GUÍA RÁPIDA DE INICIO AL GUI DEL VISUAL PRO/5

Esta guía muestra algunas, pero no todas, las capacidades gráficas del Visual PRO/5. Se incluyen breves explicaciones, ejemplos, algunas sugerencias y trucos. El término GUI va a continuarse viendo a través de esta guía y su significado se traduce a **I**nterfase **G**ráfica del **U**suario.

El manual trata con conceptos básicos de adentrar al alumno en la programación con Visual PRO/5. Se va viendo ejemplos y se realiza alguna práctica, que de fijo, es la que mejor enseñanza puede dejar al alumno.

Para dar una breve idea de la diferencia entre el PRO/5 y el Visual PRO/5, se puede decir que el PRO/5 es un lenguaje para desarrollo de aplicaciones bajo ambiente de caracteres, de una manera bastante similar a como la hace cualquier versión anterior de BBx, que corre en muchos sistemas operativos, y que sigue siendo poco exigente en cuanto a los requerimientos de memoria y equipo. El Visual PRO/5 es también un lenguaje para desarrollo de aplicaciones pero destinado a correr solo en ambiente MS Windows. Esta característica no quiere decir que no pueda correr programas hechos para ambiente de caracteres. Perfectamente en PRO/5 como en Visual PRO/5 pueden correrse cualquier programa hecho con versiones anteriores de BBx sin necesidad de hacer cambios en la programación. La advertencia que debe hacerse con el Visual PRO/5 es que la máquina en donde se vaya a usar, debe tener buena cantidad de memoria y su propio disco duro, esto no por exigencias del Visual PRO/5, sino del MS Windows.

Se puede decir que en cuanto a verbos, funciones y capacidades, el Visual PRO/5 es casi lo mismo que el PRO/5. Lo que sucede es que el Visual PRO/5 está provisto de un gran juego de mnemónicos y herramientas que son lo que nos permite interactuar con el ambiente GUI. Además de que al funcionar en ambiente MS Windows, nos permite hacer uso de los diferentes ODBC disponibles, para poder accesar las Bases de Datos correspondientes a cada ODBC, por medio del SQL estándar propio de cada base de datos. Esta capacidad de usar SQL nos permite OPCIONALMENTE accesar todos los archivos propios de BBx como si fueran una sola Base de Datos, de manera que el programador no tendría que estar dando OPEN a cada archivo, sino que hace un solo OPEN a la base de datos definida, y a partir de ahí continua manejando los datos con instrucciones SQL, en donde el acceso a los datos es idéntico a como se hace a las tablas de cualquier base de datos.

Con relación a las pantallas gráficas, estas pueden ser confeccionadas de dos maneras: 'a pie' y con el editor de pantallas que se ofrece con el Visual PRO/5.

Decir que migrar aplicaciones para ambiente de caracteres a ambiente gráfico sea fácil, sinceramente, no es así. Quizás programas como actualizaciones, generadores de reportes y algunos otros que no requieran mucha entrada de datos de parte del usuario, si son fáciles de migrar. Lo que son programas para ingreso masivo de datos y de consultas, se puede decir que hay que hacerlos nuevos.

Cuando se hace un programa para ambiente GUI, debemos olvidarnos de la forma en que hacíamos el programa para ambiente de caracteres, el cual generalmente consistía en un grupo de instrucciones que generalmente se ejecutan en el mismo orden en que están escritas dentro del programa en donde el mismo programa es el que controla los eventos que se puedan dar. En una pantalla gráfica, los eventos los genera el usuario del programa, la captura de datos es hecha por Windows y el Visual PRO/5 lo único que debe hacer es controlar los eventos que MS Windows le

reporta. Los datos en una pantalla gráfica están almacenados en los objetos definidos en el contexto de la ventana y Visual PRO/5 lo que hace es tomarlos para la acción que se necesite efectuar.

Contexto es todo el espacio que abarca la pantalla gráfica en su interior. Objetos son todas las figuras que vemos dentro del contexto: (cajas de grupos, botones, radio botones, barras de scroll, botones de lista, cajas de cierre, etc). Eventos son los que se generan cuando el operador del programa ocasiona cuando nueve el ratón o cuando con los botones del ratón hace click sobre alguno de los objetos, o cuando oprime alguna de las teclas del teclado como el ENTER, etc.

Más adelante, en la sección **Interacción SYSGUI** se ofrecen algunos consejos para ayudarle a entender el manejo o control de estos eventos. Dominando esa parte el resto será más fácil.

En lo que siga del manual, las siguientes páginas van a ir siendo discutidas y analizadas con el instructor del curso, además de que se verán algunos ejemplos no cubiertos aquí. Se recomienda antes de asistir a cada clase, tratar de llegar a la lección habiendo leído primero las páginas que supuestamente van a ser vistas. Esa es la forma de sacar el mejor provecho al curso. Aún mejor sería si por aparte el alumno tuviera oportunidad de realizar alguna práctica adicional, sea en su casa o lugar de trabajo.

Se recomienda a los alumnos que frecuentemente hagan un respaldo en disket de las prácticas que van realizando en clase. Recordemos que las computadoras también son utilizadas por otros alumnos, por lo cual y aunque cada uno use directorios diferentes, no tenemos una buena garantía de lo que pueda suceder a toda la información almacenada en los discos duros.

- 0 -

EL DISPOSITIVO SYSWINDOW

El dispositivo SYSWINDOW es un emulador de terminal extendido del Visual PRO/5. Se comporta como una sofisticada terminal de caracteres, respondiendo a un gran grupo de comandos del dispositivo tradicional de terminal del BBx.

El uso más básico de SYSWINDOW es para proveer una ventana en la terminal del Visual PRO/5 pudiendo así iniciar en modo de consola. La línea de alias de abajo, o algo muy similar a ella, es añadida al archivo **config.bbx** automáticamente cuando Visual PRO/5 es instalado.

```
ALIAS T0 SYSWINDOW "Console Terminal"
```

MODOS DE SYSWINDOW

Los modos pueden ser añadidos a la línea de alias para cambiar el comportamiento del SYSWINDOW en diferentes formas. Lo más usado para migrar al GUI son los modos TITLE=, MINIMIZED e INVISIBLE.

El modo TITLE=

El modo TITLE= le permite escoger su propio título de ventana, reemplazando por omisión del título el valor "Visual PRO/5 ". Normalmente usted podrá utilizar el nombre de su aplicación. El string suministrado aparecerá en la barra de títulos de la ventana de consola y también debajo del icono cuando la ventana es minimizada. (También es posible cambiar el título de la ventana en tiempo de corrida con el mnemónico 'TITLE', y cambiar el icono utilizado cuando la ventana es minimizada con el mnemónico 'MINIMICON'. Esto será repasado posteriormente.)

```
ALIAS T0 SYSWINDOW "Terminal Consola" TITLE="Contabilidad".
```

Los modos MINIMIZED e INVISIBLE

El modo MINIMIZED causa que la ventana comience minimizada (exhibida como un icono).

INVISIBLE causa que ésta sea inicialmente invisible. Estas son útiles si su aplicación comenzara con una interfase completamente gráfica. El usuario puede interactuar con las ventanas del GUI para que no se vea confundido o distraído por la ventana de la consola.

¡NOTA! Si la propiedad de "Correr Minimizada" es utilizada para lanzar Visual PRO/5 desde Windows, este se comportará justamente como si el modo MINIMIZED fuera utilizado.

¡NOTA! Un SYSWINDOW no puede comenzar con minimizado e invisible a la vez.

¡NOTA! El dispositivo SYSWINDOW automáticamente saltará si la entrada fuera requerida. Esto significa que si usted intenta probar los modos MINIMIZED o INVISIBLE (o la propiedad de "Correr Minimizada") sin escribir un programa, no funcionará. La ventana aparecerá inicialmente de la manera en que usted la deseó, pero luego inmediatamente saltará hasta aceptar entradas con el cursor en modo READY.

Para probar el modo INVISIBLE (sustituya MINIMIZED si usted lo desea):

Varíe el **config.bbx** así:

```
ALIAS T1 SYSWINDOW "Ventana Invisible" INVISIBLE wait5.pgm:
```

Entre y SALVE este programa con nombre wait5.pgm

```
 5 REM ventana saltará para entrada en 5 segundos
10 WAIT 5
20 END
```

Ejecute el programa así:

```
vpro5 -tT1 wait5.pgm
```

Ausentes de esta sección están varios modos que recorren las preferencias del usuario para el tamaño de la ventana y la posición, y la opción del tipo de letra, etc. Mientras algunas aplicaciones tienen una necesidad genuina de forzar estos parámetros a valores particulares, ellos son dejados generalmente para que el usuario los adecue.

M N E M Ó N I C O S S Y S W I N D O W

El control adicional sobre la terminal es suministrado con los mnemónicos para el SYSWINDOW. Aquí hay algunos de los más útiles para migrar al GUI.

'TITLE'(nuevotítulo$)

Este mnemónico establece el título para el SYSWINDOW en tiempo de corrida.

El título es el texto mostrado en la barra de título de la ventana, y debajo del icono cuando es minimizada. Por omisión, el título que aparece es "Visual PRO/5".

Si usted simplemente quisiera ponerle un determinado nombre a su aplicación, es más simple usar el modo TITLE=, mostrado anteriormente. Sin embargo, si sus necesidades de aplicación son para cambiar el título de la ventana dinámicamente en tiempo de corrida, así es como lo puede hacer.

```
PRINT 'TITLE'("Mi Super aplicación: Menú Principal"),
```

'MINIMIZE' 'HIDE' 'RESTORE' 'SHOW'

Si su aplicación utiliza tanto ventanas del GUI como ventanas de caracteres, usted puede querer minimizar (convertir en icono) u ocultar (hacer invisible) el SYSWINDOW cuando usted cambia otra vez sobre una ventana de GUI. Cuando usted vuelve a cambiar, usted podría querer mostrar (hacer visible) y restaurar (des-minimizando) el SYSWINDOW de nuevo. Esto puede ayudar al usuario a seguir la pista para saber cuál ventana está esperando entrada en un momento.

Estos mnemónicos no toman parámetros. Justamente se imprimen al dispositivo SYSWINDOW para tomar la acción.

```
PRINT 'HIDE',          ; REM hacer la ventana invisible
PRINT 'SHOW',          ; REM hacer la ventana visible nuevamente
PRINT 'MINIMIZE',      ; REM iconificar ventana
PRINT 'RESTORE',       ; REM des-hacer evento de MINIMIZE
```

¡CUIDADO! Como se discutió en la sección con los modos INVISIBLE y MINIMIZE, el SYSWINDOW no permanecerá minimizado o invisible si alguna entrada al usuario estuviera requerida. Por consiguiente, si usted simplemente tecleara las declaraciones de programa anteriores para ejecución inmediata, el comportamiento no será como esperaba. Las acciones de 'HIDE' y 'MINIMIZE' serán ejecutadas, y luego deshechas inmediatamente. Para verificar que el mnemónico 'HIDE' trabaja (sustituya 'MINIMIZE' si usted lo deseara); tratando sería:

```
10 PRINT 'HIDE',
20 WAIT 5
30 END
```

'FOCUS' 'RAISE'(newtitle$)

Estos son también útiles en aplicaciones que utilizan tanto ventanas GUI como de caracteres. Cuando se quiera cambiar a una ventana de caracteres (SYSWINDOW), usted podría querer emitir uno o ambos de estos mnemónicos. 'FOCUS' asegura que subsiguientes entradas de teclado serán dirigidas al SYSWINDOW. 'RAISE' asegura que el SYSWINDOW no esté oculto por cualquier parte de cualquier otra ventana. (esto "eleva" la ventana a la parte superior de la "orden de apilado").

Estos mnemónicos no utilizan parámetros. Son dados justamente a como se muestra para traer el SYSWINDOW al frente y establecer el enfoque de teclado.

```
PRINT 'FOCUS','RAISE'
```

Observe que estas dos acciones son distintas, y puedan ser utilizadas independientemente. Esto es posible (aunque rara vez aconsejable) para una ventana que no está delante para recibir entrada de teclado.

'SETCURSOR'(curid)

El cursor del ratón es normalmente un símbolo de flecha. Usted puede cambiar la apariencia del cursor del ratón mientras está sobre su SYSWINDOW emitiendo este mnemónico. Hay cuatro posibilidades:

0 Flecha (estandar)
1 Cruz (usado para diversas cosas)
2 Barra vertical (para edición de texto)
3 Espera (señal al usuario para indicar que el programa está ocupado).

Por ejemplo, la opción tres puede ser usada durante operaciones con largos archivos.

```
1000 PRINT "Copiando archivo ... favor esperar..."
1010 PRINT 'SETCURSOR'(3), [...]
....
1080 PRINT 'SETCURSOR'(0),
1090 PRINT "Hecho."
```

'MINICON'(filename$, índice)

Por omisión, el icono del Visual PRO/5 es exhibido cuando el SYSWINDOW está minimizado. Para utilizar su propio icono, simplemente envíe este mnemónico al dispositivo SYSWINDOW.

Usted debe especificar el nombre del archivo que contiene el icono. Este puede ser un archivo tipo .ICO, .EXE o .DLL. El parámetro de índice selecciona el icono del archivo (en caso de contener más de uno) a ser utilizado. Al primer icono le corresponde el índice 0, al segundo el índice 1, y así sucesivamente. Los directorios en la variable de entorno PATH son utilizados para la búsqueda del archivo, pero no en los directorios definidos en la variable PREFIX del BBx.

```
>PRINT 'MINICON'("PROGMAN.EXE",1),
```

Nota: El mnemónico 'MINICON' es solamente útil para cambiar el icono que esté exhibido cuando su aplicación está corriendo. Para cambiar un icono deseado el usuario tiene que hacer un 'click' en el inicio de su aplicación, utilizando el Administrador de Programas del Windows, u otras herramientas del tablero principal.

'ASK'(titulo$, iconid, mensaje$, botón1$, [botón2$ [, botón3$]])

El mnemónico 'ASK' puede ser útil por sus características gráficas en Visual PRO/5, especialmente si aún utiliza la revisión 1.0x. Con el, usted puede aprovechar para generar preguntas de diálogo gráfico con el usuario del programa. Aún si usted está utilizando el dispositivo de SYSGUI para crear sus propios diálogos de costumbre, puede encontrar que el mnemónico 'ASK', envía preguntas al dispositivo del SYSWINDOW, satisfaciendo sus necesidades en algunos casos.

Con 'ASK', una ventana de diálogo es exhibida con su correspondiente título, un número de icono opcional, un mensaje de su opción, y uno a tres botones para respuesta del usuario. Cuando el usuario da clicks en un botón, la respuesta que esté programada para ese botón es inyectada en el 'buffer' del teclado, justamente como si el usuario hubiera simplemente mecanografiado la respuesta. El usuario puede también dar 'click' en el cierre de la caja de diálogo, generando un !ERROR=29 en la sentencia PRINT o WRITE. Con la ayuda de un atrape de error, esta viene a ser efectivamente una cuarta posibilidad de respuesta, que puede ser hecha para comportarse idénticamente a una de las otras respuestas, o tratada de otro modo.

El texto en **titulo$** aparece en la barra de título de diálogo. El parámetro **iconid** puede tener cualquiera de las siguientes opciones:

0 no muestra ningún icono
1 muestra un icono de "información (i)"
2 muestra un icono de "advertencia (!)"
3 muestra un icono de "pregunta (?)"
4 muestra un icono de "error fatal (x)"

El texto en **mensaje$** es desplegado dentro del diálogo como el cursor de usuario.

Uno a tres botones en secuencia pueden ser suministrados. El texto en el string del botón suministra tanto la etiqueta de botón como el string de respuesta. Generalmente usted querrá que el string de la etiqueta y respuesta sean diferentes. Esto es hecho para listar el string de respuesta después de una coma en el string del botón. Cualquier caracter en la etiqueta del botón puede ser cambiado a una tecla caliente precediéndola con un ampersand. En el siguiente ejemplo se pretende aclarar esto.

Es un pequeño programa con el que una pregunta Si/No es asumida, con "S" y "N" definidas como teclas calientes. Si el usuario hace un "click" en la caja de cierre, una respuesta de "No" es asumida. Observe la adición de terminadores ($0A$) al final de los string de respuesta, para un procesamiento de entrada fácil. Note también que el "echo" es apagado antes de la variable destinada para obtener la respuesta, y que una coma final es suministrada. La función de los mnemónicos 'EE' y 'BE' es para evitar que los string de respuesta y terminadores de línea se muestren en la pantalla. Este ejemplo también toma ventaja del hecho de que el INPUT imprime cualesquiera ítems que no pueden ser modificados.

```
0010 REM "Ejemplo para 'ASK'
0100 INPUT (0,ERR=0120)'ASK'("Si o No",3,"Favor indicar si o
0100:no.","&Si:S"+$0A$,"&No:N"+$0A$),'EE',A$,'BE',
0110 GOTO 0130
0120 LET A$="N"
0130 PRINT "Respuesta fue ",A$
```

¡CUIDADO! El mnemónico 'ASK' puede generar un !ERROR 29 con la instrucción PRINT o WRITE si el usuario opera la caja de cierre. Asegúrese de controlar este error.

El número 3 que se está usando antes del comentario "Favor indicar si o no." sirve para seleccionar el tipo de icono que automáticamente se muestra en el extremo izquierdo de la ventana cuando es desplegada. Ese puede ser un valor del 0 al 4.

¡NOTA! Probablemente usted querrá el "echo" apagado mientras usted entra el string de respuesta. Asegúrese luego de que el "echo" sea restablecido en todos los casos, aún bajo la condición de error. Para evitar que innecesarios terminadores de línea se muestren en la pantalla, utilice una coma después del mnemónico 'ASK'.

'GETS'(título$, prompt$ [,default$])

El mnemónico 'GETS' provee un conveniente modo gráfico abreviado para indicar al usuario un string de entrada. A diferencia de 'ASK', 'GETS' no siempre servirá como un diálogo creado con el dispositivo de SYSGUI. No obstante, 'GETS' es simple de usar, y puede bastar hasta que otros diálogos sean creados.

El diálogo es mostrado con un título (**titulo$**), una literal con una sugerencia al usuario (**prompt$**), y una caja de edición, que inicialmente siempre está vacía. Sin embargo, si el parámetro opcional (**default$**) es suministrado, la caja de edición contendrá el string indicado al iniciar.

Tres botones de pulsar aparecen debajo de la caja de edición: OK, Cancelar, y Restaurar. El botón OK confirma el texto en la caja de edición como correcto y lo introducirá, agregando un caracter de terminación de línea, en el flujo de entrada del SYSWINDOW, justamente como si el usuario haya

digitado el texto por el teclado. Un 'click' en el botón de Cancelar (o la caja de diálogo) generará un !ERROR 29 con el PRINT or WRITE que envió el mnemónico 'GETS'. Será necesario controlar el error. Tanto el OK como el Cancel (así como la caja de cierre) remueven el diálogo.

El botón de Restaurar causa la reversión del texto en la caja de edición para retornar al texto suministrado en el parámetro **default$**, o para revertir a un estado de espacios si ningún valor inicial fue suministrado en el parámetro **default$**. El botón de Restaurar no remueve el diálogo o introduce cualquier texto.

En el siguiente ejemplo, al usuario se le preguntará un nombre. Si ninguno es digitado, el valor que se asumirá es "Jorge". Si el usuario oprime el botón de Cancelar, la literal "Anónimo" es asumida en su lugar.

```
0010 REM "Ejemplo para 'GETS' (Pedir un nombre)
0100 INPUT (0,ERR=0120)'GETS'("Registrar nombre","Favor de entrar
0100:su nombre","Jorge"),'EE',N$,'BE',
0110 GOTO 0130
0120 LET N$="Anónimo"; REM "Nombre que se asume si dan Cancelar
0130 PRINT "El nombre ingresado fue ",N$
```

¡CUIDADO! El mnemónico 'GETS' puede generar un !ERROR=29 en la instrucción PRINT o WRITE si el usuario opera la caja de cierre o el botón de Cancelar. Asegúrese de controlar este error.

¡NOTA! Usted probablemente quiere que el "echo" esté apagado mientras se digita el string de respuesta. Asegúrese que el "echo" se vuelva a encender en todos los casos, aún bajo la condición de error. Para evitar que innecesarios terminadores de líneas se muestren en la pantalla, utilice una coma después del mnemónico 'GETS'.

'FILEOPEN' (titulo$, path$, "", ext$)
'FILESAVE' (titulo$, path$, nombre_sugerido$, ext$)

Estos mnemónicos son muy útiles en la invocación de diálogos estándar de Abrir y Salvar archivos. Con estos mnemónicos, no solamente obtiene la ventaja de una interfase gráfica estándar para seleccionar archivos, si no que usted también obtiene las capacidades nativas de "browse" a través de los directorios. En máquinas en red, al usuario se le permite conectar nuevos dispositivos de red en el proceso de elegir un nombre de archivo para abrirlo o salvarlo.

Para utilizar 'FILEOPEN', todo lo que usted tiene que hacer es suministrar texto propio para la barra de título del diálogo (prompt$), un path inicial (el usuario puede navegar a otra parte si lo necesitara), una variable que es ignorada (esta es un sostenedor de lugar, para preservar similitudes entre 'FILEOPEN' y 'FILESAVE'), y un nombre de archivo opcional de tres letras "extensión" (ext$). Si usted no desea limitar el diálogo de Apertura del Archivo a cualquier extensión particular de tres letras, suministre un string vacío para ext$. No incluya el período inicial en ext$.

Para utilizar 'FILESAVE', suministre su texto para la barra de título (titulo$), un path inicial (el usuario puede navegar), un nombre de archivo sugerido para presentar al usuario como un valor por omisión (nombre_sugerido$), y una "extensión" opcional del nombre del archivo de tres letras (ext$). Si usted no deseara restringir qué tipos de archivo el usuario puede ver , suministre un string vacío para ext$. Al igual que con 'FILEOPEN', no incluya el punto inicial en ext$.

Tanto para 'FILEOPEN' como 'FILESAVE', una vez que el diálogo es desplegado, permanece activo hasta que el usuario haga una selección o 'click' en Cancelar (o la caja de cierre). Si el usuario hace una selección, el path completo del nombre del archivo seleccionado es introducido en el flujo de entrada del teclado, con un terminador de línea al final, de modo que una posterior instrucción de INPUT puede leerla fácilmente. Si el usuario hace 'click' para Cancelar (o en la caja de cierre), el texto "**::CANCEL::**", más un terminador de línea es introducido. Si un error ocurriera durante el procesamiento (muy raro), el texto "**::BAD::**" es introducido.

Ninguno de estos mnemónicos realmente causan aperturas de archivos o cambios en los directorios. Permiten simplemente una cierta cantidad de navegación y exploración, y retornan el nombre del path y archivo elegido al programa de llamada. Hay una excepción: si el usuario escoge un archivo que ya existe desde el diálogo File Save, una posterior caja de diálogo aparecerá, y el usuario tiene que confirmar el remplazo de ese fichero. Si el usuario hace 'click' en "No", el diálogo File Save continúa activo. Si el usuario hace 'click' en "Si", el diálogo File Save es removido y el nombre seleccionado es introducido en el buffer del teclado para su lectura. EL ARCHIVO NO ES REALMENTE SOBREESCRITO. Esto es simplemente asumiendo lo que su aplicación hará en este punto.

En el siguiente ejemplo, el usuario puede escoger el siguiente programa para correrlo desde el diálogo gráfico File Open.

```
0010 REM "Ejemplo para 'FILEOPEN' (Iniciar corrida de un programa)
0100 INPUT 'FILEOPEN'("Correr Siguiente",DSK("")+DIR(""),"","",
0100:'EE',F$,'BE',
0110 IF F$<>"::BAD::" AND F$<>"::CANCEL::" THEN RUN F$
```

¡NOTA! EL tercer parámetro de 'FILEOPEN' no es utilizado -- pasa justamente un string vacío. Usted puede querer apagar el 'echo' antes de leer el string de respuesta, y usar una coma al final, después del mnemónico, para conservar texto extra para el despliegue.

'MOUSE'(col, fila, cols, filas,[,simple_resp$ [,doble_resp$]])
'AMOUSE' (col, fila, cols, filas,[,simple_resp$ [,doble_resp$]])

Estos mnemónicos son de ayuda para aplicaciones en ambiente de caracteres. Lo que hacen es sensibilizar una región del área de acción de despliegue. Nos referimos a estas regiones sensibilizadas como "área caliente del ratón". Si el botón del ratón es presionado y mantenido, y el cursor del ratón es arrastrado sobre el SYSWINDOW, el área caliente del ratón iluminará el cursor del ratón cuando pase sobre ella. Si el botón de ratón es oprimido una o dos veces sobre unas de estas áreas, un string de respuesta es introducido en el buffer del teclado, justamente como si el usuario lo hubiera mecanografiado.

La aplicación más útil para 'MOUSE' y 'AMOUSE' es en la confección de menús, ya que es generalmente simple disponer el ratón para operar el menú tan rápidamente como el teclado. El string de respuesta pueden ser preparado a las teclas del menú, de modo que el único cambio para el código es la adición de un mnemónico 'MOUSE' o 'AMOUSE' dentro del loop que presenta los ítems del menú.

La diferencia entre 'MOUSE' y 'AMOUSE' está en la naturaleza del área caliente del ratón así creado. 'MOUSE' crea un área caliente para el ratón que son anexadas a un 'WINDOW' o pantalla

virtual de caracter particular. Si la ventana es movida, el área caliente del ratón también se mueve. Si la ventana es destruida, el área caliente del ratón desaparece. También, si una región incluyendo el área caliente del ratón es corrida o limpiada, el área caliente del ratón desaparecerá. Areas calientes del ratón de 'MOUSE' son solamente activas cuando la ventana que las contiene está en uso actual. Areas calientes del ratón de 'AMOUSE' son anexadas para todo lo desplegado en el SYSWINDOW, y perduran independientemente de cualquier creación y destrucción de ventana, limpieza o corrimiento por "scroll".

Para crear un área caliente del ratón, suministre una columna y fila para la esquina superior izquierda, más el ancho en cols y la altura en filas. Finalmente, especifique al menos un string de respuesta. El primer string de respuesta es introducido en un simple 'click', el segundo en un doble-click.

Para limpiar una o más áreas calientes del ratón, emita un mnemónico de llamada sin strings de respuesta. Toda área caliente del ratón intersectando la región especificada será limpiada. 'MOUSE'(0,0,0,0) y 'AMOUSE' (0,0,0,0) son formas rápidas para limpiar toda área caliente del ratón. Areas calientes del ratón 'MOUSE' (no 'AMOUSE') también pueden ser limpiadas simplemente limpiando la ventana.

Poniendo metacaracteres especiales en el string de respuesta, es posible detectar que botón del ratón es operado, así como la condición de las teclas de control y conmutación. La metasintaxis es como sigue:

Dentro de un string de respuesta,	esto... es sustituido con..
%%	un simple %
%b	0, 1 o 2 (botón ID)
%c	0 o 1 (control de estado)
%s	0 o 1 (shift estatus)

¡CUIDADO! 'MOUSE' y 'AMOUSE' fueron creados para servir a diferentes necesidades. 'MOUSE' es implementado como un nivel más alto de 'AMOUSE'. Utilizándolos juntos en el mismo dispositivo de SYSWINDOW al mismo tiempo puede provocar resultados impredecibles.

¡CUIDADO! Si usted quisiera responder a cada 'click' del ratón, asegúrese de especificar un string de respuesta para el doble-click. Siempre que el usuario haga 'click' muy rápidamente será suficiente para generar un doble-click, el string de respuesta para el doble-click será introducido. Si ninguno fuera suministrado, ningún string será introducido por aquellos 'clicks' que seán considerados (por Windows) para ser el segundo de un par de doble-clicks.

El siguiente es ejemplo de un programa que utiliza el mnemónico 'MOUSE' con un programa que no necesariamente corre en una pantalla gráfica:

```
0010 REM "Demo para uso del mnemonico MOUSE
0020 BEGIN
0030 LET var$=var$+" 0= Salir     ",var=LEN(var$)
0040 LET var$=var$+" 1= Incluir   "
0050 LET var$=var$+" 2= Excluir   "
0060 LET var$=var$+" 3= Cambiar   "
0070 LET var$=var$+" 4= Consultar "
0080 LET var$=var$+" 5= Listar    "
0090 REM
0100 PRINT 'WINDOW'(50,2,17,8,"< Opciones >"),'CS',
0110 FOR a=1 TO LEN(var$) STEP var; LET lin=INT(a/var),
0110:opc$=var$(a,var);
```

```
0110:PRINT @(0,lin),opc$,'MOUSE'(0,lin,var,1,opc$),; NEXT a
0120 REM
0130 INPUT (0,SIZ=var)'CH',opc$,
0140 PRINT 'POP','LF',"Respuesta fue ",opc$
0150 REM
```

'PLAYSOUND' (filename$, opción)

Este mnemónico sirve para darle el atractivo a su aplicación para un ambiente con Multimedia, ya que usted puede emitir sonidos mediante el uso de los archivos .WAV. El mnemónico 'PLAYSOUND' puede ser utilizado para interpretar o detener archivos de juegos de sonido en diversos modos. Por supuesto, esto solamente trabajará si una tarjeta de sonidos está instalada en su máquina. (Cuando algún sonido no pueda ser interpretado, no será dado error alguno.)

Para comenzar a interpretar un sonido, emite un mnemónico 'PLAYSOUND' con el nombre del archivo **filename$**, y un valor para el parámetro de **opción**, como sigue.

