



# VFP Y LOS MENSAJES DE WINDOWS I

## Agregar un icono a la barra de título

Desde que inicie con VFP, ninguna actualización en cuanto a métodos, eventos y/o propiedades me había hecho feliz como la función BINDEVENTS; por qué, pues porque nos da la posibilidad de interceptar los mensajes de Windows y eso es, particularmente, magnífico.

### INTRODUCCIÓN

Vamos a empezar con algo de teoría.

En Windows prácticamente todo son ventanas como los formularios, botones, etc; pero por qué digo que no todo son ventanas; en el caso particular de VFP; los controles son dibujados por el mismo VFP y no crean controles a partir de ventanas.

Pero, cómo sabe una ventana cómo comportarse; bueno la ventana recibe constantemente mensajes del sistema operativo y realiza acciones determinadas ante esos mensajes que van en función del tipo de ventana que es; por ejemplo un botón; al recibir un mensaje de que se presionó el botón izquierdo del Mouse (WM\_LBUTTONDOWN) éste cambiará su apariencia simulando que se presionó el botón; a diferencia de un control ListBox que cuando recibe el mismo mensaje seleccionará uno de los Ítems; como ven responden de diferente manera a un mismo mensaje dependiendo del tipo de ventana que sea por lo que podemos decir que:

*Cada ventana pertenece a una clase (Botón, ListBox, etc) y asume la apariencia y el comportamiento definidos para esa clase.*

El sistema operativo (Windows) está constantemente comunicando a todas la ventanas abiertas prácticamente todo lo que sucede en el sistema o al menos,



todo lo que afecta a cada ventana; así por ejemplo Windows notifica (enviando un mensaje a las ventanas) cuando se ha cambiado la hora del reloj, pero no envía el mensaje a todas las ventanas; para éste mensaje solamente notifica a las ventanas de tipo "Top-Level" por lo que no esperen recibir este mensaje en una ventana de la clase CommandButton.

Dado que VFP está limitado en este aspecto no profundizaré mucho en el tema; solamente se darán algunas referencias de lo necesario para entender el tema.

## DÓNDE SE PROCESAN LOS MENSAJES

Todas las ventanas tienen una función que procesa los mensajes recibidos por la ventana; normalmente llamada WndProc; ésta función ya viene definida para cada clase de ventana y dependiendo de la clase será el comportamiento que tendrá sobre los mensajes que reciba.

## FUNCTION WndProc

Esta es la función que recibe los mensajes del sistema y está definida (en C) como:

```
typedef LRESULT (CALLBACK* WNDPROC)(HWND, UMSG, WPARAM, LPARAM);
```

Parámetro	Descripción	Tamaño
HWND	Handle de la ventana	32 bits
UMSG	Mensaje específico	32 bits
WPARAM	Primer parámetro del mensaje	32 bits
LPARAM	Segundo parámetro del mensaje	32 bits
LRESULT	Valor de retorno	32 bits



Los valores de wParam y lParam pueden ser cualquier tipo de datos como numérico, carácter, puntero, etc; esto depende del mensaje.

## SUBCLASIFICACIÓN

Esta técnica tiene como finalidad interceptar los mensajes enviados a las ventanas, una vez interceptados los mensajes podemos hacer 3 cosas:

- Hacer que la ventana simplemente ignore el mensaje.
- Hacer que la ventana trate el mensaje por ella misma (según lo que diga su función Windows real)
- Tratar el mensaje por nosotros mismos definiendo el comportamiento de la ventana ante el mensaje, y una vez hecho esto incluso (si lo deseamos) hacer que la ventana trate el mensaje por ella misma.

## COMO SE INTERCEPTAN LOS MENSAJES

En palabras sencillas se hace “re-dirigiendo” los mensajes a una función definida por nosotros y procesarlos; la función definida por nosotros debe aceptar los mismos parámetros que la función WndProc.

### Y cómo se “re-diriguen” los mensajes...

Utilizando las funciones GetWindowLong y SetWindowLong.

Función	Para que nos sirve
GetWindowLong	Permite obtener el handle de la función WndProc que tiene por default la ventana
SetWindowLong	Permite asignar reasignar la función que procesará los mensajes



Ejemplo:

Obtener el handle de la función WndProc de un formulario

```
DECLARE INTEGER GetWindowLong IN Win32API INTEGER, INTEGER
#DEFINE GWL_WNDPROC -4

Local hDefaultWndProc
hDefaultWndProc = GetWindowLong(ThisForm.hWnd, GWL_WNDPROC)
```

Usando solamente el VFP no se puede asignar una nueva función que intercepte los mensajes ya que para esto se necesita el asignarle la dirección de memoria de la nueva función y eso en VFP no se puede hacer; por lo que omitiré el paso para asignar la nueva función.

