna cosas y otras no.
Simplemente deberías de crearte el fichero preprocesado y yo lo tendrías listo.
¿ Impresionante , verdad ? ;)

4.6. Llamadas al API de Windows.

¿ Cuantas veces hemos leído que podemos hacer llamadas al API de Windows en otros lenguajes ?

Miles de ejemplos en C o VB, tienen llamadas al API de Windows, incluso internamente Fivewin se vale de dichas llamadas.

Pero , ¿ *como podemos implementar dichas llamadas nosotros mismos desde Fivewin* ?

En Fivewin, hay llamadas que son muy fáciles de implementar, más incluso que en VB.

El milagro que realiza dicha proeza es el comando **DLL** o **DLL32**.
La diferencia entre ellas es que uno es usado para DLLs de 16bits y el otro para el de 32 bits.

Usando el comando DLL / DLL32.

Una de las cosas más potentes que tenemos a nuestra disposición es la posibilidad de hacer uso de llamadas a las DLLs.

Y que es una DLL ? Para entenderlo , diremos que una DLL es como cuando compilas obtienes un fichero OBJ, el cual si le pasamos TLIB.EXE obtendremos una librería.

¿ Correcto ? Bien, pero esa lib que obtienes tiene la limitación que debe ser usada por el lenguaje por el cual fue creado, en un principio.

Ahora imagina que no obtienes el LIB, si no que obtienes el fichero DLL, el cual lo podemos usar independientemente del lenguaje a usar y en tiempo de ejecución.

Esto, tienes una connotaciones tremendas.

Eso significa que nosotros , vil programador de Harbour / Fivewin, podemos acceder a la potencia y al uso de las DLL sin renunciar a nuestra sintaxis del lenguaje.

Por ejemplo, partiendo de una llamada al API de Windows, como puede ser **AnimateWindow**.

He escogido esta función por la sencilla razón que no la tenemos disponible en Fivewin de forma nativa, de esta manera nos aseguraremos completamente de lo que hacemos funciona.

Lo primero de todo es hacernos con la documentación.
Aquí, si no tenemos la documentación no hacemos nada, y cual es mejor medio para hacernos con ella ? Pregunta estúpida, INTERNET.
O también es de muy listo hacerse con un lenguaje tipo C , VB o Delphi, los cuales tienes miles de ejemplos y libros escritos.
De lo mejorcito es buscar sitios referentes a Visual Basic, por la sencilla razón que este lenguaje se nos adaptará mejor si tenemos los ejemplos.
(Bueno, es que me es mas fácil leer código en Basic que en Pascal, quizas porque yo soy de la generación del Gw-Basic ;)).

Bueno, al fin de cuentas lo único que nos interesa es el prototipado de la función **AnimateWindow.**

Y es la siguiente escrita en Visual Basic :

```
Declare Function AnimateWindow Lib "user32" (ByVal hwnd As Long,;
        ByVal dwTime As Long, ByVal dwFlags As Long) As Boolean
```

Ahora nosotros vamos a traducirla para Fivewin, siguiendo paso a paso los diferentes aspectos que forman parte de la declaración del uso del comando DLL o DLL32:

Declare Function	**DLL32 (DLL a secas para 16 bits)**

AnimateWindow	**AnimateWin**(para su uso también en fivewin 16 bts) El nombre que usaremos para nuestra función. Si estáis usando Fivewin 16 bits, recordad que la longitud de una función no puede ser superior a diez, Y ahora veremos como solucionarlo.
ByVal hwnd As Long ByVal dwTime As Long ByVal dwFlags As Long	**hWnd AS LONG** **dwTime AS LONG** **dwFlags AS LONG**
As Boolean	**AS BOOL**
Lib "user32"	**LIB "USER32.DLL"** Hay que definir la extensión porque también podemos hacer llamadas a un EXE que contiene funciones.
PASCAL FROM "AnimateWindow"	El nombre de la función realmente, contenida en el API. (Usar tipo PASCAL. Generalmente muchas funciones del API están constituidas de tipo PASCAL. Si creéis que todo va bien , las definiciones , tipos etc,,. y sigue sin funcionar, probar a quitar la cláusula PASCAL. Quizás sea por esto, aunque nunca se sabe ;))

Como podéis observar, esto no tiene ninguna complicación.

La llamada nos quedaría de esta manera:

DLL32 Function
AnimateWin(hWnd **AS LONG** ,dwTime **AS LONG**, dwFlags **AS LONG**) ;
AS BOOL PASCAL FROM "AnimateWindow" LIB "USER32.DLL"

Nuestra llamada dentro de nuestro .prg no puede será más simple :

AnimateWin(**oWnd:hWnd, 320, nOr(16,131072)**)

El 'gran inconveniente' que nos encontraremos será aceptar en la conversión de tipos, ya que Fivewin y Harbour no tienen todos los tipos que tiene el C.
Podemos decir que cuando nos requieren en una llamada a un API un tipo HDC , para 32bits, nosotros deberemos de tratarlo como un **LONG**.

Esto quizás sea un poco de entenderlo , pero es que el lenguaje C es un lenguaje fuertemente tipificado y cada cosa debe de tener un tipo determinado cosa que a nosotros , ni fu ni fa ;)

Pero es bueno que experimentéis por vosotros mismos las posibilidades que nos ofrece el comando DLL32.(Para dll de 16 bits, usar DLL).

Los diferentes tipos que podemos hacer uso son :

(Cabecera DLL.CH)

```
#define VOID    0
#define BYTE    1
#define CHAR    2
#define WORD    3
#define _INT    4
#define BOOL    5
#define HDC     6
#define LONG    7
#define STRING  8
#define LPSTR   9
#define PTR     10
#define _DOUBLE 11
#define DWORD   12
```

Estos son lo únicos tipos de que disponemos.
A partir de aquí se nos abre un mundo de inmensas posibilidades, pero..., siempre hay un pero, cuando es una llamada a una estructura como lo hacemos?

Bueno, si ejecutáis FwApi.exe tenéis disponible un montón de posibilidades que podéis realizar a través del Api de Windows, donde podéis contemplar diferentes maneras de hacer llamadas al API de Windows.
Siempre hay un pero, pero eso lo veremos en el próximo capitulo:

- **Manejando estructuras tipo C. Usando la clase TSTRUCT**

4.6.1. Manejando estructuras tipo C.
Usando la clase TSTRUCT

Cuando por necesidades de una llamada a un API necesitemos hacer uso de una estructura a un puntero, usaremos la clase TStruct.

El funcionamiento de dicha clase es muy cómodo.

Imaginemos la siguiente llamada al API de Windows, **DrawFrameControl.**
Dicha llamada al API nos servirá para pintar los gráficos de los botones de las ventanas  .
Esto son unos bitmaps que pone a disposición Windows, evitando que nosotros tengamos que simularlos gastando más recursos nuestra aplicación para ello. Hay diferentes combinaciones , como hacer que se muestren pulsados, etc... Ver FwApi.Exe para ver dichos ejemplos.

Según el SDK de Microsoft , la definición es la siguiente (en VB):

```
Declare Function DrawFrameControl Lib "user32" Alias
      "DrawFrameControl" (ByVal hDC As Long, lpRect As RECT,
       ByVal un1 As Long, ByVal un2 As Long) As Long
```

Donde el único inconveniente para poder usar dicha funciona es 'algo' que nos puede sonar a chino: **lpRect AS RECT**

lpRect , Large Point Rect, Puntero Largo a una *estructura RECT*.
Seguimos mirando y vemos como esta formada la estructura (en VB) :

```
Private Type RECT
      Left     As Long
      Top      As Long
      Right    As Long
      Bottom As Long
End Type
```

En Visual Basic, las definiciones de las estructuras son así, donde también en podemos verlo como typedef struc { ... } RECT .
Bien , pues para nosotros nada más fácil:

```
STRUCT sRECT
  MEMBER Left    AS LONG
  MEMBER Top     AS LONG
  MEMBER Right   AS LONG
  MEMBER Bottom  AS LONG
ENDSTRUCT
```

Pero esto no es suficiente.¿Ostras,y esto como lo podemos hacerlo funcionar ?

Bien, después de estar mirando y mirando, y no encontrar nada, como casi siempre en este mundo que es Fivewin, y después de tropezar 1000 miles con la misma piedra, llegamos a la siguiente conclusión:

REGLAS DE ORO. Atentos que esto ES MUY IMPORTANTE
1.- Declaración de la variable sRECT MAS otra que la usaremos de 'puente' por ejemplo, Local uBuffer
2.- Definimos la estructura.
3.- Atentos! uBuffer:= sRECT:cBuffer, la que contiene el Buffer de la estructura se lo asignamos a la variable puente, uBuffer.
4.- Hay que pasar la variable 'puente' uBuffer por REFERENCIA!
5.- Cada vez que modifiquéis algún miembro de la estructura, acordaros que ANTES de llamar a la función, necesitáis hacer el paso 3, para refrescar la variable que hacemos servir de puente, si no, no vera los cambios.
6.- Deberías de declarar en el comando DLL32 el tipo como AS LPSTR.

¿ Pero realmente que es lo que hacemos ?

La respuesta es algo enrevesada pero tiene su lógica.

La razón de ello es que nosotros no podemos enviar directamente el miembro interno *cBuffer*, pues es imposible, pues el manejo de los arrays de Harbour y C no tienen nada que ver.
A la función **DrawFrameControl** le es imposible hallar de esa forma el puntero de la estructura.

Ahora bien, como decimos aquí, echa la ley hecha la trampa:

uBuffer := sRECT:cBuffer

@uBuffer, aquí **SI** que le estamos diciendo donde esta, pues le estamos pasando realmente la dirección de memoria donde esta el buffer de la estructura.

Entonces nuestra llamada al API de Windows quedaría de esta manera :

```
DLL32 FUNCTION
 DrawFraCtl(hDC AS LONG , lpRECT AS LPSTR, uType AS
LONG,uFlags AS LONG );
 AS LONG FROM "DrawFrameControl" PASCAL LIB "USER32.DLL"
```

Si miramos el ejemplo FwApi, tendréis disponibles varias llamadas al API de Windows, donde encontrareis mas ejemplos.

Pero, seguimos con los peros, hay a veces que es común indicarle a un miembro de la estructura su longitud. Como lo podemos solucionar ? Bien sencillo. Imaginemos que tenemos la siguiente estructura :

```
STRUCT sInit
  MEMBER dwSize  AS LONG
  MEMBER dwICC   AS LONG
ENDSTRUCT
```

Donde dwSize nos dice el SDK que debe ser la longitud de la estructura.

La manera de indicárselo es haciendo uso del **método SizeOf()**.
El método **SizeOf()** nos devuelve la longitud de la estructura que hemos definido.¿ He dicho método ? SI. Señores, aquí la importancia de la POO.

El manejo de las estructuras es una clase, la clase TStruct. Impresionante , no?

A lo que íbamos, para indicarle al miembro de la estructura dwSize el tamaño, nada más fácil:

sInit:dwSize := sInit:SizeOf() // Longitud de la estructura

En el caso de necesitar un miembro de la estructura la longitud, este seria, según la REGLA DE ORO, **la 2.5**, entre el 2 y 3 paso, antes , lógicamente, que la asignación de la variable puente de la variable de instancia **cBuffer**, *uBuffer := sRECT:cBuffer.*

Hay varios tipos de variables. Aplicaremos lo mismo que las que usa el comando DLL32.

También podemos iniciar un miembro en la misma declaración :
MEMBER dwICC AS LONG **INIT 256**.

