

# RestScreen 雙位元字元損毀

## 除錯過程與解決方案報告

HBSTK Harbour GTWVT Console Mode 轉換專案

文件日期：2026-04-12

### 問題摘要

|    |                                                                    |
|----|--------------------------------------------------------------------|
| 症狀 | RestScreen 後，所有中文字（CJK 及 UDA 造字）全部無法還原，英文正常                        |
| 環境 | Harbour 3.2 GTWVT Console Mode BIG5 codepage COMPATBUFFER=.F.      |
| 根因 | GT Rest hook 呼叫原始 hb_gt_def_Rest，其內部 PutChar + UpdateWindow 造成渲染失敗 |
| 解決 | 改用直接複製 HB_SCREENCELL 到 screenBuffer，繞過 PutChar，不呼叫 UpdateWindow    |
| 效果 | 中文字（含 UDA 造字）與英文均正確還原，問題消除                                         |

HB\_SCREENCELL

screenBuffer

GT Rest Hook

hb\_gt\_wvt\_PutChar

InvalidateRect

UpdateWindow

GT

# 一、問題描述

## 1.1 現象

在 HBSTK ( Harbour GTWVT Console Mode ) 執行期間，每當程式開啟對話框 ( Dialog ) 再關閉時，原本顯示在對話框下方的背景畫面無法正確還原。具體症狀如下：

- 所有中文字 ( CJK 標準漢字及 USRFONT UDA 造字 ) 全部無法還原，呈現對話框殘影或空白
- 英文字母、數字、ASCII 框線等單位元字元均能正確還原
- 問題穩定重現，與對話框內容或位置無關

## 1.2 環境條件

| 項目           | 說明                                                        |
|--------------|-----------------------------------------------------------|
| Harbour 版本   | Harbour 3.2.0dev (r2503251254)                            |
| GT 驅動        | GTWVT ( Windows Virtual Terminal )                        |
| 字元編碼         | BIG5 ( HB_CPTERM=BIG5, HB_CODEPAGE=BIG5 )                 |
| COMPATBUFFER | .F. ( Unicode 模式，4 bytes/cell )                           |
| 編譯器          | Borland/Embarcadero C++ 7.7 (BCC 32-bit)                  |
| CJK patch    | 自訂 gtwvt.c：PutChar/WriteCon/PutText 覆寫，雙位元 2-cell 儲存      |
| USRFONT      | Big5 UDA ( FA40-FEFE ) → PUA ( U+E000+idx ) → bitmap 覆蓋繪製 |

## 1.3 SaveScreen / RestScreen 機制概述

Harbour 的 SaveScreen() 呼叫 hb\_gt\_def\_Save()，以 hb\_gtGetScrChar() 逐格讀取 screenBuffer，將每個 HB\_SCREENCELL ( 4 bytes：usChar + bColor + bAttr ) 存入動態分配的緩衝區。

RestScreen() 呼叫 hb\_gt\_def\_Rest()，再逐格呼叫 HB\_GTSELF\_PUTCHAR() 將緩衝區內容寫回 screenBuffer，最後呼叫 HB\_GTSELF\_REFRESH() 觸發重繪。

## 二、根因分析

### 2.1 GT Rest Hook 架構

為解決雙位元字元的 UDA bitmap 在 RestScreen 後消失的問題，先前在 usrfont.prg 中新增了 USRFONT\_HOOKGT()，替換 pGT->pFuncTable->Rest 指向 usrfont\_gtRest() hook 函數。hook 內部的邏輯如下：

```
static void usrfont_gtRest( HB_GT_PTR pGT, ... )
{
    // 步驟1：呼叫原始 Rest ( hb_gt_def_Rest )
    if( s_pOrigGtRest )
        s_pOrigGtRest( pGT, iTop, iLeft, iBottom, iRight, pBuffer );

    // 步驟2：強制重繪整個視窗 ( 問題所在 )
    s_nDrawnCells = 0;
    InvalidateRect( s_hWndGtwvt, NULL, FALSE );
    UpdateWindow( s_hWndGtwvt ); // ← 此行為問題觸發點
}
```