0 interpretar normalmente
1 interpretar asincrónicamente (llamada regresa antes de que el sonido es hecho)
2 interpretar continuamente (girado, y asincrónico)

Para detener un sonido que es interpretado asincrónicamente, utilice un string vacío ("") para **filename$**, y cualquiera valor para el parámetro de **opción**. Ejemplo:

```
PRINT 'PLAYSOUND'("/windows/media/logoff.wav",1)
```

Ausentes de la lista de mnemónicos anteriores son algunos que recorren preferencias de usuario para tamaño y posición de ventana, opción de fuente, etc. Mientras algunas aplicaciones tienen una necesidad genuina de forzar estos parámetros a valores particulares, generalmente es mejor dejar estas posibilidades al usuario.

MSGBOX( ) Función para Crear una Caja de Diálogo para dar Mensajes

Esta función se implementó hasta después de la revisión 1.0x de Visual PRO/5 y con todo lo que ofrece, realmente se convierte en algo más atractivo de usar que el mnemónico 'ASK'. MSGBOX es una belleza de función MSGBOX, ya que crea una caja de diálogo para mensajes y retorna un valor que identifica el botón seleccionado por el usuario.

Sintaxis: **`MSGBOX(str1{,expr}{,str2}{,ERR=lineref})`**

Descripción de parámetros

Parámetro	Descripción
str1	String con la expresión para el mensaje a ser mostrado en la caja de diálogo.
expr	Expresión numérica para el número y tipo de botón, tipo de ícono, botón a asumir por omisión (default), y tipo de diálogo modal (vea abajo los valores a ser usados).
str2	String con expresión para el título de la caja de diálogo. Si este parámetro es

omitido, el nombre del programa es usado como título.

ERR=lineref Número de línea o etiqueta a donde el proceso bifurcara si un error ocurre durante la ejecución.

Todas las siguientes tablas contienen las opciones de valor para el parámetro **expr**. Para crear un valor final, seleccione un número de cada conjunto. El valor que se asume por omisión en cada conjunto es 0.

Valores para Botones

Valor	Botón(es) a desplegar
0	**OK**
1	**OK** y **Cancel**
2	**Abort**, **Retry**, e **Ignore**.
3	**Yes**, **No**, y **Cancel**.
4	**Yes** y **No**.
5	**Retry** y **Cancel**.

Valores para Iconos

Icono	Valor desplegado
0	Ninguno.
16	Signo Stop.
32	Signo de Interrogación.
48	Signo de Admiración.
64	Símbolo de Información.

La posición del ícono en la caja del mensaje es determinada por el sistema operativo Windows y podría cambiar.

Valores por omisión para los Botones del MSGBOX

Si dentro de la expresión numérica ponemos uno de los siguientes valores como tercer parámetro, lograremos que el botón indicado aparezca como seleccionado:

Valor	Botón a ser seleccionado
0	Primero
256	Segundo
512	Tercero

Valores de Modos

Valor	Descripción
0	Aplicación modal. Causará que el usuario tenga que responder a la caja con el mensaje, antes de continuar trabajando con la aplicación actual.
26144	Sistema modal. Requiere que el usuario responda a la caja de mensaje antes de continuar trabajando en alguna aplicación. Este valor solamente puede ser establecido cuando el usuario responda inmediatamente a un mensaje que toma prioridad sobre cualquier otra actividad.

MSGBOX() automáticamente parte las líneas de mensajes en el borde derecho de la caja de diálogo. Para forzar la ruptura de una línea, inserte un delimitador de línea con el hexadecimal ($0A$) antes del primer carácter de lo que sería la nueva línea. Ejemplo:

m=MSGBOX(“El formato de la fecha es incorrecto”+$0A$+”Utilice el formato DD/MM/AAAA para digitarla”,16,””)

Los Valores retornados por el MSGBOX

El valor retornado por MSGBOX() identifica el botón que el usuario seleccionó, como sigue:

Valor	Botón Seleccionado
1	OK
2	Cancel
3	Abort
4	Retry
6	Yes
7	No

Ejemplos

Ejemplo 1
El siguiente crea un diálogo que despliega "Esto es un mensaje" y un OK como botón. Además, utilizará el nombre del programa como título de la caja y retornará un valor de 1.

```
LET x=MSGBOX("Esto es un mensaje")
```

Ejemplo 2
El siguiente crea un diálogo que muestra el contenido de MENSAJE$ como el texto de mensaje; que contiene los Botones Cancel y Retry y un ícono con el signo de interrogación; y despliega el contenido de TITULO$ como el título de diálogo.

```
LET x=MSGBOX(mensaje$,5+32+256,titulo$)
```

Seleccionando el botón Cancel (valor por omisión) retorna un valor de 2.
Seleccionando el Botón Retry retorna un valor de 4.

Otros ejemplos:

LET m=MSGBOX(“Desea cancelar la impresión”,4+32,”Cancelar”)
LET m=MSGBOX(“La impresora no está lista”,16,”Error”)

USO DEL FIN( ) DEL SYSWINDOW

Al igual que con cualquier otro dispositivo, la función FIN() regresa información útil relativa a la condición de ese dispositivo. Siguientes son algunos valores regresados por la función FIN() en un canal SYSWINDOW que podría ser usado cuando migre al GUI.

FIN(syswindow)

Con esta función normal FIN() obtenemos información de cualquier dispositivo de terminal. De interés particular son los campos {**maxcols**} y {**maxrows**}. Estos pueden ser cambiados cuando el SYSWINDOW es cambiado de tamaño por el usuario. (Esto solamente ocurre si la caja "Lock Cols/Rows" en el diálogo de Fuentes/Tamaño no está marcada o chequeada).

El siguiente ejemplo nos trae e imprime las dimensiones actuales del SYSWINDOW abierto al canal 0.

```
0010 DIM F$:TMPL(0,IND=0)
0020 F$=FIN(0)
0030 PRINT "Cols",F.MAXCOLS,",  Filas",F.MAXROWS
>run
Cols 80, Filas 25
```

FIN(syswindow,IND=1)

Esta función FIN() nos retorna información que es útil para aplicaciones que necesitan seguir el movimiento y actividad del botón del ratón mientras está sobre el área de despliegue del SYSWINDOW. Esta es una rara necesidad, pero puede ser utilizada para algunos efectos interesantes, como con un texto de varias líneas {drag-and-drop}.

El siguiente ejemplo nos informa si el botón izquierdo del ratón está oprimido, y donde el cursor del ratón está localizado (o ha sido localizado) dentro del SYSWINDOW. Ejecútelo algunas veces, moviendo el ratón y oprimiendo a veces el botón izquierdo del ratón, para notar el efecto completo.

```
0001 PRINT 'CS',"Para apreciar los movimientos del MOUSE: muévalo,
0001:oprimiendo y soltando el botón"
0010 DIM F$:TMPL(0,IND=1)
0020 F$=FIN(0,IND=1)
0030 PRINT "El botón del ratón está ",
0040 IF AND(BIN(F.MOUSE_BUTTONS,1),$01$)=$01$ THEN PRINT "oprimido"
0040:ELSE PRINT "sin oprimir"
0050 PRINT"Cursor en col",F.MOUSE_TERM_COL," lín.",F.MOUSE_TERM_ROW
0060 WAIT 1; GOTO 20
```

Múltiples dispositivos SYSWINDOWS

Visual PRO/5 inicia cada sesión en el canal 0 abriendo un SYSWINDOW (ej: "T0"). Sin embargo, es posible abrir más dispositivos SYSWINDOW en otros canales (ej: "T1"). Esto puede ser útil para el despliegue de texto adicional. En lugar de subdividir la terminal, usted puede crear otra terminal en una ventana gráfica separada, e imprimir texto ahí también. Así mismo, es igualmente posible obtener entrada de datos desde un SYSWINDOW que no sea el canal 0.

Hay muy pocas diferencias entre una "consola" SYSWINDOW (canal 0, apertura implícita) y un SYSWINDOW "no_consola" (otro canal que no es 0, apertura explícita). Básicamente, son estas:

Ventanas de consola siempre tienen la misma barra de menús del Visual PRO/5.

Ventanas de no_consola pueden tener un menú simplificado (Edición e Impresión solamente) o ningún menú, dependiendo esto del modo de MENU.

Ventanas de consola tiene un diálogo de fuentes / tamaño.
Ventanas de no_consola solo le pueden ser asignadas nuevas fuentes y dimensiones del programa o los modos.

Todos los SYSWINDOWS responden igualmente a los cambios del Mapeador de Color.
Ventanas de no_consola tienen el 'echo' puesto en 'off' desde el inicio, o sea, no se ve lo que uno escribe.

Aparte de todas ésas diferencias, un SYSWINDOW es tan bueno como cualquier otro. Usted puede utilizar esa capacidad para expandir su uso a una pantalla gráfica, sin tener que hacer elaboradas mejoras gráficas.

Para abrir un SYSWINDOW adicional, usted necesitará una línea de ALIAS extra en su archivo **config.bbx**. Luego solo abra el dispositivo por su nombre. En este caso, los modos pueden ser indispensables para poder ver la ventana y actuar de la forma en que usted lo quiera.

¡CUIDADO! Si dos diferentes tareas de Visual PRO/5 abren el mismo alias de SYSWINDOW, en cada una se creará su propia ventana para el dispositivo del alias. No hay forma de obtener múltiples tareas de Visual PRO/5 que abran la misma ventana al mismo tiempo.

Este es un ejemplo trivial, en el que su **config.bbx** deberá contener las siguientes líneas:

```
ALIAS T0 SYSWINDOW "Ventana principal"
ALIAS T1 SYSWINDOW "Ventana extra"
```

Declaraciones necesarias en el programa para hacer un despliegue en el SYSWINDOW T1:

```
OPEN (1)"T1"
```

Cuando el dispositivo es abierto, una ventana es creada, con la fuente y tamaño de fuente que se asumen por omisión, en el tamaño de valor por omisión (80x25). Para obtener otra fuente o atributos de tamaño, tendrá que utilizar modos o mnemónicos.

Modos adicionales de SYSWINDOW

Estos modos de SYSWINDOW son más útiles para abrir SYSWINDOWS en otro que canal que no sea el 0, como se describió en la sección anterior. No fueron mencionados anteriormente, ya que la adecuación de la ventana de la consola es delegada generalmente al usuario. Sin embargo, todo lo del modo "MENU" aplica igualmente al modo de consola SYSWINDOW. Los siguientes parámetros para la adecuación de cada SYSWINDOW, **se definen en la línea de alias del SYSWINDOW de interés en el config.bbx.**

XPOS= YPOS=

Estos modos determinan la ubicación de la esquina superior izquierda del SYSWINDOW. La unidad está en pixeles.

¡CUIDADO! Esta es la esquina superior izquierda del *área de cliente* (la parte que mantiene texto) la cual está siendo posicionada. Esto significa que XPOS=0, YPOS=0 colocarán la ventana de modo que la barra de título no puede ser vista.

COLS= ROWS=

Estos son usados para definir el tamaño de la ventana. Especifican la cantidad de filas y columnas deseadas. Dependiendo de la fuente en uso, el tamaño real de la ventana podría variar.

FONT= FONTSIZE=

Estos son utilizados para especificar la fuente a utilizar con el SYSWINDOW. Los valores no son portables. Solamente fuentes de punto fijo (fixed-pitch) pueden ser usadas. El tamaño es especificado en puntos.

```
LOCKMODE=Normal
LOCKMODE=VARYFONTS
LOCKMODE=SCROLLBARS
```

Utilice uno de estos para determinar el comportamiento del SYSWINDOW cuando este sea cambiado de tamaño. LOCKMODE=NORMAL es el valor que se asume por omisión. Este trabaja de la misma forma que el botón de chequeo {checkbox} y los radio botones {radio buttons} en el diálogo de Fuentes/Tamaño, excepto que estos pueden trabajar con SYSWINDOWS de no consola (las cuales no tienen un diálogo de Fuentes/Tamaño) también.

MENU

SYSWINDOWS de no consola normalmente no tienen una barra de menú. Si este modo es especificado, una barra de menú es provista, la cual contiene los menús de Edición e Impresión.

INVERT

Este causa la “Inversión de Luz/Oscuridad“ en un menú con ítems marcables en el menú de impresión a ser inicialmente marcado.

MAXIMIZED

Causa que el SYSWINDOW inicialmente sea maximizado. Observe que el tamaño es aún significativo, ya que regresará a ese tamaño cuando sea restaurado.

Este es un mejor ejemplo en su **config.bbx**, de un SYSWINDOW que no sea de consola:

```
ALIAS T1 SYSWINDOW "Ventana extra" title="Monitor de trabajo",XPOS=100,
YPOS=100, COLS=40, ROWS=10, FONT="Courier", FONTSIZE=10, MENU
```

Mnemónicos adicionales para SYSWINDOW

Estos mnemónicos no fueron mencionados anteriormente, ya que la adecuación de la ventana de consola es generalmente delegada al usuario. Sin embargo, aplican igualmente a la consola SYSWINDOW.

‘MAXIMIZE’

Este maximiza la ventana. Un ‘RESTORE’ la vuelve a minimizar. Otro uso para 'MAXIMIZE' es para restaurar el SYSWINDOW luego de haberlo minimizado. Ejemplo:

```
0010 REM "Ejemplo para 'MAXIMIZE'
0020 BEGIN
0030 PRINT 'MINIMIZE',
0040 WAIT 3
0050 PRINT 'MAXIMIZE'
```

‘FONT’(nombrefuente$,tamaño)

El mnemónico ‘FONT’ cambia el nombre de la fuente familiar (nombrefuente$) y fija el tamaño (tamaño). El nombre de la fuente no es transportable, y tiene que representar una fuente de punto fijo (fixed-pitch) disponible en el sistema. En la mayoría de los sistemas Windows, “Courier“, “Courier New“, y “Fixedsys” trabajarán. Vea el botón de lista en el diálogo de Fuentes/Tamaño. Ahí se muestran todas las fuentes que pueden trabajar con el mnemónico ‘FONT’.

'SIZE'(cols,filas)

Utilice este mnemónico para forzar un SYSWINDOW a tener un tamaño dado. El comportamiento puede variar dependendiendo del valor del modo LOCKMODE (visto anteriormente). Las siguientes líneas dan una idea de los efectos del 'FONT' y el 'SIZE':

```
0010 REM "Ejemplo para 'FONT' y 'SIZE'
0020 PRINT 'FONT'("BBX",23),'CS',"BBX #23"; WAIT 3
0030 PRINT 'SIZE'(40,10),'LF',"Cambio a 40x10"; WAIT 3
0040 PRINT 'SIZE'(40,15),'LF',"Cambio a 40x15",'LF'; WAIT 3
0050 PRINT 'CS'
0060 PRINT 'FONT'("Courier",10),'LF',"Courier #10 en 40x15";WAIT 3
0070 PRINT 'SIZE'(60,20),'LF',"Cambio a 60x20",'LF'; WAIT 3
0080 PRINT 'FONT'("Courier New",12),'LF',"Courier New #12"; WAIT 3
0090 PRINT 'FONT'("Courier New",15),"Courier New #15"
0100 PRINT 'FONT'("BBX",15),'LF',"BBX #15"
0110 PRINT 'SIZE'(80,25),'LF',"Restaura a 80x25"
```

No se confunda los parámetros arriba mostrados para 'FONT' y 'SIZE', ya que para su uso con las pantallas GUI llevan más parámetros.

El dispositivo SYSPRINT

Visual PRO/5 puede imprimir ya sea directamente a un puerto de LPT (el cual puede dirigir la salida a otra parte si el puerto está proyectado a una impresora de red) o a través del Print Manager del Windows. El hecho de imprimir directamente al puerto, permite control preciso sobre la impresora que exactamente sea usada, así como de los códigos y caracteres que van a la impresora. Imprimiendo por medio del Print Manager del Windows presenta al programador un reporte con un dispositivo impresor uniforme con un conjunto mínimo de capacidades, y permite al usuario (a través del panel de control o diversos diálogos estándar) seleccionar qué impresora, disposición de página, etc. va a usar.

El dispositivo SYSPRINT representa un importante paso en el camino al GUI porque se comporta mucho como un dispositivo impresor ordinario, no obstante, atado al sistema del GUI, permite al usuario utilizar diálogos gráficos estándar para cambiar parámetros de la impresión. El alias de SYSPRINT es integro o directo.

```
alias PD sysprint "Print Manager" font=Courier,cols=80,rows=62,PREVIEW
alias LP /dev/prn "Epson" CR,CP=0F,SP=12,CPCOLS=132,SPCOLS=80,EPON=0E,
                                          EPOFF=14,EPCOLS=2,EPLINES=1
```

Cuando el canal es abierto con "PD", es hecho un intento para establecer la impresión de un contexto gráfico. Si esto fallara, un !ERROR=13 es reportado con el OPEN.

Modos de SYSPRINT

Hay algunos modos que afectan el comportamiento de SYSPRINT. Por omisión, un OPEN al SYSPRINT causa que se use la impresora de valor por omisión con las posiciones de valor por omisión sea abierta (configurada en el Panel de Control).

DIALOG

Con el modo DIALOG, el OPEN causa que el diálogo de impresoras estándar del Windows aparezca. Si el usuario hace un 'click' para Cancelar, un !ERROR=13 es reportado. De otra manera, el canal es abierto a cualquier impresora que el usuario seleccione. La disposición de página, calidad de impresión y otros atributos pueden también ser seleccionados en ese momento.

Ejemplo: `OPEN (1,MODE="DIALOG")"PD"`

SETUP

Este es similar al modo DIALOG, excepto que el diálogo del Print Setup... es mostrado en lugar del diálogo Print ...

PREVIEW

El parámetro PREVIEW en la línea de alias de la impresora, nos provee la facilidad de que todo reporte que sea enviado hacia esa impresora, primero sea cargado "en pantalla", para que desde ahí lo podamos visualizar, de una manera tal que podemos viajar a través de todo el reporte y poder decidir al final si lo imprimimos todo o solo las páginas que nos interese o si del todo desechamos la impresión. Solo está disponible a partir de la revisión 2.0x.

JOBID=

Imprime los trabajos presentados en el Print Manager del Windows siempre que tengan un nombre. Este nombre aparece en las listas de cola del Print Manager. Por omisión, un trabajo de SYSPRINT tiene el nombre "Visual PRO/5 Documento", pero usted podría llamarlo a como guste con este modo.

FONT=DEVICE
FONT=SYSTEM

Con FONT=DEVICE (el valor por omisión), el dispositivo de SYSPRINT intentará localizar una fuente de dimensiones aceptables que sea nativa para la impresora y así puedan imprimir rápidamente y que se mire bien. Con FONT=SYSTEM, un tipo de letra (tipo verdadero) de la

fuente siempre es usado, dando más flexibilidad pero velocidades de impresión más lentas. Al igual que con la mayoría de los detalles de esta guía, no se pretende que este tema sea del todo entendido. Consulte el manual para más detalle.

El dispositivo SYSPLOT

Si usted tiene software que utilice un dispositivo tradicional de ploteo de BBx (nombre del alias inicia con una letra D), usted puede lograr un GUI instantáneo actualizado ofreciendo la opción de plotear una ventana gráfica. Lograr esto es tan fácil como añadir una línea de alias en el **config.bbx** como la siguiente:

```
ALIAS D0 SYSPLOT
```

Cuando el canal es abierto, una ventana gráfica es creada para representar la superficie que puede ser trazada. Permanece visible hasta que el canal sea cerrado.

Modos de SYSPLOT

Para ejercer control de refinamiento sobre el área de ploteo, usted puede suministrar los siguientes modos opcionales en el ALIAS que corresponda, en el **config.bbx**.

XPOS= YPOS=

Estos modos controlan la ubicación de la ventana de ploteo en pixeles. El punto que se posiciona es la esquina superior izquierda del área del cliente (superficie para ploteo). Por lo tanto, suministrando XPOS=0, YPOS=0 dejará la barra de título como 'apagada'.

WIDTH= HEIGHT=

Estos suministran el ancho y altura de la ventana de ploteo en pixeles.

Un ejemplo más elaborado de un alias de SYSPLOT sería el siguiente.

```
ALIAS D0 SYSPLOT "Ventana de Ploteo" XPOS=100,YPOS=100,
      WIDTH=400, HEIGHT=300
```

El dispositivo SYSGUI

El dispositivo SYSGUI es una poderosa herramienta gráfica que permite a un programa de Visual PRO/5 crear y manipular elaboradas ventanas y diálogos gráficos. Este es el que más utilizaremos en lo que siga del curso y en el trabajo real.
Para abrir el dispositivo SYSGUI, usted necesitará un alias como el siguiente en su archivo **config.bbx**

```
ALIAS X0 SYSGUI
```

El nombre de alias tiene que comenzar con X, pero no necesariamente tiene que ser X0. Ya que nada responde realmente a él, X0 sería como cualquier otro nombre. Para preparar el comienzo interactivo con SYSGUI, simplemente abra el dispositivo así:

```
GUI=UNT
OPEN(GUI)"X0"
```

Ninguna ventana gráfica se creará hasta que los comandos hayan sido enviados al canal SYSGUI.

¡CUIDADO! Es posible tener varios canales abiertos al SYSGUI, pero los dos canales se comportarán como uno sólo. Esto es, que algunos modos cambian u otras definiciones afectarán todos los canales de SYSGUI igualmente abiertos.

Cuando el último canal abierto al SYSGUI es cerrado, todas las ventanas de SYSGUI y sus contenidos son destruidos automáticamente.

Interacción SYSGUI

Toda la interacción con SYSGUI corresponde a una de tres categorías.

1. Mnemónicos para acción inmediata. Si una acción inmediata es deseada, como por ejemplo crear una ventana o deshabilitar un control (¡nombrado MUY poco!), un mnemónico es utilizado.
2. El CTRL() y FIN() para examinar componentes del sistema GUI. Si su programa necesita conocer el estatus actual de un control o ventana, u otra condición del sistema como la resolución desplegada, información de la impresora, etc., CTRL () y FIN() son las funciones a ser usadas.
3. READ RECORD para recibir el informe de los eventos del usuario. Cuando el usuario opera un control gráfico o de otra manera interactúa con el sistema GUI, un informe de las acciones del usuario, llamadas un *evento*, puede ser generado y colocado en una cola (la *cola de eventos*).

   Cualquier programa completo que utiliza una interfase gráfica usará todos los tres métodos mostrados arriba. Como un ejemplo típico, un diálogo para seleccionar preferencias de usuario será mostrado con mnemónicos, y luego, un READ RECORD en un loop esperará a que el botón OK sea tocado con un click del ratón. En ese punto, CTRL () examinaría la condición de las diversas cajas de chequeo (check boxes), botones de radio (radio buttons) y barras de desplazamiento (scroll bars), para determinar qué posiciones el usuario ha elegido. Luego un mnemónico podría ser usado para destruir el diálogo.

   A manera de recomendación, estos son los pasos que podrían seguirse para la confección de cualquier programa que utilice pantallas gráficas.

1) Definir la pantalla, ya sea manualmente o mediante el editor de pantallas.
2) Abrir archivos o Base de Datos a utilizar.
3) Leer los datos que necesiten ser cargados a los objetos o listas de ítems en el contexto.
4) Hacer que la pantalla gráfica se haga visible.
   Controlar eventos reportados por Windows, bajo las siguientes prioridades:

- Primero se trata de controlar en cual contexto (pantalla) ocurrió el evento.
- Luego se debe controlar la acción a seguir con el resto de eventos. Una buena forma de lograrlo es por medio de sub-rutinas adecuadas a la necesidad de cada evento.

Ejercicio de SYSGUI #1: Hola Mundo

Antes de foguearnos más abajo con algunos de los detalles del dispositivo SYSGUI, vamos a intentar trabajar con un ejemplo. Este es un programa clásico que seguiremos llamando "Hola, Mundo", con una interfase gráfica. Una ventana aparecerá con el título "Hola". Contiene un botón único, marcado "OK". La ventana desaparece después de que el usuario haga click en la caja de cierre o en el botón "OK". Esto deberá ayudarnos para introducir algunos de los conceptos y técnicas de SYSGUI de una forma manual, y posteriormente podamos agregar otros detalles.

```
0010 OPEN (1)"X0"
0020 PRINT (1)'SEMICHARS'
0030 PRINT (1)'WINDOW'(50,50,50,50,"Hola",$83$); REM "El $83$ hace que
0030:tenga opción de esconderse=$80$, maximizarse=$01$ y cerrarse=$02$
0040 PRINT (1)'BUTTON'(1,5,15,40,0,"OK",$$);  REM "Id, Col, Lin, Anch,
0040: Alto, Título
0050 DIM EVENT$:TMPL(1)
0060 READ RECORD (1,SIZ=LEN(EVENT$))EVENT$
0070 END
```

Mecanografíe este programa y sálvelo como Ejerc1. Intente correrlo con un SETTRACE.

¡SUGERENCIA! SETTRACE es más útil que siempre para la depuración de programas GUI, desde la ventana de consola si no es utilizada para cualquier otra cosa.

Intente correrlo paso por paso a través de él. (Usted podría usar <Alt>+<Tab> para ver la ventana del GUI si lo hace paso por paso). Note que la ventana puede ser cambiada de tamaño sin que ninguna declaración de programa sea ejecutada. Todo eso es manejado por el sistema del GUI. A menos que usted pregunte por reportes de eventos opcionales, usted puede quedarse quieto y permitir al sistema del GUI hacer una gran cantidad del trabajo!

Ahora vamos a comentar cada una de las líneas de ese programa.

```
0010 OPEN (1)"X0"
```

Esto exactamente nos prepara el ambiente para interactuar con el dispositivo de SYSGUI.

```
0020 PRINT (1)'SEMICHARS'
```

Esto establece la unidad de medida para la colocación de ventanas y controles a una unidad llamada "semicaracteres". 'SEMICHARS' no debe ser usado para colocar dibujos, solamente campos. Bajo la modalidad de semicaracteres, normalmente una pantalla (completa) se compone de 400 por 300 semicaracteres.

¡CUIDADO! Por omisión las ventanas y controles son posicionados en unidades de pixel, pero esto podría ser un problema debido a los tamaños variables de fuentes del sistema en uso en diferentes máquinas. Por ejemplo, si un botón fue hecho lo suficiente y justamente alto para acomodar el texto "OK" en una máquina, podría resultar demasiado pequeño en otra máquina donde una fuente más grande del sistema se tenga en uso.

En vista de que la fuente del sistema es una función del 'driver' de despliegue, y además es establecida de acuerdo a la preferencia del usuario, Visual PRO/5 no puede cambiar nada al respecto. Sin embargo, éste puede, ajustar el tamaño y localización de todo objeto gráfico para compensar la diferencia de las fuentes entre los distintos sistemas, y esto es exactamente al determinar cuál unidad de "semicaracteres" es considerada.

Un semicaracter es definido como de un cuarto del ancho de un caracter típico en la fuente del sistema actual y de un octavo la altura. Un pixel siempre es igual a un punto en el despliegue, pero un "semichar" podría ser igual a 2.4 pixeles a través y de 2.1 pixeles verticalmente, o alguna otra combinación rara. El tamaño exacto de un semicaracter no es lo que es importante (aunque es posible determinar esto en tiempo de corrida), pero el hecho es que controles con texto (como el botón en nuestro ejemplo) pueden ser hechos para mirarse bien en todos los despliegues.

El editor de recurso de BASIS soporta semicaracteres, que hacen la unidad de fácil manejo. Para máxima portabilidad, la unidad de semicaracteres debería ser utilizada en todas las ventanas donde existan controles que contengan una línea única de texto (el cual podría ser truncado si la fuente es larga). De toda manera, es aceptable y más conveniente, que utilizar pixeles.

```
0030 PRINT (1)'WINDOW'(50,50,50,50,"Hola",$83$)
```

El mnemónico 'WINDOW' es usado para crear ventanas gráficas. Usted lo utilizará una y otra vez. Los primeros cuatro parámetros describen la ubicación y tamaño de la ventana (xloc, yloc, ancho, altura). En este caso, hemos situado la ventana a 50 semicaracteres de la esquina superior izquierda a la vez en ambas coordenadas, y el tamaño de ventana de 50 por 50 semicaracteres. Entiéndase bien, que las coordenadas se refieren a los semicaracteres de la pantalla completa.

El siguiente parámetro es el título de ventana inicial. La ubicación, tamaño y título pueden ser completamente cambiados en cualquier momento con otros mnemónicos como 'SIZE' y 'TITLE'. El último parámetro requerido es un string de banderas o condiciones. Este es un string binario que contiene la suma de uno o más bits de bandera. Un séptimo parámetro puede opcionalmente ser añadido, que es una máscara de evento, también en un string binario. La máscara de evento puede ser usada para requerir informe de eventos opcionales.

¡CUIDADO! Los parámetros de máscara de banderas y evento son de cuatro bytes (ocho dígitos hexadecimales) de largo. Si menos bytes fueran especificados, los dígitos que falten serán asumidos a la izquierda a como ocurre con el vector de SETOPTS. Una bandera puesta como 01 se asumiría como 00000001. Es justamente un número.

El parámetro de banderas especificado aquí tiene una combinación de tres banderas. Estas son:

00000001 ventana puede ser cambiada de tamaño por el usuario

00000002 ventana tiene una caja de cierre
00000080 ventana puede ser minimizada por el usuario

Trate de cambiar los parámetros de la bandera por algunos de estos bits. (Valores buenos para el intento son 03, 01 y 02). ¡Usted tiene el control!

```
0040 PRINT (1)'BUTTON'(1,5,15,40,0,"OK",$$)
```

El mnemónico 'BUTTON' añade un control de botón a la ventana actual. El primer parámetro es un ID (identificador) para el botón. El ID puede ser un número cualquiera desde el 1 al 32767, aunque ID 1 es utilizado típicamente para los botones OK. Los siguientes cuatro números son la ubicación (x, y) y dimensiones (ancho, altura) del botón, en las unidades actuales de la ventana previamente definida (semicaracteres).