Ahora bien, como yo, alguien habrá pensado que podemos también iniciar el miembro dwSize como :

MEMBER dwSize AS LONG INIT sInit:SizeOf(),
pero esto es INCORRECTO, pues todavía no es posible saber la longitud de la estructura , pues todavía no ha inicializado dwICC, y la longitud que obtenemos es INCORRECTA.

Mucho cuidado con esto, que puede ser motivo de más de un dolor de cabeza.

5.4.3.- Clase Timprime

¿ Que es la clase TImprime ?

La Clase TImprime es una clase de impresión para Fivewin
.
La clase TImprime en conjunción con una clase que herede sus características, montar un listado o una factura o cualquier otro tipo de documento por la impresora es espectacular.

El mecanismo de esta clase no tiene sentido sin hacer uso de la herencia y además hay que utilizar otra clase, TUtilPrn, que es la que verdaderamente imprime en las hojas, la cuál veremos a continuación.

¿ Entonces para que sirve TImprime ? TImprime te sirve para controlar el inicio / final de Impresión, los saltos de pagina, impresión de un MEMO automático y como broche final dispones de un contador del total de paginas, pagina 1 de ...X, todo ello trabajando sobre centímetros.

Una de las cosas que pienso cuando intento realizar una clase es que debería haber de ser lo más compacta posible. Es por ello que , aunque TUtilPrn es anterior a TImprime, no significa que yo hubiese fusionado las dos para obtener solamente una Clase.

Pero eso es un error. La ventaja es que por ejemplo, yo no tengo porque usar TImprime si me da la gana, y usar simplemente TUtilPrn con la clase TPrinter o TReport directamente, que es posible hacerlo.

Como puedes observar la idea principal no es hacer una clase que lo haga todo , más bien es hacer un conjunto de clases que puedan ser independientes unas de otras , pero que conjuntamente se convierta
en una poderosa herramienta.

Veras que usando la Clase TImprime en conjunción con TUtilPrn y a través de la herencia de tu clase, obtendrás unos resultados que con las clases Standards de Fivewin no puedes obtener.

5.4.3.2.- Ventajas sobre las clases TReport y TPrinter.

La principal diferencia entre las clases Standards de Fivewin y este conjunto de clases son básicamente tres :

1.- **Rapidez.** La rapidez y la claridad son , a mi modo de ver, lo mejor que tienen estas dos clases TImprime / TUtilPrn. Básicamente hay que seguir un patrón de trabajo y punto.

2.-**Espectacularidad**. Realmente , hacer este trabajo manualmente en Fivewin es posible, pero el tiempo requerido es demasiado elevado.
Consulta mi Mini-Faq sobre Reportes, y veras las filigranas que hay que realizar para hacer algo potable. Puedes dibujar realmente lo que quieras sin ningún esfuerzo.

3.- **Sencillez.** Con este juego de clases TImprime y TUtilPrn es como si estuvieras imprimiendo como en Clipper a través del SAY, lo que cambia es que las líneas y filas son CMS en vez de posiciones del papel.

5.4.3.3.- COMANDOS

La Clase TImprime tiene unos comandos básicos donde el fichero TUTILPRN.CH son compartidos por las dos clases y que son :

IMPRIME [INIT | NEW] [<cSpool>]
Este comando inicializa la Clase TImprime , y automáticamente inicializa también la clase TUtilPrn y también la clase TPrinter
cSpool := Opcional. Al nombre que veremos en el spooler de Impresión.

IMPRIME END [<lPageCont>]
Finaliza la Clase TImprime.
lPageCont := .T. Por defecto nos pone contador de paginas. .F. Sin contador de paginas.

Nota:
Estos dos comandos son similares a los comandos de la Clase TPrinter.

PRINTER oPrn / ENDPRINT

Indicar también que podemos hacer uso de los comandos PAGE / ENDPAGE de la clase TPrinter sin ningún tipo de problema para saltar de pagina

---

IMPRIME [<nLinea>,<nFila>] DATETIME
Coloca la Fecha / Hora tal y como lo hace la clase TReport.
nLinea, nFila es opcional. Por defecto sale en la parte superior derecha.

IMPRIME MEMO <cText> ;
[COL <nCol>] ;
[LINES <nLine>];
[FONT <oFont>] ;
[COLOR <nClrText>];
[SEPARATOR <nSeparator>] ;
[JUMP]

Imprime un campo **MEMO** <cText> automáticamente.

Si el campo memo estuviese vació, simplemente no haría nada.

COL <nCol> en la fila / columna indicada. La línea no se la damos pues es la propia clase que la controla ;). Por defecto 1.

LINES <nLine> Es usado por MEMOLINE y MLCOUNT dentro del Método Memo. Cantidad de caracteres a imprimir. **Por defecto 60**

FONT <oFont> Fuente a usar para el MEMO.

COLOR <nClrText> Color de la Fuente.

SEPARATOR <nSeparator> Separación en CMS que tiene que haber cuando cambie de línea. **Por defecto es 0.5 cms**

JUMP Salta la una línea automáticamente. El salto producido será igual a *SEPARATOR.*

ISEPARATOR < nSpace > Salta una línea de Separación en cms.

IMPRIME EJECT**.**
Realiza un salto de página.

Estos son solamente las comandos necesarios, ya que no se requieren nada más. Como podéis observar esta clase es muy simple pero de una potencia tremenda.

5.4.3.4.- DATAs y METHODs

Como puedes observar las variables de instancias son muy pocas.

DATA oUtil
Objeto TUtilPrn.
Como puedes observar aquí, TImprime necesita de la Clase TUtilPrn para poder funcionar adecuadamente. Bueno, simplemente inicializa la clase TUtilPrn para ser usado para más comodidad y no estar inicializando cada vez la TUtilPrn por cada impresión que queramos hacer.

DATA oPrn
Objeto TPrinter

DATA oFnt,oFnt2,oFnt3,oFnt4,oFnt5,oBrush,oPen
Objetos Predeterminados de Fonts, Brush y Pen.

DATA cSpool INIT "UtilPrn"
Descripción para el Spooler de Impresión.

DATA nLinea INIT 1
Línea donde Imprimir.

DATA nFila INIT 1
Fila donde Imprimir.

DATA nEndLine
Indica donde esta el final de la hoja para realizar un salto de pagina.
El final de la hoja es muy fácil de adivinar:

nEndLine := ::oPrn:nVertSiZe() / 10 - 1.

Estamos diciendo que , cogemos el tamaño físico de la hoja en vertical, esta en m/m, lo pasamos a CMS(/ 10), y por seguridad le restamos un(-1) CMS.(La seguridad es por algunos drivers de impresión que no dejan llegar antes el limite teórico del tamaño del papel).

METHOD Init(cSpool) CONSTRUCTOR
Inicializamos Impresión y objetos internos(TUtilPrn,oFonts,etc...)
Igual que el Comando IMPRIME INIT

METHOD Initiate(cSpool)
Lo mismo que Init( )

METHOD End(lPageCount)
Finalizar objetos. Lo mismo que el comando IMPRIME END <lPageCount>

METHOD EndInit(lPageCount)
Lo mismo que End()

METHOD CompLinea(nSuma)
Realiza un salto de pagina si ::nLinea > ::nEndLine, en cristiano, si la línea donde voy a imprimir es mayor que el máximo teórico que me permite la hoja, realizo un salto de pagina.

METHOD Separator(nSpace)
Realiza un salto de línea de <nSpace>, en CMS, el cual a su vez hace una llamada a CompLinea, para saber si tiene todavía espacio para imprimir o debe de saltar pagina.

METHOD PageCount()
Aquí tenéis el famoso problema a lo de la pagina 1 de ..X.
Al contrario que lo que hacen muchos con la clase TReport al usar el method Rebuild(), este método es **MUCHISIMO MAS POTENTE**, puesto **QUE NO VUELVE A GENERAR** otra vez el listado.
Esta solución es la más simple y eficaz que encontrarás.

METHOD Eject()
Realiza un salto de página.
Igual que el comando IMPRIME EJECT

MESSAGE Date METHOD _DateTime(nLinea, nFila)
Poner Fecha / Hora en la Línea <nLinea> y Fila <nFila>.
Igual que el comando IMPRIME DATETIME.

MESSAGE MesFecha METHOD _MesFecha_(dDate)
Uso interno. Devuelve el mes en castellano de una fecha dada.

METHOD MEMO()
Equivalente al comando IMPRIME MEMO

5.4.3.5.-Nuestro primer Listado

Para realizar nuestro primera impresión usando la clase Timprime con TUtilPrn, procederemos a crear una clase que herede de Timprime.

Como comentario indicar que todas las impresiones que queráis hacer se basara en POO.

Si no tenéis ni idea de POO, Programación Orientado al Objeto, tenéis a vuestra disposición varios documentos sobre el asunto, en este caso, yo publique una guía sobre Objects, el motor de objetos que trae Fivewin.

También ya otro muy bueno de **Manu Expósito.**
De entrada doy por sentado que ya disponéis de conocimientos de POO.

Si miráis el Test.prg que acompaña a esta ayuda, veréis que no tiene ninguna complicación.
Empecemos pues a definir la clase.

Vamos a realizar , un listado de pendientes de cobro, por ejemplo, en líneas generales y después podemos observar completamente el código fuente aquí mismo comentado también.

*1.- CLASS TpentCob **FROM Timprime***
Nunca una línea de código dijo tanto ;)
Aquí estamos disponiendo de todas las cualidades de la Clase Timprime a través de la herencia.

2.- Declarar las datas o variables de instancias que necesites.

3.- Declarar un método constructor para tu clase.

4.- **OBLIGATORIO** es incorporar el método este ***método Separator*** en tu nueva clase

```
METHOD Separator( nSpace ) CLASS TPENTCOBRO

  // Si habido un salto de pagina despues de una linea de separacion
    if Super:Separator( nSpace ) // Atentos. Llamada a la TImprime.
      ::Cabecera()               // Opcional , eso   depende de ti
      ::nLinea := 3              // Posición donde empezara a imprimir
    endif

  Return Self
```

Vamos a explicar que hace realmente este método y veréis el porque es obligatorio.

Cuando estemos realizando una impresión , para un salto de línea debemos de disponer de un método que nos realice tal efecto, ¿ de acuerdo ?, bien pero a que además quieres que la clase te controle automáticamente si debe de saltar de pagina, ¿ no ? , pues este método si te fijas bien, verás que hace una llamada al método de la clase Superior, a la clase Timprime, que es la que verdaderamente es la que nos controla el salto de pagina.

Después si queréis que realiza una determinada tarea cada que vez que se produce un salto de pagina, pues lo ponéis. En este caso en concreto, bueno yo utilizo esto en todos mis impresiones, es crearse un método , por ejemplo, cabecera, que hace la función de imprimirme las cajas ,etc.. en todas las páginas.

Ya esta! Ya no es necesario nada más! Ahora simplemente programas como quieres que salga el listado como lo harías en clipper y punto. Que quieres un control de Grupos etc,... pues con tus IF..ENDIF de toda la vida , con tus While , DO..CASE , te lo montas como quieras.

Aquí no hay eventos, ni cosas raras, que ha veces no sabes muy bien donde tienes que meter tu código ,etc...

Si miras el Test.prg verás que más fácil no puede ser ;) (Al menos para mi).
Y después solamente debemos de usar los comandos de la Clase TutilPrn que será la nos facilitará las cosas.

Ahora os dejo disfrutar y recuerda que solamente depende de tu imaginación.

