GTWVT + USRFONT + ASCFONT

合併為單一 LIB 之研究報告

2026-05-06

一、現有架構總覽

目前三個模組分散在不同位置，透過 extern 變數和 WndProc 子類別化（subclassing）耦合：

模組	位置	編譯方式	產出
gtwvt	gtwvt_patch/gtwvt.c + gtwvt.h	build_gtwvt.bat (BCC32 單獨編譯)	gtwvt.lib → 安裝到 Harbour32/lib/
usrfont	c_code/usrfont_c.c (#include 6 個子檔)	hbmk2 編譯進 Stk32.exe	usrfont_c.obj（嵌入 EXE）
ascfont	c_code/usrfont_ascfont.c （被 usrfont_c.c #include）	同 usrfont（單一編譯單元）	同上

1.1 三層渲染架構

第一層：Big5 Codepage Hook — 攔截 wcharGet/Put/Len，將 FA-FE UDA 對應到 Unicode PUA

第二層：GT RestScreen Hook — 直接操作 screenBuffer，繞過 PutChar

第三層：WndProc Hook — 攔截 WM_PAINT，在 PaintText 之後用 ScanAndDrawUda 疊加 bitmap 造字

1.2 現有耦合：extern 變數橋接

變數	定義位置	寫入者	讀取者	用途
g_nCellW / g_nCellH	usrfont_c.c	gtwvt PaintText	ScanAndDrawUda	字元格像素尺寸
g_nMarginLeft / Top	usrfont_c.c	gtwvt PaintText	ScanAndDrawUda	最大化偏移量
g_bHookPaint	usrfont_c.c	usrfont_api (HOOKWND)	gtwvt PaintText	TRUE double buffer 開關
g_hPaintBuf	usrfont_c.c	gtwvt PaintText	ScanAndDrawUda	PaintText 離螢幕 DC

變數	定義位置	寫入者	讀取者	用途
g_bCaretHidden	usrfont_c.c	gtwvt UpdateCaret	ScanAndDrawUda	游標隱藏狀態
g_nFontLfWidth	usrfont_c.c	gtwvt FitSize	usrfont	字型寬度

1.3 WndProc 子類別化機制

usrfont_api.c 的 USRFONT_HOOKWND 用 SetWindowLongPtrW(GWLP_WNDPROC) 將 UsrFontWndProc 插入 GTWVT 視窗。必須用 W（Unicode）版本，才能保留 PUA E311-F8FF 字元不被 ANSI 轉換丟失。

攔截訊息：WM_PAINT（疊加 UDA）、WM_CHAR/WM_IME_CHAR（PUA 字元分配 FA-FE slot）、WM_TIMER(9988)（INSERT 鍵游標同步）、WM_ERASEBKGND（防背景清除）

二、USRFONT 檔案結構

usrfont_c.c 用 #include 方式將 6 個子檔合成一個編譯單元，共享 static 變數：

檔案	行數	職責
usrfont_c.c	744	主入口、全域變數定義、Big5 codepage hook、#include 所有子檔
usrfont_int.h	93	內部標頭：常數(GLYPH_WIDTH=24)、型別(GLYPH_INFO, ASCF_ENTRY, DRAWN_CELL)
usrfont_cache.c	274	ScaleGlyph、CaptureGlyph、EnsureAlpha、DrawUdaAlphaBlend
usrfont_render.c	345	SaveUsrFontUmap/LoadUsrFontUmap（快取持久化）、DrawOneGlyph
usrfont_ascfont.c	685	ASCFONT 渲染：A-path（快取 alpha blend）、B-path（直接 TextOutW）
usrfont_scan.c	400	ScanAndDrawUda — 核心掃描迴圈，分派到 USRFONT/ASCFONT/動態 Unicode
usrfont_wndproc.c	190	UsrFontWndProc — WM_PAINT/WM_CHAR/WM_TIMER 攔截
usrfont_api.c	1128	17 個 HB_FUNC：LOAD/HOOKWND/HOOKCDP/REFRESH/SETPALETTE 等

2.1 ASCFONT 渲染路徑

A-path（快取 alpha blend）：從 ASCFONT.BCACHE 載入預烘焙的 alpha mask，直接混色。~0.1ms/glyph。

B-path（直接 TextOutW）：CreateFontW + TextOutW
即時渲染。用於首次渲染或快取未命中。~5-10ms/glyph。

FontDirect=3 (UDAFONT)：用嵌入的 ASCFONT-STK.TTF，PUA 重映射 + CaptureGlyphAlpha，免磁碟 I/O。

2.2 快取檔案

檔案	格式	內容	用途
USRFONT.24M	原始二進制	24x24 1-bit bitmap，每字 72 bytes	造字字型（最多 785 字）
USRFONT.UMAP	"UFNT" + W/H + count + alpha	動態 Unicode 對應 + 縮放後 alpha mask	USRFONT bitmap 快取

檔案	格式	內容	用途
ASCFONT.UMAP	256 x 2 bytes	ASCII byte → Unicode 映射表	ASCFONT 字元對應
ASCFONT.BCACHE	"ASCF" + W/H + count + alpha	13 個控制字元的 alpha mask	ASCFONT 渲染快取
ASCFONT-STK.TTF	TrueType	PUA 映射的華康框線字型	FontDirect=3 向量字型