Note la altura de cero. Algunos controles tienen un valor de dimensión que se asume "por omisión". Los botones tienen un valor de altura que se asume por omisión, que puede ser solicitado suministrando una altura de 0. Usando dimensiones con valor que se asume por omisión puede ayudar a que sus controles sean agradables para ser utilizados en otras aplicaciones. En este caso, el botón será lo justamente alto como para acomodar el texto adecuadamente.

El sexto parámetro es el título. Este es el texto que aparece dentro del botón. El último (sétimo) parámetro es un string de banderas. En este caso, hemos suministrados un string vacío, el cual es equivalente al número cero (00000000). Ninguna de las banderas opcionales queda establecida.

En este punto, la ventana y botón son mostrados, y todo lo que continúa es para esperar a que el usuario tome alguna acción.

```
0050 DIM EVENT$:TMPL(1)
```

La nueva función TMPL() permite recuperar un string con la información que es retornada por la función FIN(), y en algunos casos especiales (los canales SELECT Y SYSGUI) pueden ser invocados sin el parámetro IND= para recuperar un TEMPLATE o patrón para el registro de información mismo.

En este caso, estamos recuperando el TEMPLATE o patrón para datos de evento, que pueden entonces ser leídos desde el canal SYSGUI. Recordar que eventos son reportados debido a la interacción del usuario con el sistema del GUI. No suministramos un parámetro de máscara de evento cuando se crea la ventana, de modo que recibiremos solo el evento obligatorio (no opcional). Afortunadamente, un botón que se presiona y una caja de cierre que se opera, ambos son eventos obligatorios, de modo que son automáticamente reportados

```
0060 READ RECORD (1,SIZ=LEN(EVENT$))EVENT$
```

El programa esperará con este READ RECORD hasta que un evento sea reportado o el programa sea interrumpido. Usted puede interrumpir el programa haciendo click con el ratón en la ventana de la consola (para establecer el foco de teclado a el) y luego oprimiendo <Ctrl >+C. Cambiando de tamaño la ventana se genera un evento opcional pero ese no es pasado por la máscara de eventos, de modo que las únicas acciones de usuario que hará que el programa sea completado son el click en la caja de cierre o el click en el botón OK.

Como un ejercicio final, vamos a convertir este programa en algo más completo, para una aplicación GUI con Visual PRO/5. Para hacer eso, vamos a crear una ventana de terminal invisible en el **config.bbx**.

```
ALIAS T2 SYSWINDOW "Terminal Invisible" INVISIBLE
```

Luego entonces cambie el programa de modo que termine con un RELEASE, no con un END.

```
0070 RELEASE
```

Ahora también vamos a adaptar un icono cualquiera.

```
0035 PRINT (1)'MINICON'("PROGMAN.EXE",10)
```

Ahora modificaremos un icono en Windows para que invoque el comando de esa línea (y utilice un icono cualquiera a opción suya).

```
vpro5 -tT3 Ejerc1
```

El programa Ejerc1 ahora correrá y terminará sin que aparezca la ventana de la consola.

Ejercicio de SYSGUI #2: Haciendo que las cosas trabajen

En este Ejercicio, vamos a trabajar en modo inmediato, para realizar un experimento con algunos de los mnemónicos que trabajan con el sistema del GUI.

Vamos a comenzar creando una ventana y poniendo un botón en ella, similarmente a como se hizo con el programa Hola Mundo. Observemos que esta vez no usaremos banderas para la creación de ventanas.

```
0001 REM "EJERC2 Práctica de pág.25/26
0010 BEGIN; OPEN (1)"X0"
0020 PRINT (1)'SEMICHARS'
0030 PRINT (1)'WINDOW'(50,50,50,50,"Test",$$)
```

¡NOTA! Desde que usted está trabajando en modo inmediato, el SYSWINDOW inmediatamente llegará al frente, para obtener su siguiente línea de entrada. Lo normal es que al dar ENTER, la ventana de GUI que usted está creando aparezca y desvanezca en forma inmediata. Todo eso es correcto. Usted puede hacer que reaparezca oprimiendo <Alt>+<Tab>. Escriba ahora:

```
0040 PRINT (1)'BUTTON'(1,5,15,40,0,"ok",$$)
```

El uno en negrita corresponde al ID asignado al botón.

En este punto, la ventana debería ser similar a la del ejemplo de Hola Mundo, excepto que la ventana no tiene caja de cierre, no puede ser minimizada, y tampoco cambiada de tamaño.

Seguidamente, trate ahora de crear otro botón usando como ID siempre el número 1.

```
0050 PRINT (1)'BUTTON'(1,5,30,40,0,"Cancelar",$$)
```

Córralo y verá que un ERROR=29 será dado, cuyo TCB(10) será igual a 12.

¡SUGERENCIA! Los mnemónicos de SYSGUI generalmente dan ERROR=29 como falla, pero el valor del TCB(10) generalmente es de gran ayuda. Recuerde que un código de TCB(10) positivo generalmente es dado cuando el error es detectado desde el interior del lenguaje (contrario a como ocurre con los errores originados en el sistema operativo). Cuando el TCB(10) es positivo, reste uno para obtener el código de error real. TCB(10)=12 forma un !ERROR=11, que normalmente es "Llave Duplicada o Faltante". En este caso, significa que ya hay un botón con ese ID.

Intente el mismo comando, pero con un nuevo ID para el nuevo botón, como sigue:

```
0050 PRINT (1)'BUTTON'(2,5,30,40,0,"Cancelar",$$)
```

Ahora usted si podrá ver el segundo botón.

¡NOTA! El ID 1 está reservado generalmente para botones OK, y el ID 2 está reservado para botones de Cancelar. Además de esos, al programador se le pide dejar los valores desde 1 a 99 como "reservados para utilización futura". Esto simplifica la situación un poco – para más detalles, vea el manual del Visual PRO/5.

Ahora tenemos una ventana con dos botones en ella. Vamos a trabajar con ellos. Comenzaremos experimentando para aprender como hacer cosas invisibles.

¡SUGERENCIA! El ID 0 es un identificador especial que se refiere siempre a la ventana actual. En los comandos que siguen, estaremos utilizando ID 0 para operar en toda la ventana, e IDs 1 y 2 para operar en los botones de control.

Trate digitando lo siguiente:

```
0060 WAIT 1; PRINT (1)'HIDE'(1)
0070 WAIT 1; PRINT (1)'SHOW'(1)
```

Los **WAIT 1** se agregan para que observe en forma pausada los cambios que se van dando en la pantalla. Apreciará muy bien que el botón OK se hace invisible y de nuevo visible. El primer número entre paréntesis es el ID del objeto a ser ocultado o mostrarlo. Recuerde que 0, 1 y 2 son válidos para este ejemplo. Experimente ocultando y mostrando los dos botones y la ventana misma. Los mnemónicos 'HIDE' y 'SHOW' soportan más de un parámetro, para cambiar más de un ítem a la vez. Ej:

```
PRINT (1)'SHOW'(0,1,2)
```

Con la línea anterior se asegurará que la ventana y ambos botones serán visibles de nuevo.

Ahora usted está listo para otro par de mnemónicos que son muy similares al 'HIDE' y 'SHOW'. Trate agregando estas líneas.

```
0080 WAIT 1
0090 PRINT (1)'DISABLE'(1)
0100 WAIT 1
0110 PRINT (1)'ENABLE'(1)
```

El mnemónico 'DISABLE' deshabilita el uso de campos definidos para entrada de datos del usuario. En el caso de un control, el ítem es rellenado con un color gris para indicar al usuario que no puede usarlo. En el caso de una ventana, simplemente no deja que responda. Las ventanas y

controles desactivados no pueden ser movidos, cambiados de tamaño, activados con click, o operados de otra manera, hasta que sean permitidos de nuevo. Experimente con 'DISABLE' y 'ENABLE' como se muestra arriba. Cuando lo haya hecho, digite la siguiente línea para asegurarse de que todo es permitido de nuevo.

```
PRINT (1)'ENABLE'(0,1,2)
```

Seguidamente, vamos a utilizar el mnemónico 'TITLE' para cambiar el texto de título de la ventana y los controles.

```
PRINT (1)'TITLE'(0,"Nuevo Título de Ventana")
PRINT (1)'TITLE'(1,"okay")
```

Experimente con el mnemónico 'TITLE', cambiando los títulos de los controles por lo que usted quiera. ¿Qué ocurre si el mensaje es demasiado largo para el campo? Cuando usted haya hecho la prueba, agregue lo siguiente para restablecer los títulos.

```
0120 WAIT 1
0130 PRINT (1)'TITLE'(0,"TEST")
0140 WAIT 1
0150 PRINT (1)'TITLE'(1,"OK")
0160 WAIT 1
0170 PRINT (1)'TITLE'(2,"CANCELAR")
0180 WAIT 1
```

Los mnemónicos 'MOVE' y 'SIZE' le permiten cambiar la ubicación y posición de algo una vez que ha sido creado. Intente eso, para hacer que el botón OK quede más alto.

```
0190 REM "Siguiente línea cambia de tamaño el botón OK
0200 PRINT (1)'SIZE'(1,40,30)
0210 WAIT 1
```

El primer parámetro del 'SIZE' es el ID del control (0 para la ventana) a ser afectado, y los siguientes dos parámetros son los nuevos valores X y Y, en unidades actuales (semicaracteres).

El botón OK ahora estará tan alto que absorbe al botón de Cancelar. Ahora vamos a hacer la ventana misma más alta

```
0220 PRINT (1)'SIZE'(0,50,100); rem "El cero subrayado se
0220:refiere a la ventana.
0230 WAIT 1
```

y entonces moveremos el botón de Cancelar hacia abajo.

```
0240 PRINT (1)'MOVE'(2,5,50)
0250 WAIT 1
```

Ahora que usted conoce como los mnemónicos 'MOVE' y 'SIZE' trabajan, experimente sus propios movimientos y cambios de tamaño de los tres objetos gráficos (ID 0, 1 y 2) que están mostrados en este momento. ¿Que ocurre si usted localizara un botón afuera del área visible en una ventana? ¿Puede usted moverla detrás y ver de nuevo? ¿Puede una ventana ser hecha mayor que la mostrada?

Vamos a aprender manipulación más general con el mnemónico: 'DESTROY'. Usted puede deshacerse de cualquier control o ventana con solo generar un 'DESTROY'(id) con el ID del objeto a ser destruido. Deshagámonos ahora del botón de Cancelar.

```
0260 PRINT (1)'DESTROY'(2)
0270 WAIT 1
```

Si no quiere esconder un objeto, puede entonces destruirlo y el objeto se pierde por completo. El número del ID puede ser reutilizado en este punto.

A diferencia de 'HIDE', 'SHOW', 'ENABLE', y 'DISABLE', el mnemónico 'DESTROY' solo puede aceptar un parámetro. 'DESTROY'(0) elimina la ventana actual y todos sus contenidos de un solo golpe. Ahora vamos a intentar, acabar este ejercicio. En este caso, hay dos cosas que se pueden destruir de inmediato, la ventana y el botón OK.

```
0280 PRINT (1)'DESTROY'(0)
```

En este punto, usted puede crear una nueva ventana, o cerrar simplemente el canal SYSGUI.

```
0290 CLOSE (1)
```

Ejercicio de SYSGUI #3: Explorando la función CTRL()

La función CTRL() le permite examinar el estado del sistema del GUI en diversos modos. Por ejemplo, usted puede utilizar la función CTRL "obtener texto" para recuperar el texto de título de la mayoría de los controles. Para probarlo mecanografíe el siguiente programa:

```
0010 OPEN (1)"X0"
0020 PRINT (1)'SEMICHARS'
0030 PRINT (1)'WINDOW'(50,50,70,70,"CTRL test",$$)
0040 PRINT (1)'EDIT'(101,10,10,30,15,"",$$)
0050 ESCAPE
```

Al dar RUN a ese programa, el contexto se desplegará y 'saltará' debido a la instrucción ESCAPE. Para lograr ver el contexto que genera, debe oprimir < Alt> + <Tab>.

La mayor parte de este programa ya le debe ser muy familiar. El nuevo mnemónico que está usando es el 'EDIT', que crea un control llamado Caja de Edición. Este es un control estándar de Windows para obtener una línea única de entrada del usuario. La sintaxis es idéntica a la indicada para el mnemónico 'BUTTON. En este ejemplo, estamos asignando control ID 101 a la caja de edición, y suministramos un string vacío para el texto del título. Cualquier texto de título que se suministre será preestablecido como el texto actual en la caja de edición.

Una vez que usted ha corrido el programa, usted deberá poder ver la nueva ventana del GUI y editar el control. Haciendo un Click del ratón dentro de la caja de edición producirá que usted vea una barra intermitente, la cual es conocida como "signo de intercalación", e indica donde el texto será insertado. Entonces, para lograr FOCO en dicho control, hágale click y luego proceda a digitar algún texto.

Ahora utilizaremos la función CTRL para atisbar en el control y ver lo que usted ha entrado. Digite este comando después de haber digitado algo en la caja de edición:

PRINT CTRL(1,101,1) → (Canal GUI, ID del Control y Tipo de Función)

Usted podrá ver el texto que fue digitado en la caja de edición. El primer argumento de la función CTRL() es el número de canal. En este caso, el dispositivo SYSGUI fue abierto por medio del canal 1. El segundo argumento es el ID del control o ventana que está siendo examinada. Como de costumbre, el ID 0 es un distintivo para la ventana en uso. Solo por diversión, trate de imprimir CTRL(1,0,1). Verá el texto del título que se le dio a la ventana: CTRL test.

El tercer argumento es la "función", que dice al CTRL qué tipo de información usted quiere realizar. La función 1 es "obtener texto". Frecuentemente "obtener texto" recupera el texto de título del control o ventana, pero no siempre. Como un ejemplo, cuando el objeto en cuestión es una caja de lista, "obtener texto" retorna el texto del ítem seleccionado de la lista en uso. Existen al menos quince diferentes funciones CTRL(), pero no todas son normalmente usadas.

Si el tercer argumento no fuera suministrado, Visual PRO/5 retorna la más comúnmente-utilizada o "interesante" **función** para ese control. Esta es siempre ya sea "obtener texto" (función 1) u "obtener valor" (función 2). Generalmente, aquellos controles cuyo estado está descrito principalmente por un número (lista de caja, barra de desplazamiento, caja de chequeo, etc.) asumido por omisión para "obtener valor". Todo lo demás es contrario al valor por omisión para "obtener texto".

Para que haga la prueba, digite estas dos líneas!!

```
PRINT CTRL(1,101)
PRINT CTRL(1,0)
```

Como usted puede ver, la función "obtener texto" es el valor por omisión en estos casos. Si a usted no le molesta esto con las funciones por omisión, no necesita preocuparse. Usted siempre puede especificar el tercer argumento y obtener la función que usted desee.

La función 0 es "obtener rectángulo". Eso retorna la ubicación del rectángulo unificador del control o ventana especificada. El rectángulo es retornado como cuatro strings consecutivos binarios de dos bytes. Este es un patrón que trabajará con CTRL() función 0.

```
X:U(2), Y:U(2), W:U(2), H:U(2)
```

Vamos a intentar encontrar la ubicación y tamaño de la ventana y de la caja de edición. Para eso adicione las siguientes líneas al programa definido en la página anterior.

```
0060 DIM RECT$: "X:U(2), Y:U(2), W:U(2), H:U(2)"
0070 RECT$=CTRL(1,0,0)
0080 PRINT RECT.X, RECT.Y, RECT.W, RECT.H
0090 RECT$=CTRL(1,101,0)
0100 PRINT RECT.X, RECT.Y, RECT.W, RECT.H
```

Ahora intente mover la ventana y encontrar los valores de nuevo. ¿Se sorprendió? La ubicación de la ventana es diferente, pero las dimensiones son las mismas. Todas las coordenadas para la caja de edición son reportadas relativamente a la ventana, de modo que no cambiaron. Usted podría encontrar alguna ligera variación de dimensión en errores de conversión debido a la conversión entre pixeles y semicaracteres.

La función 0 retorna los valores en las unidades actuales. Para ver como los pixeles y semicaracteres comparan en su sistema, trate de hacerlo digitando

```
0020 PRINT (1)'PIXELS'
```

para cambiar las unidades actuales a pixeles. Entonces busque los rectángulos dándole RUN al programa de nuevo. Note que los objetos exhibidos no cambiaron repentinamente su tamaño. Cambiando las unidades solamente serán afectados los nuevos objetos que sean posicionados y cambiados de tamaño, y como las dimensiones actuales sean reportadas por CTRL().

Vamos a combinar lo qué hemos aprendido en los ejercicios previos y hagamos un NUEVO PROGRAMA que presente una caja de edición y un pulsador en una ventana. Cada vez que el botón OK es presionado, el título de la ventana es dispuesto para concordar con el texto en la caja de edición.

```
0010 OPEN(1)"X0"
0020 PRINT (1)'SEMICHARS'
0030 PRINT (1)'WINDOW'(50,50,50,50,"Título a Variar",$$)
0040 PRINT (1)'BUTTON'(1,5,15,40,15,"OK",$$)
0050 PRINT (1)'EDIT'(101,5,30,40,15,"",$$)
0060 DIM E$:TMPL(1)
0070 REM
0080 READ RECORD(1,SIZ=LEN(E$))E$
0090 REM
0100 T$=CTRL(1,101)
0110 PRINT (1)'TITLE'(0,T$)
0120 GOTO 70
```

Nota 1: Este programa no tiene como terminar o salirse normalmente. Para volver a la ventana de la consola, tendrá que dar <Alt>+<Tab> o dar un Click fuera del contexto, y luego hacer un <Ctrl>+C para interrumpirlo.

Nota 2: Usted va a notar que ningún evento es reportado hasta que usted haga click en el botón. Las cajas de edición pueden reportarlo cada vez que sean modificadas, o cada vez que ganen o pierden foco de teclado. Sin embargo, esos eventos son opcionales y deliberadamente no los hemos solicitado. En muchos casos esto es preferible para esperar a que el usuario haga click en un botón, y entonces lea los otros controles, tanto más para que usted le siga la pista a todo y obtenga los eventos desde todos los controles.

Nota 3: Intente dar click, doble click y arrastre en la caja de edición. Usted va a observar el comportamiento estándar de Windows en los controles de edición de texto. Visual PRO/5 es completamente compatible con estos. Usted verá que se puede cortar, copiar y pegar con el la combinación estándar de teclas <Ctrl>+X, <Ctrl>+C y <Ctrl>+V. Acceso al clipboard de un menú también se tiene disponible, pero está fuera del propósito de este experimento.

Ahora mejoremos el programa anterior, cambiándole las siguientes líneas, para que vea como se optimiza el funcionamiento del mismo.

```
0030 PRINT (1)'WINDOW'(50,50,50,50,"Título a Cambiar",$00010002$);
     REM "Esa bandera activa gravedad y cierre de caja para la ventana
0070 PRINT (1)'FOCUS'(101); REM "Posiciona cursor en Objeto 101.
0090 IF E.CODE$="X" STOP; REM "Termina si cierre de caja es activado
```

Ejercicio de SYSGUI #4: El Reportero de Eventos

Hemos estado leyendo la cola de eventos para detectar eventos de botón y cajas de cierre, pero realmente nunca hemos analizado la información de los eventos. El mejor modo de experimentar esto es creando una ventana que reporte todos los eventos, entonces escriba un programa que vaya mostrando el contenido de cada evento en la pantalla. (Salve este programa, ya que luego lo volveremos a usar una y otra vez.)

```
0010 OPEN(1)"X0"
0020 PRINT (1)'SEMICHARS'
0030 PRINT (1)'WINDOW'(100,100,100,100,"Reportero de eventos",$03$,
0030:$FFFFFFFF$)
0040 PRINT (1)'BUTTON'(1,10,10,50,15,"OK",$$)
0050 DIM E$:TMPL(1)
0070 READ RECORD(1,SIZ=LEN(E$))E$
0080 PRINT E.CONTEXT, "  ",E.CODE$,E.ID,"  ",HTA(BIN(E.FLAGS,1)),
0080:E.X, E.Y
0090 GOTO 60
```

Vamos a repasar al menos las partes que ya nos son familiares en este programa:

```
0030 PRINT (1)'WINDOW'(100,100,100,100,"Reportero de eventos",$03$,
0030:$FFFFFFFF$)
```

Esta línea crea una ventana, como de costumbre. Estamos usando las banderas de opciones 01 (cambiar de tamaño) y 02 (tener una caja de cierre). El séptimo parámetro es opcional y sirve para enmascarar el evento, al cual en este caso le pusimos el "Árbol de Navidad" (todos los bits encendidos), para poder así obtener el máximo reporte de eventos. Esto normalmente solo es útil para pruebas. Generalmente, usted quisiera un pequeño reporte a como usted lo pueda tener y lograr aún que su programa haga lo qué usted quiera.

La línea 80 muestra todos los campos en la estructura del evento. El primer campo, E.CONTEXT, siempre será cero para esta demostración. El contexto es usado para identificar una ventana particular cuando el dispositivo de SYSGUI es manipulando en más que una ventana a la vez. Hemos olvidado todo lo tratado porque hasta el momento no hemos desplegado más que una ventana.

El campo E.CODE indica el tipo de evento. Hay muchos tipos de eventos. Algunos que usted podría ver en esta demostración son:

m	ratón movido
X	caja de cierre operada
B	pulsador de botón operado
F	ventana lograda o enfoque perdido
S	tamaño de ventana fue establecido o borrado
d	botón del ratón está abajo
u	botón del ratón está arriba
2	doble click en botón del ratón

El campo E.ID contiene el ID del control o ventana que está generando el reporte. En esta demostración, usted verá solamente 0 (la ventana) y 1 (el botón).

Los campos restantes (FLAGS, X, Y) no son siempre usados, pero cuando si lo están, dan el detalle acerca del evento. Por ejemplo, si un botón de ratón es oprimido {clicked}, FLAGS indica la condición de las teclas de Ctrl y Shift, y X y Y reportan la posición del cursor del ratón en el momento del click. Probablemente usted puede observar fuera de la figura cuáles campos son utilizados para otros tipos de evento.

Una vez que el programa está corriendo, intente las siguientes acciones.

mueva el ratón sobre la ventana
haga click en el botón
haga click en la caja de cierre
cambie el tamaño de la ventana

Lo primero que va a notar cuando haga lo anterior, es que hay tantos mensajes 'm' que es difícil darle seguimiento. Estos son eventos de movimiento del ratón, y es una buena idea omitir su reporte, aparte de si usted lo necesitara. Para desactivar ese reporte, debe cambiar la máscara de eventos en la línea 30 debe variar el tercer byte así: $FFFFF**E**FF$, dejando todos los otros encendidos.

Va a notar también que cuando el ratón hace click en el botón, el informe no contiene detalle alguno. Usted no será informado en cual botón del ratón ocurrió el click, donde fue el click, o la condición de las teclas de Ctrl y Shift. Hay otro tipo especial de control de botón conocido como TOOL BUTTON el cual si retorna esa información. (Sepa que botones herramienta y botones regulares no siempre son intercambiables). Intente cambiar el mnemónico 'BUTTON' en la línea 40 a 'TBUTTON'. (Todos los parámetros son los mismos). Corra el programa de nuevo y verá mucho más detalle del reporte (esta vez tipo de evento 'b') cuando el botón del "OK" es presionado.

¡NOTA! Verá que al operar la caja de cierre no hace que la ventana se aleje. Si este fuera el comportamiento que usted desea, el programa deberá preguntar por los eventos 'X' para destruir la ventana con 'DESTROY'(0).

Ejercicio de SYSGUI #5: Dibujando e Imprimiendo

Lo que en esta sección se ve, realmente solo puede interesar a quienes desean aprender a realizar trazos lineales en un contexto gráfico.

Cada ventana de SYSGUI es no solamente un tenedor potencial de controles sino también una superficie para dibujado y trazado.

¡CUIDADO! Los controles y dibujado tienen completamente diferentes coordenadas en el sistema. Es de gran ayuda imaginarse que la coordenada del sistema CONTROL está incrustada encima de la coordenada DRAWING del sistema. Cuando los controles están situados en una ventana de SYSGUI, son situados siempre encima de cualquier dibujo hecho (¡hasta agrupan ítem's dibujados en cajas obscurecidas!). No es posible dibujar sobre un control. Las unidades también son diferentes lo cual puede causar que el alineamiento sea a veces engañoso.

¡SUGERENCIA! Si usted deseara dibujar en una ventana que utiliza semicaracteres y contiene controles, y tienen el alineamiento de dibujo predicho con los controles, coloca un control invisible (una caja de grupo funciona agradablemente) en la ventana donde usted quiera dibujar. Aún cuando el control esté invisible, la función CTRL 0 reportará su ubicación. Cambie las unidades de control actuales a 'PIXELS' lo suficientemente largos como para contener el rectángulo unificador para el control invisible. Entonces usted tendrá coordenadas en pixeles para colocar su dibujo. VISUAL PRO/5 se despacha con programas de demostración que utilizan esta técnica.

Vamos a comenzar dibujando un círculo en una ventana que puede ser variada de tamaño, entonces encojamos la ventana y utilicemos Barras de Desplazamiento para ver parte del círculo a la vez. Teclee inmediatamente las siguientes líneas.

```
0010 REM "Hacer un dibujo
0020 OPEN(1)"X0"
0030 PRINT (1)'WINDOW'(100,100,200,200,"Dibujando",$0D$)
0040 PRINT (1)'ARC'(100,100,30)
```

No es necesario cambiar a semicaracteres porque no estamos ploteando algunos controles. El flag $0D$ para la creación de la ventana requiere una barra de desplazamiento vertical, una barra de desplazamiento horizontal, y una ventana que permita ser cambiada de tamaño.

¡CUIDADO! Ahora mismo usted va a notar que se ven las barras de desplazamiento. Esto es normal. La razón es que las barras de desplazamiento son solamente agregadas a una ventana si (1) lo indicamos en el flag, y (2) si parte del área de dibujo no está actualmente visible. Inicialmente todo el área de dibujo es visible.

El mnemónico 'ARC' en SYSGUI trabaja exactamente como el conocido mnemónico 'ARC' usado con los dispositivos tradicionales de ploteo del BBx. Igualmente a como es utilizado desde antes, este dibuja un círculo con un radio de 30 pixeles, y el centro posicionado en el pixel 100,100.

¡NOTA! El origen para dibujar es normalmente en la esquina superior izquierda de la ventana, y las unidades son normalmente pixeles. El origen y escala pueden ser alterados con el mnemónico 'WORD', y las unidades pueden ser cambiadas (aunque nunca a semicaracteres) con 'DRAWUNITS'.

Trate de cambiar el tamaño a la ventana. Usted notará que aún no puede ver alguna barra de desplazamiento. En lugar de eso, la imagen dibujada es adaptada al tamaño y escala de la ventana.

Para mantener la imagen el mismo tamaño que la ventana cambiada de tamaño, utilice el mnemónico 'TRACK', a como sigue.

```
0050 PRINT (1)'TRACK'(0)
```

Eso impedirá que la superficie de dibujo sea re-escaleada junto con la ventana. Ahora usted debería poder reducir la ventana y de poner en operación las Barras de Desplazamiento.

```
0060 PRINT (1)'PLOTTEXT'(150,100,"Comentario"); REM "Sirve
0060:para agregar texto a los dibujos
```

Trate de vaciar el dibujo a una impresora, si tuviera una disponible. Estos comandos deberán permitirle esa acción, ya que ocasionan el despliegue del diálogo de impresión del Windows.

```
PRINT (1)'PSETUP','PWINDOW'
```

El mnemónico 'PSETUP' nos trae el diálogo de impresión, de modo que usted puede seleccionar una impresora. Esto solamente es necesario si usted no deseara aceptar las omisiones del sistema, como están configuradas en el panel de control. 'PWINDOW' descarga los contenidos de la ventana actual a la impresora como job de impresión de una página. (Jobs de impresión de multiples-páginas son posibles con 'PBEGIN' y 'PEND').

¡NOTA! Los controles nunca son impresos por 'PWINDOW'... solamente el contenido de una superficie de dibujo de la ventana es transferido.

Ver ejemplo de 'SETUP' y 'PWINDOW' al final de esta sección.

Usted puede notar que su dibujo aparece mucho mayor o más pequeño (generalmente más pequeño) en la página que en la pantalla. La razón es que fue ploteado en unidades de pixel. Su impresora probablemente tiene un diferente tamaño de pixel que el de su pantalla. Mientras sea posible utilizar las funciones CTRL() y FID() para determinar tamaños de pixeles tanto para impresora como pantalla, es mucho más fácil de dibujar simplemente de unidades absolutas.

El mnemónico 'DRAWNITS' selecciona unidades de dibujo. 'DRAWUNITS'(0) selecciona pixeles, que es lo que se asume por omisión. 'DRAWUNITS'(1) selecciona milésimas de pulgada.

!CAUTELA! Los ítems dibujados en unidades de pixel no aparecerán del mismo tamaño absoluto cuando se imprimen. Utilice milésimas de pulgada (`'DRAWUNITS'(1)`) para la mayor parte de aplicaciones impresas.

Vamos a limpiar la ventana al color background actual (`'CLEARBG'`) y redibujar el círculo utilizando milésimas de pulgada.

```
0032 PRINT (1)'CLEARBG'
0034 PRINT (1)'DRAWUNITS'(1)
0040 PRINT (1)'ARC'(1000,1000,300)
```

Esta vez estamos dibujando el círculo con mayores números, desde unas unidades que ahora son muy pequeñas. Este círculo tiene un radio de tres décimos de una pulgada, y es ploteado en una pulgada a lo largo de cada dimensión de la esquina de la superficie de dibujo.

¡CUIDADO! Muchas impresoras no pueden dibujar la página completa. Así, la superficie dibujada es situada en una dirección desde de la esquina de la página. Si usted necesitara recuperar estas cantidades de margen, usted puede utilizar la función CTRL().