Veremos a continuación el Test.prg, un listado de facturas.

```
#INCLUDE "Fivewin.CH"
#INCLUDE "Utilprn.CH"
#DEFINE CLR_GREY 14671839

*Clase Para Listados Facturas.
* © Rafa Carmona ( Thefull )

CLASS TFACTURA FROM TIMPRIME
     DATA oDbf
     METHOD New( )
     METHOD Cabecera()
     METHOD Lineas()
     METHOD Separator()  // Este method es compartido por las dos clases
ENDCLASS
```

Tal y como habíamos visto anteriormente podemos observar que la clase Tfactura ‘hereda’ desde Timprime.

A continuación podemos observar las DATA que nos interesa , en este caso, será un objeto Database que le pasaremos a la clase.

Después declaramos todos los métodos de la clase Tfactura y el método obligatorio Separator().

```
METHOD New()  CLASS TFACTURA

  Local nOldFocus

  USE FACTURAS NEW SHARED
  DATABASE ::oDbf

 ::oDbf:bEof := {|| Nil }

 IMPRIME INIT "Listado de Facturas"  // Inicializamos clase TImprime

  nOldFocus := (::oDbf:cAlias)->( OrdSetFocus( 1 ) ) // Ordenado por Vto

   PAGE               // Nueva página para imprimir
     ::Cabecera()     // Enseña cabecera
     ::Lineas()       // Procesa líneas de facturas
   ENDPAGE            // Fin de la página

 IMPRIME END .F.  // Fin de la clase Timprime SIN contador de páginas

  (::oDbf:cAlias)->( OrdSetFocus( nOldFocus ) ) // Ordenado por el que vino

  ::oDbf:Gotop()
  ::oDbf:Close()

Return .T.
```

El método que será llamado desde la opción de un menú o desde un botón , por ejemplo, *MENUITEM "Listado Factura" ACTION TFactura():New( )*

```
 METHOD Cabecera() CLASS TFACTURA
   Local cEmpresa := " Uso Sobre TImprime by (c)2001 TheFull "
   Local oFont,oPen,oBrush
   Local nColor := 35724527

   DEFINE FONT  oFont  NAME "Arial" SIZE 0,-16 BOLD OF ::oPrn
   DEFINE PEN   oPen   WIDTH 3 COLOR CLR_GREY
   DEFINE BRUSH oBrush COLOR nColor

   IMPRIME DATETIME   // Imprimimos la fecha /hora automáticamente

   UTILPRN ::oUtil 0.5,1 SAY cEmpresa FONT ::oFnt5
```

```
 UTILPRN ::oUtil ;
    MSG { { "Listado de Facturas",oFont,CLR_RED,PAD_CENTER },;
      { "Ordenado por Codigo",::oFnt,CLR_BLUE,PAD_CENTER },;
      { "Ejemplo de Uso de TImprime / TUtilPrn", ::oFnt2,CLR_GREEN,PAD_LEFT } },;
       AT 1.25,0;
       BRUSH ::oBrush ;
       PEN oPen ;
       ROUND  20,20 ;
       SHADOW WIDTH 0.1 ;
       SHADOWPEN oPen  ;
       EXPANDBOX 0.25,2 TITLE

 UTILPRN ::oUtil ;
       MSG { { "Codigo"  ,::oFnt2 ,CLR_YELLOW,PAD_LEFT },;
            { "Factura", ::oFnt2,CLR_HBLUE , PAD_RIGHT} } AT 3.5,1;
       BRUSH oBrush ;
       PEN ::oPen ;
       SHADOW WIDTH 0.05 ;
       EXPANDBOX 0.25,.5  LEFT

 UTILPRN ::oUtil ;
       MSG "FECHA" AT 3.5,5 TEXTFONT oFont  TEXTCOLOR CLR_BLUE ;
       BRUSH oBrush ;
       PEN ::oPen ;
       SHADOW WIDTH 0.05 ;
       EXPANDBOX 0.25,.5  CENTER

 UTILPRN ::oUtil ;
       MSG { "Precio","Base" } AT 3.5,10 TEXTFONT ::oFnt2 TEXTCOLOR
CLR_BLUE ;
       BRUSH ::oBrush ;
       PEN ::oPen ;
       SHADOW WIDTH 0.1 ;
       ROUND 50,50 ;
       EXPANDBOX 0.25,.5  RIGHT

 UTILPRN ::oUtil ;
       MSG { "Precio","Total","Todo By Thefull" } AT 3.5,14 TEXTFONT ::oFnt3
TEXTCOLOR CLR_GREEN ;
       ROUND 150,150 ;
       EXPANDBOX 0.25,.5  CENTER

  oFont:End()
  oPen:End()
  oBrush:End()
```

```
   ::oUtil:Reset()  //Valores por defecto

  RETURN NIL
```

Aquí, dentro de este método podemos observar , como a través de la herencia , podemos hacer uso de una variable de instancia llamada oUtil, la cual contiene el objeto de la clase TUtilPrn ya iniciado automáticamente por la llamada IMPRIME INIT.
Podemos observar las diferentes maneras de construir una caja a través de las diferentes posibilidades que nos brinda la clase TUtilPrn.
Consultar más adelante para saber como funciona MSGBOX,etc..

```
METHOD Lineas() CLASS TFACTURA
 Local oPen,oBrush

 DEFINE BRUSH oBrush  COLOR CLR_YELLOW

 DEFINE PEN oPen STYLE PS_NULL WIDTH 0

 ::nLinea := 5   // comenzamos a 5 cms

 While !::oDbf:Eof()

     UTILPRN ::oUtil Self:nLinea,1 SAY  ::oDbf:Codigo   ;
                FONT ::oFnt3 COLOR CLR_BLUE;
                SHADOW lSombra() BRUSH ::oBrush WIDTH 16 PEN oPen

      UTILPRN ::oUtil Self:nLinea,6 SAY  ::oDbf:Fecha ;
                FONT ::oFnt3 COLOR CLR_RED   CENTER

      UTILPRN ::oUtil Self:nLinea,11.5 SAY ::oDbf:Base ;
                FONT ::oFnt2 COLOR CLR_RED   RIGHT

     UTILPRN ::oUtil Self:nLinea,17 SAY   ::oDbf:Total :
                FONT ::oFnt  COLOR CLR_GREEN RIGHT ;
               SHADOW .T.  BRUSH oBrush PEN oPen WIDTH 3

     ISEPARATOR
    ::oDbf:Skip()
 End While

 oPen:End()
```

```
  oBrush:End()

RETURN .T.
```

Como podéis apreciar, el método lineas() no contiene ningún misterio :
Hacemos uso de un bucle, mientras no sea final de fichero , y vamos escribiendo los campos que nos interesa que se impriman , donde a continuación saltamos un espacio, ISEPARATOR, y luego un registro.
Nada más!!!
Mirar si esto es potente, que nosotros ya no nos tenemos que preocupar cuando tiene que saltar la pagina, cuando debe de enseñar la cabecera, pues es la propia tecnología que aplicamos que lo hace la clase por si misma!

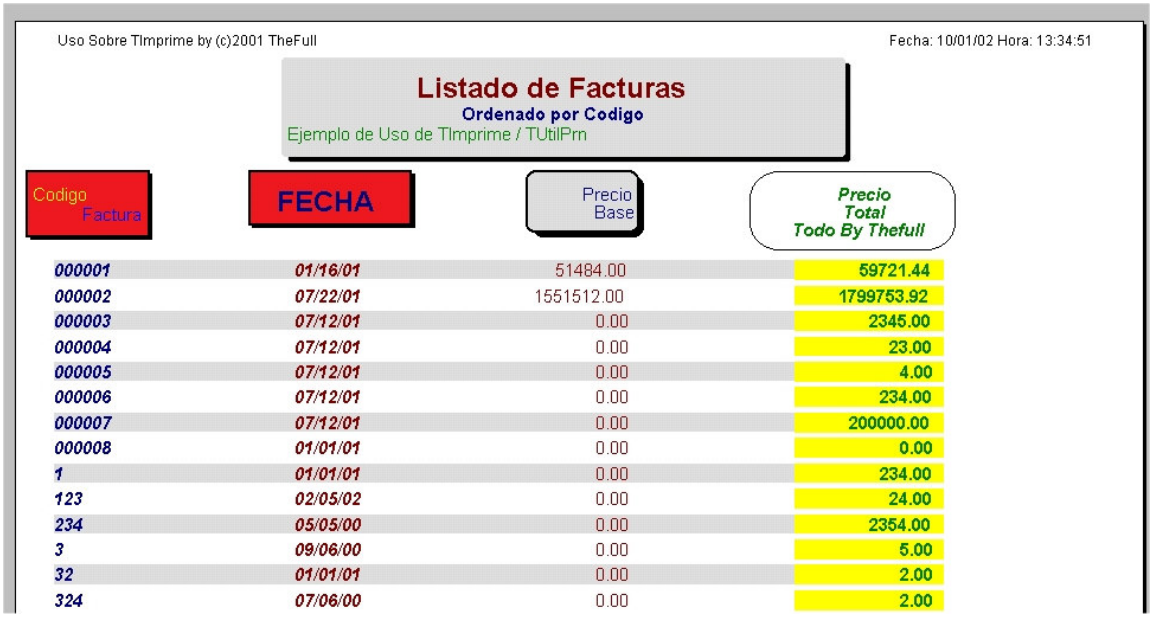
```
METHOD Separator( nSpace ) CLASS TFACTURA

   // Si habido un salto de pagina después de una línea de separación
   if Super:Separator( nSpace ) // Atentos. Llamada a la TImprime.
      ::Cabecera()
      ::nLinea := 5
   endif

Return Self
```

Aqui podeís observar como hacemos uso del método Separator(), el cual es obligatorio.

```
Static Func lSombra()

   Static lRet := .F.
   IF lRet
      lRet := .F.
   ELSE
      lRet := .T.
   ENDIF

Return lRet
```

Y por última una función que nos quitará o pondrá una sombra al texto.

Pero como bien dice el refrán , una imagen vale más que mil palabras, aquí tenéis el resultado de este código tan pequeño.
¿ Impresionante , no ? ;)



Uso Sobre TImprime by (c)2001 TheFull — Fecha: 10/01/02 Hora: 13:34:51

Listado de Facturas
Ordenado por Codigo
Ejemplo de Uso de TImprime / TUtilPrn

Codigo Factura	FECHA	Precio Base	Precio Total Todo By Thefull
000001	01/16/01	51484.00	59721.44
000002	07/22/01	1551512.00	1799753.92
000003	07/12/01	0.00	2345.00
000004	07/12/01	0.00	23.00
000005	07/12/01	0.00	4.00
000006	07/12/01	0.00	234.00
000007	07/12/01	0.00	200000.00
000008	01/01/01	0.00	0.00
1	01/01/01	0.00	234.00
123	02/05/02	0.00	24.00
234	05/05/00	0.00	2354.00
3	09/06/00	0.00	5.00
32	01/01/01	0.00	2.00
324	07/06/00	0.00	2.00

5.4.3.6.-Conclusiones

Como podéis observar, la clase no tiene ninguna complicación, ni grandes cosas ni una tecnológica que es la polla, que realiza 'tropezientas cozas' a la vez. NO! Es lo más simple que se me ha ocurrido y en contra obtengo una manera rápida y eficaz de imprimir.

Si habéis tenido la oportunidad de leeros ***la Mini-Faq***, os dejéis cuenta que para controlar los Reportes, es mucho más complicado y para hacer las cajas de los títulos con colores, etc,.. hay que sufrir, o modificar la clase original.

