How to FLUSH Without Running Out of Resources

Flushing Your Graphics

-
  JanetTerra Jul 22, 2006

Drawing graphics with Liberty BASIC is very easy. Using native Liberty BASIC commands, the programmer can draw lines, circles, boxes and other Turtle graphics. In addition, the commands **LOADBMP** and **DRAWBMP** allows the programmer to draw *bitmaps* retrieved from file.

Graphics consume memory, though, especially when graphics are meant to *persist*. A *persisted graphic* is one that will redraw itself whenever the window becomes minimized, obscured, or dragged offscreen. The Liberty BASIC command that causes a graphic to *persist* or *stick* is **FLUSH**. Run this demo to see how **FLUSH** works.

```
    Nomainwin
    WindowWidth = 787
    WindowHeight = 594
    UpperLeftX = Int((DisplayWidth - WindowWidth) / 2)
    UpperLeftY = Int((DisplayHeight - WindowHeight) / 2)

    Graphicbox #demo.g, 0, 0, 780, 560
    Open "Flushed and Non-Flushed Graphics" for Window as #demo
    Print #demo, "Trapclose [QuitDemo]"

' Put the pen in the down position and assign the color
    Print #demo.g, "Down; Fill Buttonface; Color Buttonface"

' Draw the upper concentric circles
' Place the pen in the center of the upper half
    Print #demo.g, "Place 390 180"
' Choose 5 background colors
    hue$ = "Red Yellow White Blue Green"
' Draw a colorful figure
ct = 1
For i = 100 to 10 Step -10
    Print #demo.g, "Backcolor ";Word$(hue$, ct)
    Print #demo.g, "Circlefilled ";i
    ct = ct + 1
    If ct = 6 Then
        ct = 1
    End If
Next i
' FLUSH the upper picture
```

```
    Print #demo.g, "Flush"

' Draw the lower concentric circles
' Place the pen in the center of the lower half
    Print #demo.g, "Place 390 420"
' Choose 5 background colors
    hue$ = "Red Yellow White Blue Green"
' Draw a colorful figure
For i = 100 to 10 Step -10
    Print #demo.g, "Backcolor ";Word$(hue$, ct)
    Print #demo.g, "Circlefilled ";i
    ct = ct + 1
    If ct = 6 Then
        ct = 1
    End If
Next i
' Don't FLUSH the lower picture

    Wait

[QuitDemo]
    Close #demo
    End
```

Once the program has been executed, drag the window partway off the display screen and then back onto the display screen. The **FLUSH**ed, upper figure becomes redrawn as the window becomes viewable again. The Non-**FLUSH**ed, lower figure does not.

From the Liberty BASIC **Help File**

print #handle, "flush"
This command ensures that drawn graphics 'stick'. Each time a flush command is issued after one or more drawing operations, a new group (called a segment) is created. Each segment of drawn items has an ID number. The segment command retrieves the ID number of the current segment. Each time a segment is flushed, a new empty segment is created, and the ID number increases by one. See also the commands cls, delsegment, discard, redraw, and segment.

FLUSHing does make the final display *persist*, but it does so by reexecuting each step in the sequence of the building of the **FLUSH**ed graphic. To see this reexecution of code in action, run this next demo.

