

Event-Driven GUI Programming With Notify And SENDMSG()

By Jim Douglas

Among the many new features introduced with Visual PRO/5™ v2.0 are four new graphical controls. INPUTE and INPUTN, derived from existing Visual PRO/5 verbs, allow for more control of text and numeric input. Tabbed dialogues are supported through the 'TABCTRL' mnemonic. And the new grid control allows you to present your data to the user in a matrix format which can be manipulated using a series of control messages. In order to properly support the richer set of interactions required by these new controls, BASIS has added the SENDMSG() function and the Notify event.

Intercepting Incoming Events

Before we look at the Notify event, let's review how events are handled in Visual PRO/5. The following program fragment shows the basic structure of an event-driven program:

```
sysgui=unt; open (sysgui)"X0"
print (sysgui)'window'(0,40,640,400,"Event Loop",
    $0401088f$, $ffffff$)
dim event$:tmpl(sysgui); event_len=len(event$),done=0
while !(done)
  read_record(sysgui,siz=event_len)event$
  switch asc(event.code$)
    case asc("N"); rem " Notify event
      break
    case asc("X"); rem " Close box operated
      done=1
  swend; rem " asc(event.code$)
wend; rem " !(Done)
stop
```

This program opens the SYSGUI device, creates a window, and goes into a loop which retrieves and dispatches events from the event queue. The program drops out of the loop when the user clicks on the close box, generating a Close Box Operated event (type "X"). There is also a stub for handling Notify events (type "N").

Interpreting And Acting On The Notify Event

Visual PRO/5's new graphical controls must often pass on information which won't fit in the ten-byte `event$` structure. Since changing the structure of `event$` could potentially break existing code, the new extended event information is made available through the Notify event and the NOTICE() function. Whenever Visual PRO/5 needs to pass on extended event information, it issues a Notify event, which tells you that the information has been queued for you to retrieve, as in the following example:

```
case asc("N"); rem " Notify event
```

```

dim base$:noticetpl(0,0); rem " Dim base notice template.
base$=notice(sysgui,event.x%); rem " Grab notice string.
dim notice$:noticetpl(base.objtype%,event.flags%)
notice$=base$; rem " Assign notice string to template.

switch notice.objtype%
  case 19; rem " List Button Control
  break
  case 20; rem " List Edit Control
  break
  case 104; rem " INPUTE Control
  break
  case 105; rem " INPUTN Control
  break
  case 106; rem " Tab Control
  break
  case 107; rem " Grid Control
  break
  case default; rem " Notify undefined.
  break
swend
break

```

The format of the string returned by the NOTICE() function will vary depending on which control issued the Notify event. The job of interpreting this string is made easier by the NOTICETPL() function, which returns the appropriate template for a given object type and message type.

The sample code above starts off by DIMing a temporary working string called `base$` using the template returned by the function call `noticetpl(0,0)`. This call returns a base template which is common to all notice strings and which can be used to extract the object type. (This base template is currently defined as "context:u(2), code:u(1), id:u(2), objtype:i(2)".

Once the template is defined we load the notice string into `base$` using the NOTICE() function, making the object type available to us in `base.objtype%`. By passing this object type value and the message type value `event.flags%` as arguments to the NOTICETPL() function, we can DIM the `notice$` string with a template specific to the control that triggered the original Notify event. The notice string is then copied from `base$` to `notice$`, making the full event string available through the appropriate template.

Dealing With Specific Controls

There are currently six different control types which can cause a Notify event to be issued: the four new graphical controls, plus the list button and list edit controls. As shown in the switch block above, you can determine what kind of control issued (or "fired") an event by examining the `objtype%` portion of the `notice$` string. You can also find out what kind of Notify event the control fired by examining the `flags%` portion of the `event$` string:

```

case 19; rem " List Button Control
  switch event.flags%
    case 1; rem " List Open
    break
    case 2; rem " List Selection
    break
    case 3; rem " List Close
    break
    case default; rem " Unknown Notify event
    break
  swend
break

```

The number of Notify event types for which you'll need to check

depends on the kind of control and how many of its events interest you. INPUTE and INPUTN only issue two types of Notify events: type 1 is a keypress and type 2 is an error report. At the other extreme, the grid control currently supports twenty-eight different types of Notify events.

If your window includes more than one control of a given type, you can distinguish between the different controls using the control ID (`event.id%` Or `notice.id%`). If your program uses multiple windows, you can distinguish between them using the window context (`event.context%` Or `notice.context%`).

Talking To Controls With SENDMSG()

The new controls added in Visual PRO/5 v2.0 support a much richer interaction than has been available. The SENDMSG() function allows you to send commands in the form of messages to controls, windows, or the operating system, and get back responses and attribute settings.

The syntax of the SENDMSG() function is:

```
result$=SENDMSG(sysgui,objid,msgid,int,str[,context])
```

<i>result\$</i>	Format varies based on each message type (see examples below).
<i>sysgui</i>	This is the channel on which SYSGUI has been opened.
<i>objid</i>	-1 = send message to the operating system. 0 = send message to the window in the current or specified context. 1..65535 = send message to a specific Control ID.
<i>msgid</i>	The list of legal message IDs varies based on the object type (see below).
<i>int</i>	Optional integer argument; set to 0 if unused.
<i>str</i>	Optional string argument; set to "" if unused.
<i>context</i>	Optional context number; if omitted, the current context is assumed.

The following table summarizes the SENDMSG() functions available for the operating system, any window, and various types of controls. Due to the number of messages defined for the controls, they aren't listed in detail here.

Object ID or Control Type	Message Type	Message Description
-1 (System Message)	1	Get System Colors.
0 (Window Message)	1	Get or Set Window Property.

	20	Get Window Context.
	21	Get Window Event Mask.
	22	Get Window Flags.
	23	Set window to accept or reject keyboard change requests.
List button	20..21	See 'LISTBUTTON' supplementary notes.
List edit	20..21	See 'LISTEDIT' supplementary notes.
INPUTE	20..36	See 'INPUTE' documentation.
INPUTN	20..36	See 'INPUTN' documentation.
Tab control	20..39	See 'TABCTRL' documentation.
Grid control	20..82	See 'GRID' documentation.

SENDMSG() Examples

To get the current system menu color:

```
rgb$=sendmsg(sysgui,-1,1,0,$14$)
r=asc(rgb$(1)),g=asc(rgb$(2)),b=asc(rgb$(3))
```

To get the context of the current window:

```
context=dec(sendmsg(sysgui,0,20,0,""))
```

To get the event mask of the current window:

```
mask$=sendmsg(sysgui,0,21,0,"")
```


To get the Table Information Block from a grid control:

```
tblinf$=sendmsg(sysgui,grid_id,20,0,"")
```

To get the number of tabs from a tab control:

```
tabs=dec(sendmsg(sysgui,tabctrl_id,31,0,""))
```

Conclusion

For more complete technical details about these new features and functions, see the sections of the Visual PRO/5 v2.0 Supplement manual that discuss NOTICE(), NOTICETPL(), SENDMSG(), and the new graphical INPUTE, INPUTN, tab and grid controls. You can also download the sample program notify.src, which demonstrates the Notify event and the SENDMSG() function, from the BASIS web site at www.basis.com/advantage/mag-v2n1/notify.src. 

Jim Douglas, a thirteen-year Business Basic veteran, is currently working with BASIS on Volcano™.