Te puedo asegurar, que con esta clase TImprime en conjunción con la TUtilPrn, eso lo tienes superado por goleada.

5.4.4.- Clase TUtilPrn

¿ Que es la clase TUtilPrn ?

La clase TUtilPrn es una clase para la impresión de Windows bajo Fivewin.
Esta clase esta pensada para poder ser usada por las clases TPrinter o TReport indistintamente.

Pero su finalizad principal a sido para ser usada por la clase TImprime, llegando a tener dos clases independientes entre sí, pero ligadas para hacer algo más potente.

El mecanismo de funcionamiento de la clase esta pensado que todas las coordenadas que afecten a la realización de cualquier instrucción de salida para la impresora, este expresado en CMS.

De esta manera lograremos que funcione sin problemas, aparentemente, en cualquier impresora, dependiendo del driver de tu impresora.

Una de las cosas que he implementado es la selección de objetos por defecto.
A través del comando **SELECT** <*oObjeto*> podemos definir un objeto **brush**, **Pen** o **Font** que estará disponible para su uso sin estar todo el rato pasándole el objeto a los demás métodos de impresión.

Por ejemplo
Todas las salidas usaran un Brush determinada:

```
DEFINE BRUSH oBrush COLOR CLR_RED.
```

Si ponemos UTILPRN oUtil SELECT oBrush, cuando llamemos a poner un rectángulo, *UTILPRN oUtil BOX 1,1 TO 2,2* , él sabe ya que tiene que usar el brush que habíamos seleccionado, en este caso usará automáticamente el Brush oBrush.

Y si además definimos un Pen, igualmente usara el Brush que tenia más el pen.
Por ejemplo:

```
UTILPRN oUtil SELECT oBrush
UTILPRN oUtil SELECT oPen
UTILPRN oUtil BOX 1,1 TO 2,2
```

Que ventajas me aporta este comando?
La ventaja de no tener que estar pasando continuamente los objetos para usar unas características determinadas.
De esta manera, un grupo de comandos pueden hacer uso de un Brush, Pen o Font si tener que estar todo el rato pasando argumentos a los comandos.

La idea es muy simple: Escribir cuanto menos, mejor ;)

5.4.4.2.-Ventajas que me aporta usar TUtilPrn

Las ventajas que te puede aportar esta clase es sobre todo rapidez a la hora de diseñar tus impresos de una manera rápida y sencilla.

Esta pensada sobre todo a los nuevos usuarios de Fivewin que se incorporan a esta fantástica librería, y en concreto al mundo de la impresión bajo Windows, ya que la impresión no tiene nada que ver con los listados que se hacían en Clipper.

La primera toma de contacto con una impresión en Fivewin es muy dura.
Parece sencillo , pero a medida que queramos realizar cosas muy concretas y espectaculares, empezaremos a darnos cuenta de que esto no es tan sencillo como se suponía.

Pero esto tu no lo tienes que pasar si no quieres. Nosotros ya lo hemos pasado, y créeme , solamente merece la pena si quieres ver como funciona internamente el mecanismo de impresión.

Si no quieres perder el tiempo, pues con esta clase estas más que servido. ;)

Una de las limitaciones es que esta clase solamente permite un tipo de salida, a la impresora o al Preview. No tiene salida a fichero, pues no la diseñe para tal cuestión.

La comunidad de Fivewin a creado y donado multiples clases para distintas salidas de formatos como si fuera una impresión, ya sea una salida para el Word , Excel, Html , Rtf , etc...

5.4.4.3.- Comandos

Veremos a continuación los diferentes comandos que nos proporciona la clase TUtilPrn , para hacernos la vida menos sufrida.

DEFINE UTILPRN <oUtil> [< of: PRINTER,OF> <oPrinter>] ;
[BRUSH <oBrush>] [PEN <oPen>]

Este comando inicializa la clases para usarla en un objeto TPrinter.

<oUtil>
➔ Objeto TUtilPrn.

PRINTER | OF <oPrinter>
➔ Objeto printer sobre el cual va a trabajar el objeto TUtilPrn. Si queremos hacerla servir bajo la clase TReport inicializar la clase TUtilPrn así: OF oReport:oDevice.

BRUSH <oBrush>
➔ Brush por defecto que usaremos.

PEN <oPen>
➔ Pen por defecto que usaremos.

UTILPRN <oUtil> SELECT <oObjeto> [COLOR <nColor>]

Este comando seleccionará por defecto unos objetos para ser usados por defecto. Como anteriormente explique, a través de este sencilla técnica podemos evitar tener que estar definiendo continuamente que tipo de letra vamos a usar para, por ejemplo, usar el comando UTILPRN ...TEXT...
Simplemente definiéndolo una vez, será aplicable a todos los TEXT o comandos que hagan uso de una Fuente en concreto.

<oObjeto>
➔ Puede ser un objeto Brush, Pen o Font.

<nColor>

➔ Solamente sirve para cuando pasamos un objeto Fuente. De que color será la fuente por defecto a usar.

UTILPRN <oUtil> BOX <nX>,<nY> TO <nX2>,<nY2> ;
[BRUSH <oBrush>];
[PEN <oPen>] [ROUND [<nZ>,<nZ2>]]

Este comando dibuja un rectángulo en la impresora expresado en cms

BOX <nX>,<nY> TO <nX2>,<nY2>

➔ Coordenadas de la caja en la hoja.

BRUSH <oBrush>

➔ Opcional. Brush a usar para rellenar el rectángulo.

➔ Si no se especifica , se usara por defecto el Brush seleccionado por el comando SELECT

PEN <oPen>

➔ Opcional. Pen a usar para la Caja.

➔ Si no se especifica , se usara por defecto el Pen seleccionado por el comando SELECT

ROUND <nZ>,<nZ2>

➔ Si queremos que los bordes sean redondeados.

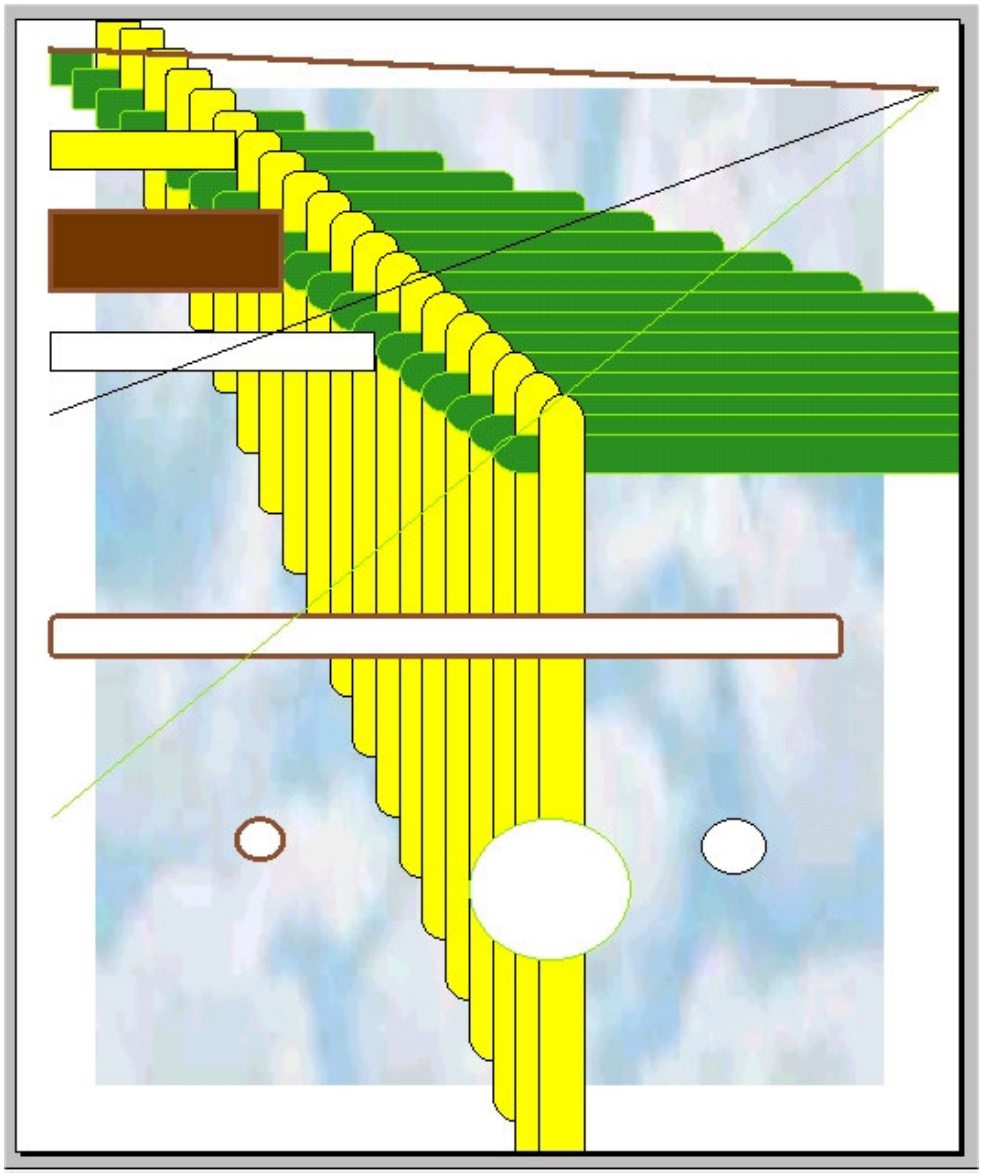
➔ **<nZ>,<nZ2>** Ángulos de la Elipse que forman las esquinas del rectángulo.

Ejemplo: Este ejemplo nos muestra como poner una imagen, cajas,lineas y círculos

```
UTILPRN oUtils 2,2 IMAGE ".\nubes.jpg"  JPG PAGE

// Veinte cajas transformandose
FOR x := 1 TO 20

  // Caja redondeada con la brocha por defecto ,pero con el lapiz que le digo
```

```
    UTILPRN oUtils BOX nLinea,nFila TO nLinea+1,nFila+x     ROUND 10*X,10*X;
                          PEN oPen2

   // Caja redondeada con la brocha amarilla ,pero con el lapiz por defecto
   UTILPRN oUtils BOX nLinea-1,nFila+1 TO nLinea+x,nFila+2 ROUND 10*X,10*X;
                         BRUSH oBrush
     nLinea += 0.5
     nFila += 0.5

   NEXT

   nLinea := 1

   UTILPRN oUtils SELECT oBrush
   UTILPRN oUtils SELECT oPen

   // Una caja usando la brocha y lapiz por defecto
   // Note que ahora los valores por defecto han cambiado.
   UTILPRN oUtils BOX nLinea+2,1 TO nLinea+3,5

   // Ahora una caja con la brocha y pen que le hemos pasado.
   UTILPRN oUtils BOX nLinea+4,1 TO nLinea+6,6 BRUSH oBrush3 PEN oPen3
   // Usa Lapiz Pen3
   UTILPRN oUtils LINEA 1,1 TO 2,20 PEN oPen3

   // Ahora, reseteamos la brocha y el lapiz. Tenemos Brocha Nula y Lapiz Negro a 1
   // y dibujamos caja para comprobar
   oUtils:Reset()
   UTILPRN oUtils BOX nLinea+7,1 TO nLinea+8,8

   // Ahora , tal y como lo tenemos por defecto, solamente usando el PEN y borders
ROund
   UTILPRN oUtils BOX nLinea+14,1 TO nLinea+15,18 PEN oPen3  ROUND
   UTILPRN oUtils LINEA 10,1 TO 2,20 // Usando el lapiz por defecto de la clase
   UTILPRN oUtils LINEA 2,20 TO 20,1 PEN oPen2 // Usando el lapiz definido

   // DIbujando unos circulos
   UTILPRN oUtils CIRCLE 20,10 WIDTH 500 PEN oPen2 BRUSH oBrushImage
   UTILPRN oUtils CIRCLE 20,5  WIDTH 150 PEN oPen3
   UTILPRN oUtils CIRCLE 20,15 WIDTH 200 BRUSH oBrush

  ENDPAGE
```

Como podéis apreciar en la siguiente imagen el resultado es espectacular.