Hay muchas, muchas funciones más para dibujo. Los mnemónicos proveen apoyo para colores variables, modos de relleno, patrones de pluma, y montones de diferentes formas y modos de dibujo. Quizás el mnemónico para dibujo más útil es 'IMAGE', el cual permite plotear un bitmap desde un archivo .BMP. Pruebe el siguiente ejemplo:

```
0010 REM "Mostrar BMP's
0020 BEGIN
0030 PRINT 'CS',"Este programa muestra los archivos .BMP
     del directorio que indique.",'LF',
     "Dar directorio a revisar:",
0040 LET path$="c:\windows"; INPUTE 26,1,30,"_",path$;
     IF path$="" THEN GOTO final
0050 LET a=POS(path$(LEN(path$))="/\");
     IF a THEN LET path$=path$(1,a-1)
0060 OPEN (1,ERR=0030)path$; LET c=UNT
0070 REM
0080 ver_si_es_bmp:
0090 READ RECORD(1,END=final)fil$;
     IF POS(".BMP"=CVS(fil$,4))=0 THEN GOTO ver_si_es_bmp
0100 OPEN (c)"X0"; PRINT (c)'WINDOW'(131,69,397,355,fil$,$$)
0110 PRINT (c)'IMAGE'(64,64,64,64,path$+"\"+fil$)
0120 PRINT (c)'SHOW'(0)
0130 WAIT 2; CLOSE (c)
0140 GOTO ver_si_es_bmp
0150 REM
0160 final:
0170 END
```

Adjunto un ejemplo del uso de 'PSETUP' y 'PWINDOW' para imprimir una ventana GUI.

```
0010 OPEN (1)"X0"
0020 PRINT (1)'WINDOW'(50,50,300,200,"",$01$)
0030 PRINT (1)'TRACK'(0)
0040 PRINT (1)'DRAWUNITS'(1)
0050 PRINT (1)'WORLD'(0,0,8000,10540)
0060 PRINT (1)'VIRTUAL'(0,0,8000,10540)
0070 PRINT (1)'IMAGE'(0,0,8000,10450,"Logotipo.bmp",1)
0080 PRINT (1)'PSETUP'
0090 PRINT (1)'PWINDOW'
0100 END
```

Ejercicio de SYSGUI #6: Botones herramienta (Tools Buttons)

En este ejercicio, introducimos un nuevo tipo de control llamado "botón herramienta". Los botones herramienta son muy utilizados, pero tienen algunas diferencias muy importantes respecto a los push buttons:

BOTONES DE PULSAR

son un objeto intrínsico o de Windows
pueden recibir enfoque de teclado
tiene significación especial cuando la navegación de teclado es permitida.

BOTONES HERRAMIENTA

son un control de hábito suministrado por BASIS
pueden verse mejor que los botones de pulsar
pueden mostrar un título regular o una imagen de un bitmap
no pueden recibir enfoque de teclado
están excluidos de la orden de navegación de teclado
pueden operar normalmente o como un "botón de conmutación"
informa una gran cantidad de detalle en cada evento
son idealmente apropiados para uso en "barras de herramientas"

Para aprovecharnos más del botón herramienta, por favor teclee en la versión modificada del reportero de eventos del Ejercicio #4. Si usted salvó el reportero original de eventos, la única diferencia con el original es que no estamos mostrando los eventos de movimiento de ratón (máscara con $FFFFF**E**FF$), y una línea 40 que dice 'tbutton' en lugar de 'button'. También usaremos ID 101 en lugar de ID 1, desde ahí no hay forma de usar un botón herramienta para el propósito estándar de que control ID 1 es supuesto a servir. De otra manera, eso es justamente el reportero de eventos del Ejercicio #4.

```
0010 OPEN (1)"X0"
0020 PRINT (1)'SEMICHARS'
0030 PRINT (1)'WINDOW'(100,100,100,100,"Reportero de Eventos",$03$,
0030:$FFFFFEFF$)
0040 PRINT (1)'TBUTTON'(101,10,10,50,15,"OK",$$)
0050 DIM E$:TMPL(1)
0060 READ RECORD (1,SIZ=LEN(E$))E$
0070 PRINT E.CONTEXT,"", E.CODE$, E.ID,"", HTA(BIN(E.FLAGS,1)),
0070:E.X, E.Y
0080 GOTO 60
```

Lo primero que usted va a notar al correr el programa es que el botón herramienta mira y actúa justamente como el regular botón de pulsar. O lo hace? Bien, conocemos ya que no se comporta de la misma manera si la navegación de teclado estuviera permitida, pero eso es algo mucho más diferente.

Intente hacer click en el botón herramienta algunas pocas veces y mire el reporte de eventos. La información de eventos realmente cambia en dependencia de *donde* usted haga click, dentro del botón. Cuánto útil esto sea depende de su aplicación. Mucho más útil es el hecho que el informe también indica si el control y/o teclas shift están oprimidas durante el click, y cuál botón del ratón es oprimido.

Aún más interesante es el hecho que el botón herramienta pueda ser hecho para operar como un botón conmutador. Para hacer esto, interrumpa y termine el programa (haga click en la ventana de la consola para enfocar, luego de Ctrl-C, y digite END), y haga este pequeño cambio:

```
0040 PRINT (1)'TBUTTON'(101,10,10,50,15,"OK",$0400$
```

Cuando usted lo corra de nuevo, va a notar que el botón herramienta ahora permanece abajo de donde usted hizo click la primer vez, y cambia el estado con cada click. Si usted quisiera el estado inicial del botón que está abajo, use justamente la bandera estándar "inicialmente chequeada". Como usted ve, un botón herramienta cambiable se comporta muy parecido a una caja de chequeo. (Si usted quisiera intentarlo, use justamente 0404 en lugar de 0400. Estos medios son "cambiables e inicialmente chequeados").

Mientras el programa está corriendo, intente dar click en el botón algunas veces. Usted va a notar que el estado del conmutador es informado junto con todo lo demás en el parámetro de banderas. El estado del conmutador (E.FLAGS) tiene un valor 10. De otra manera, todo evento reportado es idéntico, independientemente de si el botón herramienta es cambiable o no.

Los botones herramienta pueden también sostener {bitmaps} en lugar de texto ordinario. Para hacer eso, solamente establezca el título del control (ya sea inicialmente o en cualquier momento posterior) a el texto "BITMAP=", seguido por el nombre del path de un {bitmap} (archivo .BMP).

¡CUIDADO! Cuando especifique el nombre del archivo para un archivo .BMP para mostrarlo en un botón herramienta, los directorios definidos dentro del PREFIX no son utilizados para la búsqueda.

Este es un ejemplo que trabajará en las instalaciones de Windows más estándar.

```
0041 WAIT 1
0042 PRINT (1)'TITLE'(101,"BITMAP=C:\Basis\VPRO5\HELP.BMP")
```

Si no trabajara, usted va a ver justamente el texto anotado entre comillas como el título dentro del botón. Generalmente esto ocurre si el path es inválido. Si esto ocurriera, intente localizar el archivo .BMP en su sistema para suministrar así el path correcto.

Ahora interrumpa el programa (haciendo click fuera del contexto y luego oprima Ctrl-C) y ejecute los siguientes comandos de modo inmediato para interactuar con el botón herramienta.

```
0043 WAIT 1
0044 PRINT (1)'DISABLE'(101)
0045 WAIT 1
0046 PRINT (1)'ENABLE'(101)
```

Note el color gris utilizado para el texto del título o {bitmap} cuando el botón está desactivado. Asegúrese de escoger {bitmaps} con suficiente contraste para que sean distinguibles cuando el botón esté desactivado. Los botones herramienta pueden ser desactivados en cualquier estado de cambio, y el despliegue reflejará aún el estado actual. El botón puede ser conmutado mientras esté desactivado, pero solamente desde una sentencia de programa, nunca por el usuario.

Aquí hay dos nuevos mnemónicos.

```
0047 WAIT 1; PRINT (1)'CHECK'(101)
0048 WAIT 1; PRINT (1)'UNCHECK'(101)
```

Estos operan el botón conmutador sin necesidad de usar el ratón. Los eventos no son generados en este caso. Similarmente como 'ENABLE'/'DISABLE' y 'HIDE'/'SHOW', estos pueden operar en una lista de IDs. Estos pueden también ser usados con cajas de chequeo y botones de radio (discutidos posteriormente), e ítems de menú chequeables.

¡NOTA! El mnemónico **'CHECK'** siempre causará que un botón herramienta sea cambiado a estado "chequeado" o "deprimido", aunque el botón no sea cambiable. Normalmente usted no querría utilizar 'CHECK' excepto con botones herramienta cambiables.

```
0049 WAIT 1; PRINT (1)'TITLE'(101,"Hola")
```

El mnemónico 'TITLE' le permite cambiar el título de botón en cualquier momento. Desde el título codifica cualquier información del BITMAP, este puede también ser usado para cambiar el BITMAP utilizado por este botón.

En la siguiente sección se muestra un programa en donde se donde se pone en práctica el uso de los mnemónicos aprendidos en esa sección.

Ejercicio de SYSGUI #7: Controles Chequeables (CHECKBOX y RADIOBUTTON)

En este ejercicio, aprenderá a usar las **cajas de chequeo** y los **botones de radio**. Estos son similares en concepto y operación, excepto que los botones de radio operan en grupos exclusivos (chequeando uno y des-chequeando otros en el grupo) y operando las cajas de chequeo independientemente.

Estaremos reutilizando nuestro viejo amigo el reportero de eventos, del ejercicio 4. Si aún lo tiene salvado, solo necesitará re-entrar la línea 40, del listado de abajo.

```
0010 OPEN (1)"X0"
0020 PRINT (1)'SEMICHARS'
0030 PRINT (1)'WINDOW'(100,100,100,100,"Reportero de Eventos",$03$,
0030:$FFFFFEFF$)
0040 PRINT (1)'CHECKBOX'(101,10,10,50,15,"Chequear",$$)
0050 DIM E$:TMPL(1)
0060 READ RECORD (1,SIZ=LEN(E$))E$
0070 PRINT E.CONTEXT," ",E.CODE$, E.ID," ",HTA(BIN(E.FLAGS,1)), E.X, E.Y
0080 IF E.CODE$<>"X" GOTO 60
```

Cuando este programa es corrido crea por si mismo una caja de chequeo en una ventana. Trate de operar la caja de chequeo y ver los eventos que son generados. Usted obtiene unos eventos del tipo 'c' cuando una caja de chequeo es chequeada o des-chequeada, y las banderas indican si estuvo chequeada o des-chequeada (01 y 00).

Si usted quisiera la caja de chequeo inicialmente puede ser chequeada, use el valor 04 en la bandera (inicialmente chequeado) en la lista de parámetros 'checkbox'. El que sigue es un ejemplo

de como usted puede cambiar la línea 40 para obtener una caja de chequeo que inicialmente esté chequeada (04), y también inicialmente desactivada (01).

```
0040 PRINT (1)'CHECKBOX'(101,10,10,50,15,"Chequear",$05$)
```

Mientras el programa está corriendo, interrúmpalo (haga click en la ventana de la consola a enfocar, y luego oprima Ctrl-C). Ahora vamos a interactuar con la caja de chequeo directamente.

```
0041 WAIT 1; PRINT (1)'DISABLE'(101)
0042 WAIT 1; PRINT (1)'ENABLE'(101)
```

Las declaraciones anteriores hacen que el control se desactive, y active nuevamente. Usted va a ver el indicador de color gris cuando es desactivado. En este momento esos mnemónicos ya le deberían ser muy familiares.

```
0047 WAIT 1; PRINT (1)'CHECK'(101)
0048 WAIT 1; PRINT (1)'UNCHECK'(101)
```

Estos mnemónicos pueden ser usados en cualquier momento para cambiar el estado de los controles chequeables (botón herramientas cambiable, cajas de chequeo, botones de radio). Pueden también ser usados en ítems de menú chequeables. Una lista de IDs puede ser provista en la lista de parámetros, al igual que 'ENABLE'/'DISABLE' y 'SHOW'/'HIDE'.

```
0049 WAIT 1; PRINT (1)'TITLE'(101,"No chequear")
```

Como siempre, el mnemónico 'TITLE' puede ser utilizado para cambiar un título de un control en cualquier momento. En el caso de una caja de chequeo, el título es el texto de etiqueta mostrado a continuación de la caja de chequeo.

Ahora vamos a experimentar con la función CTRL ( )

Antes de que pasemos a los Botones de Radio, vamos a aprender a usar la función CTRL() para obtener el estado de una caja de chequeo. Utilizando los mismos mnemónicos del programa que tenemos cargado en memoria asegurémonos de que la caja de chequeo esté habilitada. Entonces digite:

```
PRINT DEC(CTRL(1,101,2))
```

Esto imprimirá ya sea un 0 o un 1, dependiendo esto del estado actual de la caja de chequeo. Intente conmutación con el ratón, y obtenga la función 2 de CTRL nuevamente para ver el cambio.

Recuerde que la función 2 de CTRL es "obtener valor". La línea de arriba permite obtener el estado actual del botón, que normalmente solamente nos puede interesar si el botón es cambiable.

Trate de usar la función 2 del CTRL (como se muestra arriba) dos veces, intercambiando entre el botón y el ratón.

También podrá ver que la función 2 nos sirve para recuperar el título de un TOOL BUTTON, el cual, si lo que está exhibiendo es un BITMAP, el texto del título es retornado como se estableció, con la sintaxis del BITMAP=.

```
PRINT CTRL(1,101,1)
```

¡CUIDADO! En todos los casos, el valor por omisión para la función CTRL es 2 (“obtener valor”), mientras que el valor por omisión para la función CTRL para los botones de pulsar es 1 (“obtener texto“).

Para practicar un poco más con el CTRL y con los mnemónicos y funciones aprendidos en este capítulo, podemos escribir el siguiente programa, el cual nos muestra un contexto en el que aparece un botón para indicar OK, dos cajas para chequeo y dos campos para mostrar números. Cada vez que sea oprimido el botón OK se inicia un ciclo del 1 al 10, en el que van siendo mostrados los números pares o impares, según lo hayamos indicado en los botones de chequeo.

```
0010 REM "Ejerc7a Opción de mostrar números pares/impares del 1 al 10 (1-1)
0020 BEGIN ; OPEN (1)"X0"
0030 PRINT (1)'SEMICHARS'
0040 PRINT (1)'WINDOW'(100,100,100,100,"Prueba con Controles",$03$,$FFFFFEFF$)
0050 PRINT (1)'CHECKBOX'(101,10,5,50,15,"Impares",$$); PRINT (1)'CHECK'(101)
0060 PRINT (1)'CHECKBOX'(102,10,20,50,10,"Pares",$$)
0070 PRINT (1)'TEXT'(103,65,8,15,10,"- - - -",$$); REM "Ponga guiones normales
0080 PRINT (1)'TEXT'(104,65,22,15,10,"- - - -",$$)
0090 PRINT (1)'TEXT'(105,8,38,84,40,"La idea es que cada vez que oprima OK se
0090:genere un ciclo del 1 al 10, para mostrar los números que indique arriba
0090:.",$$)
0100 PRINT (1)'BUTTON'(1,20,80,50,15,"OK",$$); REM "ID del Boton OK es 1.
0110 DIM e$:TMPL(1)
0120 REM
0130 lee_eventos:
0140 READ RECORD(1,SIZ=LEN(e$))e$
0150 PRINT e.context," ",e.code$,e.id," ",HTA(BIN(e.flags,1)),e.x,e.y
0160 REM
0170 IF e.code$="X" THEN PRINT (1)'DESTROY'; STOP
0180 IF e.code$<>"c" THEN GOTO ver_si_ok
0190 REM "Se nos reporta que usuario activó una caja de chequeo, entonces hay
0190:hay que desactivar la otra
0200 PRINT (1)'UNCHECK'(103-e.id+100); REM "Desactiva CHECKBOX 101 o 102
0210 PRINT (1)'TITLE'(105-e.id+100,"- - - -"); REM "Afecta TEXT 103 O 104
0220 GOTO lee_eventos
0230 REM
0240 ver_si_ok:
0250 IF e.code$<>"B" THEN GOTO lee_eventos
0260 IF e.id<>1 OR DEC(CTRL(1,101,2))+DEC(CTRL(1,102,2))=0 THEN GOTO lee_event
0260:os
0270 IF DEC(CTRL(1,101,2))=1 THEN LET caja=103,inicio=1 ELSE LET caja=104,inic
0270:io=2
0280 FOR t=inicio TO 10 STEP 2; PRINT (1)'TITLE'(caja,STR(t:"0000")); WAIT 1;
0280:NEXT t
0290 GOTO lee_eventos
```

Ya en la realidad, el programa anterior quedaría mejor implementado si cambiáramos los 'CHECKBOX' por 'RADIOBUTTON', pero como aún no los hemos estudiado, dejémoslo así. Este cambio le queda de tarea para cuando aprenda a usarlos en lo que sigue de esta sección.

Para poder ver más los **botones de radio**, necesitaremos cambiar el programa en dos modos.

(1) Necesitamos crear al menos dos Botones de Radio, ya que operan siempre como un conjunto, y

(2) Necesitaremos añadir una línea al programa para agrupar los botones después de creados.

Las siguientes tres líneas cambiarán al reportero de eventos que hemos estado utilizando para hacer ese trabajo.

```
0040 PRINT (1)'RADIOBUTTON'(101,10,10,50,15,"Opción 1",$04$)
0041 PRINT (1)'RADIOBUTTON'(102,10,30,50,15,"Opción 2",$$)
0042 PRINT (1)'RADIOGROUP'(101,102)
```

Las primeras dos líneas ya deben serle familiares. Causan que nos sean colocados controles de botones de radio, con valores 101 y 102 en sus ID. El 04 es la bandera de "inicialmente chequeado".

¡NOTA! No es un requisito activar un botón de radio inicialmente, pero es buena práctica de programación, ya que al usuario nunca se le debe permitir des-chequear todos los botones.

La línea 42 introduce el mnemónico 'RADIOGROUP'. Coge una lista de dos o más controles de IDs de botones de radio y los asigna a un grupo. Los botones de radio no operarán hasta que sean asignados a un grupo. Usted puede crear tantos grupos de botones de radio como usted guste en una ventana. Nunca asigne un mismo botón de radio a más de un grupo.

Ahora vamos a correr el programa y observar el reporte de eventos. Cada uno de los botones de radio genera eventos del tipo 'c', indicando cuál botón ha sido chequeado. Ningún reporte de "des-chequeado" será generado para los botones de radio.

Interrumpa el programa y trate de operar los botones de radio directamente con las siguientes declaraciones de programa:

```
PRINT (1)'CHECK'(101)
PRINT (1)'CHECK'(102)
```

Observe que al chequear el botón de radio de un grupo, causa que todos los demás botones de radio del grupo se des-chequeen. Esta es la naturaleza de los botones de radio. El mnemónico 'UNCHECK' no trabaja con los botones de radio.

Para ver cual es el estado de un grupo de botones de radio, es necesario utilizar la función **2** de CTRL ("Obtener valor") en cada uno de los botones en el grupo.

```
PRINT DEC(CTRL(1,101,2))
PRINT DEC(CTRL(1,102,2))
```

¡SUGERENCIA! Aún cuando los botones de radio y cajas de chequeo informen unos eventos del tipo 'c' cada vez que son operados, frecuentemente es más oportuno esperar a que un botón OK sea presionado, y entonces puedan ser leídos los botones con la función 2 de CTRL, para determinar cuáles están chequeados.

Ahora es un buen momento como para introducir el control GROUPBOX, el cual es una caja que agrupa controles para proveer al usuario una asociación visual entre estos. No hay reglas reales en cuanto a su colocación, solamente lineamientos generales.

Añada la siguiente línea al programa para agregar una caja de grupo alrededor los botones de radio.

```
0043 PRINT (1)'GROUPBOX'(103,5,0,80,80,"Botones de Radio",$$)
```

Esto no cambiará el comportamiento del programa, solamente la apariencia. Las cajas de grupo nunca generan eventos algunos.

¡CUIDADO! Las cajas de grupo aparecen siempre detrás de otros controles, pero no son transparentes. Obscurecerán cualesquier ítem dibujado que se superponga en la ventana.

¡CUIDADO! Hay tres modos en que los botones de radio pueden ser "agrupados", y esto es importante para evitar confundirlos. La primera forma en que puedan ser agrupados es que sean programados para operar como un conjunto. Los botones de radio no trabajan hasta que sean agrupados de esta manera. Esto puede ser hecho con los mnemónicos 'RADIOGROUP' o por la asignación de los números de ID desde el interior del Editor de Recursos de BASIS. La segunda forma en que los botones de radio puedan ser agrupados, es asignando números de ID consecutivos y encendiendo el atributo de la bandera "group" para todas ellos. Esta forma es solamente para navegación de teclado, y no afecta el modo en que operan. Esto simplemente causa que podamos usar la tecla Tab para tratar todo el grupo de Botones de Radio (o cajas de chequeo, en cuanto a eso) como un conjunto, en función de navegación. La tercer forma de "agrupar" botones de radio es visualmente, dibujando un control de caja de grupo alrededor a ellos. Esto no tiene efecto en la operación del programa y es solamente para beneficio del usuario. En la mayoría de los programas, los tres métodos de agrupar son usados en acuerdo, pero es importante entender cuáles son requeridos y cuáles no, y como aplicar cada método.

Algo que no se controla, como la caja de grupo, es el control de texto estático. Es creado con el mnemónico 'TEXT', y simplemente causa un mensaje de texto (el título del control) que aparece en la ubicación prescrita. Inténtelo adicionando la siguiente línea al programa.

```
0044 PRINT (1)'TEXT'(104,10,90,50,15,"Hola",$$)
```

El uso más común de controles de texto estáticos es el etiquetado para controles que no muestran sus títulos, como cajas de listas, barras de desplazamiento, etc. Estos pueden ser utilizados libremente, pero usted debería ser consciente de que, como las cajas de grupo, no son transparentes.

Ejercicio de SYSGUI #8: Controles de Listas

Existen tres tipos de controles que permiten al usuario escoger ítems de una lista. El más básico de estos es el **LIST BOX**. Este es mostrado como una caja con una barra de desplazamiento vertical. Dentro de la caja tenemos varios ítems de texto, que pueden ser "seleccionados" por el usuario. Dependiendo del tipo de caja de lista, el usuario puede seleccionar solamente un ítem, o algún número de ítems, de la lista. A veces un click, y más frecuentemente un doble click en un ítem de la lista, puede provocar una acción especial a tomar.

Los otros tipos de controles de lista son el **LIST BUTTON** y el **LIST EDIT**. Ambos son mostrados como una caja de edición con un botón a continuación de esta. Presionando el botón se activa una lista, y seleccionar un ítem en la lista causa su selección para ser mostrado en la caja de edición, y la lista se desactiva. En el control de botón de lista, solamente selecciones de la lista pueden ser hechas, mientras que el control de la lista de edición permite al usuario teclear una entrada libremente en la caja de edición, así como para seleccionar un ítem desde la lista.

Este es nuestro bien gastado programa reportero de eventos, con una nueva línea 40 y 41.

```
0010 OPEN (1)"X0"
0020 PRINT (1)'SEMICHARS'
0030 PRINT (1)'WINDOW'(100,100,100,100,"Reportero de eventos",$03$,
0030:$FFFFFEFF$)
0040 PRINT (1)'LISTBOX'(101,10,10,80,80,"",$$)
0041 PRINT (1)'LISTADD'(101,-1,"John","Paul","George","Ringo")
0050 DIM E$:TMPL(1)
0060 READ RECORD (1,SIZ=LEN(E$))E$
0070 PRINT E.CONTEXT," ",E.CODE$, E.ID," ",HTA(BIN(E.FLAGS,1)), E.X, E.Y
0080 IF E.CODE$<>"X" GOTO 60
```

Trate de correr el programa y experimente con la caja de lista. Note que la barra de desplazamiento no opera porque aún no tenemos suficientes ítems en la lista. La barra de desplazamiento se activará automáticamente por si misma cuando suficientes ítems sean añadidos. El tipo de eventos 'l' solamente indica que un click o doble-click fue hecho en un ítem de la lista. Para determinar la selección hecha nuevamente hay que usar CTRL, como se muestra más adelante. Generalmente, solamente un ítem puede ser seleccionado a la vez.

El nuevo mnemónico '**LISTBOX**' tiene la sintaxis que ya nos es familiar para crear un control. El título es especificado como un string vacío en este caso debido a que el título no tiene importancia. Cajas de lista nunca muestran un título. El mnemónico 'LISTADD' puede ser usado para añadir detalles a una lista. Nuevos ítems son añadidos antes del índice de detalle especificado en el segundo parámetro. Se necesita un -1 como segundo parámetro para añadir los nuevos detalles al final de la lista. Existe una "forma larga" para el mnemónico 'LISTADD', para evitar colocar lotes de ítems en el parámetro de lista del mnemónico (que son de tamaño limitado). Este es un ejemplo de la forma larga:

```
0041 PRINT (1)'LISTADD'(101,-1,4),
0042 WRITE (1)"John","Paul","George","Ringo"
```

No olvide la coma al final de la línea 41. Esta es importante debido a que la forma larga de 'LISTADD' prepara el canal para recibir un número de ítems (4 en este caso) escritos luego directamente al canal. Estos ítems tienen cada uno un 'linefeed' como terminador de campo. Si la línea 41 no es finalizada con una coma, un 'linefeed' es enviado al canal inmediatamente después de la sentencia 'LISTADD', lo que causará que los primeros 4 ítems de la lista sean blancos.

¡SUGERENCIA! Observe también en línea 42 el uso de WRITE en lugar del PRINT. El WRITE inserta un 'linefeed' después de cada ítem escrito, que justamente es lo qué deseamos en este caso. Utilizaremos de nuevo este truco con controles de barras de desplazamiento, y nuevamente con menús, posteriormente.

Ahora vamos a cambiar el programa de modo que la caja de lista permitirá selecciones múltiples. Esto es fácil... solamente debemos cambiar la línea 40 agregando la bandera para "selecciones múltiples".

```
0040 PRINT (1)'LISTBOX'(101,10,10,80,80,"",$0400$)
```

Córralo de nuevo, y verá que con Ctrl-click se seleccionarán y des-seleccionarán ítems individuales, y también verá que Shif-click puede ser usado para seleccionar rangos.

En este ejemplo, ningún ítem es seleccionado inicialmente, pero podemos lograrlo con el mnemónico 'LISTSEL', el cual selecciona algunos ítems en una lista y automáticamente des-selecciona todos los otros. Vamos a añadirlo al programa.

```
0045 PRINT (1)'LISTSEL'(101,1)
```

La línea anterior provocará que el ítem "Paul" (segundo) sea seleccionado. La lista de índices está basada en cero.

Hay un grupo adicional de otros mnemónicos especiales para manipulación de los controles de tipos de lista. Corra el programa e interrúmpalo para experimentar con estos en modo inmediato. Recuerde que para volver a visualizar la ventana gráfica puede dar <Alt>+<Tab>.

```
PRINT (1)'LISTDEL'(101,0)
```

Esto elimina un ítem de la lista. El índice 0 es el que corresponde al primer ítem en la lista. Usted puede experimentar eliminando otros ítems.

```
PRINT (1)'LISTUNSEL'(101,1)
```

El mnemónico 'LISTUNSEL' des-selecciona algunos ítems de la lista. En el ejemplo anterior, el segundo ítem de la lista no es seleccionado. Inténtelo seleccionando con el ratón, y luego des-selecciónelo con esta declaración de programa.

```
PRINT (1)'LISTMSEL'(101,0)
```

El mnemónico 'LISTMSEL' es para ser usado con cajas de lista que permitan selecciones múltiples. 'LISTSEL', como usted recordará, selecciona un ítem y des-selecciona todos los demás. 'LISTMSEL' le permite seleccionar un ítem y le deja los que ya estuvieran seleccionados también activos. Naturalmente, estos solo trabajan con cajas de lista que permiten selecciones múltiples. Este ejemplo selecciona el primer ítem en una lista, dejando todos los demás ítems solos.

```
PRINT (1)'LISTADD'(101,0,"Peter")
```

Usted puede añadir ítems a una lista que ya contenga algunos. Este ejemplo añade "Peter" como el nuevo primer ítem en la lista, corriendo los otros ítems hacia abajo un espacio.

```
PRINT (1)'LISTCLR'(101)
```

Para borrar todos los ítems de una lista, use 'LISTCLR'.

A como lo detallamos en la Sección 3, la función CTRL puede ser utilizada para obtener información de un control de tipo de lista en cualquier momento. Recordemos que la función CTRL 1 ("Obtener Texto") recupera el texto de todos los ítems seleccionados en la lista. Intente seleccionar uno o más ítems y emitir este comando.

```
PRINT CTRL(1,101,1)
```

Usted verá el texto de todos los ítems seleccionados en la lista. Un caracter {linefeed} es añadido después de cada uno excepto del último ítem listado. Para ver el texto de absolutamente todos los detalles en la lista (seleccionados o no), use la función de CTRL 7 ("Obtener Todo el Texto"). En este caso, un caracter de {linefeed} es añadido después de a todos los ítems, incluyendo el último.

```
PRINT CTRL(1,101,7)
```

A veces es más fácil operar con el índice de ítem seleccionado(s) que con las etiquetas (labels). La función de CTRL 2 ("Obtener Valor") nos da un código-binario de dos bytes por cada ítem seleccionado. Estos están concatenados, de modo que la longitud del string es siempre dos veces la cantidad de ítems seleccionados. Para obtener cada uno de los índices en el string debemos utilizar la función DEC.

```
A$=CTRL(1,101,2)
```

La función de CTRL 3 ("Obtener Cuenta") nos retorna dos valores binarios de dos bytes consecutivos. El primero nos informa la cantidad de ítems seleccionados. El segundo nos informa del número total de ítems en la lista.

```
A$=CTRL(1,101,3)
```
→ DEC(A$(1,2)) = Cantidad registros seleccionados
DEC(A$(3,2)) = Cantidad de registros en la lista.