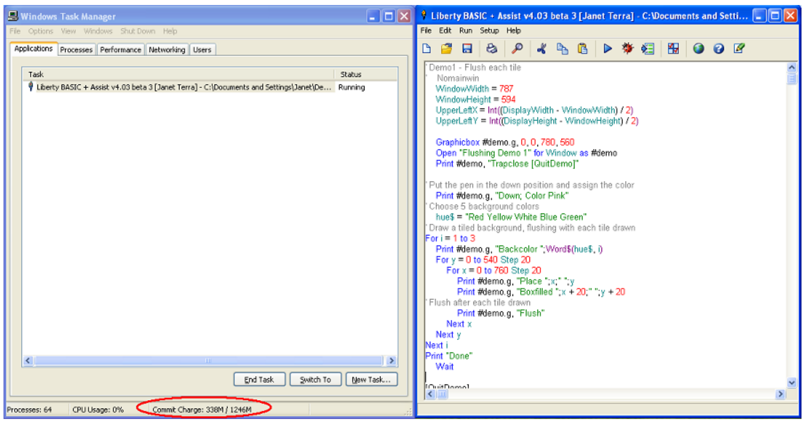
Warning: This demo WILL cause flashing. DO NOT run this demo if you have a history of seizures.

```
    Nomainwin
    WindowWidth = 787
    WindowHeight = 594
    UpperLeftX = Int((DisplayWidth - WindowWidth) / 2)
    UpperLeftY = Int((DisplayHeight - WindowHeight) / 2)

    Graphicbox #demo.g, 0, 0, 780, 560
    Open "Windows Repainting FLUSHED Graphics" for Window as #demo
    Print #demo, "Trapclose [QuitDemo]"

' Put the pen in the down position and assign the color
    Print #demo.g, "Down; Fill Buttonface; Color Buttonface"

' Fill the box with concentric circles
' Place the pen in the center of the graphicbox
    Print #demo.g, "Home"
' Choose 5 background colors
    hue$ = "Red Yellow White Blue Green"
' Draw a colorful figure 5 times
    ct = 1
    For i = 1 to 5
        For j = 700 to 10 Step -10
            Print #demo.g, "Backcolor ";Word$(hue$, ct)
            Print #demo.g, "Circlefilled ";j
            ct = ct + 1
            If ct = 6 Then
                ct = 1
            End If
        Next j
    Next i
' FLUSH the upper picture
    Print #demo.g, "Flush"

    Wait

[QuitDemo]
    Close #demo
    End
```

When the code has been executed, maximize and restore the window several times. Watch how the window *repaint* itself. For more information on *segments* and **FLUSH**ed graphics see Segments and Flushing - a Graphics Tutorial by Alyce Watson in the Liberty BASIC Newsletter, Issue #102.

The Problem with Memory and Graphics

It is obviously good programming technique to keep your graphics **FLUSH**ed. This is especially true when the program calls for the graphics to change throughout the program. Unfortunately, **FLUSH**ing comes at a high cost and that price can very quickly deplete your system's resources. One way to keep an eye on resources is to open ***Task Manager*** and watch the *Commit Charge* numbers in the lower right corner.


Windows Task Manager
File Options View Windows Shut Down Help
Applications Processes Performance Networking Users
End Task
Switch To
New Task...
Liberty BASIC + Assist v4.03 beta 3 [Janet Terra] - C:\Documents and Setti...
File Edit Run Setup Help

Task	Status
Liberty BASIC + Assist v4.03 beta 3 [Janet Terra] - C:\Documents and Settings\Janet\De...	Running

Processes: 64 CPU Usage: 0% Commit Charge: 338M / 1246M

```
' Demo1 - Flush each tile
'    Nomainwin
    WindowWidth = 787
    WindowHeight = 594
    UpperLeftX = Int((DisplayWidth - WindowWidth) / 2)
    UpperLeftY = Int((DisplayHeight - WindowHeight) / 2)

    Graphicbox #demo.g, 0, 0, 780, 560
    Open "Flushing Demo 1" for Window as #demo
    Print #demo, "Trapclose [QuitDemo]"

' Put the pen in the down position and assign the color
    Print #demo.g, "Down; Color Pink"
' Choose 5 background colors
    hue$ = "Red Yellow White Blue Green"
' Draw a tiled background, flushing with each tile drawn
For i = 1 to 3
    Print #demo.g, "Backcolor ";Word$(hue$, i)
    For y = 0 to 540 Step 20
        For x = 0 to 760 Step 20
            Print #demo.g, "Place ";x;" ";y
            Print #demo.g, "Boxfilled ";x + 20;" ";y + 20
' Flush after each tile drawn
            Print #demo.g, "Flush"
        Next x
    Next y
Next i
Print "Done"
    Wait

[QuitDemo]
```