UTILPRN <oUtil> LINEA <nX>,<nY> TO <nX2>,<nY2> [PEN <oPen>]

Dibuja una línea desde nX,nY hasta nX2,nY2 con un PEN determinado.

LINEA <nX>,<nY> TO <nX2>,<nY2>

➔ Dibuja una línea desde nX,nY hasta nX2,nY2 con un PEN determinado.

[PEN <oPen>]

- ➔ Opcional. Pen a usar para dibujar la línea.
- ➔ Si no se especifica , se usara por defecto el Pen seleccionado por el comando SELECT

(Ver ejemplo de Box)

UTILPRN <oUtil> CIRCLE <nX>,<nY> WIDTH <nWidth> ;
[BRUSH <oBrush>] [PEN <oPen>]

- ➔ Dibuja un circulo en las coordenadas nX, nY con un tamaño determinado , <nWidth>, en cms.

BRUSH <oBrush>

- ➔ Opcional. Brush a usar para rellenar el circulo.
- ➔ Si no se especifica , se usara por defecto el Brush seleccionado por el comando SELECT

PEN <oPen>

- ➔ Opcional. Pen a usar para el circulo.
- ➔ Si no se especifica , se usara por defecto el Pen seleccionado por el comando SELECT

(Ver ejemplo de Box)

UTILPRN <oUtil> ELLIPSE <nX>,<nY> TO <nX2>,<nY2> ;
[BRUSH <oBrush>] [PEN <oPen>]

- ➔ Dibuja una elipse desde las coordenadas nX, nY hasta nX2, nY2

BRUSH <oBrush>

- ➔ Opcional. Brush a usar para rellenar la elipse.
- ➔ Si no se especifica , se usara por defecto el Brush seleccionado por el comando SELECT

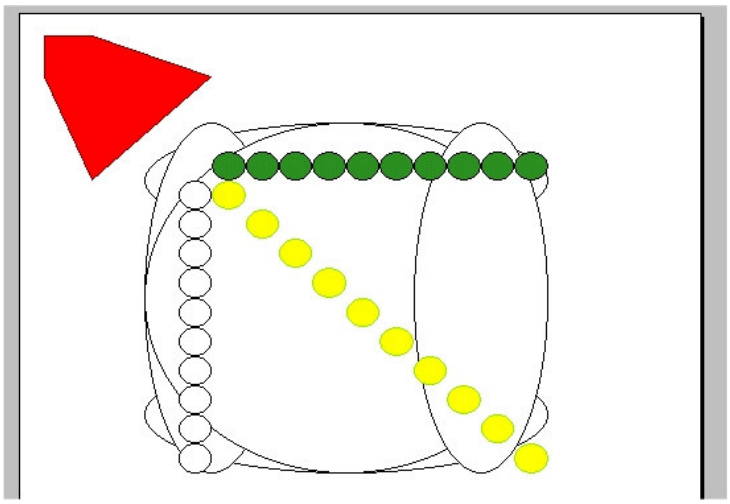
PEN <oPen>

- ➔ Opcional. Pen a usar para la elipse.
- ➔ Si no se especifica , se usara por defecto el Pen seleccionado por el comando SELECT

Ejemplo: Unas Elipses y un polígono

```
PAGE
  //Elipses y Poligonos
  UTILPRN oUtils SELECT oPen4
  UTILPRN oUtils SELECT oBrush4
  UTILPRN oUtils POLYPOLYGON {100,100},{100,300},{300,800},;
                    {800,300},{300,100},{100,100}

  oUtils:Reset()
  UTILPRN oUtils ELLIPSE 4,4 TO 8,16
  UTILPRN oUtils ELLIPSE 16,4 TO 12,16

  UTILPRN oUtils ELLIPSE 4,4 TO 16,8
  UTILPRN oUtils ELLIPSE 4,4 TO 16,16

  UTILPRN oUtils ELLIPSE 4,16 to 16,12
  FOR x := 1 TO 10
    UTILPRN oUtils ELLIPSE 5+x,5 TO 6+x,6 PEN oPen BRUSH oBrush4
    UTILPRN oUtils ELLIPSE 5,5+x TO 6,6+x PEN oPen BRUSH oBrush2
    UTILPRN oUtils ELLIPSE 5+x,5+x TO 6+x,6+x PEN oPen2 BRUSH oBrush
  NEXT
   ENDPAGE
```

Y el resultado de este código tan simple es la siguiente imagen:



UTILPRN <oUtil> <POLY,POLYPOLYGON> <aPoly,...> ;
[MODE <nMode>][BRUSH <oBrush>][PEN <oPen>]

➔ Dibuja un polígono dado unos vectores <aPoly>.

[MODE <nMode>]
➔ Tipo de relleno del Polígono

BRUSH <oBrush>
➔ Opcional. Brush a usar para rellenar el circulo.
➔ Si no se especifica , se usara por defecto el Brush seleccionado por el comando SELECT

PEN <oPen>
➔ Opcional. Pen a usar para el circulo.
➔ Si no se especifica , se usara por defecto el Pen seleccionado por el comando SELECT

(Ver ejemplo de la Elipse)

UTILPRN <oUtil> [<nX>,<nY>] IMAGE <cFile> [SIZE <nX2>,<nY2>] ;
[JPG][PAGE][RASTER <nRaster>]

➔ Muestra una imagen en nX,nY.
➔ SIZE espeficica el tamaño que tendra la imagen.
➔ JPG. Hace uso de la librería nViewLib para visualizar Jpg,gif,etc...
➔ PAGE. La imagen ocupa toda la hoja.
➔ RASTER Operación raster a aplicar a la imagen.

Ejemplo: Ejemplo de cómo poner una imagen y un texto

```
PAGE

   // Para imprimir una imagen en el formato actual de papel ,
   //simplemente le decimos la imagen que es y la clausula PAGE y yasta!
    UTILPRN oUtils IMAGE ".\logotipo.jpg" JPG PAGE
    // Podemos usar tambien la clausula JPG para poner Bitmaps
    UTILPRN oUtils 1,1 IMAGE ".\logotipo.jpg" SIZE 10,10 JPG
    // Estilo de fondo
   UTILPRN oUtils 10,15 SAY "TEXTO OPACO"  COLOR CLR_YELLOW ;
                        FONT oFontGrande MODE 2
   UTILPRN oUtils 21,05 SAY "TEXTO TRANSPARENTE "COLOR CLR_WHITE ;
                        FONT oFontGrande MODE 1
 ENDPAGE
```

Con este simple código obtenemos resultados inesperados ;)

```
UTILPRN <oUtil>  [<nRow>,<nCol> SAY <cText> ];
              [ FONT <oFont> ][ COLOR <nClrText> ][ MODE <nBkMode> ];
              [ LEFT,RIGHT,CENTER ];
              [ [ [ SHADOW <blShadow> ] [IMAGE <cImage>] ;
                  [BRUSH <oBrush>]  [PEN <oPen>];
                  [WIDTH <nX>] [ HEIGTH <nY> ] ] ]
```

➔ Coloca un texto en las posiciones nRow,nCol determinadas en cms. Si definimos la cláusula SHADOW, además, podemos definir si la sombra será una imagen o un brush, que ancho y alto queremos y además que Pen usáremos para el contorno del sombra.

FONT

➔ Tipo de Fuente a usar. Por defecto usa la fuente seleccionada o la fuente por defecto del objeto printer si no hubiese ninguna.

COLOR

➔ Color de la fuente.

MODE

➔ 0 Opaco 1 Transparente. Respecto al fondo

LEFT,RIGHT,CENTER.

➔ Alineación del texto.

SHADOW

➔ Indicamos que tipo de fondo queremos para nuestro texto. Cuando usamos esta cláusula del preprocesado, como podemos observar arriba en la declaración del comando, podemos acceder a otras cláusulas adicionales para definir exactamente que tipo de sombra.

IMAGE

➔ Imagen que usaremos como sombra.

BRUSH

➔ Brocha que usaremos como sombra. Tiene preferencia que IMAGE.

PEN

➔ Pen que usaremos como para la sombra.

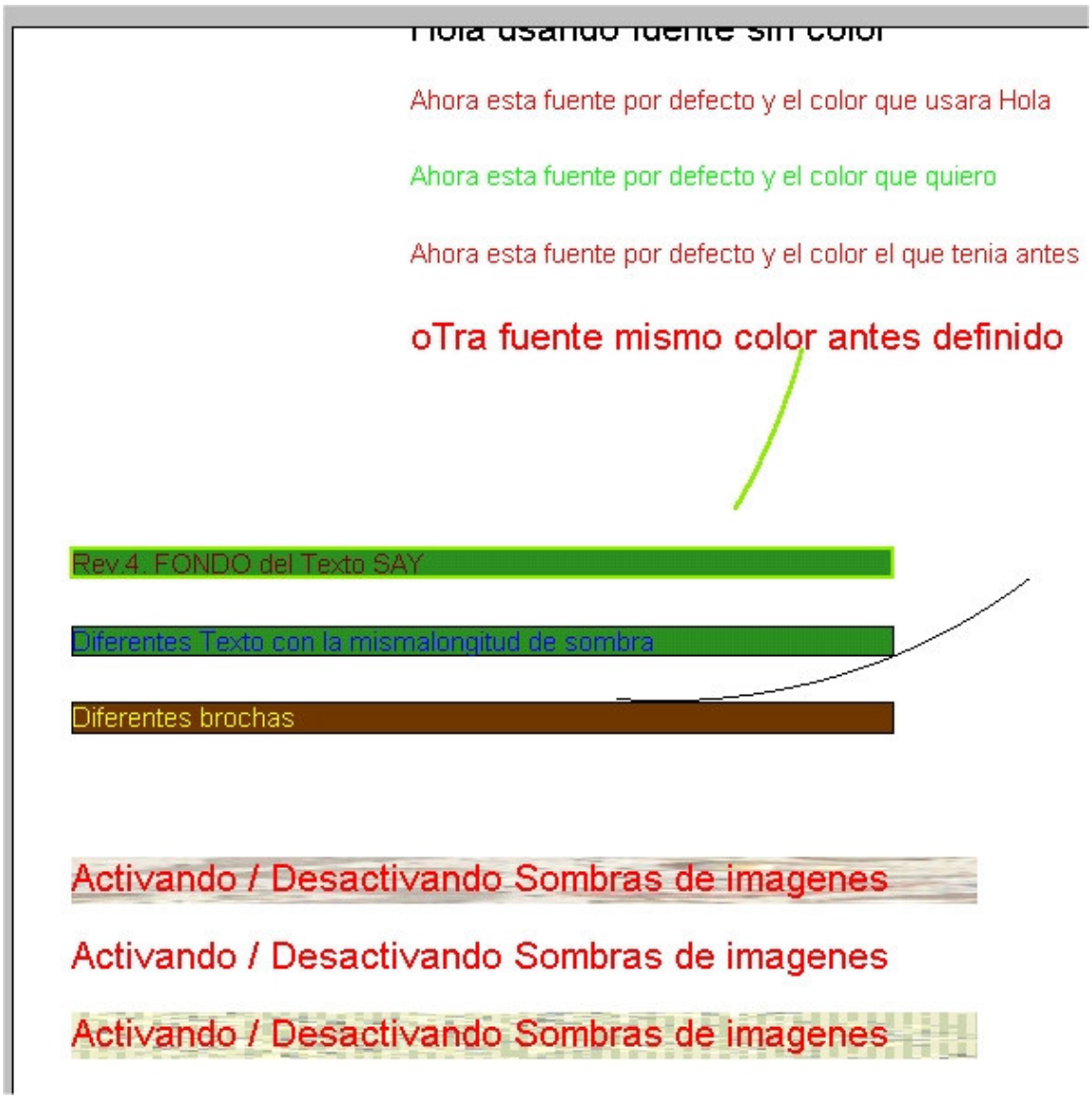
WIDTH <nX> HEIGTH <nY>

➔ Ancho y Alto de la sombra. Por defecto es el ancho y alto del texto a escribir.