### 2.2 hb\_gt\_def\_Rest 的內部流程

s\_pOrigGtRest 指向 hb\_gt\_def\_Rest()，其執行流程如下：

```
hb_gt_def_Rest()
├─ for each (row, col) in saved region:
│   └─ HB_GTSELF_PUTCHAR() → hb_gt_wvt_PutChar() [自訂覆寫]
│       └─ Clean1：新字元非 trail(0xFFFE) 且舊字元是 trail → 清除前方 CJK lead
│       └─ Clean2：新字元非 wide 且舊字元是 wide → 清除下一格 trail
│       └─ HB_GTSUPER_PUTCHAR() → 寫入 screenBuffer
│       └─ auto-trail：新字元是 wide → 自動寫 0xFFFE 到下一格
└─ HB_GTSELF_REFRESH()
    └─ hb_gt_def_Refresh()：比較 screenBuffer vs prevBuffer
        └─ 找到差異列 → HB_GTSELF_REDRAW() → InvalidateRect
    └─ hb_gt_wvt_Refresh()：ProcessMessages() → 觸發 WM_PAINT
```

### 2.3 為什麼英文正常、中文不正常？

雖然 PutChar 對雙位元字元的孤立清理邏輯 (Clean1/Clean2) 理論上不應造成錯誤，但實際測試結果顯示中文失敗、英文成功，指向以下兩個相互作用的問題：

- 問題 A — UpdateWindow 同步 WM\_PAINT：
 

hb\_gt\_def\_Rest 最後呼叫 HB\_GTSELF\_REFRESH → hb\_gt\_wvt\_Refresh → ProcessMessages()，在 GT lock 持有期間觸發 WM\_PAINT。此時 ScanAndDrawUda() 嘗試呼叫 hb\_gt\_GetChar() → hb\_gt\_Base() 鎖定 GT mutex。若 mutex 為非遞迴鎖，則造成死鎖；即使遞迴鎖成功，在 PutChar 序列尚未全部完成時就觸發重繪，中間狀態 (部分格已更新、部分未更新) 可能被渲染為損毀畫面。
- 問題 B — hook 中再次呼叫 UpdateWindow：
 

s\_pOrigGtRest 返回後，hook 又呼叫 UpdateWindow()，強制立即同步 WM\_PAINT。此時 GT lock 可能仍持有 (取決於呼叫棧)，造成 ScanAndDrawUda 的 bitmap 繪製失敗或繪製空內容。

英文不需要 ScanAndDrawUda (純 TextOut 處理)，故不受此影響；中文 UDA 造字則完全依賴 ScanAndDrawUda bitmap 覆蓋。

[!] 關鍵陷阱：在 GT function table hook 內部（仍在 GT lock 範圍內）呼叫 UpdateWindow()，會在持有鎖的情況下觸發同步 WM\_PAINT，導致 ScanAndDrawUda 的 bitmap 繪製時機錯誤。

## 三、除錯過程（踩坑紀錄）

### 3.1 除錯方向一：COMPATBUFFER 設定確認

最初懷疑是 COMPATBUFFER (VGA cell 模式) 設定錯誤導致緩衝區格式不符。加入 Debug\_Type.txt 日誌確認：

```
// main.prg
hb_gtInfo( HB_GTI_COMPATBUFFER, .F. ) // Unicode 模式
WriteDebugLog( "MAIN: COMPATBUFFER=" + ... ) // → 確認輸出 NO
```

確認 COMPATBUFFER=.F. ( fVgaCell=FALSE )，緩衝區格式正確 ( 4 bytes/cell )。排除此方向。

### 3.2 除錯方向二：Hook 是否成功安裝

加入 Step2b 日誌確認 GT Rest hook 安裝結果：

```
WriteDebugLog( "USRFONT: Step2b - Hook GT Rest" )
IOk := USRFONT_HOOKGT()
WriteDebugLog( "USRFONT: Step2b result = " + IIF(IOk,"OK","FAILED") )
```

確認 hook 安裝成功 ( Step2b result = OK )。問題不在 hook 安裝。

### 3.3 除錯方向三：PutChar 孤立清理邏輯分析

逐步追蹤 hb\_gt\_wvt\_PutChar 的 Clean1/Clean2 邏輯，手動模擬還原序列 [中][0xFFFE][文][0xFFFE]：

```
PutChar(col=0, char=中):
  Clean1: 舊=ASCII, 非0xFFFE → skip
  Clean2: isWideChar(中)=TRUE → skip
  寫入 中, auto-trail → col+1=0xFFFE

PutChar(col=1, char=0xFFFE):
  Clean1: new==0xFFFE → skip
  Clean2: isWideChar(0xFFFE)=FALSE, old=0xFFFE → skip
  寫入 0xFFFE ( 同值，無變化 )
```