Hasta el momento solamente hemos considerado el tipo de control de una caja de lista. Cambiémonos ahora al '**LISTBUTTON**', para lo cual vamos a cambiar el programa de modo que la línea 40 nos genere un control de botón de lista, como sigue.

```
0040 PRINT (1)'LISTBUTTON'(101,10,10,80,80,"",$$)
```

El botón de lista no puede permitir selecciones múltiples, de modo que tuvimos que eliminarle la bandera. Por lo demás, el programa es idéntico. Trate de correrlo y observe la diferencia.

El mnemónico 'LISTSEL' en la línea 45 aún trabaja, así que "Paul" aparece como la selección inicial. Normalmente, el control del botón de lista ocupa solamente la porción principal de su rectángulo unificador, pero el tamaño puede ser más grande (como en este ejemplo tiene 80 x 80 semicaracteres) para permitir tener espacio para la lista cuando este es activado. Listas de ítems de doble-click no son posibles, pero selecciones son todavía reportadas como eventos tipo "l" (ele).

Es por eso que, un botón de lista opera idénticamente a una caja de lista. Interrumpa el programa e intente practicar con algunos de los mnemónicos que vio anteriormente, como 'LISTSEL', 'LISTADD', 'LISTDEL', y 'LISTCLR'. La función CTRL trabaja igual de bien, aun cuando la cantidad de ítems seleccionados siempre sea 0 o 1.

El control del mnemónico '**LISTEDIT**' muestra algo parecido al 'LISTBUTTON', pero se comporta de otro modo, ya que una entrada de texto de formato libre puede ser hecha. En muchos formas, el control del 'LISTEDIT' es un cruce entre una caja de edición y un botón de lista. El programa puede poblar la lista, pero solamente el usuario puede tomar ítems de la lista y usarlos para poblar la caja de edición. El programa nunca podrá establecer una selección actual, pero puede en su lugar forzar texto en una porción de la caja de edición del control utilizando el mnemónico 'TITLE'.

Por lo tanto tenemos que hacer dos cambios a nuestro programa. Primero, tenemos que crear una lista de edición en lugar de un botón de lista. Eso es fácil (cambie la línea 40). Segundo, no podemos utilizar 'LISTSEL' ya más (eliminar línea 45). Para establecer el contenido inicial del control, vamos a especificar justamente un título cuando creamos el objeto. La otra opción sería utilizar el mnemónico 'TITLE'.

```
0040 PRINT (1)'LISTEDIT'(101,10,10,80,80,"Paul",$$)
DELETE 45
```

Cuando usted corra el programa, usted va a notar que trabaja muy igualmente a la forma en que dijimos, con la excepción que usted puede ahora digitar cualquier cosa en el control. Los eventos del tipo 'l' (ele) no son reportados por más tiempo, ya que la operación de la lista es solamente una manera de como los contenidos de control pueden ser cambiados. Los eventos tipo 'f' (enfoque del control de edición) y 'e' (modificación del control de edición) son reportados en lugar de eso, justamente como estos son reportados para una caja de edición ordinaria.

Los mnemónicos para manipular listas como ('LISTADD', 'LISTCLR', 'LISTDEL') aún trabajan, pero algunos mnemónicos para ejecutar selecciones ('LISTSEL', 'LISTMSEL', 'LISTUNSEL') no trabajarán. Para cambiar el estado del control desde el interior de un programa, usted tiene que utilizar el mnemónico 'TITLE'.
Similarmente, las funciones de CTRL se comportan igualmente al modo en que trabajan las cajas de edición. La función de CTRL 1 ("Obtener Texto") lee el texto desde el control. Si se necesitara que un dato actual sea mostrado en la lista, el programa tendrá que hacer esa determinación por sí mismo.

Interrumpa el programa y mecanografíe lo siguiente.

```
PRINT (1)'TITLE'(101,"Peter")
```

Podrá notar que aún cuando "Peter" no está en la lista, se puede poner en la caja, ya sea por acción del programador o del usuario.

```
PRINT CTRL(1,101,1)
```

La función de CTRL 1 ("Obtener Texto") muestra el texto actual en el control. Trate de cambiar nuevamente el contenido del control con el ratón, y obtenga el valor de CTRL de nuevo. Usted puede ver que siempre está actualizado.

Los siguientes tres programas son para poner en práctica el aprendizaje de todo lo aprendido en esta sección. El primero abre un contexto con dos ventanas o listas de nombres, en donde la idea es marcar nombres en cualquiera de las listas y poder pasarlos a la otra, al oprimir el botón llamado PASAR.

```
0001 REM "Ejerc8a Practica pasando ítems entre 2 listas de un contexto
0010 OPEN (1)"X0"
0020 PRINT (1)'SEMICHARS'
0030 PRINT (1)'WINDOW'(60,100,200,120,"Cantantes",$03$,$FFFFFEFF$)
0040 REM
0050 PRINT (1)'LISTBOX'(101,10,10,80,80,"",$0400$); REM Lista izquierda
0060 PRINT (1)'LISTADD'(101,-1,"John","Paul","George","Ringo","Madonna",
0060:"M.Jackson","Sting","Pink Floyd","Aaaaa","Bbbbbb","Cccccc")
0070 REM
0080 PRINT (1)'LISTBOX'(102,110,10,80,80,"",$0400$); REM Lista derecha
0090 PRINT (1)'LISTADD'(102,-1,"Maria","Jose","Miguel","Angel","Pablo")
0100 PRINT (1)'TEXT'(105,10,110,180,10,"La idea es marcar y pasar ítems
0100:de una ventana a otra.",$$)
0110 REM
0120 PRINT (1)'BUTTON'(1,88,90,25,15,"Pasar",$$); REM "Define botón que
0120:servirá para indicar trasiego de ítems de una ventana hacia la otra
```

```
0130 DIM e$:TMPL(1)
0135 REM
0140 eventos:
0150 READ RECORD(1,SIZ=LEN(e$))e$
0160 PRINT e.context," ",e.code$,e.id," ",HTA(BIN(e.flags,1)),e.x,e.y
0170 IF e.code$="X" THEN PRINT (1)'DESTROY'; STOP
0180 IF e.code$="B" THEN GOTO pasar
0190 REM
0200 REM "Si hubo actividad en lista Izq. Des-selecciona los de la Der.
0210 IF e.id<>101 THEN GOTO 0250
0220 LET a$=CTRL(1,101,1),radio=DEC(CTRL(1,101,2)); PRINT a$,radio
0230 PRINT (1)'LISTUNSEL'(102,-1); GOTO eventos
0240 REM
0250 REM "Si hubo actividad en lista Der. Des-selecciona los de la Izq.
0260 IF e.id<>102 THEN GOTO eventos
0270 LET a$=CTRL(1,102,1),radio=DEC(CTRL(1,102,2)); PRINT a$,radio
0280 PRINT (1)'LISTUNSEL'(101,-1); GOTO eventos
0290 REM
0300 pasar:
0310 LET ven=101; REM "Detectar si hubo ítems marcados
0320 LET ven$=CTRL(1,ven,2); IF ven$="" THEN IF ven=101 THEN LET
0320:ven=ven+1; GOTO 0320 ELSE GOTO eventos
0330 LET a$=CTRL(1,ven,3); PRINT (1)'LISTADD'(203-ven,-1,DEC(a$(1,2))),
0330:CTRL(1,ven,1); REM "Agrega ítems seleccionados en la otra ventana
0340 FOR a=LEN(ven$)-1 TO 1 STEP -2
0350 PRINT (1)'LISTDEL'(ven,DEC(ven$(a,2))); REM "Elimina ítems seleccio
0350:nados en lista original
0360 NEXT a
0370 GOTO eventos
```

El siguiente programa es una variación del anterior. Lo que tiene diferente es que en lugar de un contexto nos muestra tres, en donde se permite por aparte para cada uno, poder hacer la misma migración de nombres del ejemplo inicial. Podrá observar que se pone en uso un nuevo mnemónico llamado 'CONTEXT', del cual veremos dos funciones: una para definir un CONTEXTO o ventana y la otra, para navegar o migrar de un CONTEXTO hacia otro.

```
0001 REM "Ejerc8b Práctica controlando dos contextos
0010 OPEN (1)"X0"
0020 PRINT (1)'SEMICHARS'
0030 FOR ven=0 TO 2
0040 IF ven=0 THEN PRINT (1)'WINDOW'(90,32,190,110,"Cantantes",$03$,
     $FFFFFEFF$)
0050 IF ven=1 THEN PRINT (1)'CONTEXT'(1),'WINDOW'(5,160,190,110,"Fut
     bolistas",$03$,$FFFFFEFF$)
0060 IF ven=2 THEN PRINT (1)'CONTEXT'(2),'WINDOW'(202,160,190,110,"O
     cupaciones",$03$,$FFFFFEFF$)
0065 REM
0070 REM "Define objetos inmediatamente después de haber definido cada
0070:CONTEXTO
0080 PRINT (1)'BUTTON'(1,82,90,25,15,"Pasar",$$)
0090 PRINT (1)'LISTBOX'(101,10,10,80,80,"",$0400$)
0100 PRINT (1)'LISTBOX'(102,100,10,80,80,"",$0400$)
```

```
0110 REM
0120 ON ven GOTO cant,futb,ocup
0122 REM
0128 cant: REM "Define y carga lista de Cantantes
0130 PRINT (1)'LISTADD'(101,-1,"John","Paul","George","Ringo","Madonna",
     "M.Jackson","Sting","Pink Floyd","Aaaaa","Bbbbbb","Cccccc")
0140 PRINT (1)'LISTADD'(102,-1,"Maria","Jose","Miguel","Angel","Pablo")
0145 GOTO 0240
0150 REM
0160 futb: REM "Define y carga lista de Futbolistas
0170 PRINT (1)'LISTADD'(101,-1,"Pele","Maradona","El Kaiser","Luis Gar
     cia","Reinaldo","Teofilo Cubillas")
0180 PRINT (1)'LISTADD'(102,-1,"Gabelo","Ronald Gomez","Wilmer Lopez",
     "Mauricio Montero","Rolando Fonseca")
0190 GOTO 0240
0200 REM
0210 ocup: REM "Define y carga lista de Ocupaciones
0220 PRINT (1)'LISTADD'(101,-1,"Mecanico","Bombero","Fontanero","Ciru
     jano","Abogado","Chofer","Guarda")
0230 PRINT (1)'LISTADD'(102,-1,"Secretaria","Doctora","Miscelanea",
     "Misionera","Visitadora Social","Maestra","Microbiologa","Super
     visora")
0240 NEXT ven; DIM e$:TMPL(1)
0250 REM
0260 main_loop: REM "Obtiene y muestra eventos causados por el usuario
0270 READ RECORD(1,SIZ=LEN(e$))e$
0280 PRINT e.context," ",e.code$,e.id," ",HTA(BIN(e.flags,1)), e.x, e.y
0290 IF e.code$="X" THEN PRINT (1)'DESTROY'; STOP
0295 PRINT (1)'CONTEXT'(e.context); REM "Se posiciona en CONTEXTO activa
0295:por el usuario
0300 IF e.code$="B" THEN GOTO pasar
0310 REM
0320 REM "Ctrl actividad en lista de la izquierda (ID=101)
0330 IF e.id<>101 THEN GOTO 0370
0340 LET a$=CTRL(1,101,1),radio=DEC(CTRL(1,101,2)); PRINT a$,radio
0350 PRINT (1)'LISTUNSEL'(102,-1); GOTO main_loop
0360 REM
0370 REM "Ctrl actividad en lista de la derecha (ID=102)
0380 IF e.id<>102 THEN GOTO main_loop
0390 LET a$=CTRL(1,102,1),radio=DEC(CTRL(1,102,2)); PRINT a$,radio
0400 PRINT (1)'LISTUNSEL'(101,-1); GOTO main_loop
0410 REM
0420 pasar:
0430 LET ven=101
0440 LET ven$=CTRL(1,ven,2); IF ven$="" THEN IF ven=101 THEN LET
     ven=ven+1; GOTO 0440 ELSE GOTO main_loop
0450 LET a$=CTRL(1,ven,3); PRINT (1)'LISTADD'(203-ven,-1,DEC(a$(1,2))),
     CTRL(1,ven,1)
0460 FOR a=LEN(ven$)-1 TO 1 STEP -2
0470 PRINT (1)'LISTDEL'(ven,DEC(ven$(a,2)))
0480 NEXT a
0490 GOTO main_loop
```

El último ejemplo de esta sección nos vuelve a mostrar los mismos tres contextos del programa anterior, con la diferencia de que ahora permitirá hacer el trasiego de nombres dentro del mismo contexto, o bien, desde un determinado contexto hacia cualquiera de los demás.

```
0001 REM "Ejerc8c Practica controlando tres contextos
0010 OPEN (1)"X0"
0020 PRINT (1)'SEMICHARS'
0030 FOR ven=0 TO 2
0040 IF ven=0 THEN PRINT (1)'WINDOW'(90,32,190,110,"Cantantes",$03$,$FFFFFEFF$
0040:)
0050 IF ven=1 THEN PRINT (1)'CONTEXT'(1),'WINDOW'(5,160,190,110,"Futbolistas",
0050:$03$,$FFFFFEFF$)
0060 IF ven=2 THEN PRINT (1)'CONTEXT'(2),'WINDOW'(202,160,190,110,"Ocupaciones
0060:",$03$,$FFFFFEFF$)
0070 REM
0080 PRINT (1)'BUTTON'(1,82,90,25,15,"Pasar",$$)
0090 PRINT (1)'LISTBOX'(101,10,10,80,80,"",$0400$)
0100 PRINT (1)'LISTBOX'(102,100,10,80,80,"",$0400$)
0110 REM
0120 ON ven GOTO cant,futb,ocup
0122 REM
0128 cant:
0130 PRINT (1)'LISTADD'(101,-1,"John","Paul","George","Ringo","Madonna","M.Jac
0130:kson","Sting","Pink Floyd","Aaaaa","Bbbbbb","Cccccc","Dddddd")
0140 PRINT (1)'LISTADD'(102,-1,"Maria","Jose","Miguel","Angel","Pablo")
0145 GOTO 0240
0150 REM
0160 futb:
0170 PRINT (1)'LISTADD'(101,-1,"Pele","Maradona","El Kaiser","Luis Garcia","Re
0170:inaldo","Teofilo Cubillas")
0180 PRINT (1)'LISTADD'(102,-1,"Gabelo","Ronald Gomez","Wilmer Lopez","Maurici
0180:o Montero","Rolando Fonseca")
0190 GOTO 0240
0200 REM
0210 ocup:
0220 PRINT (1)'LISTADD'(101,-1,"Mecanico","Bombero","Fontanero","Cirujano","Ab
0220:ogado","Chofer","Guarda")
0230 PRINT (1)'LISTADD'(102,-1,"Secretaria","Doctora","Miscelanea","Misionera"
0230:,"Visitadora Social","Maestra","Microbiologa","Supervisora")
0240 NEXT ven
0250 REM
0260 main_loop: DIM e$:TMPL(1)
0270 READ RECORD(1,SIZ=LEN(e$))e$
0280 PRINT e.context," ",e.code$,e.id," ",HTA(BIN(e.flags,1)),e.x,e.y
0290 IF e.code$="X" THEN PRINT (1)'DESTROY'; STOP
0295 LET context_act=e.context; PRINT (1)'CONTEXT'(e.context)
0300 IF e.code$="B" THEN GOTO pasar
0310 REM
0320 REM "Ctrl actividad en lista de la izquierda (id=101)
0330 IF e.id<>101 THEN GOTO 0370
0340 LET a$=CTRL(1,101,1),radio=DEC(CTRL(1,101,2)); PRINT a$,radio
0350 FOR ven=0 TO 2; PRINT (1)'CONTEXT'(ven); PRINT (1)'LISTUNSEL'(102,-1); IF
0350: context_act<>ven THEN PRINT (1)'LISTUNSEL'(101,-1)
0355 NEXT ven; GOTO main_loop
0360 REM
```

```
0370 REM "Ctrl actividad en lista de la derecha (id=102)
0380 IF e.id<>102 THEN GOTO main_loop
0390 LET a$=CTRL(1,102,1),radio=DEC(CTRL(1,102,2)); PRINT a$,radio
0400 FOR ven=0 TO 2; PRINT (1)'CONTEXT'(ven); PRINT (1)'LISTUNSEL'(101,-1); IF
0400: context_act<>ven THEN PRINT (1)'LISTUNSEL'(102,-1)
0405 NEXT ven; GOTO main_loop
0410 REM
0420 pasar:
0425 FOR context_fte=0 TO 2
0427 PRINT (1)'CONTEXT'(context_fte)
0430 LET ven=101
0435 REM "Obtiene indice de Ítems seleccionados
0440 LET ven$=CTRL(1,ven,2); IF ven$="" THEN IF ven=101 THEN LET ven=ven+1; GO
0440:TO 0440 ELSE GOTO 0445
0442 EXITTO 0448
0445 NEXT context_fte; GOTO main_loop
0447 REM
0448 PRINT (1)'CONTEXT'(context_act)
0450 LET a$=CTRL(1,ven,3,context_fte); PRINT (1)'LISTADD'(203-ven,-1,DEC(a$(1,
0450:2))),CTRL(1,ven,1,context_fte)
0455 PRINT (1)'CONTEXT'(context_fte)
0460 FOR a=LEN(ven$)-1 TO 1 STEP -2
0470 PRINT (1)'LISTDEL'(ven,DEC(ven$(a,2)))
0480 NEXT a
0490 GOTO main_loop
```

Cuando la cantidad de ítems a ser cargados en un LISTBOX, LISTEDIT, LISTBUTTON, o en un GRID es mucha, ocurre un cierto parpadeo en el objeto que resulta un poco molesto a la vista. Si le interesa hacer desaparecer el parpadeo, tiene que hacer uso de los mnemónicos **LISTSUSPEND** y **LISTRESUME.** Se coloca el primero exactamente antes de iniciar la carga de los ítems y el segundo al finalizar la carga.

Ejercicio de SYSGUI #9: Controles de barra de desplazamiento (Scroll Bars)

En ejercicios previos, hemos vistos como las barras de control pueden servir como una parte integral de otro control (ej: cajas de lista), y como pequeñas decoraciones de ventana (ver la parte oculta de una superficie dibujada). Ahora consideraremos el uso directo de barras de desplazamiento como un control único.

Antes de que pueda ser usada, una barra de desplazamiento tiene que ser asignada a un rango y una proporción. El rango especifica el ancho de la pista de scroll. La proporción especifica una fracción entera del ancho de la pista e indica cuánto se deslizará parte del control). El rango "reasigna un espacio" en la pista del scroll, semejante a la proporción que se coge del rango total. Por ejemplo, si el rango va desde 1 al 10 y la proporción es de 3, entonces la barra de desplazamiento puede tener 7 posiciones: del 1 al 7. Para tener una proporción de 3 y 10 posibles posiciones, sería necesario tener un rango con 3 valores mas en el borde superior: 1 hasta 13.

¡CUIDADO! El rango del scroll primeramente tiene que ser establecido, y luego la proporción.

El siguiente programa es el reporteador de eventos ya conocido, con nuevas líneas: 40, 41 y 42.

```
0010 OPEN (1)"X0"
0020 PRINT (1)'SEMICHARS'
0030 PRINT (1)'WINDOW'(100,100,100,100,"Reportador de Eventos",$03$,
0030:$FFFFFEFF$)
0040 PRINT (1)'HSCROLL'(101,10,10,80,10,"",$$)
0041 PRINT (1)'SCROLLRANGE'(101,1,13); REM Define inicio y final de la
0041:barra.
0042 PRINT (1)'SCROLLPROP'(101,3); REM El 3 indica el STEP.
0050 DIM E$:TMPL(1)
0060 READ RECORD (1,SIZ=LEN(E$))E$
0070 PRINT E.CONTEXT," ",E.CODE$, E.ID," ",HTA(BIN(E.FLAGS,1)), E.X, E.Y
0080 IF E.CODE$<>"X"GOTO 60
```

Corra el programa y observe el comportamiento. El tipo de evento 'p' es reportado siempre que la barra de desplazamiento es reposicionada. El campo .X retorna la nueva posición. Arrastrando el pequeño cuadro causa el reposicionamiento directo. Haciendo click en las flechas causa el movimiento de una unidad. Haciendo click en la pista a continuación del cuadrito causa que el cuadrito pase a "página arriba" o "página abajo", lo que significa que se mueve por la cantidad proporcional. 10 posiciones son disponibles, pero desde la proporción tiene 3, un rango desde 1 hasta 13 es requerido.

¡NOTA! Los límites del rango de scroll pueden ser cualquier valor desde el -32768 al +32767.

Interrumpa el programa y emita este mnemónico especial.

```
PRINT (1)'SCROLLPOS'(101,4)
```

Noten que la barra de desplazamiento salta inmediatamente a la posición 4. Ahora mueva la barra de desplazamiento con el ratón y lea su posición con la función de CTRL 2 ("Obtener Valor").

```
PRINT DEC(CTRL(1,101,2))
```

El parámetro $$ de la línea 40 podría ser sustituido por:

0000 para ENABLEar la ventana.
0001 para DISABLEar la ventana.
0010 inicialmente sea invisible.

El mnemónico 'VSCROLL' puede ser usado en el lugar de 'HSCROLL' para crear una barra de desplazamiento vertical. De otra manera, opera exactamente igual. Haga la siguiente sustitución en la línea 40.

```
0040 PRINT (1)'VSCROLL'(101,10,10,10,80,"",$$)
```

Ejercicio de SYSGUI #10: Controles de Edición multilínea

Para edición de texto más complicada, cuando una simple caja de edición no lo hace, Visual PRO/5 provee el control de edición multilínea. El control de edición multilínea permite la edición de una o más líneas de texto (llamémoslos "párrafos"), cada uno de los cuales pueden ocupar una o más líneas físicas, cada una hasta un máximo de 256 caracteres. El control puede tener un borde opcional, opcionales barras de desplazamiento, y pueden ser usados con cualquier fuente.

Para experimentar con el control de edición multilínea, vamos a utilizar el programa reporteador de eventos de nuevo. Solamente la línea 40 es nueva. Quite la 41 y la 42.

```
0010 OPEN (1)"X0"
0020 PRINT (1)'SEMICHARS'
0030 PRINT (1)'WINDOW'(100,100,100,100,"Reportador de Eventos",$03$,
0030:$FFFFFEFF$)
0040 PRINT (1)'TXEDIT'(101,10,10,80,80,"",$10$)
0050 DIM E$:TMPL(1)
0060 READ RECORD (1,SIZ=LEN(E$))E$
0070 PRINT E.CONTEXT," ",E.CODE$,E.ID," ",HTA(BIN(E.FLAGS,1)), E.X, E.Y
0080 GOTO 60
```

El parámetro de título en la línea 40 esta vacío, pero si alguno es proveído, será insertado como el primer párrafo de texto. La bandera 10 genera un borde alrededor del control. Esto ayudará a que usted vea donde están los límites.

Asegúrese de gastar algún tiempo experimentando con esto. Note que cuando el ratón se mueve sobre el control, este se convierte como una raya vertical para la entrada de texto. Si hace click con el ratón en el control, aparece un "signo de intercalación" que centellea, mostrando donde se procederá con la entrada de texto. Usted puede entonces comenzar a mecanografiar y va a ver que el control informa inicialmente una ganancia de foco ('f'), y con cada tecla que se oprima, reportará un cambio ('e').

Si usted entrara líneas largas, el scroll de la caja funcionará automáticamente, y podrá entrar tantas líneas como desee. La caja puede ser scroleada con las teclas de flecha, página arriba, página abajo, inicio y final, o arrastrando el ratón. Con el ratón usted puede arrastrar fuera una selección de texto, y la elimina o digita sobre el que ya existe. Usted puede también utilizar el teclado para las funciones estándar de Windows: ^C (copiar) ^X (cortar) y ^V (pegar). Posteriormente, vamos a ver como enganchar un menú de comandos sobre el panel de control.

Dos adiciones muy útiles al control son la capacidad de arrollar palabras y barras de desplazamiento. Vamos a añadir ambos.

¡SUGERENCIA! Si la opción de arrollar palabras estuviera encendida, no hay necesidad de definir una barra de desplazamiento horizontal. El texto no se saldrá del borde.

```
0040 PRINT (1)'TXEDIT'(101,10,10,80,80,"",$16$)
```

El valor de la banderas 16 es la combinación de la bandera de borde (10), la bandera de barra de desplazamiento vertical (04) y la bandera de arrollar palabras (02). Corra esta versión del programa y observe la diferencia en el comportamiento. Ahora, la caja nunca scrolea horizontalmente. Los párrafos largos simplemente doblan hacia más de una línea física, rompiendo

la línea al final de las palabras, hasta donde sea posible. Una barra de desplazamiento vertical permite navegación fácil si hay muchos párrafos.

Una variación adicional que es extremadamente útil es la opción de ventana-completa. Si usted creara un control de edición multilínea con un ancho y altura de 0, esto automáticamente hará que cambie de tamaño para ocupar el área completa de su ventana padre. !Inténtelo! (Nota: Cambiando las banderas a 06, desde un control de ventana-completa siempre tendrá un borde suministrado por la ventana padre. El control tendrá un borde que no es propio.)

```
0040 PRINT (1)'TXEDIT'(101,0,0,0,0,"",$06$)
```

En particular, trate de entrar líneas largas con muchas expresiones en ellas, luego cambie el tamaño de la ventana y vea como el arrollamiento de palabras redistribuye todo el texto.

Una vez que usted está a gusto con la operación del control de edición multilínea, interrumpa el programa e intente estas sentencias de modo inmediato.

```
PRINT (1)'TXADD'(101,0,"Nueva primer línea","Nueva segunda línea")
PRINT (1)'TXADD'(101,-1,"Nueva última línea")
```

Si usted practicó los ejemplos del Ejercicio #8, usted va a reconocer que la sintaxis para 'TXADD' es idéntica para el mnemónico 'LISTADD'. El tercero y subsiguientes parámetros a ser añadidos son opcionales. El segundo parámetro indica donde se puede añadir nuevo material. Un número de párrafo válido indica que el nuevo texto debería ir antes de ese párrafo, así, por medio de un 0 se indica añadir antes del párrafo 0, el primer párrafo. Los números de párrafo son basados en -cero- Si el segundo parámetro es -1, significa que serán añadidos nuevos párrafos al final.

Como 'LISTADD', 'TXADD' tiene una "forma larga", para adicionar grandes cantidades de texto directamente desde el canal. Ver el siguiente ejemplo y observe la coma al final de la primera línea. La segunda línea es un WRITE, no un PRINT.

```
PRINT (1)'TXADD'(101,0,3),
WRITE (1)"primero","segundo","tercero"
```

La forma larga de 'TXADD' prepara el canal para recibir un número de párrafos terminados con {linefeed} (en este caso, 3) afuera del parámetro de lista del mnemónico. Estos párrafos son recopilados y luego son insertados como un conjunto en el control. En este ejemplo, el número de párrafo antes del cual se va a insertar tuvo 0, de modo que las nuevas líneas son agregadas al comienzo.

La sentencia PRINT prepara el canal, y la coma al final de la línea es necesaria para suprimir el {linefeed} final que de otra manera sería impreso en el canal. En este caso, ese {linefeed} sería interpretado como el fin del primer párrafo (en blanco), si este fuera mandado a imprimir.

La sentencia WRITE entonces envía los tres {linefeed} de los párrafos. Usaremos WRITE en lugar de PRINT porque automáticamente añade un {linefeed} a cada ítem escrito.

```
PRINT (1)'TXAPPEND'(101,0,"xxx")
```

El mnemónico 'TXAPPEND' añade texto al final de un párrafo existente. Este ejemplo fallaría si al menos no existe un párrafo (número de párrafo 0) en el control.

```
PRINT (1)'TXDEL'(101,1)
```

Este mnemónico elimina un párrafo por su número. En este caso, el segundo párrafo (número de párrafo 1) es eliminado, si existiera. Si usted quisiera eliminar todos los párrafos, hágalo con 'TXCLR'.

```
PRINT (1)'TXCLR'(101)
```

Hay varias funciones de CTRL que obtienen información acerca del control de edición multilínea. Haga pruebas con cada una de ellas al menos dos veces, haciendo algunos cambios para el texto que haya dentro.

La función de CTRL 3 ("obtener cuenta") retorna el número de párrafos existentes en el control.

```
PRINT DEC(CTRL(1,101,3))
```

La función 1 ("obtener texto") retorna el texto del párrafo existente, sin un {linefeed} al final. El párrafo actual es el que muestra el signo de intercalación en la pantalla. Esto puede ser establecido desde el programa (vea 'TXSELECT' en su manual de referencias del Visual PRO/5), pero es más frecuentemente establecido por el usuario.

```
PRINT CTRL(1,101,1)
```

La función 7 ("obtener todo el texto") retorna el texto de TODOS los párrafos, con un {linefeed} después de cada uno, incluyendo el último.

```
PRINT CTRL(1,101,7)
```

La función 2 ("obtener valor") retorna el número del párrafo actual con un string binario de dos bytes.

```
PRINT DEC(CTRL(1,101,2))
```

Finalmente, hay una función de CTRL muy especial con la que se puede obtener el texto de cualquier párrafo por número. Esta es la función 5 ("obtener párrafo por número"). !Aquí se puede ver como trabaja!

```
PRINT CTRL(1,101,5,0)                    ; REM obtener primer párrafo
PRINT CTRL(1,101,5,1)                    ; REM obtener segundo párrafo
PRINT CTRL(1,101,5,DEC(CTRL(1,101,3))-1) ; REM obtener último párrafo.
```

¡NOTA! El cuarto argumento de la función CTRL es normalmente un contexto específico. Vamos a cubrir este detalle más adelante. El propósito es obtener información por medio de CTRL acerca de los ítems en cualquier ventana en cualquier momento, convenientemente. Sin embargo, la función 5 es un caso especial -- el cuarto argumento es el número de párrafo para la función 5 solamente.