Each **FLUSH** command increases this *Commit Charge*, very quickly to the point of slowing and eventually crashing the program. Open ***Task Manager*** while this next demo runs. You may want to change the

```
For i = 1 to 5
```

to

```
For i = 1 to 3
```

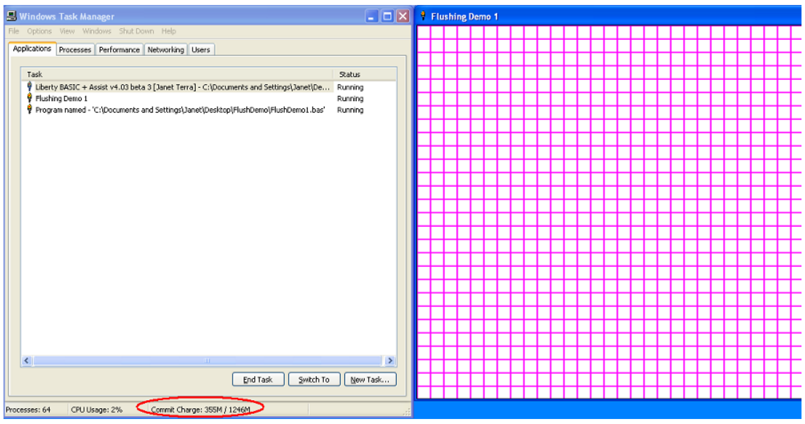
to be sure the program doesn't crash. If you encounter a crash, highlight liberty.exe in Task Manager > Applications and press End Process. While the program is running, the *Commit Charge* gradually, yet steadily, increases. Even when Liberty BASIC is dutifully closed, the resources aren't completely restored to the pre-run level.

The Full Demo

```
' Demo1 - Flush each tile
'    Nomainwin
    WindowWidth = 787
    WindowHeight = 594
    UpperLeftX = Int((DisplayWidth - WindowWidth) / 2)
    UpperLeftY = Int((DisplayHeight - WindowHeight) / 2)

    Graphicbox #demo.g, 0, 0, 780, 560
    Open "Flushing Demo 1" for Window as #demo
    Print #demo, "Trapclose [QuitDemo]"
```

```
' Put the pen in the down position and assign the color
    Print #demo.g, "Down; Color Pink"
' Choose 5 background colors
    hue$ = "Red Yellow White Blue Green"
' Draw a tiled background, flushing with each tile drawn
For i = 1 to 5
    Print #demo.g, "Backcolor ";Word$(hue$, i)
    For y = 0 to 540 Step 20
        For x = 0 to 760 Step 20
            Print #demo.g, "Place ";x;" ";y
            Print #demo.g, "Boxfilled ";x + 20;" ";y + 20
' Flush after each tile drawn
            Print #demo.g, "Flush"
        Next x
    Next y
Next i
Print "Done"
    Wait

[QuitDemo]
    Close #demo
    End
```

Look again at the *Commit Charge*.


Windows Task Manager
File Options View Windows Shut Down Help
Applications Processes Performance Networking Users
End Task Switch To New Task...
Flushing Demo 1

```
Task                                                                              Status
Liberty BASIC + Assist v4.03 beta 3 [Janet Terra] - C:\Documents and Settings\Janet\De...   Running
Flushing Demo 1                                                                   Running
Program named - 'C:\Documents and Settings\Janet\Desktop\FlushDemo\FlushDemo1.bas'   Running

Processes: 64    CPU Usage: 2%    Commit Charge: 355M / 1246M
```

Here it has risen from 338 to 355. The *Commit Charge* will continue to rise with each **FLUSH** until the program crashes.