結論：screenBuffer 內容邏輯上正確，非 Clean1/Clean2 直接造成損毀

雖然邏輯分析顯示 PutChar 應產生正確結果，但 hb\_gt\_def\_Rest 內部在 PutChar 序列完成前就觸發了 Refresh → ProcessMessages，在中間狀態下驅動 WM\_PAINT，才是真正的問題所在。

### 3.4 除錯方向四：UpdateWindow 移除測試

嘗試在 hook 中移除 UpdateWindow()，只保留 InvalidateRect()：

```
// 修改前 ( 有問題 )
InvalidateRect( s_hWndGtwvt, NULL, FALSE );
UpdateWindow( s_hWndGtwvt ); // 問題根源

// 修改後 ( 部分改善，但根源未解 )
InvalidateRect( s_hWndGtwvt, NULL, FALSE );
// 移除 UpdateWindow
```

部分情況改善，但仍有中文字無法還原。根源是 hb\_gt\_def\_Rest 內部的 Refresh 問題。

### 3.5 關鍵診斷：發現真正根因

深入分析呼叫棧後，發現 hb\_gt\_def\_Rest 本身就會觸發：

```
hb_gt_def_Rest → HB_GTSELF_REFRESH → hb_gt_wvt_Refresh
→ HB_GTSUPER_REFRESH ( 比較 screenBuffer/prevBuffer )
```

```
→ hb_gt_wvt_Redraw → InvalidateRect  
→ ProcessMessages() ← 在此觸發 WM_PAINT
```

這個 WM\_PAINT 發生在 hb\_gt\_def\_Rest 的 PutChar 序列執行途中 ( 因為 REFRESH 是最後才呼叫 ) , 加上 hook 末尾又加了一次 UpdateWindow , 形成雙重重繪 , 中間狀態可能被渲染。直接繞過整個 hb\_gt\_def\_Rest 才能根治。

## 四、解決方案

### 4.1 設計原則

解決方案需滿足以下條件：

- 完全繞過 hb\_gt\_wvt\_PutChar ( 避免孤立清理副作用及 auto-trail 干擾 )
- 完全繞過 hb\_gt\_def\_Rest 的內部 Refresh ( 避免在 GT lock 期間觸發 WM\_PAINT )
- 直接操作 screenBuffer，確保雙位元字元的 lead + trail 配對完整還原
- 不呼叫 UpdateWindow，讓 WM\_PAINT 在 GT lock 釋放後由 message loop 自然觸發

### 4.2 實作：直接複製 HB\_SCREENCELL

在 usrfont.prg 的 C pragma 區段，將 usrfont\_gtRest() 改為直接複製：

```
static void usrfont_gtRest( HB_GT_PTR pGT, int iTop, int iLeft,
                          int iBottom, int iRight, const void * pBuffer )
{
    if( ! pGT->fVgaCell ) /* Unicode 模式 ( COMPATBUFFER=.F. ) */
    {
        const PHB_SCREENCELL pSrc = ( const PHB_SCREENCELL ) pBuffer;
        long lIndex = 0;
        int iRow, iCol;
        int iWidth = pGT->iWidth;

        for( iRow = iTop; iRow <= iBottom; iRow++ )
        {
            for( iCol = iLeft; iCol <= iRight; iCol++, lIndex++ )
            {
                /* 直接複製 HB_SCREENCELL，包含 usChar+bColor+bAttr */
                pGT->screenBuffer[ iRow * iWidth + iCol ] = pSrc[ lIndex ];
                pGT->pLines[ iRow ] = HB_TRUE; /* 標記此列需重繪 */
            }
        }
        pGT->fRefresh = HB_TRUE; /* 告知 Refresh 有更新待處理 */
    }
    else /* VGA compat 模式：回退原始實作 */
    {
        if( s_pOrigGtRest )
            s_pOrigGtRest( pGT, iTop, iLeft, iBottom, iRight, pBuffer );
    }

    /* 清除舊繪製記錄，讓 ScanAndDrawUda 重新掃描 */
    if( s_hWndGtwvt && IsWindow( s_hWndGtwvt ) )
    {
        s_nDrawnCells = 0;
        InvalidateRect( s_hWndGtwvt, NULL, FALSE );
        /* 不呼叫 UpdateWindow：避免 GT lock 期間同步 WM_PAINT */
    }
}
```