## LLAMANDO A LA FUNCIÓN WndProc PREDETERMINADA

Algunas veces queremos que un determinado mensaje sea procesado por la función WndProc predeterminada eso lo podemos hacer utilizando la función CallWindowProc de la siguiente manera:

```
DECLARE INTEGER CallWindowProc IN Win32API INTEGER, INTEGER,
INTEGER, INTEGER, INTEGER

CallWindowProc(hDefaultWndProc; && handle a la function WndProc
predeterminada
    uhWnd; && handle de la ventana
    uMsg; && Mensaje a procesar
    wParam; && Primer parámetro del mensaje
    lParam) && Segundo parámetro del mensaje
```



## Y EN VFP

Bueno, en VFP no sería propiamente una subclasificación a como lo hemos mencionado, podríamos decir que nosotros le decimos que mensajes queremos procesar y VFP se encarga de hacer la subclasificación y cuando el mensaje es uno de los que le indicamos que queremos procesar nos lo envía al procedimiento que nosotros le definimos.

Pero por qué no es una subclasificación, viendo lo anterior la subclasificación cambia la función WndProc por una nuestra y de ahí procesamos TODOS los mensajes; en VFP sólo procesamos los que necesitamos, sólo es cuestión de conceptos.

## COMO INTERCEPTAR MENSAJES EN VFP

Antes de empezar, esto es **válido únicamente para VFP 9.0 o superior** y para las ventanas, hasta donde sé en VFP solamente son los formularios. Cómo se reconoce que es una ventana, toda ventana tiene un handle (propiedad hWnd de los formularios)

Para interceptar los mensajes utilizamos la función BINDEVENTS de la siguiente manera:

```
BINDEVENTS(ThisForm.hWnd, ; && handle de la ventana
            WM_ACTIVATE, ; &&Mensaje a interceptar
            ThisForm, ; &&En qué objeto está la función que procesará los
mensajes
            "MessageHandleProc") && Nombre del método donde se
procesaran los mensajes
```

*Se necesita ejecutar una función BINDEVENTS por cada mensaje que se quiere interceptar*

La función que procesará los mensajes debe aceptar los siguientes parámetros

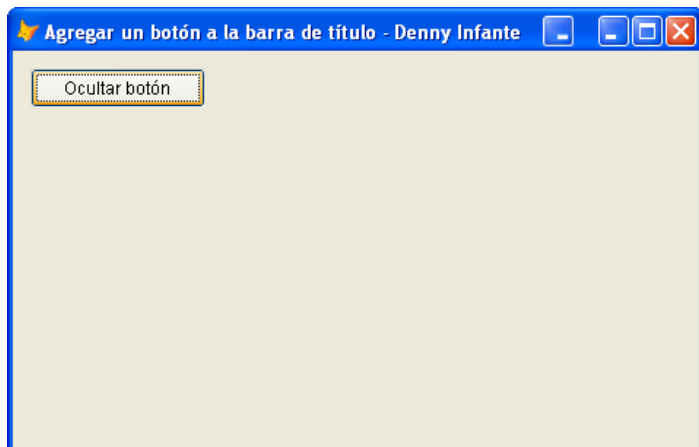
Parámetro	Descripción
hWnd	handle de la ventana
uMsg	Mensaje a procesar
wParam	Primer parámetro del mensaje
lParam	Segundo parámetro del mensaje

De forma general el procedimiento se define como

```
PROCEDURE MessageHandleProc  
    PARAMETERS hWnd, Msg, wParam, lParam  
ENDPROC
```

## Agregar un botón en la barra de título...

Algunos dirán, pero para qué modificar el comportamiento de las ventanas si estas hacen bien su trabajo, pues que les parecería algo sencillo cómo esto:



Agregar un botón en la barra de título, yo se que muchos no le encontrarán utilidad pero quizá otros sí como por ejemplo GetRight que en las ventanas de "download" tienen un botón igual que hace que se minimice como un IconTray.