Ejercicio de SYSGUI #11: Menús (con revisión 1.0x)

Las ventanas de SYSGUI pueden tener barras de menú. La forma de hacerlo es activando la bandera "tener barra de menú" (0800) cuando se crea la ventana, y luego usando el mnemónico 'SETMENU' para reemplazar el menú fijo asumido por omisión para su nuevo menú.

He aquí el programa reporteador de eventos del Ejercicio #4, el cual tiene la bandera "tener barra de menú" activada. También hemos cambiado el botón de pulsar en una caja de edición, para demostrar que el tablero del menú funcionará posteriormente. (sólo las líneas 30 y 40 han sido cambiadas).

```
0010 OPEN (1)"X0"
0020 PRINT (1)'SEMICHARS'
0030 PRINT (1)'WINDOW'(100,100,100,100,"Reportador de Eventos",$0803$,
0030:$FFFFFEFF$)
0040 PRINT (1)'EDIT'(101,10,10,50,15,"",$$)
0050 DIM E$:TMPL(1)
0060 READ RECORD (1,SIZ=LEN(E$))E$
0070 PRINT E.CONTEXT,"", E.CODE$, E.ID,"", HTA(BIN(E.FLAGS,1)), E.X, E.Y
0080 GOTO 60
```

Trate de correrlo así como está. Usted notará que obtiene el menú fijo asumido por omisión. Hasta contiene un ítem, que es chequeable. Intente operarlo y observe los eventos del comando de menú ('C'). El valor 04 en la bandera es la condición de chequeado, y aplica solamente para ítems de menús chequeables. El número que usted ve en el campo de ID indica que comando de menú es enviado. Los números en el menú asumido por omisión no tienen significado especial.

Obviamente, para hacer cualquier cosa útil vamos a necesitar nuestro propio menú. Trate de adicionar las siguientes líneas de código y córralo de nuevo. Considere la coma que va al final de la línea 41 y el uso de WRITE en lugar de PRINT en las líneas restantes. Note también el espacio antes la palabra "Exit" y las comas dentro de comillas.

```
0041 PRINT (1)'SETMENU',                ←Poner coma del final
0042 WRITE (1)"File,1,,"+$0A$+" Exit,2,,"+$0A$
```

El mnemónico 'SETMENU' prepara el canal para recibir un menú. El menú mismo consiste de una línea de texto por cada ítem del menú, indentado con espacios antepuestos para mostrar el nivel en la jerarquía. El menú es terminado con una línea en blanco. En el ejemplo mínimo de arriba, la barra del menú contendrá una opción única, la del menú de Archivo. El menú de Archivo activado contiene a su vez un ítem, con opción de Salir (Exit). Observe que " Exit" está indentado, para indicar que debe aparecer dentro de la opción "File" del menú. Finalmente, las comillas del final ("") proveen un {linefeed} extra para terminar el menú.

Cada ítem u opción del menú consiste de cuatro parámetros separados por comas.

Write (1)"Ítem, Etiqueta, Acelerador, Bandera"

- Item: generalmente se usa para el nombre que va a llevar la opción.
- Etiqueta: se usa para IDentificar la opción
- Acelerador
- Bandera: es para dar condiciones a la opción: S, C, U, D.

El primero es el ítem con el título del menú, con espacios antepuestos para especificar el lugar en la jerarquía de los ítems. El segundo campo es para el ítem de "etiqueta" (tag). La etiqueta es

justamente un número utilizado para identificar ese ítem. Los tags de menú son controles IDs muy usados, excepto que son usados de otro modo, de manera que la existencia de un control con el mismo número como una etiqueta de menú no provocará un problema. El tercer campo es donde cualquier tecla de acelerador puede ser especificada. El cuarto campo puede contener banderas especiales acerca de esos ítems del menú.

Corra el programa de nuevo y note la apariencia del menú "File", con el comando "Exit" debajo. Cuando el comando "Exit" es seleccionado, unos eventos tipo 'C' llegan con el campo de ID igual a 2, que es la etiqueta del comando "Exit". Cualquier número desde 1 al 31999 puede ser usado para tags o etiquetas de menú. Del número 32000 en adelante están reservados para uso de BASIS. Actualmente solamente 3 de esas etiquetas de menú son reservadas. Estos son los comandos para cortar (32027), copiar (32028) y pegar (32029) en el tablero.

Los ítems del menú pueden tener teclas de mnemónicos asociadas con estos. Los menús de mnemónicos son opciones por teclado para la navegación de menús. Vamos a añadir mnemónicos a nuestro ejemplo, y al mismo tiempo, vamos a añadir un menú de edición con comandos por teclado. En las siguientes líneas es muy importante recordar los espacios previos al nombre de cada opción.

```
0042 menu$="&File,1,,"+$0A$+" &Exit,2,,"+$0A$
0043 menu$=menu$+"&Edit,3,,"+$0A$+" Cu&t,32027,,"+$0A$
0043:+" &Copy,32028,,"+$0A$+" &Paste,32029,,"+$0A$
0044 WRITE (1)menu$
```

Con los mnemónicos a como se muestra, <Alt+F> nos mostrará el menú de archivos, y <Alt+E> nos dará acceso al menú de edición. Dentro de estas opciones, las que aparezcan con letras subrayadas harán funcionar los comandos de menú correspondiente. Haga una selección de texto en la caja de edición y córtelo y péguelo desde el menú. Las funciones del tablero siempre están disponibles con las secuencias de teclado estándar (^X para cortar, ^C para copiar, ^V para pegar), pero pueden ser hechas disponibles desde el menú por medio de los tags de menú estándar, como se muestra. Observe que el uso del tablero no genera eventos especiales.

Hasta el momento no hemos especificado alguna bandera en nuestro menú de ítems. Hay cuatro banderas disponibles:

S	Ítem es separador
C	Ítem es chequeable, e inicialmente chequeado
U	Ítem es chequeable, pero no chequeado inicialmente
D	Ítem es desactivado desde el inicio.

Vamos a alterar nuestro ejemplo de modo que el menú de archivo tiene un comando "Print" que desde el inicio aparece desactivado, un ítem "Paint" que será chequeable y aparecerá inicialmente chequeado, y un separador sobre la línea de Salida (Exit).

```
0042 menu$="&File,1,,"+$0A$+" &Print,4,,D"+$0A$+" Paint,5,,C"+$0A$
0042:" sep,6,,S"+$0A$+" &Exit,2,,"+$0A$
```

Si usted corriera esta versión, usted va a ver que ahora es posible chequear y des-chequear la opción de "Paint". Usted puede también ver el Comando "Print" desactivado, y el **sep**arador (declarado con la letra "S"). El nombre usado para el Ítem del **sep**arador no tiene importancia, y nunca se muestra.

Ahora vamos a interactuar con el menú utilizando directamente mnemónicos. Interrumpa el programa e intente estas instrucciones inmediatamente.

```
PRINT (1)'TITLE'(-4,"&Paint")
```

Ahora el comando "Print" nos trae el comando "Paint". Ningún espacio es usado debido a que el espacio no es realmente parte del título del ítem. Los espacios indican simplemente niveles de anidamiento de menús al mnemónico 'SETMENU'.

Para evitar confundir IDs de controles con tags de menú, mnemónicos y funciones CTRL que operan a la vez donde ambos usan el *negativo* de la etiqueta de menú. Este puede ser un número de control 4 también, pero no sería afectado por este comando. El -4 indica que la etiqueta de menú número 4 es la que va a ser usada.

Ahora vamos a habilitar ese comando.

```
PRINT (1)'ENABLE'(-4)
```

Si usted accesa el menú va a ver que el ítem no es gris por más tiempo.

Los mnemónicos 'CHECK' y 'UNCHECK' trabajan con ítems de menú chequeables. Pruebe con estos, atisbando en el menú antes y después de cada evento.

```
PRINT (1)'CHECK'(-5)
PRINT (1)'UNCHECK'(-5)
```

La función CTRL puede ser usada, como de costumbre, para obtener información acerca de menús y de ítems de menús. El más básico, aunque muy rara vez usado, es la función 6 de CTRL ("obtener menú"). La función 6 es desempeñada en la ventana padre y retorna el árbol del menú completo, en la sintaxis de 'SETMENU'.

```
PRINT CTRL(1,0,6)
```

Recuerde que el ID 0 se refiere siempre a la ventana completa. Esta función recuperará todo el menú, completo con el doble {linefeed} al final.

Para obtener información en ítems de menú individuales, usted puede usar la función 1 ("obtener texto") para obtener el título, y la función 2 ("obtener valor") para obtener la condición chequeado de ítems chequeables.

Para obtener el título del ítem número 4 del menú, usar la siguiente instrucción:

```
PRINT CTRL(1,-4,1)
PRINT CTRL(1,-1,1)          →    No devuelve nada.
```

Para ver si un ítem chequeable está chequeado en este momento, ponga lo siguiente.

```
PRINT DEC(CTRL(1,-5,2))     →    0=Sin chequear, 1=Chequeado.
```

Como usted puede ver, trabajar con menús es un poco tedioso, pero no difícil. Una vez que su menú está colocado, el menú individual de ítems funciona mucho más bonito con controles especiales. El menú puede ser alterado en pequeñas formas con simples mnemónicos (como cambiar títulos y deshabilitar ítems), o puede ser reemplazado completamente con 'SETMENU' en cualquier momento.

Ejercicio de SYSGUI #12: Las Barras de condición y las Colas para mensajes

La barra de condición es un control especial que le permite mostrar mensajes de texto en la parte inferior de una ventana. Para ver una barra de condición, vamos a utilizar una versión modificada de el programa "Hola, Mundo" del Ejercicio #1. Solamente la línea 45 es nueva.

```
0010 OPEN (1)"X0"
0020 PRINT (1)'SEMICHARS'
0030 PRINT (1)'WINDOW'(50,50,50,50,"Hola",$83$,$FFFFFEFF$)
0040 PRINT (1)'BUTTON'(1,5,15,40,0,"OK",$$)
0045 PRINT (1)'STATBAR'(102,0,0,0,0,"Click OK to exit",$$)
0050 DIM E$:TMPL(1)
0060 READ RECORD (1,SIZ=LEN(E$))E$
0070 END
```

La barra de condición aparece como un rectángulo en la parte inferior de la ventana. Por omisión este es coloreado con un gris claro y separado del resto de la ventana por una línea de negro. (Los colores pueden ser optimizados). La barra de condición ocupará siempre el ancho completo de la ventana, aunque la ventana es cambiada de tamaño (hágalo!), y siempre será lo suficientemente alta como para acomodar el tipo de letra (font) que utiliza. La fuente que se asume por omisión para una barra de condición es la que se tenga instalada en la máquina mas parecida a "Helvetica". (Esto, también, es optimizable). Desde el tamaño y ubicación de una barra de condición son determinados o bien, el usuario no tiene que especificar un rectángulo unificador cuando crea el Control. Como los controles de cajas de grupo y cajas de texto estático, las barras de condición también nunca generan eventos. Las barras de condición son muy útiles para guiar o informar al usuario de una condición actual o de lo que ellos deberían hacer después.

¡NOTA! Las barras de condición toman un área lejos de la ventana padre.

Las barras de condición son particularmente útiles cuando son combinadas con *colas*. Una cola es un par de mensajes que pueden ser asociados con un control. Si una cola está activa, con solo pasar el cursor del ratón sobre un objeto, causa que una ventana amarilla pequeña se aparezca con el primero de un par de mensajes, llamado la "**cola corta**". Si una barra de condición aplicable es localizada, el texto en la barra de condición es cambiado temporalmente por el segundo parámetro para mensajes, llamado la "**cola larga**". Si el ratón es movido o oprimido, la barra de condición obtiene su contenido anterior, y la ventana amarilla desaparece.

Vamos a añadir una cola a nuestro programa.

```
0046 PRINT (1)'CUE'(1,"OK button","Press this!")
```

Corra el programa y observe el comportamiento cuando el ratón es movido encima del pulsador y allí mantenido.

¡CUIDADO! Debido a la forma en que las colas sean implementadas, su uso puede provocar algunos efectos secundarios indeseables en el comportamiento de la interfase del usuario. Si el uso de colas es limitado a barras de herramienta (ventanas hijas se llenarán principalmente con botones herramientas), y esto casi nunca será objetable. Algunos efectos secundarios pueden ser que los

controles en una ventana que se sobrepone conteniendo colas, no podrían operar, y que las porciones de barras de desplazamiento de cajas de lista y los controles de edición multilínea no podrían responder al primer golpe en algunas circunstancias. Estamos trabajando para minimizar estos inconvenientes. Revise el archivo **readme** del Visual PRO/5 para obtener información más actual.

El Editor de Pantallas GUI

En esta sección vamos a aprender a confeccionar pantallas gráficas por medio del editor suministrado con el Visual PRO/5. Confeccionar una pantalla con esta herramienta implica que la producción de un programador sea bastante más rápida, ya que no se tiene que pasar por los trabajos de estar calculando ventanas, tamaños de objetos, ni posicionamientos.

El motivo por el cual no se les enseñó desde el principio a usar este editor, es para que conocieran más a fondo como interactúa el Windows con el Visual PRO/5. En las primeras páginas del manual se dijo que todo programa para consultas o captura de datos, en realidad no tiene ni una sola instrucción de INPUT de BBx, ya que en la realidad este trabajo es delegado a Windows.

Resource editor – Editor de la revisión 1.0x

Para crear un nuevo recurso (con la revisión 1.0x) debe darse click en el icono del editor de recursos. Esto hace que nos aparezca una ventana de mediano tamaño, en donde podemos apreciar todos los objetos o figuras que podemos utilizar con el editor. Se supone que al tener cierto fogueo con Windows, usted está familiarizado con todas esas figuras, y por eso no vamos a detallar mucho sobre ellas.

Si a usted le interesa diseñar un recurso más grande que esa pantalla, puede maximizarla haciendo click en el pequeño rectángulo de la esquina superior derecha.

El siguiente paso para la creación del recurso es hacer click sobre la palabra Windows ubicada en el menú de la esquina superior izquierda y posteriormente haciendo otra vez click sobre la opción de crear un nuevo recurso. Esto hace que nos aparezca un nuevo contexto cuadriculado, el cual se convertirá en la superficie que vamos a utilizar para colocar los objetos que sean de nuestro interés. Este contexto podemos moldearlo al tamaño que nos interese, corriendo con el RATÓN las líneas delimitadoras que nos interesen.

Tanto la ubicación de este contexto como sus líneas delimitadoras, podemos variarlas de posición colocándoles el cursor del RATÓN encima y arrastrándolo hacia donde nos interese.

Usted podrá ver que el contexto tiene un letrero o título en inglés que dice: "Untitled". Este podemos variarlo a lo que sea nuestra necesidad, al igual que otras características propias del contexto. Para lograrlo debemos colocar el cursor del RATÓN dentro del contexto y hacer un doble click. Hecho esto nos aparece una ventana con varios parámetros que podemos variar según sea la

necesidad. Para conocer las cualidades y usos de estos parámetros debe tomar ayuda del correspondiente manual del Visual PRO/5.

Ahora vamos a hacer un pequeño comentario de cada uno de los objetos o iconos que aparecen desplegados en la parta alta del editor de recursos. Al oprimir cualquiera de esos iconos estaremos activando su uso. Cuando eso es hecho, ocurre que hacia donde llevemos el puntero del MOUSE, estaremos en posibilidad de crear ese tipo de botón, en el entendido de que previamente ya ha sido creado el contexto o ventana GUI. Dicho de otra manera, no podemos crear un objeto sin tener un determinado contexto donde alojarlo.

Sepa también que los objetos que se detallan abajo, además de poder ser creados con el editor de recursos, también pueden ser creados por medio de mnemónicos en tiempo de corrida del programa. Inclusive, uno puede crear un contexto con sus respectivos objetos por medio del editor de recursos, y luego en tiempo de corrida del programa, podemos adicionarle nuevos objetos, y también modificar los previamente creados.

1) Puntero: En realidad este no es un objeto de Windows, Este icono debe ser constantemente utilizado para estar desactivando la condición de nuevas inserciones del objeto que tengamos activado.

2) Push Button: Se utiliza para que el usuario tome acciones como Continuar, Cancelar, Calcular, Terminar, Salir, etc. Se caracteriza porque al ser luego usado por el usuario, hace la simulación de que se hunde y retorna a la posición normal.

   Sintaxis.: PRINT (gui)'BUTTON'(id,x,y,w,h,title,flags)
   Ejemplo: PRINT (gui)'BUTTON'(1,5,15,40,0,"Ok",$$)

3) Check Box: Estas cajas de chequeo se utilizan para que el usuario pueda en una aplicación hacer indicaciones sobre lo que esté procesando. Ejemplo, al incluir un cliente podría preguntarse: Se le envía estado de cuenta, Necesario trámite de cobro con original, Se le carga flete al facturar, etc. Diciéndolo de otra manera, estas cajas hacen la función de afirmar o desmentir una condición.

   Sintaxis.: PRINT (gui)'CHECKBOX'(id,x,y,w,h,title,flags)
   Ejemplo: PRINT (gui)'CHECKBOX'(113,155,63,58,14,"Calcular",$$)

4) Radio Button: Funcionan de manera bastante similar a las Cajas de Chequeo, solo que generalmente estos se agrupan para poder indicar una sola condición de varias que podrían darse. Por ejemplo, podemos decir que una persona pertenece a una determinada raza pero no a todas a la vez, que es simpatizante de un determinado partido político pero no de varios a la vez, que pertenece a una determinada religión o doctrina, pero no a varias a la vez. Al ser definidos en grupos Windows controla que siempre haya uno activo y que al ser activado otro, los demás automáticamente son desactivados.

   Sintaxis.: PRINT (gui)'RADIOBUTTON'(id,x,y,w,h,title,flags)
   Ejemplo: PRINT (gui)'RADIOBUTTON'(105,464,32,62,18,"Y...",0005),;rem "Check/Disab

5) Horizontal Scroll Bar: Se utiliza en contextos un poco grandes, en donde por lo general los datos no caben en la pantalla y necesitamos desplazarnos hacia la derecha o la izquierda, para poder visualizarlos.

   Sintaxis.: PRINT (gui)'HSCROLL'(id,x,y,w,h,title,flags)
   Ejemplo: PRINT (gui)'HSCROLL'(101,10,10,80,10,"",$$)

6) Vertical Scroll Bar: Lo mismo del anterior, solo que en este caso el desplazamiento es hacia arriba y hacia abajo.

   Sintaxis.: PRINT (gui)'VSCROLL'(id,x,y,w,h,title,flags)
   Ejemplo: PRINT (gui)'VSCROLL'(101,10,10,10,80,"",$$)

7) Static Text: Se utiliza en aquellos casos en que necesitamos mostrar texto o rótulos dentro de un determinado contexto.

   Sintaxis.: PRINT (gui)'TEXT'(id,x,,w,h,title,flags)
   Ejemplo: PRINT (gui)'TEXT'(104,13,65,43,14,"Periodos",8000)

8) Edit Box: Son los objetos que necesitamos para digitación de datos. Se caracteriza porque su título va dentro de la caja.

   Sintaxis.: PRINT (gui)'EDIT'(id,x,y,w,h,title,flags)
   Ejemplo: PRINT (gui)'EDIT'(105,62,20,40,14,"0",$$)

9) Custom Edit Box: Este objeto nos permite la digitación de texto en una o varias líneas. Su título es asumido como parte inicial del texto a ser digitado.

   Sintaxis.: PRINT (gui)'TXEDIT'(id,x,y,w,h,title,flags)
   Ejemplo: PRINT (gui)'TXEDIT'(101,10,10,80,80,"Asunto: ",16)

10) List Box: Esta caja sirve para mostrar listas de ítems, en donde el usuario por lo general usa marcar uno o varios de estos, dependiendo esa acción de la opción que se le esté dando al usuario.

    Sintaxis.: PRINT (gui)'LISTBOX'(id,x,y,w,h,title,flags)
    Ejemplo: PRINT (gui)'LISTBOX'(103,16,94,120,224,"",0400)

11) List Button: Funcionan de una manera elegante. Por ejemplo: el usuario debe hacer una indicación de un nombre entre doce posibles opciones. La caja inicialmente deberá mostrar uno de todos los nombres, asumiéndolo a conveniencia por programación. Si el usuario desea otro deberá hacer click en la flecha que tiene el botón y automáticamente nos son desplegados los doce nombres, para que marquemos el que nos interesa.

    Sintaxis.: PRINT (gui)'LISTBUTTON'(id,x,y,w,h,title,flags)
    Ejemplo: PRINT (gui)'LISTBUTTON'(108,13,126,72,54,"",$$)

12) List Edit: Es muy similar al anterior, solo que este permite que además de elegir entre los que tengamos almacenados en la lista de ítems, podamos optar por digitar otro diferente al que indiquemos en 'title'.

    Sintaxis.: PRINT (gui)'LISTEDIT'(id,x,y,w,h,title,flags)
    Ejemplo: PRINT (gui)'LISTEDIT'(112,105,127,37,53,"Azul",$$)

13) Group Box: Esta figura no sirve para mostrar ni ingresar datos. Su uso está dirigido para que dentro de su contorno coloquemos todos los Radio Botones que necesitemos para controlar una determinada condición.

    Sintaxis.: PRINT (gui)'GROUPBOX'(id,x,y,w,h,title,flags)
    Ejemplo: PRINT (gui)'GROUPBOX'(122,3,84,159,66,"Opciones",$$)

14) Tool Button: Este botón funciona de una manera similar al Push Button. La diferencia es que es más especial porque permite que dentro de su superficie podamos desplegar algún literal o archivo .BMP. Ver más características en sección Ejercicio de SYSGUI #6.

    Sintaxis.: PRINT (gui)'TBUTTON'(id,x,y,w,h,title,flags)
    Ejemplo: PRINT (gui)'TBUTTON'(108,13,126,72,54,"BITMAP=/windows/Waves.bmp",$$)

15) Status Bar: Es utilizada para mostrar mensajes de ayuda o de interés al usuario del programa.

    Sintaxis.: PRINT (gui)'STATBAR'(id,x,y,w,h,title,flags)
    Ejemplo: PRINT (gui)'STATBAR'(115,0,0,0,0,"Mover el puntero sobre los controles para obtener ayuda",$$)

El uso de estos objetos es algo que debemos aprender a dominar bien, ya que de eso depende bastante que una aplicación nos funcione bien o no.

Podemos observar que conforme vamos agregando objetos en un contexto, automáticamente el editor de recursos les va asignando un número consecutivo que inicia con el 101, para que luego ya dentro de las líneas del programa, podamos referirnos a estos por medio de ese identificador.

Cualquier objeto ya colocado dentro del contexto puede ser movido, eliminado o modificado. Para eso primero debemos tocar el botón del Puntero (#1) y luego el objeto que necesita ser intervenido. Si le damos click una sola vez podemos moverlo o eliminarlo. Si necesitamos modificarlo es necesario hacer un doble click sobre este.

El título que inicialmente se le asume a la mayor parte de esos objetos al ser creados ("Untitled"), puede ser variado desde un inicio, dando doble click sobre el objeto y escribiendo el título adecuado. Luego, en tiempo de corrida del programa, esos títulos pueden ser variados por medio del mnemónico 'TITLE'.

Para salvar los cambios hechos se oprime el botón OK. Usar Cancel para salirse sin salvar nada.

Es importante recordar que dentro de un archivo de recursos podemos guardar varios contextos, el cual luego con el verbo RESGET podemos seleccionar, a como se indica en el programa al final de esta sección.

ResBuilder – Editor de la revisión 2.0x

Este utilitario está disponible desde la revisión 2.0x de Visual PRO/5 y permite la creación y modificación de pantallas gráficas, de una forma bastante más fácil para el programador.

Al ponerlo en funcionamiento es más cómodo si desde el inicio es maximizado, para tener así una mayor amplitud en el área de trabajo.

De los dos recuadros que aparecen, en el izquierdo podemos apreciar cuatro iconos con las opciones que se pueden realizar:

- Form → Para crear nuevas pantallas.
- Child Window→ Para crear ventanas hijas.
- Menu → Para crear menús.
- Image List → Para crear listas de imágenes.

Cómo crear una nueva pantalla?

- Posicionamos el puntero del mouse en el icono Forms.
- Apretamos el **botón derecho** del mouse.
- Seleccionamos agregar (add) y nos aparece el contexto vacío para la nueva pantalla, junto con un recuadro que nos muestra las propiedades del contexto. Ahí podemos cambiar el Título del contexto, el ID, el nombre con que le identificará en el árbol que se va formando en el recuadro izquierdo, el color de fondo del contexto y todas las demás propiedades que ahí se muestran. Haciendo click en la etiqueta **Flags** que aparece dentro de las propiedades de la forma, podemos activar o desactivar algunas cualidades que tienen los contextos:

  - Allways on top → Dejarlo chequeado si queremos que el contexto se mantenga fijo.
  - Close box → Chequeado si queremos que el contexto tenga su cajita de cierre.
  - Custom color palette→ Chequearlo si queremos que el usuario cambie los colores del contexto a su criterio.
  - Dialog Behavior → Dejarlo siempre chequeado.
  - Dialog Border → Chequeado si queremos que el contexto mantenga su borde.
  - Disabled → Chequearlo si queremos declararlo deshabilitado.
  - Enter as Tab → Chequearlo si queremos que el ENTER funcione igual que el Tab.
  - Gravity → Dejarlo chequeado si queremos que luego el usuario del programa pueda arrastar la pantalla.
  - Horizontal scroll bar→ Chequearlo si queremos que tenga una barra de scroll horizontal.
  - Initially maximized → Chequearlo si queremos que inicialmente aparezca maximizado.
  - Initially minimized → Chequearlo si queremos que inicialmente aparezca minimizado.
  - Invisible → Chequearlo si queremos que inicialmente aparezca invisible.
  - Keyboard navigation→ Chequearlo si queremos permitir navegación de teclado.

- Manage syscolor event
- Minimizable → Chequearlo si queremos que luego lo puedan minimizar.
- No title bar → Chequearlo si queremos que no muestre el título del contexto.
- Sizable → Chequearlo si queremos que el usuario lo pueda cambiar de tamaño.
- Vertical scroll bar → Chequearlo si queremos que tengan una barra de scroll vertical.

- En la parte inferior izquierda de la pantalla, aparecen tres iconos con un pequeño enrejado.
  - El de la izquierda nos sirve para llenar el contexto de unos puntitos que luego nos servirán para guiarnos en la colocación de los objetos que va a contener el contexto.
  - El del centro sirve para ayudarnos con el alineamiento automático de los objetos.
  - El de la derecha sirve para ayudarnos a que objetos de una misma especie, puedan ser hechos de un solo tamaño.

  Conforme vayamos agregando objetos veremos como su referencia también va apareciendo en el árbol del recuadro izquierdo, con su tipo de icono, su ID y su nombre. A manera de consejo, se recomienda que los ID de las etiquetas sean ciento y resto (1xx), mientras que los ID de las cajas para edición sean doscientos (2xx), de manera que haya concordancia entre ambos. Por ejemplo que 121 y 221 sean para el campo llamado CUENTA. Recuerde que si en el futuro necesitara variar esta numeración, es posible hacerlo en las propiedades del objeto, con el gran cuidado de no utilizar un ID asignado a otro objeto.

  Para el texto de las etiquetas (Static text) se recomienda que la primer letra sea mayúscula y minúsculas el resto del texto. Eso le da una mejor estética a la pantalla y muestra la delicadeza de quien la hizo.

ResBuilder: Barra de objetos disponibles:

Normalmente esta la vemos en forma alargada horizontal en la parte superior izquierda de la pantalla. Ahí nos muestra todos los objetos que podemos utilizar para colocar en el contexto que vamos a crear. Esta barra, si se desea, podemos cambiarla de posición con solo arrastrarla a la zona que nos resulte más cómoda.

Los iconos de objetos mostrados en la barra son:

- **Pointer** (flecha) para des-seleccionar cualquier objeto de la barra previamente seleccionado.
- **Static text** (ABC) para poner etiquetas de texto relacionado a un objeto que contendrá datos.
- **Edit box** (abc) sirve para colocar los objetos o cajas que luego contendrán información, sea para despliegue o digitarla. Su contenido se puede variar con el mnemónico 'TITLE'. A manera de consejo, se recomienda que en las propiedades su Nombre (Name) sea puesto, para que en el árbol de objetos que se va formando en el recuadro izquierdo de la pantalla, aparezcan con dicho nombre, ojalá el mismo que corresponde al campo en el diccionario.
- **Check Box** (X) sirve para colocar botones chequeables, o sea, para que el usuario del programa luego pueda condicionar cierta información.
- **Radio Button** (O) sirve para colocar radio botones, los cuales luego servirán para que el usuario del programa pueda escoger una de entre al menos dos opciones agrupadas. Estos grupos se numeran para poder luego identificar los Radio Botones de cada grupo. Como una regla, en cada

grupo siempre debemos indicar cual de los Radio Botones es el que inicialmente va a quedar activado. Los Radio Botones tienen un funcionamiento un poco similar a los Check Box.