Ejemplo: Podemos observar en este ejemplo distintas formas de enseñar un texto corriente , pero que aplicando un poco de color , los resultados son impactantes ;)

```
PAGE
     UTILPRN oUtils ARC 1,1 TO 16,15 START 1,270 END 270,360
     UTILPRN oUtils ARC 1,1 TO 16,10 START 180,270 END 360,270 PEN oPen2

     UTILPRN oUtils 5,5 SAY "Hola" COLOR CLR_RED
     UTILPRN oUtils 6,5 SAY "Hola usando Fuente" COLOR CLR_HBLUE ;
                            FONT oFont
     UTILPRN oUtils 7,5 SAY "Hola usando fuente sin color" FONT oFont
*Atentos. El color es para todas las fuentes. NO PARA UNA FONT EN CONCRETO!
    UTILPRN oUtils SELECT oFontDefault COLOR CLR_HRED

    UTILPRN oUtils 8,5 SAY "Ahora esta fuente por defecto y el color que usara Hola"
     UTILPRN oUtils 9,5 SAY "Ahora esta fuente por defecto y el color que quiero"  ;
                            COLOR CLR_HGREEN
  UTILPRN oUtils 10,5 SAY "Ahora esta fuente por defecto y el color el que tenia antes"
  UTILPRN oUtils 11,5 SAY "oTra fuente mismo color antes definido  " FONT oFont

     UTILPRN oUtils 14,1 SAY "Rev.4. FONDO del Texto SAY "  COLOR CLR_RED ;
                     SHADOW .T.  PEN oPen2 WIDTH 10
     UTILPRN oUtils 15,1 SAY "Diferentes Texto con la misma"+;
                        "longitud de sombra " ;
                     COLOR CLR_HBLUE ;
                     SHADOW .T.  WIDTH 10
     UTILPRN oUtils 16,1 SAY "Diferentes brochas" ;
                     COLOR CLR_YELLOW ;
                     SHADOW .T. BRUSH oBrush3 PEN oPen WIDTH 10

      FOR X := 1 TO 10
      UTILPRN oUtils 17+X ,1 SAY "Activando / Desactivando Sombras de  imagenes" ;
                     COLOR CLR_HRED MODE 1 ;
                     SHADOW lSombra() IMAGE cGetFile( "*.*", "<cTitle>", , ;
                                          "c:\d\fw20\bitmaps\.") FONT oFont
        NEXT
      ENDPAGE
```

Y el resultado obtenido es esta imagen. Como podéis observar el comando SAY de la clase TUtilPrn es bastante potente en cuanto opciones.



```
UTILPRN <oUtil> ;
        MSG [ <cText>  [ AT <nRow>,<nCol> ];
        [TEXTFONT <oFont>] [TEXTCOLOR <nClrText>] ];
        [<nX>,<nY> TO <nX2>,<nY2>] ;
        [ BRUSH <oBrush>];
        [ PEN <oPen> ] ;
        [ <lRound: ROUND >  [ <nZ>,<nZ2>  ] ];
        [ <lShadow: SHADOW> [ WIDTH <nShadow> ] ];
        [SHADOWBRUSH <oBrushShadow>];
        [SHADOWPEN <oPenShadow>];
        [ EXPANDBOX <nAlto>,<nAncho> ] ;
        [ <lMode: TITLE> ] ;
        [ <lNoBox: NOBOX> ] ;
        [ LEFT,RIGHT,CENTER ]
```

Este comando quizás sea el mas completo y útil que posee la **UTILPRN**, puesto que con el lograremos , aparte de un efecto visual elegante, una rapidez asombrosa para dotar de cajas de mensajes a nuestros impresos.
Este comando tiene múltiples combinaciones, puede trabajar de diversas formas, para hacer diferentes salidas. No se ha creado otros comandos parecidos puesto que con este la mayoría de efectos ya se consiguen por si solos.

El método que crea el contenido tienen en cuenta cada uno de las opciones que actives y actuara en consecuencia. Por ejemplo:
Si queremos simplemente una caja con sombra, UTILPRN oUtil MSG 1,1 TO 10,10 SHADOW,
en cambio si hacemos UTILPRN oUtil MSG "TEXTO" 1,1 TO 10,10 AT 5,5 la preferencia la marca AT puesto que colocara el texto primeramente, ignorando la cláusula TO.

Es por ello que hay que jugar con lo que uno quiera hasta darse cuenta que TO, en este caso es totalmente innecesario. El comando TO simplemente esta para crear cajas vacías con sombras, sin Texto.¿ Y que utilizad tiene esto ? Pues un formulario, por ejemplo, que se tiene que escribir a mano ;)

MSG [<cText>]

➔ Texto literal o un array a mostrar o un array bidimensional de tres elementos. Los elementos del array será mostrados línea por línea automáticamente. Como puedes ver , hay tres maneras de mostrar un texto en una caja.

1.- *Texto simple :* ***MSG "Hola"***

2.- *Array de varios elementos :*

MSG { "Esto es","un array","unidimensional" }

3.- *Array bidimensional de cuatro elementos { cText, oFont, nColor,nAlign } por elemento:*

```
MSG { { "Color Rojo",    oFont      ,CLR_RED,   PAD_CENTER },;
      { "Fuente nueva",oFontNew ,NIL       ,  PAD_LEFT } }
```

Atentos, si no se pasa la fuente , el color o la alineación **, hay que pasar un NIL**. Si no se especifica fuente , cogerá la fuente especificada por defecto en el comando MSG cText .TEXTFONT oFont y sino cogerá la que hayamos escogido con el comando SELECT.
El color y la alineación la misma condición que la fuente.
No será por posibilidades!!! XD

[AT <nRow>,<nCol>]

➔ Posición donde colocar la caja. Atentos al uso de **EXPANDBOX** que te variara la posición real de la caja. Si hacemos uso de **TITLE** , la fila <nCol> será ignorada, por que será recalculada.

[TEXTFONT <oFont>] [TEXTCOLOR <nClrText>]]

➔ Objeto fuente por defecto para el texto y color de la fuente opcional.

[<nX>,<nY> TO <nX2>,<nY2>]

➔ Posición de la caja. Atentos la cláusula **AT** tiene prioridad sobre **TO**.

➔ **TO** esta especialmente indicado para hacer cajas con sombra sin texto.

[BRUSH <oBrush>]

➔ Opcional. Brush a usar para rellenar el rectángulo.

➔ Si no se especifica , se usara por defecto el Brush seleccionado por el comando SELECT

[PEN <oPen>]

- ➔ Opcional. Pen a usar para la Caja.
- ➔ Si no se especifica , se usara por defecto el Pen seleccionado por el comando SELECT

[ROUND <nZ>,<nZ2>]

- ➔ Si queremos que los bordes sean redondeados.
- ➔ **<nZ>,<nZ2>** Ángulos de la Elipse que forman las esquinas del rectángulo.

[SHADOW [WIDTH <nAncho>]]

- ➔ Si queremos sombra y opcionalmente un tamaño . Por defecto el ancho de la sombra es de 0.25cms

[SHADOWPEN <oPenShadow>]

- ➔ Que pen usaremos para la caja de la sombra.

[EXPANDBOX <nAlto>,<nAncho>]

- ➔ Expande la caja por el Alto y Ancho especificado. Por defecto 0.5 cms por alto y por ancho.

[TITLE]

- ➔ Si queremos que la caja sea centrada con respecto al tamaño de la hoja.

[NOBOX]

- ➔ Le indicamos que no queremos dibujar ninguna caja alrededor del texto .

[LEFT, RIGHT , CENTER]

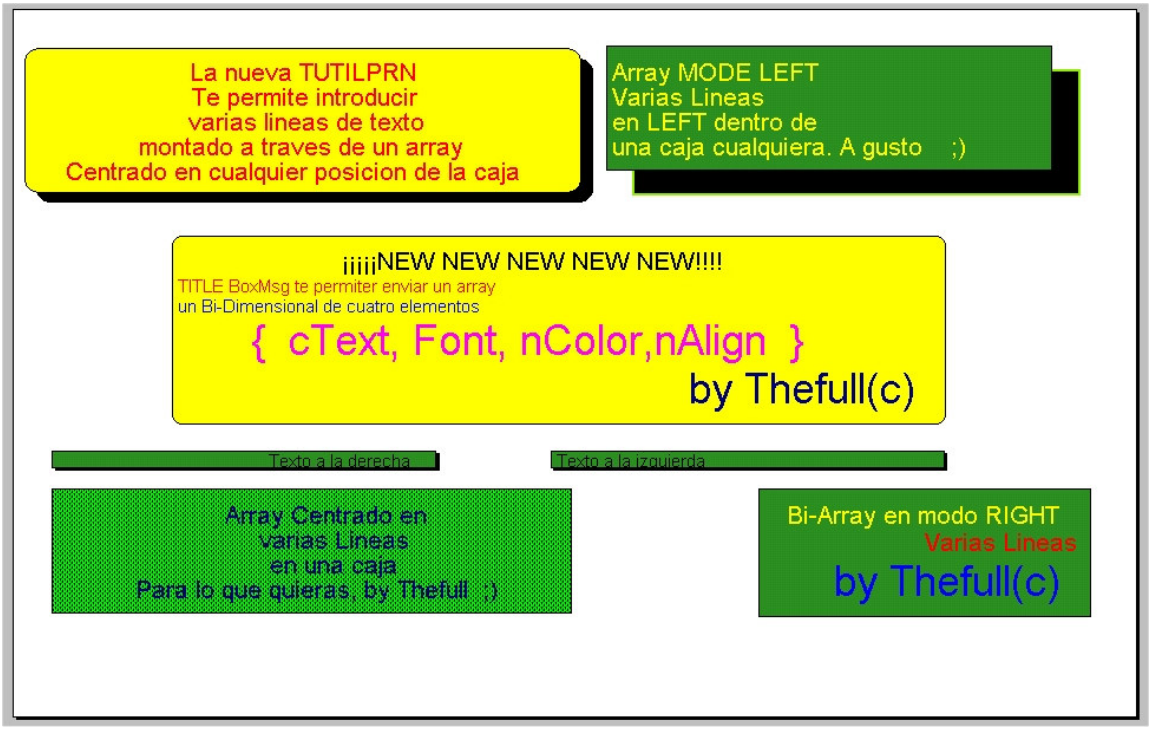
- ➔ Alineación del texto en la caja. Por defecto es centrado , por lo cual no hay ni que especificarlo si lo queremos centrado. Esto afecta también a la cláusula **TITLE.**