Para hacer esto es necesario interceptar los siguientes mensajes:



Mensaje	Descripción
WM_NCPAINT	Cuando se requiere pintar el área "no cliente" (barra de título y bordes)
WM_NCACTIVATE	Cuando se activa el área "no cliente"
WM_ACTIVATE	Cuando se activa la ventana
WM_NCHITTEST	Para saber en que parte de la ventana nos encontramos (En la barra de título, en cual borde, en algún botón de la barra de título, etc.)
WM_NCLBUTTONDOWN	Cuando se presiona el botón izquierdo del Mouse sobre el área "no cliente" de la ventana.

### Cómo se procesaron los mensajes...

Primero les recomiendo que lean la documentación de cada uno de los mensajes; si no tienen el MSDN Library lo pueden consultar en <http://msdn.microsoft.com>.

El código presentado aquí no es el código del ejemplo, éste es solamente ilustrativo.

### ***WM\_NCPAINT, WM\_NCACTIVATE, WM\_ACTIVATE***

No queremos cambiar el comportamiento de éstos mensajes; pero queremos saber cuando se dibuja el área "no cliente" para dibujar nuestro botón por lo que, primero llamamos a procedimiento WndProc por default (para que pinte el área "no cliente") y después a nuestro procedimiento DibujaBoton() para dibujar el botón.

En el procedimiento en que procesamos los mensajes:

```
PARAMETERS hWnd, uMsg, wParam, lParam
DO CASE
...
CASE INLIST(uMsg, WM_NCPAINT, WM_NCACTIVATE, WM_ACTIVATE)
    lRet = CallWindowProc(hDefaultWndProc, hWnd, uMsg, wParam,
    lParam)
    This.DibujaBoton("Normal")
...
```



ENDCASE

RETURN lRet

### ***WM\_NCHITTEST***

Este procedimiento nos indica en que parte de la ventana nos encontramos

PARAMETERS hWnd, uMsg, wParam, lParam

DO CASE

CASE uMsg = WM\_NCHITTEST

Local lcPosicion

De acuerdo con la documentación el resultado de procesar este mensaje en la función WndProc por default devuelve la posición por tanto, llamamos a la función WndProc por default para saber en donde estamos.

lcPosicion = CallWindowProc(hDefaultWndProc, hWnd, uMsg, wParam, lParam)

IF lcPosicion = HTCAPTION THEN                      && Si se encuentra en la barra de título

Función que indica si el Mouse está sobre el botón que dibujamos

lcMouseOverBtn = This.MouseOverBtn()

IF lcMouseOverBtn THEN

This.DibujaBoton("MouseOver")

lRet = HTMD\_MOUSEOVER

ELSE

This.DibujaBoton("Normal")

ENDIF

...

ENDCASE

RETURN lRet

### ***WM\_NCLBUTTONDOWN***

Este mensaje nos indica que se hizo clic sobre el área no cliente

DO CASE

CASE uMsg = WM\_NCLBUTTONDOWN

IF wParam = HTMD\_MOUSEOVER

This.DibujaBoton("MouseClicked")

ELSE

This.DibujaBoton("Normal")

...

ENDCASE

RETURN lRet





## ASPECTOS IMPORTANTES A TOMAR EN CUENTA

- La subclasificación es una técnica muy “poderosa” pero al mismo tiempo riesgoso ya que, si no se procesan correctamente los mensajes se puede llegar a tener comportamientos erráticos en las ventanas y su aplicación dejará de funcionar adecuadamente.
- Todo mensaje procesado debe regresar un valor (`Return lcValorDevuelto` en el procedimiento donde se procesan los mensajes); para mayor información refiéranse a la documentación de mensaje
- Algunos mensajes son enviados a las “Top-level Windows” como es el caso de `WM_TIMECHANGE`, así que si intentan interceptar el mensaje en una ventana hija dentro de una aplicación MDI no lo van a conseguir.
- Esta es una pregunta para el que me pueda quitar la duda; por qué si la ventana del ejemplo no es formulario de nivel superior el ejemplo no funciona; si se supone que todos los formularios son ventanas.
- Y sobre lo mismo del punto anterior, en VFP he experimentado algunos problemas con los mensajes, por ejemplo con `WM_NCHITTEST`, no me ha regresado el valor del `HTCLIENT` a pesar de encontrarme en el área cliente; no se si se deba a la forma en que VFP maneja los mensajes.

Espero que les sea de utilidad

**Denny Infante**

**ADVERTENCIA:** El código de éste artículo es proporcionado “como está”; no me hago responsable por ningún tipo de daño atribuible al uso del código en éste artículo ni del código fuente del ejemplo que lo acompaña.

Úselo bajo su propio riesgo :o)