- **Push Button** ([ ]) sirve para colocar botones que luego servirán al usuario del programa para activar botones con funciones como Cancelar, Continuar, Imprimir, etc. El nombre de esta función se debe escribir en el campo para Texto de las propiedades del Botón.
- **Group Box** ([ABC] sirve para agrupar juegos de Radio Botones. Se muestra en la pantalla como un recuadro con un título, que adorna y agrupa un juego de Radio Botones.
- **Custom Edit** [=] sirven para colocar cajas que permitirán mostrar o ingresar información que es un poco extensa, sea de una o varias líneas. Funciona como un pequeño editor de texto multilínea en donde cada línea de texto está restringida a una longitud máxima de 256 caracteres.
- **List Box** sirve para colocar objetos que luego permitirán al usuario escoger o seleccionar con el mouse, uno o varios ítems de la lista que contenga. Para este tipo de objeto se recomienda usar un tipo de letra (font) de tamaño fijo como `Courier new`. Si la cantidad de ítems sobrepasa la capacidad de despliegue dibujado, de manera automática le aparece una barra de scroll vertical.
- **List Button** sirve para colocar objetos que contienen una flecha en su costado derecho, la cual al ser activada, hace que la caja se abra hacia abajo para mostrarnos una lista de ítems disponibles. Al dibujarlo, hay que ampliarlo hacia abajo, para que de esta manera señalemos hasta donde 'se abrirá' en el despliegue que hará cuando el usuario del programa lo active.
- **List Edit** es muy similar al anterior, solo que este permite digitación en su caja, lo cual sirve como patrón de búsqueda entre los ítems cargados en la lista del objeto, o bien, para asumir el dato que sea digitado.
- **Vertical Scroll Bar** sirve para colocar un objeto que nos muestra una barra de scroll vertical.
- **Horizontal Scroll Bar** sirve para colocar un objeto que nos muestra una barra de scroll horizontal.
- **Grid Control** sirve para colocar objetos que proporcionan al usuario un cuadriculado que suministra el mismo efecto de estar trabajando con una hoja electrónica, pero restringido a la configuración que el programador le asigne. Es especial para programas que necesiten virtualizar o mostrar todos los registros de un determinado archivo, de manera que el usuario del programa pueda navegar y hacer las modificaciones que si lo condicionáramos, automáticamente podrían repercutir sobre el archivo involucrado. El "grid" puede contener tanta data como sea posible en la cantidad de memoria virtual que contenga la computadora. Es viable tener varios miles de líneas con poca memoria y puede llegar a varios millones de líneas si tiene suficiente memoria. El "grid" puede simplificar grandemente el manejo de "scrolling" y de edición de campos individuales. Varios de los utilitarios que incluye VP5 2.xx como el "Configurator", usan el "grid" y le pueden servir de ejemplo.
- **Tab Control** sirve para colocar objetos que dan la sensación de estar trabajando con varias carpetas, en donde para cada una se muestran los objetos necesarios para la aplicación. Dentro de cada Tab se puede colocar todo el resto de objetos aquí definidos.
- **Horizontal line** sirve para el trazado de líneas horizontales, donde el programador lo indique con el mouse, en el contexto que está creando.
- **Vertical line** sirve para el trazado de líneas verticales, donde el programador tenga la necesidad de hacerla.
- **Image** sirve para colocar recuadros que luego servirán para mostrar el contenido de los archivos .BMP.
- **INPUTE** [ABC] sirve para colocar cajas que sirvan para la digitación de valores alfanuméricos, especialmente cuando se necesita enmascarar el dato a ser entrado, igual a como se hace en ambiente de caracteres.
- **INPUTN** [123] sirve para colocar cajas que servirán para la digitación de valores numéricos, que de fijo necesiten ser enmascarados. Por ejemplo montos, números de cuenta, etc.

- **Child Window** sirve para la creación de ventanas hijas en un contexto principal.
- **Tool Button** ([ ]) son botones que funcionan de manera muy similar a un Push Button. Además de otras diferencias, se caracteriza porque el Tool Button permite que en su superficie podamos desplegar un archivo .BMP.

ResBuilder: Propiedades de los objetos:

Todos los objetos, al igual que el contexto donde residen, cada uno tiene propiedades que podemos variar según nuestras necesidades. Para lograrlo colocamos el puntero del mouse sobre el objeto deseado y ahí damos un <Click> izquierdo, para que así nos aparezca el recuadro con las propiedades, dentro de las cuales tenemos:

- **Control Type** → Solo nos indica el tipo de control.
- **Control ID** → Nos muestra el número con que ResBuilder identifica dicho objeto. Si lo deseamos, podemos variarlo siempre y cuando otro no vaya a repetirse con el nuevo número.
- **Name** → Este va apareciendo en el árbol de figuras que a la izquierda se va formando.
- **Text** → Sirve para que en algunos objetos como los de Texto Estático, podamos indicar el texto que va a permanecer fijo, con el tamaño y tipo de letra que indiquemos.
- **Initial Content** → Permite que desde un inicio el objeto aparezca con el texto que aquí definamos, con las características de alineado que se ofrecen.
- **X position** → Corresponde al número de línea dentro del contexto, en donde la esquina superior izquierda del objeto está posicionado.
- **Y position** → Corresponde al número de columna dentro del contexto, en donde la esquina superior izquierda del contexto está posicionado.
- **Width** → Indica la longitud o ancho del objeto.
- **Height** → Indica la altitud del objeto.
- **Fore color** → Sirve para dar un color deseado a las letras que se desplieguen dentro del objeto.
- **Back color** → Sirve para poner un determinado color al fondo del objeto.
- **Font** → Sirve para dar un tamaño y tipo de letra diferente al que desde un inicio se asume.
- **Short Cue** → Es una pequeña etiqueta amarilla con un mensaje corto, que se muestra en tiempo de corrida del programa, cuando el usuario coloca el puntero del mouse sobre el objeto.
- **Long Cue** → Es una etiqueta más grandecita que la anterior, con un mensaje un poco más largo, que también se muestra en el contexto, al colocar el puntero del mouse sobre el objeto.
- **Disabled** → Permite que el objeto inicialmente aparezca habilitado o deshabilitado.
- **Invisible** → Permite que el objeto inicialmente aparezca visible o invisible.
- **Client Edge** → Para que el objeto se vea como hundido.
- **Raised Edge** → Para que el objeto se vea como sobresaltado.
- **In Group** → Necesario para agrupar Radio Botones y Tabs.
- **Justification** → Se utiliza en algunos objetos para indicar si el Texto inicialmente definido en las propiedades, se alinea hacia la izquierda, a la derecha o al centro.
- **Button Group** → Se utiliza con los Radio Botones para indicar el número de grupo al que pertenece cada Radio Botón.
- **Text Left** → Se utiliza en con los Check Box o con los Radio Botones, para indicar si el Check o el Radio Botón se alinean a la izquierda o a la derecha de la etiqueta.
- **Checked** → Se utiliza con los Check Box o con los Radio Botones, para indicar si desde un inicio se muestra activados o desactivados.
- **Password Entry** → Condición que debemos activar cuando en el objeto luego van a digitarse caracteres como el de un password o clave secreta.

- **Flags** → Sirve para condicionar algunas cualidades del objeto como:
  . Border
  . Ignore Tabs.
  . Word Wrap.
  . Horizontal Scrollbar.
  . One Paragraph.
  . Overstrike Mode.
  . Read Only → No permitir edición.
  . Vertical Scrollbar
- **Max Par Length** → Sirve para que indiquemos la longitud máxima que pueda tener el texto en un objeto Custom Edit. El máximo soportado es 256 caracteres.
- **Styles** → Similar a Flags, sirve para condicionar algunas cualidades de los Tabs.

ResBuilder: Alineamiento de objetos:

Si tenemos definidos un grupo de objetos de una misma especie, los cuales no nos han quedado bien alineados en la pantalla, o los necesitamos de un mismo tamaño, o el espacio de separación entre uno y otro no es uniforme, podemos valernos de la ayuda que el ResBuilder nos ofrece para estos efectos.

Sabemos que con solo hacer clic sobre cualquier objeto, queda marcado para que luego podamos 'arrastrarlo' con el mouse para ubicarlo donde mejor nos convenga. De igual manera, cuando el objeto está marcado, podemos variar su tamaño, 'jalándolo' por el extremo que necesitemos estirar o encoger.

Para emparejar varios objetos en su tamaño o separación existe en la parte baja inferior de la pantalla del ResBuilder, una Barra de Herramientas para Alinear Objetos. La explicación para el uso de estos botones se detalla en el otro manual que se la ha dado, en la página 26 de la sección para ResBuilder. Para realizar este tipo de alineamiento, primero debemos mantener oprimida la tecla <Ctrl> y luego vamos dando <Clic izquierdo> sobre los objetos que requieran el alineamiento. Para todos los tipos de alineamiento que requiera, debemos considerar que el primero que marquemos será el que se utilice como machote para los demás. Una vez que todo el grupo de objetos ha sido marcado, soltamos el botón <Ctrl> y en la Barra de Herramientas damos clic al botón para alineamiento que estemos ocupando. Estando marcado todo el grupo de objetos, también podemos arrastrarlo como un todo, hasta reubicarlo donde mejor convenga. Para lograr este arrastre hay que soltar la tecla <Ctrl>, volver a mantenerla apretada y con el mouse arrastramos el conjunto de objetos al lugar deseado.

Cómo lograr el despliegue de los archivos .BRC ?

Después de haber utilizado el ResBuilder, la pantalla dibujada queda salvada en un archivo que se identifica porque su nombre termina con **.BRC**.

Los datos salvados en un archivo **.BRC** no contienen código de programación. Solamente sirven para que luego, hagamos su despliegue por medio de los mnemónicos que aparecen en negrita en el siguiente ejemplo de programa:

```
0010 REM "Programa que despliega Recursos
0020 BEGIN ; INPUT 'LF',"Dar nombre del archivo .BRC a desplegar: ",recurso$;
0020:IF recurso$="" THEN STOP
0030 IF POS("."=recurso$)=0 THEN LET recurso$=recurso$+".BRC"
0040 LET a%=RESOPEN(recurso$,ERR=no_existe)
0050 LET a$=RESGET(a%,1,1); REM "Tercer valor indica el ID del contexto
0060 RESCLOSE (a%)
0070 LET gui=UNT; OPEN (gui)"X0"
0080 PRINT (gui)'RESOURCE'(LEN(a$)),a$
0090 DIM e$:TMPL(gui)
0100 READ RECORD(gui,SIZ=LEN(e$))e$
0110 STOP
0120 REM
0130 no_existe:
0140 INPUT (0,SIZ=1)"No encuentro ese archivo .BRC (Dar ENTER) ",*; GOTO 0020
0150 END
```

Explicación de lo que ocurre en algunas de las líneas del programa anterior:

0020 y 0030 Sirven para obtener el nombre del recurso.
0040 Se abre el archivo con el recurso indicado. De no existir bifurca a indicar que NO_EXISTE.
0050 Se carga los datos del recurso en la variable A$.
A como se indica en el REM de la línea 0050, en un mismo archivo de recursos, podemos guardar varios contextos. Habíamos dicho que un contexto es igual a una determinada pantalla gráfica. En este caso se está utilizando el contexto número uno.
0060 Se cierra el canal usado para leer el recurso.
0070 Se abre el canal para el SYSGUI.
0080 Se hace el despliegue del recurso cargado en A$.
0090 Se genera el TEMPLATE con que van a ser leídos los **e**ventos desde el SYSGUI.
0100 Este READ RECORD es el que nos suministra los **e**ventos reportados por SYSWINDOWS.

Debe tenerse presente que la ruta del archivo **.BRC** hay que darla completa, ya que el RESOPEN no utiliza los directorios definidos en el PREFIX.

Eso es todo. Lo único que quedaría por hacer es el control y manejo de los eventos que nos sean reportados. Recordemos que los eventos se originan de las acciones que tome el usuario del programa, sean echas por medio del RATÓN o por medio del teclado.

Mnemónico GRID – Crear un Control Grid

Sintaxis

'GRID'(id,x,y,w,h,flags$,initrows,initcols,maxcols{,colheadheight,colheadid} {,rowheadwidth,rowheadid})

Descripción

El mnemónico 'GRID' sirve para crear un control grid. Es mejor crear grids y todos los otros controles en un archivo de recursos que por medio de mnemónicos. Usando ResBuilder para definir un grid, atributos como los encabezados de columnas pueden ser puestos sin escribir código de Visual PRO/5 y pueden ser cambiados sin modificar el código. Todos los atributos de un grid pueden ser cambiados en tiempo de corrida del programa, incluyendo el número de líneas, sin que se pierda la flexibilidad.

En versiones previas a la revisión 2.10 de Visual PRO/5, la bandera usada para desplegar líneas verticales es 0020. Para la revisión 2.10 y posteriores, debe usarse la bandera 8000. Cuando necesite migrar aplicaciones a revisiones previas es necesario cambiar la bandera $0020 a 8000 o también, activando el byte 7 de SETOPTS con el bit 08.

Parámetro	Descripción
Id	Un entero positivo que debe ser usado para identificar el grid. Este número debe ser único entre los controles en la ventana en uso.
x	Posición horizontal de la esquina superior izquierda del grid, basado unidades de la escala asignada al CONTROL, relativas al interior de la ventana que lo contiene.
y	Posición vertical de la esquina superior izquierda del grid, basado unidades de la escala asignada al CONTROL, relativas al interior de la ventana que lo contiene.
w	Ancho del grid entero basado en unidades de la escala asignada al CONTROL.
h	Ancho del grid entero basado en unidades de la escala asignada al CONTROL.
flags$	Puede ser un string vacío o un string binario de dos bytes compuesto de la suma de algunos de los siguientes valores hexadecimales:

Parámetro	Descripción
0001	Establece que el control grid inicialmente esté deshabilitado.
0002	Crea un control grid manejado como una columna de encabezado.
0004	Crea un control grid manejado como una línea de encabezado.
0008	Permite al usuario cambiar de tamaño las columnas del grid.

0010 Establece que el grid inicialmente sea invisible. Luego, el grid puede hacerse visible con este comando:

```
print(sysgui)'show'(id)
```

0020 Mostrar líneas verticales entre columnas en Visual PRO/5 2.0x. En Visual PRO/5 2.10 y posteriores, usar la bandera 8000 para mostrar líneas verticales. Por omisión de este valor en Visual PRO/5 2.10 y posteriores ocurre que no salen las líneas. Sin embargo, en las revisiones anteriores se quiere compatibilidad, el bit 08 en el byte 7 debe ser activado, lo cual causará que las líneas verticales sean mostradas.

0040 Mostrar líneas horizontales entre líneas.
0080 Incluir una barra de scroll horizontal en el control.
0100 Incluir una barra de scroll vertical en el control.
0400 Crear un botón de intercambio para "click on, click off".
0800 Crear un borde tridimensional pausado alrededor del control. El efecto que notará es que el cuadriculado se ve como hundido dentro del rectángulo, reduciendo levemente el área visible del grid.
1000 Crear un borde tridimensional levantado alrededor del control. El efecto que notará es que el cuadriculado se ve como sobrepuesto en todo el rectángulo, reduciendo levemente el área visible del grid.
8000 Mostrar líneas verticales entre las columnas.

En revisiones de Visual PRO/5 previas a la 2.10, la bandera para esta característica es 0020. Cuando se migre desde versiones previas, es necesario cambiar la bandera por 8000 o cambiar en el byte7 de SETOPTS el bit 08.

initrows Número inicial de líneas en el grid.
Initcols Número inicial de columnas en el grid.
maxcols Máximo número de columnas que el grid podría llegar a tener. El número de columnas puede ser modificado en tiempo de corrida. Definiendo más columnas de las necesarias en este parámetro podría hacer que se agote la memoria.
colheadheight Párametro opcional para definir el alto de los encabezados de las columnas en pixeles. Solo funciona cuando la bandera 0004 está activa.
colheadid Parámetro opcional para definir un único control ID para el grid que muestra los encabezados de las columnas. Solo funciona si la bandera 0004 es activada.
rowheadwidth Parámetro opcional para definir el ancho de los encabezados de las líneas en pixeles. Solo funciona cuando la bandera $0020 está activa.
rowid Parámetro opcional para definir un único control ID para el grid que muestra los encabezados de las líneas. Solo funciona cuando la bandera 0002 es activada.

Ejemplos

El código que sigue crea un control grid con el ID 100 que contendrá tres líneas y tres columnas, con líneas verticales y horizontales, efecto de cuadriculado sobresaltado (raised edge), y sin encabezados de columna ni de línea:

```
sysgui=unt; open (sysgui)"X0"
print (sysgui)'window'(120,150,550,100,"Grid Ejemplo",$0082$,$FF$)
print (sysgui)'grid'(100,20,20,500,54,$9040$,3,3,5)
escape
```

El siguiente código crea un control grid cuyo ID es 100, el cual tendrá tres líneas y tres columnas, con líneas verticales y horizontales, efecto de cuadriculado hundido (client edge), y con encabezados de columna y de línea. Observe que en este ejemplo, los vínculos del rectángulo son un poco más extensos que el ejemplo de arriba y que el cuerpo del grid no está truncado:

```
sysgui=unt; open (sysgui)"X0"
print (sysgui)'window'(80,300,650,120,"Grid Ejemplo",$0082$,$FF$)
print (sysgui)'grid'(100,20,20,580,74,$8846$,3,3,5,18,101,30,102)
escape
```

Esta cualidad del Visual PRO/5 nos permite interactuar con diferentes Bases de Datos, de manera que fácilmente podemos manejar información sin hacer uso de los tradicionales verbos del BBx, sino que por medio del famoso SQL, lenguaje fabricado hace muchos años por I.B.M. y adoptado luego como un estándar para diferentes manejadores de Bases de Datos.

Una base de datos es con conjunto de datos almacenados como "un todo", en donde la información se guarda en tablas de datos compuestas de columnas y líneas. La información fácilmente puede ser accesada mediante diferentes ordenamientos y relacionarse entre las diferentes tablas de datos.

Generalmente para toda base de datos existe un software llamado 'driver' u ODBC, el cual le permite a 'paquetes o lenguajes' externos poder accesar sus datos y ese es precisamente el medio que utiliza el Visual PRO/5 para ofrecer esta ventaja. Por lo tanto, sin existe el ODBC para una base de datos externa, podemos decir que tanto Visual PRO/5 como otros, podrán entrar a dicha base de datos.

Basis International también a fabricado su propio ODBC, para que desde otros lenguajes o paquetes puedan accesar la Base de Datos del Visual PRO/5. Digo Base de Datos del Visual PRO/5, porque si quisiéramos, podemos seguir trabajando bajo el esquema de archivos, a como siempre lo hemos hecho, y si quisiéramos, también podemos seguir trabajando los datos propios o nativos del Visual PRO/5 como una Base de Datos completamente relacional, en donde ya no es necesario estar dando OPEN a cada archivo, sino que se da un solo OPEN a la Base de Datos, como "un todo" y por una sola vez.

Entiéndase bien, que el ODBC de Basis no es necesario para que el Visual PRO/5 pueda manejar la Base de Datos propia. Este ODBC es necesario solamente para otros lenguajes o paquetes puedan accesar la Base de Datos del Visual PRO/5. Los únicos ODBC necesarios para el Visual PRO/5 son aquellos propios de la Base de Datos externa que quisiéramos accesar.

Dentro del panel de control del Windows, existe uno o dos iconos para ODBC. Uno es para ODBCs de 16 bits (Windows 3.1 y 3.11) y el otro para ODBCs de 32 bits (Windows 95, 98 y NT). Resulta siempre más rápido el desempeño con los de 32 bits. Al hacer doble click sobre el icono de los ODBC nos aparece un contexto que nos muestra las diferentes Bases de Datos declaradas en esa máquina. Un mismo ODBC perfectamente puede ser utilizado para accesar diferentes Bases de Datos en una máquina. Por ejemplo: podrían tenerse varias bases de datos de FOX y uno necesitaría decir en alguna parte cuál es la que le interesa. Sobre los ODBC ya no vamos a hablar más, ya que eso es tema aparte del que ahora nos interesa.

Parte de lo que se quiere dar a entender en el párrafo anterior, es que las Bases de Datos externas se pueden ver definidas en el correspondiente icono en el panel de control del Windows. Claro debe quedar entonces, que si se opta por trabajar bajo el concepto de una Base de Datos, es completamente necesario entender que debemos contar con un Diccionario de Datos, lo cual nos garantiza un mejor control y ordenamiento de la información almacenada, mayor facilidad para el desarrollo de aplicaciones y su posterior mantenimiento.

Si nuestro deseo es trabajar con la Base de Datos del Visual PRO/5 como tal, inicialmente necesitamos definirla en un archivo de texto llamado **sql.ini**, definido en el mismo directorio donde

está el ejecutable del Visual PRO/5 (vpro5.exe), el cual estará apuntando hacia un archivo config.TPM por cada base de datos ahí declarada.

El siguiente es un ejemplo del contenido de un archivo **sql.ini**

```
; Top of config file          <- Esta línea es un comentario opcional


[BASIS Data Sources]          <- Esta línea debe ser escrita exactamente así.
Chile Company                 <- Nombre de la primer base de datos.
CD-Store                      <- Nombre de la segunda base de datos.
Cuentas por Cobrar            <- Nombre de otra base de datos.
                              <- Esta línea en blanco es obligatoria.
[Chile Company]               <- Declaración de ubicación de la primer Base.
CONFIG=C:\BASIS\TOOLS\CHILEDD\config.tpm
                              <- Línea en blanco obligatoria.
[CD-Store]                    <- Declaración de ubicación de la segunda Base.
CONFIG=C:\BASIS\TOOLS\GUIBUILD\CD-STORE\config.tpm
                              <- Línea en blanco obligatoria.
[Cuentas por Cobrar]          <- Declaración de ubicación de la tercer Base.
CONFIG=C:\basis\CURSOVP5\BBDICT\Curso.tpm

; Bottom of config file       <- Comentario opcional igual que la primer línea
```

En este archivo **sql.ini** se definen tres Bases de Datos y el directorio donde estas se encuentran ubicadas está definido en el archivo cuyo nombre termine con **.tpm** de cada una de las líneas inferiores. El nombre de una base de datos puede estar compuesto por varias palabras. Lo importante de recordar es que posteriormente la Base de Datos debe ser abierta con ese mismo nombre, respetando las letras mayúsculas, minúsculas y espacios intercalados entre palabras.

Observación muy importante: en el archivo config.bbx en su última línea de texto aparece lo siguiente: **#sql**
Es necesario por medio de algún editor eliminar el signo # para que la operación del SQL bajo Visual PRO/5 quede habilitada.

Empezando a probar el SQL

En el entendido de que este no es un curso de SQL, vamos a ir viendo pequeños programas que nos mostraran como interactuar con la Datos que nos interese. Debemos comprender que aunque SQL es un estándar entre las diferentes Bases de Datos, generalmente todas tienen ciertas características propias dentro de la sintaxis del lenguaje. Dicho de otra manera, un resultado equivalente con diferentes Bases de Datos, podría requerir la sintaxis propia para cada una.

Para aclarar más lo descrito en el párrafo anterior, se puede decir que con las Bases de Datos externas, no es Visual PRO/5 quien ejecuta las instrucciones de SQL, él se limita únicamente a

enviárselas al manejador de la Base de Datos a través del ODBC. Cualquier error que ocurriera con la sintaxis enviada es captado por el Visual PRO/5 y podríamos desplegado si lo quisiéramos.

Para familiarizarlo con la sintaxis del SQL vamos a mostrar los verbos utilizados con BBx para uso y acceso a los archivos y el equivalente dentro del SQL.

Comando en Visual PRO/5	**Equivalente en SQL**
MKEYED "file"	CREATE TABLE
ERASE "file"	DROP TABLE
OPEN (canal)"file"	no se da por archivo pero si a la Base de Datos
READ	SELECT
WRITE nuevo registro	INSERT
EXTRACT/WRITE	UPDATE
REMOVE	DELETE

Para saber si tengo posibilidad de accesar alguna Base de Datos se escribe:

>PRINT SQLLIST(0) → y me deberán aparecer los nombres de estas. Ejemplo:

Chile Company
CD-Store
Cuentas por Cobrar
MS Access 97 Database
dBASE Files
Excel Files
FoxPro Files
Text Files
BASIS Chile Company Database

Con esa lista de Bases de Datos, lo único que nos quedaría por hacer es abrir la que necesitemos y comenzar a trabajar. Para abrir una Base de Datos usamos la siguientes instrucción:

```
>SQLOPEN (1)"Chile Company"
```

Existe una forma rápida de saber cuáles son las tablas de datos de la Base de Datos abierta escribiendo:

```
>PRINT SQLTABLES(1)
```
→ y aparecerán los nombres de las tablas disponibles.

Lo que nos queda por ver es como accesar una tabla de datos. Para eso se necesita aún tres pasos:

1) Preparar la sintaxis del SQL a ejecutar, por ejemplo:
```
>SQLPREP (1)"select * from item price<4.5"
```

2) Obtener la estructura o formato de su contenido, por ejemplo así:
```
>DIM REG$:SQLTMPL(1)
```

3) Hacer que se ejecute el SQL así:

```
>SQLEXEC (1)
```

4) Instrucción para indicar que los registros de la tabla van a ser leídos secuencialmente:

```
>REG$=SQLFETCH(1,ERR=FINAL)
```

5) Con una línea como la siguiente se logra el despliegue de algunos de los campos del registro:

```
>PRINT reg.item_num$," ",reg.description$," ",reg.price
```

Los siguientes son ejemplo de varios programas que utilizan SQL. El primero define una tabla de datos llamada PRUEBA y permite insertarle registros. Note que con Bases de datos nativas ya no se permite usar tipos de campo NUMERIC/DECIMAL, pero puede usar FLOAT/DOUBLE/REAL. El uso de parámetros como el signo de pregunta (?) con SQL, hace más parametrisable sus aplicaciones y se presta para desarrollar funciones y rutinas de propósito general. Observe como se aplica en las líneas 0080 y 0130.

```
0010 REM "Crear Tabla y agregar nuevos registros
0020 BEGIN
0030 LET db=SQLUNT; SQLOPEN (db)"Cuentas por Cobrar"; REM "Abre Base de Datos
0040 SQLPREP (db)"Drop table PRUEBA"; SQLEXEC (db,ERR=0050); REM "Elimina Tabl
0040:a PRUEBA
0050 INPUT (0,LEN=3)"De que longitud quiere que sea el campo para el nombre: "
0050:,long:(128); IF long=0 THEN STOP
0060 SQLPREP (db)"Create table PRUEBA (num_orden char(4) primary key, cliente
0060:char("+STR(long)+"), tot_orden real)"
0070 SQLEXEC (db)
0080 SQLPREP (db)"Insert into PRUEBA values(?,?,?)"
0090 loop:
0100 INPUT (0,LEN=4)'LF',"No.Orden> ____",@(10),ord$; IF ord$="" OR CVS(ord$,4
0100:)="F" THEN GOTO final
0110 INPUT (0,LEN=long)"Cliente.> ",clt$; IF clt$="" OR CTL>1 THEN GOTO loop
0120 INPUT (0,LEN=10)"Total...> ",tot; IF CTL>1 THEN GOTO 0120
0130 SQLEXEC (db)ord$,clt$,tot
0140 GOTO loop
0150 REM
0160 final:
0170 INPUT (0,LEN=1)'LF',"Quiere hacer variaciones a los registros recien incl
0170:uidos? (ENTER=Si, 1=No): ",mant$:(""=0180,"1"=0190)
0180 RUN "upda_tab.bbx"
0190 END
```

El siguiente programa lee todos los registros arriba insertados y los despliega por pantalla.

```
0010 REM "Lista Tabla de Clientes desde BBx
0020 BEGIN
0100 SQLOPEN (1)"Cuentas por Cobrar"
0130 SQLPREP (1)"Select * from PRUEBA"
0140 LET a$=SQLTMPL(1); SQLEXEC (1)
0145 DIM r$:a$
0150 LET r$=SQLFETCH(1,ERR=final)
0160 PRINT r.num_orden$," ",r.cliente$,r.tot_orden
0180 GOTO 0150
0190 REM
0200 final:
0210 END
```

Este programa utiliza la misma tabla de datos del programa anterior y permite variar el nombre del cliente. Ahí se puede observar la forma en que definimos el campo que va a ser variado por medio de un parámetro indicado con el signo de pregunta (?).

```
0010 REM "Actualiza campo en tabla de datos PRUEBA
0020 BEGIN
0040 LET db=SQLUNT; SQLOPEN (db)"Cuentas por Cobrar"
0060 REM
0070 pide_cambio:
0080 INPUT (0,LEN=4)'LF',"Dar un numero de Orden: ",orden$; IF orden$="" THEN
0080:GOTO final
0090 SQLPREP (db)"select * from PRUEBA where num_orden='"+orden$+"'"
0100 DIM ord$:SQLTMPL(db); LET long=LEN(ord.cliente$)
0110 SQLEXEC (db)
0120 LET ord$=SQLFETCH(db,ERR=error)
0130 PRINT ord.cliente$,"  ",ord.tot_orden
0150 REM
0160 INPUT (0,LEN=long)"Dar nuevo nombre para el cliente:",'LF',nuevo$; IF nue
0160:vo$="" THEN GOTO pide_cambio
0170 SQLPREP (db)"update PRUEBA set cliente=? where num_orden='"+orden$+"'"
0180 SQLEXEC (db)nuevo$
0190 GOTO pide_cambio
0200 REM
0210 error:
0220 IF ERR=2 THEN INPUT (0,SIZ=1)"No existe ",*; GOTO pide_cambio
0230 REM
0240 final:
0250 END
```

El programa que sigue es una modificación del anterior para que pueda eliminar registros

```
0010 REM "Elimina registros de Ordenes de Compra en PRUEBA
0020 BEGIN
0040 SQLOPEN (1)"Cuentas por Cobrar"
0060 REM
0070 pide_num_orden:
0080 INPUT (0,LEN=4)'LF',"Dar numero Orden a eliminar: ",orden$; IF orden$=""
0080:THEN GOTO final
0090 SQLPREP (1)"select * from PRUEBA where num_orden='"+orden$+"'"
0100 DIM ord$:SQLTMPL(1)
0110 SQLEXEC (1)
0120 LET ord$=SQLFETCH(1,ERR=error)
0130 PRINT ord.cliente$,"  ",ord.tot_orden
0150 REM
0160 INPUT (0,LEN=1)"Elimina ese registro? (ENTER=Si, 1=No): ",elimina$:(""=01
0160:70,"1"=pide_num_orden)
0170 SQLPREP (1)"delete from PRUEBA where num_orden='"+orden$+"'"
0180 SQLEXEC (1)
0190 GOTO pide_num_orden
0200 REM
0210 error:
0220 IF ERR=2 THEN INPUT (0,SIZ=1)"No existe ",*; GOTO pide_num_orden
0230 REM
0240 final:
0250 END
```

El siguiente programa nos hace un tabulado de salarios, con cortes de control por compañía, bajo la condición de que solo considere los salarios menores o iguales a 500,000. Observe como se aplica el parámetro COUNT(*) para obtener el total de registros por compañía y SUM(salario) para indicar que ocupamos la sumatoria de salarios por compañía.

```
0010 REM "Tabula salarios por compañía ignorando salarios mayores de 500000
0020 BEGIN
0030 LET pla=SQLUNT; SQLOPEN (pla)"Cuentas por Cobrar"
0040 SQLPREP (pla)"select numcia, count(*), sum(salario) from empleado
0040:WHERE salario<=5000000 GROUP by numcia"
0050 DIM emp$:SQLTMPL(pla)
0060 SQLEXEC (pla)
0070 LET emp$=SQLFETCH(pla,ERR=0090)
0080 PRINT emp.numcia$,emp.col002
0085 GOTO 0070
0090 END
```

Este pequeño programa lee un archivo de FOX y nos muestra los datos ordenados por nombre del asociado:

```
0010 REM "Leer base datos FOX
0020 BEGIN
0070 SQLOPEN (1)"Fox Asociacion"
0100 SQLPREP (1)"select * from asociado order by asociado.nombre"
0110 DIM dato$:SQLTMPL(1)
0120 SQLEXEC (1)
0130 LET dato$=SQLFETCH(1,ERR=final)
0140 PRINT dato.ced$,"  ",dato.nombre$
0150 GOTO 0130
0160 REM
0170 final:
0180 END
```

El que sigue nos muestra los datos provenientes desde dBASE.

```
0010 REM "Listar Registros de archivo.dbf Banco Popular"
0020 BEGIN
0030 SQLOPEN (1)"dBASE Files"
0040 SQLPREP (1)"select * from 109362"
0050 DIM r$:SQLTMPL(1); SQLEXEC (1)
0060 LET r$=SQLFETCH(1,ERR=final)
0070 PRINT r.cedula$,"  ",PAD(r.nombre$,32),r.ahotot
0080 GOTO 0060
0090 final:
0100 END
```

Con los ejemplos mostrados se muestra como se pueden efectuar las operaciones básicas de acceso a una determinada Base de Datos, utilizando instrucciones básicas y estándar de SQL. Procesos

más sofisticados pueden ser realizados siempre y cuando el manejador de la Base de Datos en uso externo lo permita.