Using GETBMP and DRAWBMP

What can the programmer do, then, to make graphics *persist* without consuming large resources of memory? One way is to not **FLUSH** each step or *segment* of the draw, but to capture the final output as a *bitmap* and draw the whole display in one step. **GETBMP** captures the graphic display from the screen rather than loading it from a file. **DRAWBMP** then draws that *bitmap*. When a *bitmap* is drawn, the whole picture is drawn as one *segment* rather than a series of individual *segments*.

```
    Nomainwin
    WindowWidth = 787
    WindowHeight = 594
    UpperLeftX = Int((DisplayWidth - WindowWidth) / 2)
    UpperLeftY = Int((DisplayHeight - WindowHeight) / 2)

    Graphicbox #demo.g, 0, 0, 780, 560
    Open "Capturing a Bitmap" for Window as #demo
    Print #demo, "Trapclose [QuitDemo]"

' Put the pen in the down position and assign the color
    Print #demo.g, "Down; Fill Buttonface; Color Buttonface"
' Draw some concentric circles
' Choose 5 background colors
    hue$ = "Red Yellow White Blue Green"
' Position the pen
    Print #demo.g, "Place 100 100"
' Set the counter to 1
    ct = 1
' Draw concentric circles
For i = 100 to 10 Step -10
    Print #demo.g, "Backcolor ";Word$(hue$, ct)
    Print #demo.g, "Circlefilled ";i
    ct = ct + 1
    If ct = 6 Then
        ct = 1
    End If
Next i
' Get the bitmap
    Print #demo.g, "Getbmp ConcentricCircles 0 0 200 200"
' Let mouse click select a position
    Print #demo.g, "When leftButtonUp [DrawBitmap]"

' Wait for mouseclick
    Wait

[DrawBitmap]
' Get the mouse coordinates
    x = MouseX
    y = MouseY
' Clear the screen - Cls clears all graphics memory as well
    Print #demo.g, "Cls"
```

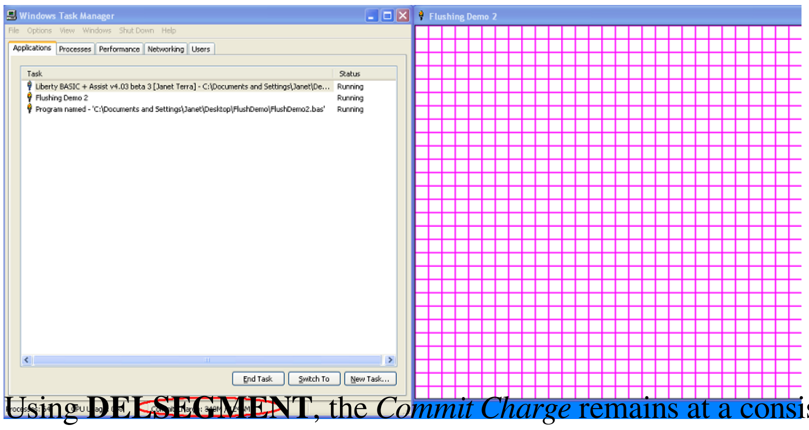
```
' Position the upper left corner at x, y
    Print #demo.g, "Drawbmp ConcentricCircles ";x;" ";y
' Flush the drawing
    Print #demo.g, "Flush"

    Wait

[QuitDemo]
    Close #demo
    End
```