Ejemplo: Diferentes MsgBox

```
PAGE
   UTILPRN oUtils ;
        MSG { "La nueva TUTILPRN","Te permite introducir",;
             "varias lineas de texto","montado a traves de un array",;
             "Centrado en cualquier posicion de la caja"} ;
        AT 1.25,1 TEXTFONT oFont  TEXTCOLOR CLR_HRED ;
```

```
        BRUSH oBrush ;
        SHADOW WIDTH 0.2 ;
        EXPANDBOX 0.2,.5 ROUND 75,75

   UTILPRN oUtils MSG {"Array MODE LEFT","Varias Lineas",;
                 "en LEFT dentro de","una caja cualquiera. A gusto    ;)" };
         AT 1.25,11.5 TEXTFONT oFont TEXTCOLOR CLR_YELLOW;
         BRUSH oBrush2 ;
         SHADOW WIDTH 0.5  ;
         SHADOWPEN oPen2 ;
         EXPANDBOX 0.25,.5 LEFT

    UTILPRN oUtils MSG { { "¡¡¡¡¡NEW NEW NEW NEW"+;
                                      NEW!!!!",oFont,CLR_BLACK,NIL },;
                   { "TITLE BoxMsg te permiter enviar un array",oFontDefault ,;
                     CLR_HRED,PAD_LEFT  },;
                   { "un Bi-Dimensional de cuatro elementos" ,oFontDefault ,;
                     CLR_HBLUE,PAD_LEFT },;
                   { "  {  cText, Font, nColor,nAlign  } " ,oFontGrande, ;
                     CLR_HMAGENTA,PAD_CENTER },;
                   { "by Thefull(c)",oFontGrande, CLR_BLUE,PAD_RIGHT } };
         AT 5,4 TEXTFONT oFont TEXTCOLOR CLR_BLUE;
         BRUSH oBrush ROUND ;
         EXPANDBOX 0.25,1 TITLE

   UTILPRN oUtils MSG {"Array Centrado en ","varias Lineas",;
                 "en una caja","Para lo que quieras, by Thefull  ;) " };
         AT 10,2 TEXTFONT oFont TEXTCOLOR CLR_BLUE;
         BRUSH oBrushNubes ;
         EXPANDBOX 0.25,1

    UTILPRN oUtils MSG { { "Bi-Array en modo RIGHT",oFont, CLR_YELLOW, },;
                   { "Varias Lineas"         ,oFont, CLR_HRED , },;
                   { "by Thefull(c)"         ,oFontGrande, CLR_HBLUE, } };
         AT 10,14 TEXTFONT oFont TEXTCOLOR CLR_BLUE;
         BRUSH oBrush2 ;
         EXPANDBOX 0.25,.25 RIGHT

   UTILPRN oUtils SELECT oFontDefault
   UTILPRN oUtils MSG "Texto a la derecha"       AT 9,3 SHADOW WIDTH .05;
                        EXPANDBOX 0,2 RIGHT
   UTILPRN oUtils MSG "Texto a la izquierda"     AT 9,12 SHADOW WIDTH .05;
                         EXPANDBOX 0,2 LEFT
ENDPAGE
```

Como podéis observar en la siguiente imagen , el comando MsgBox nos permite realizar muy diferentes efectos.
(El código de arriba es el resultado de la imagen de abajo)



Otra imagen usando el Comando MSGBOX usando NOBOX para hacer etiquetas.

5.4.4.4.- DATAs y METHODs

La clase TutilPrn se componen de las siguientes DATAS. Observar que he declarado unas DATAs y METHODs como LOCAL, que es lo mismo que HIDDEN, para que no se manipulen directamente , puesto que la modificación de cualquiera de ellas puede provocar que no funcione como debería.

De todas maneras, no nos sirven para nada, ya que son básicamente, para el uso de otros METHODs.

DATA oPrinter
➔ Objeto Printer en el que trabajar

DATA oBrush
➔ Brocha a usar la clase por defecto

DATA oPen
➔ Lápiz a usar la clase por defecto

DATA nColor **INIT** 0
➔ Color de la fuente a usar por defecto

LOCAL:
DATA nAnchoFont, nAltoFont,nAnchoPage
➔ Ancho y Alto de la Font y Ancho de la hoja

DATA cLongText
➔ Este data puede contener el texto mas largo dentro de un array o Texto pasado.

DATA aAltoFonts
➔ Alto de las Fuentes declaradas del array bidimensional. Esto nos será de gran utilidad a la hora de calcular la próxima , nRow, donde colocar la siguiente cadena del array

DATA nPos

➔ Nos informa cual es la posición del array que contiene el ancho mayor que obtendremos. De esta manera , sabremos cual es el ancho del texto para poderlo centrar dentro de la caja.

Los METHODs que componen la clase TUtilPrn son, al igual que las DATAs, algunos PUBLIC, y solamente unos cuantos son LOCAL, puesto que como hemos mencionado anteriormente en las DATAS, simplemente son usados para ayudar a otros METHODs, que fuera de su ámbito, en este caso la clase, no tiene ningún sentido su modificación, y lo único que conseguiremos será un desastre.

Los que contengan su propio comando, nos remitiremos al comando, no vamos a explicar dos veces la misma cosa ;)

Como se puede observar la clase dispone de unos cuantos métodos muy simples.

Lo que es un poco complejo son los métodos CalFonts y TextLines, dado que hay que calcular todo lo relativo a los anchos y altos de las fuentes, etc...

METHOD New(oPrinter, oBrush, oPen) **CONSTRUCTOR**

➔ Mirar comando **DEFINE TUTILPRN**

METHOD Default()

➔ Llamada desde method New. Inicializa oBrush y oPen si no le pasemos a New un objeto pen y un objeto Brush.

➔ Por defecto oPen tiene un ancho de 1.

➔ Por defecto oBrush tiene un style NULL

METHOD Reset() **INLINE (::oPen := NIL,::oBrush := Nil ,::Default())**

➔ Introduce valores por defecto a los objetos oPen y oBrush.

METHOD Sel(oObject, nColor) **SETGET**

➔ Seleccionar un objeto para usarlo por defecto. Mirar comando **SELECT.**

METHOD End() **INLINE (::Reset(),::oPen:End(),::oBrush:End())**
➔ Cogemos nuestros Pen y Brush por si acaso y nos lo cargamos.

METHOD Box(nArriba,nIzq,nAbajo,nDerecha ,oBrush, oPen,lRound,..)
➔ Mirar el comando **BOX**.

METHOD Linea(nArriba, nIzq, nAbajo, nDerecha, oPen)
➔ Mirar el comando **LÍNEA**.

METHOD SayImage(Arriba,nIzq,nAncho,nAlto,xImage,lImage,lPage,...)
➔ Mirar el comando **IMAGE**

METHOD PrintImage(nRow, nCol, xImage, nWidth, nHeight, nRaster)
➔ Este método es usado por SayImage para imprimir ficheros a través de la DLL.

METHOD Poly(aVerters ,nMode, oBrush, oPen)
➔ Mirar comando **POLY**.

METHOD Circle(nArriba,nIzq, nWidth,oBrush,oPen)
➔ Mirar el comando **CIRCLE**.

METHOD _Ellipse(nArriba,nIzq,nAbajo,nDerecha,oBrush,oPen)
➔ Mirar el comando **ELLIPSE**

METHOD Text(cText,nRow,nCol,oFont, nClrText, nBkMode, cPad ,;
blShadow, oBrush, oPen, nAncho, nAlto, cImage)
➔ Mirar el comando **SAY**

METHOD BoxMsg(nArriba,nIzq,nAbajo,nDerecha,oBrush,;
oPen,lRound,nZ,nZ2,;
cText,nRow,nCol,oFont,nClrText,nBkMode,nAlto,;
nAncho ,lShadow,nShadow,;
oBrushShadow, oPenShadow ,lTitle, cPad)
➔ Mirar el comando **MSG**.

LOCAL:
METHOD TextLines(Text,nRow,nCol,oFont,nClrText,nBkMode,;
nLinesArray, nMode,;
nAncho, lTitle)

- Este method es utilizado por BoxMsg para imprimir el texto pasado como array ya sea de una o dos dimensiones. También se encarga de calcular nCol y nRow de acuerdo con las características del array, teniendo en cuenta anchos y altos de las fuentes escogidas.

METHOD AnchoMayor(cText, oFont)

- Calcula un ancho determinado por una cadena de texto y una fuente.

METHOD CalFonts()

- Calcula las DATAS nAnchoFont y nAltoFont para cuando el texto pasado es un array dos dimensiones, introduce los altos de las fuentes en la data aAltoFonts y nos devuelve en que posición, ::nPos, se encuentra el mayor ancho de texto+fuente.

METHOD IsArrayBi(aArray)

- Devuelve .T. si es un array Bidimensional. A sido escrita para facilitar la lectura del código fuente.

5.5.1.-Clase TDatabase.

La clase TDatabase, nos permite manejar el mantenimiento de las bases de datos como un objeto, ganando una mayor rapidez y compactación de código fuente.

(Atención. Hay otra clase, TDbf de Manu Expósito que es mucho más potente que la clase Tdatabase, pero su filosofía es completamente diferente y esto no es aplicable a dicha clase pues su funcionamiento es diferente)

Por ejemplo ,antiguamente hacíamos :

```
Local nVar := 0
Local cNombre := Space(30)

USE CLIENTES INDEX CLIENTES SHARED ALIAS "CLIENTES"

SEEK(cNif)

IF FOUND()
  cNombre := Field->Nombre
  nVar    := Field->Numero
ENDIF

@1,1 SAY "NOMBRE" GET cNombre PICTURE "@!"
@2,1 say "numero" get nVar    PICTURE "99"
READ

IF LASTKEY() # 27
  REPLACE cNombre with Field->Nombre
  ...
ENDIF
```

o bien, podíamos hacer el *GET Field->NOMBRE* , sobre el registro físico.