Para aquellos que aún no usan SQL recuerden que una de las ventajas principales de usar SQL en sus aplicaciones es la independencia de Base de Datos que se obtiene. Es decir, si se usa SQL para el manejo de archivos su aplicación es portátil y potencialmente se puede instalar con una Base de Datos externa como Oracle, Sybase, Fox, etc. Esto puede hacer su aplicación más mercadeable entre usuarios que tengan aplicaciones en otras bases de datos. Por ejemplo, puede vender un sistema de Nómina que actualice un Mayor General residente en Oracle.

El uso de SQL requiere cambios en la aplicación ya que los comandos y manejo de archivos es diferente. Por lo general deben asumir un tiempo de aprendizaje si no han usado SQL anteriormente. También SQL es menos eficiente que el acceso normal que siempre ha ofrecido el BBx (READ RECORD, WRITE RECORD, etc.) por su estilo de manejo de conjuntos de registros (record sets) en lugar de registros individuales. Por otro lado, al manejar conjuntos puede ser más eficiente en modo cliente/servidor ya que se minimiza la actividad entre el servidor y el cliente al procesar grupos de registros. Con la tendencia hacia procesadores más veloces la menor eficiencia en acceso navegacional es en algunos casos un factor de menor peso que lograr portabilidad en el sistema de archivos.

El ODBC de Basis emite una documentación que muestra la sintaxis de SQL que permite el Visual PRO/5, para trabajar con la Base de Datos nativa o propia del Visual PRO/5.

Dynamic Data Exchange

Dynamic Data Exchange es una facilidad que también nos brinda el Visual Pro/5, ya que con esta opción podemos accesar documentos que estén cargados en la memoria de la estación de trabajo, como lo son documentos de Word, Excel, Lotus y otros. Por ejemplo, desde Excel podemos abrir una hoja electrónica cargada de datos que perfectamente podemos leer o modificar desde Visual Pro/5. El requerimiento para este tipo de programa es que por aparte se necesita conocer los parámetros necesarios para interactuar con el documento en memoria, ya que estos parámetros no son algo propio del Visual Pro/5.

El programa que a continuación se muestra permite cargar las celdas de una hoja que tengamos abierta con Excel. Está bastante documentado con REMs que van explicando el proceso. Note que en la línea 0130 se hace la advertencia de que el programa debe condicionarse para cuando se usa Excel en español o en inglés.

```
0010 REM "Muestra como CARGAR datos en una hoja de Excel en memoria
0020 REM "Se requiere que el archivo Excel exista y que Excel este abierto
0030 BEGIN
0040 REM "La siguiente linea comprueba que Excel este abierto
0050 LET xls=UNT; OPEN (xls,MODE="DDECLIENT",ERR=xls_no_abierto)"EXCEL:SYSTEM"
0060 INPUT "Dar nombre del archivo de Excel (ENTER=Ninguno): ",arch$:(""=final
0060:,LEN=1,18); IF POS(".xls"=CVS(arch$,8))=0 THEN LET arch$=arch$+".xls"
0070 LET cmd$="[OPEN(""C:\temp\"+arch$+""")]"+$00$
0080 REM "La siguiente linea carga el archivo indicado en memoria
0090 WRITE RECORD(xls,KEY="DDE_EXECUTE",ERR=xls_no_abierto)cmd$
0100 CLOSE (1); REM "Despues de ponerlo en memoria hay que cerrar el canal
0110 REM "La siguiente linea abre el acceso hacia el Excel en memoria
0120 OPEN (1,MODE="DDECLIENT")"EXCEL:c:\temp\"+arch$
0130 REM "Abajo, para Excel en Ingles debe usar RxCx y para Espanol FxCx
0140 REM
0150 LET cli=UNT; OPEN (cli)"CLIENTE.tpl"; FIND (cli)tpl$; CLOSE (cli); OPEN (
0150:cli)"CLIENTE.dat"; DIM cli$:tpl$
0160 REM
0170 leer:
0180 READ RECORD(cli,END=final)cli$
0190 LET lin=lin+1
0200 WRITE (xls,KEY="F"+STR(lin)+"C1")cli.codigo$
0210 WRITE (xls,KEY="F"+STR(lin)+"C2")cli.nombre$
0220 WRITE (xls,KEY="F"+STR(lin)+"C3")STR(cli.region)
0230 GOTO leer
0240 REM
0250 xls_no_abierto:
0260 INPUT (0,SIZ=1)"Favor de abrir Excel antes de continuar ",*; GOTO 0030
0270 REM
0280 arch_no_abierto:
0290 RETRY
0300 REM
0310 final:
0320 END
```

Un poco similar al anterior, el siguiente programa nos enseña como podemos leer los datos en las celdas de una hoja electrónica que necesariamente primero debe ser abierta (cargada en memoria) por Excel. Igualmente al programa anterior, hay que tomar la previsión para los casos en Excel esté en inglés o en español.

```
0010 REM "Muestra como LEER un archivo de Excel
0020 REM "Se requiere que el archivo Excel exista y que Excel lo tenga abierto
0030 BEGIN
0040 REM "La siguiente linea comprueba que Excel este abierto
0050 LET xls=UNT; OPEN (xls,MODE="DDECLIENT",ERR=xls_no_abierto)"EXCEL:SYSTEM"
0060 INPUT "Dar nombre del archivo de Excel (ENTER=Ninguno): ",arch$:(""=final
0060:,LEN=1,18); IF POS(".xls"=CVS(arch$,8))=0 THEN LET arch$=arch$+".xls"
0070 LET cmd$="[OPEN(""C:\My document1\"+arch$+""")]"+$00$
0080 REM "La siguiente linea carga el archivo indicado en memoria
0090 WRITE RECORD(xls,KEY="DDE_EXECUTE",ERR=arch_no_existe)cmd$
0100 CLOSE (1); REM "Despues de ponerlo en memoria hay que cerrar el canal
0110 REM "La siguiente linea abre el acceso hacia el Excel en memoria
0120 OPEN (1,MODE="DDECLIENT")"EXCEL:c:\My document1\"+arch$
0130 REM "Abajo, para Excel en Ingles debe usar RxCx y para Espanol FxCx
0140 REM
0150 leer:
0160 LET lin=lin+1
0170 READ RECORD(xls,KEY="F"+STR(lin)+"C1")a$
0180 READ RECORD(xls,KEY="F"+STR(lin)+"C2",ERR=final)b$
0190 READ RECORD(xls,KEY="F"+STR(lin)+"C3",ERR=final)c$
0200 PRINT a$,"-",b$,"-",c$
0210 INPUT ":",*
0220 GOTO leer
0230 REM
0240 xls_no_abierto:
0250 INPUT (0,SIZ=1)"Favor de abrir Excel antes de continuar ",*; GOTO 0030
0260 REM
0270 arch_no_existe:
0280 RETRY
0290 REM
0300 final:
0310 END
```

Para pasar datos desde Pro/5 o Visual Pro/5 hacia Excel también existen otras buenas formas de lograrlo:

1) Por medio de un archivo de texto en el que se utiliza 09 como separador de campos. Ese hexadecimal le es más natural a Excel, ya que por 'default' ese es el separador que internamente utiliza.

2) Por medio de un archivo de texto en el que se utilizan comas como separador de campos y luego 'pegarlo' por medio de los verbos y funciones CLIP..... que ofrece el Visual PRO/5. Funciona muy bien pero tiene el inconveniente de dar problema si existen comas dentro de los datos a ser enviados (como a veces ocurre en las direcciones). Lo que hace Visual Pro/5 con esos mnemónicos es colocarlo en el portapapeles de Windows, y después en Excel se le da clic al botón pegar y listo. Como dije, con anticipación se necesita crear un archivo de texto (String), y enviar el reporte a este archivo con los campos separados por comas. Luego, al finalizar el reporte se cierra el canal del archivo de texto y se le da las siguientes instrucciones, donde Clip$ es igual al nombre del archivo de texto

```
csv=clipregformat("CSV")
cliplock
clipfromfile csv,clip$
clipunlock
```

El siguiente es una muestra de un programa que logra hacer eso:

```
0010 REM "Clip_xls Carga datos en archivo de texto a una hoja de Excel
0020 BEGIN
0030 LET sep$=",",txt=UNT,temp$="temp.txt"; OPEN (txt,MODE="O_CREATE")temp$
0040 PRINT (txt)"articulo"+sep$+"Descripcion"+sep$+STR(3132500)
0050 PRINT (txt)"codigo"+sep$+"nombre"+sep$+STR(316354458)
0060 CLOSE (txt)
0070 LET csv=CLIPREGFORMAT("CSV")
0080 CLIPLOCK
0090 CLIPFROMFILE csv,temp$
0100 CLIPUNLOCK
0110 LET m=MSGBOX("Ahora puede pegar los datos en una hoja de excel",64,"Excel
0110:")
0120 ERASE temp$
0130 END
```

Nota: Muchas de estas palabras tienen usos y significados más generales. En el sentido que se le da a las palabras, aquí nos referimos solamente el Visual PRO/5, SYSGUI.

Acelerator: (aligerador) una combinación de teclas que pueden sustituirse por un ítem de menú, hasta cuando el menú no esté activo.

bitmap: (mapa de bits) una imagen gráfica almacenada simplemente como un arreglo de pixeles, listando valores de color para cada uno.

border: (borde) el borde de una ventana gráfica.

border decorations: ver window decorations.

bounding rectangle: (coordenadas del rectángulo) cuatro números que describen la ubicación de la esquina superior izquierda, el ancho y altura de un control gráfico o ventana.

button: (botón) un control GUI estándar que emula el comportamiento de un botón de pulsar físico.

caret: (cursor) el "cursor" usado para marcar el punto de inserción de texto en las cajas de edición y controles similares.

chars: (caracteres) una unidad de medida igual a la altura exacta y ancho típico de un caracter en la fuente del sistema en uso.

check box: (caja chequeable) un control GUI estándar con dos estados, chequeado y des-chequeado. Cajas chequeadas pueden ser usadas en grupos pero no son exclusivos de los botones de radio.

checkable menu ítem: un ítem en un menú {pull-down} con un check marcado como variable, cuando opera.

checked: (chequeable) se dice que un control es chequeable cuando este es marcado. Botones de radio, cajas de chequeo, menús de ítems chequeables, y hasta botones herramienta variables pueden ser chequeados.

child window: (ventana hija) una ventana que es contenida completamente dentro de otra ventana.

client area: la porción de una ventana en donde se muestra su contenido. Se excluye el borde, la barra de desplazamiento, la caja de cierre, y cualquier otra "decoración" de borde.

clipboard: (tablero) un almacenamiento de información temporal, y facilidad de recuperación de Microsoft Windows y otros ambientes de graficación similar. Las operaciones que se pueden realizar tienen nombres estándar. La operación de "Cortar" elimina información de una ubicación original y la almacena en el {clipboard}. La operación de "Copiar" deja la información original en su lugar pero pone una copia de ella en el {clipboard}. Finalmente, la operación de "Pegar" copia

la información del {clipboard} en el control o documento en uso, y la inserta a partir del punto marcado por el cursor.

close box: (botón de cierre) un dibujito en la parte superior derecha de la ventana, que le permite al usuario despedir o desaparecer la ventana cuando opera.

context: (contexto) un ID numérico asociado con cada ventana de SYSGUI. Esto da la posibilidad de poder manipular varias ventanas a la vez.

control: un objeto GUI que permite recopilar entradas del usuario y las reporta a la aplicación de una manera pronosticable. Visual PRO/5 ofrece apoyo a once controles estándar de Windows y un manojo de conocidos controles de propósito especial. Ejemplos: botones de pulsar, cajas de edición, botones de radio.

CONTROL coordinates: cada ventana de SYSGUI puede mostrar controles así como detalles dibujados. Los controles son ploteados utilizando coordenadas de control. Estos pueden ser suministrados en 'PIXELS', 'CHARS' o 'SEMICHARS', y pueden ser 'escaleados' por medio del mnemónico 'SCALE'.

cue: (cola) una pequeña ventana se abre y/o un comentario en la barra de condición aparecen cuando el ratón es pasado sobre un control particular. Frecuentemente, una cola guía a usuarios principiantes a través de la operación de una aplicación. Son especialmente útiles con barras de herramientas, que pueden ser ocupadas con botones herramientas que contienen {bitmaps} en lugar de texto.

cursor: el cursor de texto es un bloque que indica donde el siguiente caracter de texto entrará una ventana de caracteres. El cursor del ratón (también conocido como el apuntador del ratón) es un símbolo pequeño (generalmente una flecha) que se mueve en la pantalla, pudiéndose rastrear el movimiento del ratón.

custom edit: un control de edición de texto multi-línea suministrado por Visual PRO/5.

default button: el botón de pulsar es una caja de diálogo que será activada si la tecla ENTER es presionada. Generalmente un rectángulo de negro espeso es dibujado a su alrededor.

dialog: (diálogo) una ventana cuyos controles son navegables por medio del teclado. Un diálogo puede ser modal (usuario tiene que responder antes de continuar) o sin modo (usuario puede dejar el diálogo y regresar posteriormente).

disabled: (desactivado) las ventanas y controles que estén desactivados son visibles pero no pueden ser movidos u operados de cualquier modo por el usuario.

docking child window: normalmente una ventana hija reside dentro del área de cliente del padre. Al rescindir de ventanas hijas quedan vinculadas por si mismas al interior del borde del padre en lugar del área del cliente.

drag: (arrastrar) desplazar el botón del ratón de un lugar a otro, manteniendo el botón de este oprimido durante el trayecto, así es como por medio del ratón se causa que objetos movibles sean movidos a través de la pantalla.

draw mode: determina lo que ocurre cuando un objeto es aproximado a la parte superior de otro. Los modos de arrastre más comunes son COPIAR (nuevos objetos son inducidos sobre los viejos) y XOR (nuevos objetos son reemplazados por viejos, de modo que un segundo dibujo del mismo nuevo objeto anulará los dibujados primero).

DRAWING coordinates: cada ventana de SYSGUI puede mostrar controles así como ítems dibujados. Los ítems dibujados son ploteados utilizando coordenadas dibujadas. Esto puede ser suministrado en pixeles (lo que se asume si no hay otra indicación) o milésimas de una pulgada. Para seleccionar unidades de dibujo, use el mnemónico 'DRAWNITS'.

Edit box: (caja de edición) un control GUI estándar que permite al usuario editar una línea única de texto.

event: el reporte generado por el sistema GUI cuando el usuario opera un control o de otra manera interactúa con un objeto gráfico.

event loop: juego de instrucciones de un programa donde una cola de eventos es leída en un loop, para responder a eventos hasta que un evento especial (quizás la operación de una caja de cierre) rompa el ciclo.

event mask: (máscara de eventos) un string de bits que indican en cierta medida qué clase de informes deberían ser generados por el sistema GUI, en respuesta a una interacción de usuario.

event queue: (cola de eventos) el buffer único en el que los informes de evento son guardados por todas las ventanas de SYSGUI. Este puede ser leído con READ RECORD, y nivelado con el mnemónico 'FLUSH'.

event-driven: dicho de un programa que es diseñado para responder a eventos. Vea también: event loop.

flags: (banderas) un string de bits usados en la creación de ventanas y controles que determinan cual de los parámetros opcionales es seleccionado.

focus: (enfoque) cualquier ventana o control de edición recibirá entrada mecanografiada por el teclado cuando tiene enfoque.

font: (fuente) un tipo de letra.

generic control: cualquiera de los controles disponibles en Visual PRO/5 son controles estándar de Windows y no controles acostumbrados. Los controles genéricos son nativos de Windows y así trabajan más rápidos y están menos bajo el control de BASIS.

gravity: (gravedad) una opción de ventana que causen que sus ventanas hijas se redistribuyan por si mismas, siempre que la ventana madre es cambiada de tamaño. Puede ser muy útil para las barras de herramientas.

group: los controles pueden ser "agrupados" en diversos modos. Pueden ser tratados como un set para navegación de teclado, y botones de radio pueden ser agrupados lógicamente de modo que se

excluyan unos a otros. Estos agrupamientos usualmente corresponden, pero esto no es un requisito. Los controles pueden también tener una caja de grupo dibujada alrededor a ellos. La caja de grupo no tiene efecto en la operación de los controles o en la navegación de teclado.

group box: (caja de grupo) un control estándar GUI que provee una asociación visual entre controles. No afecta la operación de la navegación de controles o teclado de alguna manera. Las cajas de grupo nunca generan eventos, ni consumen clicks del ratón. Son, sin embargo, opacas. Todos los demás controles aparecerán delante, pero ítems dibujados serán oscurecidos por una caja de grupo.

GUI: Interfase Gráfica del Usuario

hot key: (teclas calientes) una forma rápida por medio del teclado para desempeñar una acción que de otra manera requiere una acción del ratón, o al menos más golpes de tecla. Mnemónicos de menú y los aceleradores son ejemplos de teclas calientes. Controles como los botones de pulsar pueden también tener teclas calientes.

icon: (icono) un pequeño {bitmap} que usualmente es conectado lógicamente con ya sea una aplicación o un documento.

iconized: minimizado.

id: un número que es utilizado para identificar y distinguir objetos en un contexto de SYSGUI. Cualquier número desde el 1 al 32767 puede ser usado, aunque del 1 al 99 están "reservados ". ID 0 casi siempre es asignado a la ventana en uso.

invisible: un control o ventana puede estar existiendo, y puede ser localizado en el mismo lugar, pero es hecho invisible. Esto vuelve al control inoperable para el usuario, por supuesto, hasta que sea hecho de nuevo visible.

keyboard navigation: (navegación de teclado) los diálogos usan ciertas teclas estándar para moverse entre los controles disponibles y los operan. Seleccionando navegación de teclado se permite este comportamiento.

list box: (caja de lista) un control GUI estándar que suministra una lista de ítems (a veces con scroll) para que el usuario pueda hacer selecciones. Algunas cajas de lista pemiten múltiples selecciones, y todas pueden reportar los clicks y los doble-clicks en todos los ítems.

list button: (botones de lista) un botón GUI estándar que permite al usuario escoger uno entre una lista de posibilidades. La lista es mostrada solamente cuando la porción del botón del control es operado. la mayor parte del tiempo el control ocupa solamente la parte superior de su rectángulo unificador.

list edit: un control GUI estandar que es muy similar a un botón de lista, excepto que el usuario puede seleccionar una de las opciones disponibles *o* mecanografiar texto que no ha sido incluido en la lista.

maximize: (maximizar) para lograr que una ventana ocupe todo el espacio disponible (ventana de la pantalla o ventana padre) como sea posible. Windows provee un dibujito en la parte superior derecha de la ventana que permite esta acción.

menú: una lista de posibilidades que pueden ser operadas con el ratón. Los menús pueden contener otros menús (sub-menús).

menu ítem: (menú de opciones) un ítem en un menú. El ítem de un menú ordinario causa que un comando sea enviado cuando este es operado. Otros tipos de ítems de menú son ítems de menú chequeables (estos cambian la condición de chequeado cuando son operados), separadores (estos aparece simplemente como una línea horizontal que no puede ser operada), y otros menús (sub-menús).

menu mnemonic: si una letra o símbolo en un menú de ítems es mostrado subrayado, entonces usted puede mecanografiar la letra o símbolo que causa que ese ítem del menú sea activado. Los mnemónicos de menú trabajan solamente cuando el menú en particular está desplegado.

menu tag: (la etiqueta de menú) un número que únicamente identifica un único ítem del menú. Las etiquetas de menú son separadas desde los IDs de los controles, de modo que el mismo número puede ser usado por ambos sin ningún daño.

menubar: (barra del menú) el menú de nivel-superior que es mostrado a través de la parte superior de una ventana.

minimize: (minimizar) acción que se toma para lograr que una ventana sea representada como un icono.

modal: un diálogo modal tiene que ser despedido antes de que la aplicación pueda continuar. (ver modeless).

modeless: un diálogo sin uso puede continuar siendo mostrado mientras otras ventanas sean operadas. (ver modal).

picture button: un botón herramienta dispuesto para mostrar un {bitmap} en el área del cliente.

pixels: una unidad de medida igual a un elemento de pixel (un punto en la pantalla o dispósitivo de despliegue).

radio button: (botón de radio) un control GUI estándar que es útil solamente en grupos de dos o más elementos. Un conjunto de botones de radio operan como un selector mutuamente exclusivo.

raise: para traer una ventana al frente del orden apilado. No necesariamente la ventana tendrá enfoque.

resolution: lo más pequeño: los pixeles, lo mayor: la resolución. Frecuentemente la resolución es medida en pixeles por pulgada.

restore: (reversar) para reversar el efecto de minimizar o maximizar.

scroll bar: (barra de desplazamiento) un control o dibujo en la ventana que dispone de una flecha en cada uno de sus extremos y un cuadrito que se puede desplazar a través de la barra. La barra de desplazamiento es utilizada para seleccionar un posible valor desde un rango lineal.

semichars: (semicaracteres) una unidad de medida derivada del tamaño de un caracter típico en la fuente del sistema actual. Ploteando controles en semicaracteres es más transportable la aplicación que utilizando pixeles, si hay texto (en la fuente que esté usando el sistema) dentro de los controles.

stacking order: el orden con que se gráfica ventanas es dispuesto en la pantalla.
static text: un control GUI estándar el cual simplemente dibuja texto en la ventana padre. Nunca genera algún evento.

status bar: (barra de condición) un control de propósito especial que consiste de una línea de texto que es dibujada de todas formas a través de la parte inferior de la ventana padre.

SYSGUI: El dispositivo que Visual PRO/5 usa para conversar con el subyacente sistema gráfico.

SYSPLOT: un dispositivo del Visual PRO/5 que acepta comandos tradicionales de ploteo de BBx y despliega dibujos en una ventana gráfica.

SYSPRINT: un dispositivo del Visual PRO/5 que provee una interfase de caracteres con impresoras de Windows a través del Administrador de Impresión.

SYSWINDOW: un dispositivo del Visual PRO/5 que es un {superset} del dispositivo de terminal del BBx estándar. Generalmente es abierto por el canal 0 cuando Visual PRO/5 es iniciado.

text selection: un bloque de texto en un control de edición que es mostrado en forma destacada. Este texto será reemplazado si cualquier nuevo texto es mecanografiado, u operado si cualquiera de las funciones estándar del tablero es ejecutada. Una selección de texto puede ser hecho a través del ratón o a través de los mnemónicos ‘SELECT’ o ‘TXSELECT’. La selección de texto en uso puede ser recuperada con CTRL ().

title: el texto primario asociado con un control o ventana. Todos los controles tienen títulos, aunque algunos nunca son exhibidos.

title bar: La caja en la parte superior de la ventana que mantiene el título de la ventana.

toggle button: (botón variable) un botón herramienta que es hecho para operar como un conmutador. Con un click se deprime el botón (“chequeado”), y con otro click se invierte. Las capacidades de conmutar y visualizar pueden ser usadas conjuntamente.

tool bar: (barra de herramienta) una ventana (generalmente una ventana hija) de la cual los controles se mantienen para acceso conveniente del usuario. Frecuentemente esta es una ventana hija llenada con botones herramientas.

tool button: (botón herramienta) un control de propósito especial que se comporta parecido a botón estándar, pero no del todo. El botón herramienta reporta el ID del botón del ratón, shift y estado del estado de las teclas de control, y la posición del ratón en sus eventos. Los botones herramienta pueden mostrar un {bitmap} en lugar de un título textual, y pueden también

opcionalmente ser hecho para conmutar. Los botones herramienta no pueden tener enfoque de teclado y así no puede tampoco ofrecer navegación de teclado.

window: una pantalla gráfica lógica o una superficie en la cual controles y dibujos pueden ser colocados.

window decorations (algunas veces llamado "border decorations"): bonitos controles de objetos que residen en el borde de ventanas y que afectan la ventana de alguna forma. Son pequeños dibujos comunes que incluyen cajas de cierre, minimizar y maximizar la ventana, así como barras de desplazamiento.

Evento a ser reportado	Event Mask Bit	Atributo	Flag bit
Check or uncheck of check box or radio button	02000000	Close box	00000002
		Gravity	00100000
Click or double click on list ítem	01000000	Horizontal scroll bar	00000004
Close box operated	None-always sent	Initially disabled	00000020
Edit of list edit modified	00400000	Initially invisible	00000010
Edit, list edit, or text edit focus change	00800000	Initially maximized	00001000
Key pressed	00000400	Initially minimized	00000100
Menú selection mode	None-always sent	Keyboard navigation	00010000
Mouse button double click	00000200	Menu bar	00000800
Mouse button down	00000040	Minimizable	00000080
Mouse button up	00000080	Modal dialog style border	00040000
Mouse moved	00000100	Modal dialog behavior	00080000
Push buttton operated	None-always sent	Stay on top	00020000
Scroll bar position changed	00100000	User sizable	00000001
Tool button operated	None-always sent	Vertical scroll bar	00000008
Window resized	00000008		
Window focus change	00000004		

Code	Evento reportado	ID	Flags	x,y
2	Mouse button double click + 0=left 1=right 2=center	+ Button ID	01 <Shift> 02 <Ctrl>	Mouse Position
B	Push Button opereted	Y	N/A	N/A
b	Tool button operated Note: Low order flag bits contain mouse button ID 00=left 01=right 10=center	Y	See note 04 <Shift> 08 <Ctrl> 10 Button down	Mouse position
C	Menu selection mode	Menu tag	01 <Shift> 02 <Ctrl> 04 Now Checked	N/A
c	Check/uncheck check of check box or radio button	Y	00 Uncheck 01 Checked	N/A
d	Mouse button down + 0=left 1=right 2=center	+ Button ID	01 <Shift> 02 <Ctrl>	Mouse position
e	Edit, list edit, or text edit modified	Y	N/A	N/A
F	Window focus gain/loss	N/A	00 Lost 01 Gained	N/A
f	Edit, list edit, or text edit focus gain/loss	Y	00 Lost 01 Gained	N/A
l (ele)	Click or double click on list box ítem	Y	00 Single 01 Double	N/A
m	Mouse moved (use only when needed)	N/A	01 <Shift> 02 <Ctrl>	Muse position
p	Scroll bar control moved	Y	N/A	New position
S	Window resized	N/A	N/A	New size in pixels
t	Key pressed	Numeric key code	01 + <Shift> 02 + <Ctrl>	N/A
u	Mouse button up + 0=left 1=right 2=center	+ Button ID	01 <Shift 02 <Ctrl>	Mouse position
X	Close box operated	N/A	N/A	N/A