Capturing the Graphic Window

Using this same **GETBMP** technique, the entire window can be captured as a *bitmap*. The drawn *bitmap* is then **FLUSH**ed rather than the multitude of individual *segments*. Demo2 is a modification of Demo1, **FLUSH**ing the drawn *bitmaps*. In addition, as each *bitmap* is drawn, the prior *segment* is deleted with the **DELSEGMENT** command. This allows just one *segment* to remain in memory at any given time, thus relieving the drain on resources required for storing hundreds of *segments*.

```
' Demo2 - Deleting Segments and Flushing Bitmaps
'    Nomainwin
    WindowWidth = 787
    WindowHeight = 594
    UpperLeftX = Int((DisplayWidth - WindowWidth) / 2)
    UpperLeftY = Int((DisplayHeight - WindowHeight) / 2)

    Graphicbox #demo.g, 0, 0, 780, 560
    Open "Flushing Demo 2" for Window as #demo
    Print #demo, "Trapclose [QuitDemo]"

' Put the pen in the down position and assign the color
    Print #demo.g, "Down; Color Pink"
' Capture the initial bitmap
    Print #demo.g, "Getbmp DemoPic 0 0 780 560"
' Get the segment number
    Print #demo.g, "Segment segID"
' Flush
    Print #demo.g, "Flush"

' Choose 5 background colors
    hue$ = "Red Yellow White Blue Green"
' Draw a tiled background, flushing with each tile drawn
For i = 1 to 3
```

```
    Print #demo.g, "Backcolor ";Word$(hue$, i)
    For y = 0 to 540 Step 20
        For x = 0 to 760 Step 20
            Print #demo.g, "Place ";x;" ";y
            Print #demo.g, "Boxfilled ";x + 20;" ";y + 20
' Delete the old segment
            Print #demo.g, "Delsegment ";segID
' Unload the previous bitmap
            Unloadbmp "DemoPic"
' Capture the entire graphic as a bitmap
            Print #demo.g, "Getbmp DemoPic 0 0 780 560"
' Draw the entire window
            Print #demo.g, "Drawbmp DemoPic 0 0"
' Get the segment number
            Print #demo.g, "Segment segID"
' Flush the bitmap
            Print #demo.g, "Flush"
        Next x
    Next y
Next i
Print "Done"
    Wait

[QuitDemo]
    Close #demo
    End
```


Windows Task Manager
File Options View Windows Shut Down Help
Applications Processes Performance Networking Users
End Task Switch To New Task...
Flushing Demo 2

Task	Status
Liberty BASIC + Assist v4.03 beta 3 [Janet Terra] - C:\Documents and Settings\Janet\De...	Running
Flushing Demo 2	Running
Program named - 'C:\Documents and Settings\Janet\Desktop\FlushDemo\FlushDemo2.bas'	Running

Using **DELSEGMENT**, the *Commit Charge* remains at a consistent, safe level. ***Segment.bas*** is an example program that ships with Liberty BASIC. ***Segment.bas*** demonstrates the **DELSEGMENT** command.

When running these demos, you may notice that the **DELSEGMENT**, **GETBMP**, **DRAWBMP**, **SEGMENT**, **FLUSH**, sequence takes a bit more time than just a simple **FLUSH**. In most programs, the increased time will be negligible and virtually imperceptible.

Special Considerations with GETBMP

The **GETBMP** command captures the visible portion of the display. Areas extending beyond these visible limits will result in images of the topmost display. One way to avoid unintended capturing of another window is to keep the Liberty BASIC graphic window topmost with stylebits.

```
            Stylebits #demo, 0, 0, _WS_EX_TOPMOST, 0
```

More experienced Liberty BASIC users may want to first draw the images and then capture those images in memory. The captured *bitmap* can then be drawn to the partially visible graphic window and **FLUSH**ed like any other drawing. For more information on drawing in memory, see Drawing in Memory by Alyce Watson in the Liberty BASIC Newsletter, Issue #101.

Summary

FLUSH is a viable option for preserving graphics when only a few **FLUSH** commands are scattered throughout the program. **FLUSH**ing thousands, hundreds, or even dozens of times in your program will very quickly drain system resources. **CLS** is the most expedient way to free these graphics resources. When **CLS** isn't feasible, consider using the **DELSEGMENT**, **GETBMP**, **DRAWBMP**, **SEGMENT**, **FLUSH** sequence to keep your graphics program running smoothly and to prevent your program from crashing.