Ahora bien , con la Clase TDatabase, esto es mucho mas sencillo:

```
USE CLIENTES INDEX CLIENTES SHARED ALIAS "CLIENTES"
DATABASE oClientes      //Inicializamos objeto database después del USE

oClientes:Seek(oClientes:Nif)  //Buscamos en la dbf CLIENTES

IF oClientes:Found()            // Si lo encuentras
  oClientes:Load()     // Cargamos registro actual
ELSE
  oClientes:Blank()    // En blanco
ENDIF

@1,1 SAY "NOMBRE" GET oClientes:Nombre PICTURE "@!"
@2,1 SAY "numero"    GETt oClientes:Numero PICTURE "99"

oClientes:Save()        // Salvamos en contenido del Buffer al registro
```

La rapidez a la hora de usar objetos Databases, se hace evidente.

```
USE CLIENTES INDEX CLIENTES SHARED NEW ALIAS "CLIENT01"
```

Ahora veamos las **DATAs** que componen dichas clases :

cAlias	Nos devuelve el Alias que le hayamos asignado.
cFile	Nos devuelve el nombre del fichero .DBF asociado.
cDriver	Nos devuelve el nombre del Driver asociado.
lShared	Nos devuelve si la Dbf esta compartida.
lReadOnly	Nos devuelve si es solamente de lectura.

bBof Codeblock ha ejecutarse cada vez que la DBF es BOF()
Si os habreis fijado, cuando llegamos a principio del
fichero - 1, nos muestra un mensaje en English.
Para cambiarlo al español , portugues o lo que hablemos:
oCLientes:bBof := {||MsgInfo("Esto es el principio")}

bEof Codeblock ha ejecutarse para cuando se es final del fichero+1.
Cuando realizamos operaciones , como por ejemplo :
While !oCLientes:Eof() //Operaciones End While
y no quereis que salga el dichoso mensaje de fin de fichero,
ponerlo asin:
oClientes:bEof := {|| NIL } // No muestra mensaje

bNetError Lo mismo que bBof y bEof. Para mensajes de error de Red.

nArea El orden del area en uso.

lBuffer Nos informa del estado del buffer, por defecto es .T.
Si lo pones a .F. , estas editando en modo registro fisico
por tanto, el boton que has creado para guardar no te sirve
ni para envolicar el bocadillo de Atun. ;)
Es lo mismo que si hicieras Field->[Nombre del Campo]

aBuffer Un array que contiene los datos temporales de la Edicion.
Imagina que tenemos dos campos:
1 NOMBRE C 20 := 'Pepito los Palotes'
2 NIF N 10 '007'
Cuando haces oDbf:Load() o oDbf:Blank(), lo que hace la clase,
que por cierto es muy lista, crea el array aBuffer y mete
el contenido de los campos en aBuffer:
Por lo tanto, tenemos que :
aBuffer[1] := 'Pepito los Palotes'
aBuffer[2] := '007' , haciendo las modificaciones sobre este
array. Entonces cuando hacemos oDbf:Save(), coge los valores
de aBuffer y los descarga en los campos del Registro donde estes.

lOemAnsi cambia Oem a Ansi.(No todavia no le encuentro utilidad :-))

lTenChars Usado internamente. Ni caso.

aFldNames Array con todos los nombres de los campos.
Cogiendo el ejemplo de aBuffer, tenemos que :
aFldNames[1] := NOMBRE
aFldNames[2] := NIF

Para utilizar cualquier método , es muy fácil :

```
oDatabase:Metodo()
// salta un registro del objeto database llamado oClientes
oClientes:Skip()
```

Una aclaración:

NO MEZCLEIS CODIGO CLIPPER CON METODOS!!!

```
While !Eof()
   // operaciones
   oDbf:Skip()
end while
```

Esto que parece correcto, puede llegar a dar ***un cuelgue total***!!
Lo correcto es:

```
while !oDbf:Eof()
  //Operaciones
  oDbf:Skip()
end while
```

Os preguntareis el porque. La explicación es muy simple. Cuando hacemos uso de los objetos Database, no nos preocupamos que dbf esta activa en ese momento, cosa que si lo teniamos que tener en cuenta cuando programamos como toda la vida, esto es, deberíamos de poner el (Alias)->(DbSkip()) para no tener problemas.
En el primer While !Eof() haciendo uso de la clase Database en el Skip(), puede darse el caso que TU te pienses que la dbf activa es la del objeto, pero eso no tiene porque ser cierto, dando lugar a un bucle infinito.

Observemos el siguiente código:

```
USE CLIENTES NEW ALIAS “Clientes”
DATABASE oClientes
USE FORMAS    NEW ALIAS “Formas”
DATABASE oFormas
```

Como podemos observar la dbf activa es la última que hemos abierto, FORMAS, entonces si codificamos de la siguiente manera el código siguiente:

```
While !Eof()  // MIENTRAS SEA FINAL DE FICHERO DE FORMAS
        // Operaciones
       oClientes:Skip()
End While
```

Nosotros podemos creer, ilusos de nosotros ;), que estamos operando correctamente, pero por mucho que se ejecute el metodo Skip del objeto oClientes, eso no le afecta al Eof() , pues esta comprobando el Eof() de la dbf activa, que es la dbf Formas, creando un bucle infinito , sin más remedio que el cuelgue de nuestro programa.

Por ello, es muy importante, si hacemos uso de los objetos Database, hacerlo de la manera correcta.

La ventaja de los Objetos Database, es tambien a la hora de desarrollar nuestra aplicacion ya que no tenemos que poner el Alias(), osease :

(cAlias)->(DbGoTop()) // Esto es lo correcto en un ambiente de red

poniendo simplemente , oDbf:Gotop(), la clase ya lo hace por nosotros.

METODOS(METHODS)

New()
El llamado Constructor. Es lo mismo que DATABASE oClientes,
lo que pasa que el preprocesado lo cambia a:
oClientes := TDatabase:New()

Activate()
Selecciona la Dbf actual. Osease como si hicieras un Select.

AddIndex()
Añadir un nuevo indice al contenedor. No confundir con crear.

Append()
Añadimos un nuevo registro.

AnsiToOem()
Convertir de Ansi a Oem.

Blank()
Ponemos el buffer de Edicion en Blanco.

Bof()
Pos si estamos al principio.

Close()
Cierra la .Dbf , lo mismo que poner DbCloseArea()

CloseIndex()
Cierra Indices. Lo mismo que OrdListClear()

Commit()
Ah!! Cuantas discusiones sobre este mandato!!!
Y TODAVIA CONTINUAN!! Pos vacia buffers pendientes al
disco duro. No hace falta decir que debes de tener el
MS-DOS Ver 3.3 o Superior. jejejejejej.

Create(cFile, aStruct, cDriver)
Crear una nueva .Dbf. Igual que la funcion DbCreate( ) de Clipper.

CreateIndex(cFile, cTag, cKey, bKey, lUnique)
Crear un indice. Igual que la funcion OrdCreate( ) de Clipper.

ClearRelation()
Cierra una relacion activa.

DeActivate()
Cierra la .Dbf. Lo mismo que DbCloseArea()

DbCreate(aStruc)
Crear una .Dbf con la estructura aStruc. cFile y cDriver es el actualmente en uso.

Delete()
Marca un registro como borrado

Deleted()
Devuelve el estado de borrado del registro actual.
.T. borrado , .F. no borrado.

DeleteIndex(cTag, cFile)
Suprime un orden especificado de un contenedor de órdenes
Mirate la funcion OrdDestroy(cTag, cFile)

Eof()
Determina si es fin de fichero.

Eval()
Evalua un codeblock para los registros scopeados con una condicion

FCount()
Retorna el numero de campos de la .Dbf actual.

FieldGet()
Devuelve el valor del campo.

FieldName(nField)
Devuelve el nombre del campo de la posicion nField.

FieldPos(cField)
Devuelve la posicion del nombre del campo dentro de la estuctura .DBF.
Podriamos decir que seria la inversa de FieldName()

FieldPut(nField, uVal)
Introduce un valor uVal, en la posicion nField.

Found()
Determina si la busqueda previa (Seek()) a tenido exito.

GoTo(nRecno)
Salta al registro nRecno.

GoTop()
Ve al principio del Fichero. Igual que el DbGotop() del Clipper.

GoBottom()
Ve al final del fichero.

IndexKey(ncTag, cFile)
Devuelve la expresion del indice del Indice especificado.

IndexName(nTag, cFile)
Devuelve el nombre de un orden de la lista de órdenes

IndexBagName(nInd)
Devuelve el nombre del contenedor de órdenes de un orden específico

IndexOrder(cTag, cFile)
Devuelve la posición de un orden en la lista de órdenes actual

LastRec(nRec)
Determina el numero de registros de la .Dbf

Load()
Carga en el array aBuffer los valores de los campos del registro actual.

Lock()
Bloqueo del fichero.

Modified()
Si el registro actual a sido modificado respecto a aBuffer.

OemToAnsi()
Traduce de Oem a Ansi

Pack()
Limpieza de la .Dbf de los registros marcados para borrar.

ReCall()

Restaura un fichero borrado como no borrado.

RecCount()
Determina el numero de registros de la .Dbf. Igual que LastRec().

RecLock()
Bloqueo del registro actual.

RecNo()
Devuelve el numero de registro sobre el que estoy posicionado.

Save()
Guardar el contenido de aBuffer en el registro actual.

SetBuffer(lOnOff)
Activar si queremos buffer de edicion o no.

Seek(uExp, lSoft)
Busca una expresion de Indice(uExp), y si quiero busqueda blanda o dura

SetOrder(cnTag, cFile)
Establece el orden del indice.

SetRelation(ncArea, cExp)
Establece una relacion.

Skip(nRecords)
Pos salta nRegistos. (Como el salto del Tigre ;))
Por defecto salta 1.

Skipper(nRecords)
Salta algo. Que alguien me lo explique.

UnLock()
Desbloquea el registro o fichero anteriormente bloqueado.

Used()
Si esta en uso la .Dbf.

Zap()

Borra el contenido de la .Dbf

Si os fijáis , tenéis todas estas funciones en las NG del Harbour
Los valores devueltos por los métodos , son los mismos de los de Harbour.

Hace un tiempo después de escribir este pequeño manual sobre la clase TDatabase, y empecé a comprender como trabajaba la POO, me di cuenta de como era posible acceder a una variable de instancia con el mismo nombre que el campo de la dbf, si yo no le decía nada.

Me estaba dando cuenta que algo realizaba la clase TDatabase para que milagrosamente supiera automáticamente el nombre de mis campos de la dbf.

Entonces en el Foro de Fivewin, me dieron la explicación.
Ojo, que lo que viene a continuación se te puede quedar cara de poker ;)

- ***La clase TDatabase NO TIENE variables de instancias con el nombre de tus campos de la dbf que estas usando, si no que , a través de un 'error controlado' te permite acceder a unas variables de instancias que realmente no existen!!***

Realmente la idea es genial! Independientemente de la dbf a usar como un objeto Database, ella por si misma te permite acceder como si fuera una variable de instancia más, por el nombre del campo de la dbf.

¿ Y como funciona ?

Pues a través del método **OnError()**

```
CLASS TDATABASE ....

   ERROR HANDLER OnError( uParam1 )

ENDCLASS

METHOD OnError( uParam1 ) CLASS TDataBase
  // cMsg es igual al nombre del campo al que intentamos acceder.
  local cMsg   := __GetMessage()
  local nError := If( SubStr( cMsg, 1, 1 ) == "_", 1005, 1004 )
```

```
local nField


// Si lo que vamos a realizar es una asignacion
// El primer caracter nos lo indica "_"+Nombre del Campo
if SubStr( cMsg, 1, 1 ) == "_"
  // Localiza el campo dentro de la dbf
  if( ( nField := ::FieldPos( SubStr( cMsg, 2 ) ) ) != 0 )
  // Si lo has encontrado , modifica por el valor que te paso por uParam1
    ::FieldPut( nField, uParam1 )
  else
    // Si no existe ese campo, dara error.
    _ClsSetError( _GenError( nError, ::ClassName(), SubStr( cMsg, 2 ) ) )
  endif
else
  // Si lo que vamos a realizar va a ser una lectura
  // Localiza el campo dentro de la dbf
  if( ( nField := ::FieldPos( cMsg ) ) != 0 )
    // Si lo encuentras , devuelve el valor contenido.
    return ::FieldGet( nField )
  else
    // Si no existe campo, dara error.
    _ClsSetError( _GenError( nError, ::ClassName(), cMsg ) )
  endif
endif

return nil
```

Por lo tanto, lo que estas viendo aquí, podemos definir que cada vez que se accede a un nombre del campo a través de la clase, esta 'se cae' literalmente para solucionar el problema.

Si no tuviéramos bastantes problemas con las caídas del sistema Windows, ahora te enteras que cuando usas la clase TDatabase, esta va y se va cayendo cada tres por cuatro ;)