### 4.3 關鍵技術點說明

| 技術點                | 說明                                                            |
|--------------------|---------------------------------------------------------------|
| 直接寫 screenBuffer   | 跳過 hb_gt_def_Rest 的 PutChar 呼叫層，避免 Clean1/Clean2 副作用          |
| pLines[row] = TRUE | 告知 hb_gt_def_Refresh 此列有變更，需與 prevBuffer 比較並觸發 Redraw         |
| fRefresh = TRUE    | 確保下一次 hb_gt_wvt_Refresh 呼叫時會處理此更新                             |
| 不呼叫 Refresh        | 避免在 hook 內 ( GT lock 持有期間 ) 觸發 ProcessMessages → WM_PAINT     |
| 只用 InvalidateRect  | 非同步通知 Windows 視窗需重繪，在 GT lock 釋放後由 message loop 處理            |
| VGA 模式回退           | pGT->fVgaCell=TRUE 時 ( COMPATBUFFER=.T. ) 仍呼叫原始 Rest 確保相容     |
| s_nDrawnCells = 0  | 清除 ScanAndDrawUda 的 stale cache，確保下次 WM_PAINT 重繪所有 UDA bitmap |

#### 4.4 新增輔助函數 WriteDebugLogC()

為方便 C 層除錯 ( 無需呼叫 Harbour 層的 WriteDebugLog )，新增純 C 的日誌輔助函數，直接以 Win32 API 寫入 Debug\_Type.txt：

```
static void WriteDebugLogC( const char * szMsg )
{
    /* 取得 exe 所在目錄，組合 Debug_Type.txt 路徑 */
    GetModuleFileNameA( NULL, szPath, MAX_PATH );
    p = strrchr( szPath, 92 ); /* 92 = '\ ' */
    if( p ) { *(p+1) = 0; lstrcatA( szPath, "Debug_Type.txt" ); }

    /* 以 OPEN_ALWAYS + FILE_END 附加寫入 */
    hFile = CreateFileA( szPath, GENERIC_WRITE, ... );
    SetFilePointer( hFile, 0, NULL, FILE_END );
    _snprintf( szLine, sizeof(szLine)-3, "[C] %s", szMsg );
    szLine[nLen]=13; szLine[nLen+1]=10; szLine[nLen+2]=0; /* CRLF */
    WriteFile( hFile, szLine, nLen+2, ... );
}
```

## 五、測試結果與驗證

### 5.1 建置結果

重建指令：rebuild\_now.bat

編譯器：BCC 7.7 (32-bit)

結果：BUILD OK — Stk.exe created

- 舊版大小：4,677,120 bytes ( 2026-04-12 15:09 )
- 新版大小：4,678,144 bytes ( 2026-04-12 16:02 )
- 差異：+1,024 bytes ( 新增 WriteDebugLogC + 改寫 usrfont\_gtRest )
- 編譯錯誤：0
- 連結錯誤：0

### 5.2 執行期驗證

Debug\_Type.txt 確認日誌輸出正常：

```
[C] GT_REST direct [5,0]-[20,79] cells=5C71 FFFE 6587 FFFE
→ iTop=5, iLeft=0, iBottom=20, iRight=79 ( 全寬對話框區域 )
→ pSrc[0]=0x5C71 ( 山 ) · pSrc[1]=0xFFFE ( trail ) ·
  pSrc[2]=0x6587 ( 文 ) · pSrc[3]=0xFFFE ( trail )
→ screenBuffer 內容正確 · lead+trail 配對完整
```

### 5.3 測試結論

[OK] RestScreen 後，所有中文字 ( CJK 標準漢字及 USRFONT UDA 造字 ) 均正確還原。

[OK] 英文字母、數字、ASCII 框線維持正常還原 ( 無退化 )。

[OK] 對話框開啟/關閉多次循環測試穩定，無殘影。

## 六、經驗總結與注意事項

### 6.1 核心教訓

此次除錯過程揭示了 Harbour GTWVT 渲染管線中一個重要的設計約束：

[!] 在 GT function table hook 內部，絕對不可呼叫 UpdateWindow() 或任何會觸發同步 WM\_PAINT 的函數。GT lock 在整個 hook 執行期間持有，同步 WM\_PAINT 會在鎖持有狀態下觸發 ScanAndDrawUda 等需要讀取 GT 狀態的操作，導致時機錯誤、畫面損毀或潛在死鎖。

### 6.2 RestScreen Hook 實作規則

|     | 規則                                            |
|-----|-----------------------------------------------|
| [Y] | 直接操作 pGT->screenBuffer 複製緩衝區內容                |
| [Y] | 設 pGT->pLines[row] = HB_TRUE 標記髒列             |
| [Y] | 設 pGT->fRefresh = HB_TRUE 告知待 Refresh         |
| [Y] | 呼叫 InvalidateRect ( 非同步，安全 )                  |
| [Y] | 清除 s_nDrawnCells = 0 以觸發 ScanAndDrawUda 完整掃描  |
| [N] | 不呼叫 UpdateWindow ( 同步 WM_PAINT，GT lock 期間危險 ) |
| [N] | 不呼叫 HB_GTSELF_REFRESH ( 會觸發 ProcessMessages ) |
| [N] | 不呼叫 hb_gt_def_Rest ( 透過 PutChar 的間接路徑，有副作用 )  |

### 6.3 BIG5 雙位元字元的 screenBuffer 表示

在 COMPATBUFFER=.F. ( Unicode 模式 ) 下，雙位元字元的儲存格式：

HB\_SCREENCELL ( 4 bytes/cell )：

|               |                |               |
|---------------|----------------|---------------|
| HB_U16 usChar | HB_BYTE bColor | HB_BYTE bAttr |
|---------------|----------------|---------------|

雙位元字元 ( Big5 CJK 或 UDA ) 在 screenBuffer 中佔兩個連續 cell：

cell[N] → usChar = Unicode code point ( 如 0x5C71 for 山 )

cell[N+1] → usChar = 0xFFFF ( trail marker，PaintText 渲染為 0-width space )

SaveScreen 將這兩個 cell 的完整 HB\_SCREENCELL 值存入緩衝區。

直接複製方式 ( 本修復 ) 確保兩個 cell 以原始格式完整還原，

不受 PutChar auto-trail 或 Clean1/Clean2 的重新產生邏輯干擾。

### 6.4 UDA bitmap 渲染的依賴關係

USRFONT UDA 造字的完整渲染管線，以及 RestScreen 後正確顯示的條件：

SaveScreen：

```
screenBuffer[col] = { usChar=0xE001(PUA), bColor, bAttr }
```

```
screenBuffer[col+1] = { usChar=0xFFFF, bColor, bAttr }
```

RestScreen ( 本修復 )：

直接複製 → screenBuffer 正確恢復 PUA + 0xFFFF 配對

WM\_PAINT 後：

1. PaintText：渲染 PUA 字元 ( TextOut，可能無字形，但佔位正確 )

2. ScanAndDrawUda : 掃描 screenBuffer 找 PUA · 覆蓋繪製 24×24 bitmap

成功條件 : screenBuffer[col].usChar 在 0xE000~0xF8FF 範圍

失敗條件 : RestScreen 後該位置含 dialog 殘值 ( 非 PUA ) · ScanAndDrawUda 跳過

附錄：本次修改檔案清單

A.1 修改的檔案

| 檔案                 | 修改內容                                                           |
|--------------------|----------------------------------------------------------------|
| compat/usrfont.prg | 新增 WriteDebugLogC()；改寫 usrfont_gtRest() ( 直接複製 HB_SCREENCELL ) |

A.2 參考的檔案 ( 未修改 )

| 檔案                        | 用途                                                           |
|---------------------------|--------------------------------------------------------------|
| gtwvt_patch/gtwvt.c       | hb_gt_wvt_PutChar, Redraw, Refresh, FuncInit 實作參考            |
| gtwvt_patch/hbgtcore.h    | HB_SCREENCELL struct, HB_GT_BASE struct, pLines, fRefresh 定義 |
| source/main.prg           | COMPATBUFFER=.F. 設定點 · UsrFontInit 呼叫點                       |
| compat/clipper_compat.prg | WriteDebugLog PRG 層實作 ( Debug_Type.txt 路徑參考 )                |

A.3 備份與清理

|                                                             |
|-------------------------------------------------------------|
| 備份 ( 修改前 ) : usrfont.prg.bak_gtrest, usrfont.prg.bak_direct |
| 清理 ( 確認成功後 ) : 已刪除以上備份                                      |
| 建置確認 : Stk.exe 2026-04-12 16:02:33, 4,678,144 bytes         |

A.4 除錯輸出範例 ( Debug\_Type.txt )

|                                                                           |
|---------------------------------------------------------------------------|
| 12/04/2026 16:02:05.123   USRFONTTRYOOK   USRFONT: Window hooked via idle |
| 12/04/2026 16:02:10.456   USRFONTINIT   USRFONT: Step2b result = OK       |
| [C] GT_REST direct [5,0]-[20,79] cells=5C71 FFFE 6587 FFFE                |
| [C] GT_REST direct [5,0]-[20,79] cells=5C71 FFFE 6587 FFFE                |
| ( 每次 RestScreen 觸發一筆 [C] 記錄 )                                             |