三、合併方案研究

3.1 目標

將 gtwvt + usrfont + ascfont 合併成一個 gtwvt.lib。任何 Harbour 程式只要 link gtwvt.lib（-gtwvt），就自動擁有 UDA 造字 + ASCFONT 框線渲染 + TRUE double buffer 功能，無需額外 .c 檔和 .hbp 設定。

3.2 方案 A：多 OBJ 合併進 LIB（推薦）

原理：分別編譯 gtwvt.c 和 usrfont_c.c 成兩個 .obj，用 tlib 打包成一個 .lib。

```
bcc32 -c ... -o gtwvt.obj gtwvt.c
bcc32 -c ... -o usrfont.obj usrfont_c.c
tlib gtwvt.lib +gtwvt.obj +usrfont.obj /P64
copy gtwvt.lib E:\Harbour32\lib\win\bcc\gtwvt.lib
```

stk32.hbp 修改：移除 `..\c_code\usrfont_c.c`（第 211 行），保留 `-gtwvt` 和 `usrfont.prg`。

優點：

- 改動最小，兩個 obj 各自獨立編譯
- extern 變數橋接不變，零程式碼修改
- BCC ICE 風險低（各檔案獨立編譯）
- 可立即實作

缺點：

- WndProc hook 仍需外部 SetWindowLongPtrW（Harbour 啟動後才呼叫 USRFONT_HOOKWND）
- 仍需手動呼叫 UsrFontInit()，不是全自動

3.3 方案 B：深度整合（WndProc 內建）

原理：將 UsrFontWndProc 的邏輯直接寫進 gtwvt.c 的 hb_gt_wvt_WndProc。不需外部 subclassing，GTWVT 自己在 WM_PAINT 尾端呼叫 ScanAndDrawUda。

優點：

- 真正的「一個 LIB 全搞定」— link gtwvt.lib 就有造字功能
- 不需 SetWindowLongPtrW，更乾淨
- extern 變數可改為 pWVT 結構體成員，狀態管理更統一

缺點：

- 需大量修改 gtwvt.c 和 usrfont 程式碼
- BCC ICE 風險高（gtwvt.c 已 5700 行，加入 usrfont 後超過 8000 行）
- WM_CHAR/WM_IME_CHAR 的 PUA 攔截需插入 gtwvt 的 HandleWideChar 流程
- 風險高、工期長

3.4 方案 C：callback 機制（折衷）

原理：gtwvt.c 在 WM_PAINT 結尾預留 callback（函式指標），usrfont 註冊 ScanAndDrawUda。不需 SetWindowLongPtrW，但也不用改 WndProc 主體。

```
/* gtwvt.h */
typedef void (*HB_GT_WVT_PAINT_HOOK)(HDC hdc, RECT *pRect);
void hb_gt_wvt_SetPaintHook(HB_GT_WVT_PAINT_HOOK pFunc);

/* gtwvt.c WM_PAINT 結尾 */
if( s_pPaintHook ) s_pPaintHook( hdc, &ps.rcPaint );

/* usrfont_api.c 初始化時 */
hb_gt_wvt_SetPaintHook( ScanAndDrawUda );
```

優點：

- gtwvt.c 改動極小（加 3 行 callback 呼叫）
- usrfont 仍可獨立維護
- 不需外部 SetWindowLongPtrW
- 其他程式也可以用同樣的 callback 機制擴展 gtwvt

缺點：

- WM_CHAR/WM_IME_CHAR 攔截仍需處理（需第二個 callback）
- 需要修改 gtwvt.h 加入 callback 型別

四、建議實作步驟（方案 A + C 混合）

建議分兩階段：先用方案 A 快速合併 LIB，再用方案 C 優化 hook 機制。

Phase 1：多 OBJ 合併（方案 A）

步驟	內容	風險
1	修改 build_gtwvt.bat：增加編譯 usrfont_c.c → usrfont.obj	低
2	修改 tlib 指令：gtwvt.lib +gtwvt.obj +usrfont.obj	低
3	修改 stk32.hbp：移除 usrfont_c.c 行	低
4	確認 include path：usrfont_int.h 和 Harbour headers	中（BCC ICE）
5	測試：rebuild gtwvt.lib → rebuild Stk32 → 驗證造字/ASCFONT/hover	低

Phase 2：callback 機制（方案 C）

步驟	內容	風險
1	gtwvt.h 加 HB_GT_WVT_PAINT_HOOK 型別和 Set 函式宣告	低
2	gtwvt.c WM_PAINT 結尾加 callback 呼叫	低
3	gtwvt.c WM_CHAR 加 pre-process callback（PUA 攔截）	中
4	usrfont_api.c HOOKWND 改為呼叫 Set*Hook 註冊函式	中
5	usrfont_wndproc.c 重構為 callback 函式（移除 SetWindowLongPtrW）	中
6	移除 EnumWindows 窗口搜索（改由 gtwvt 內部提供 hWnd）	低

Phase 3：extern → pWVT 整合（方案 B 部分）

將 g_nCellW/g_nCellH/g_nMarginLeft/g_nMarginTop/g_bHookPaint/g_hPaintBuf 從 extern 變數改為 pWVT 結構體成員。讓 ScanAndDrawUda 透過 callback 參數接收 pWVT，直接存取，不再依賴全域變數。

這是最乾淨的最終架構，但改動最大，建議在 Phase 1+2 穩定後再考慮。

五、風險分析

風險	影響	對策
BCC ICE	usrfont_c.c 加入 gtwvt.lib 時 BCC32 內部錯誤	保持兩個 .obj 分開編譯（方案 A），不合成單一 .c
符號衝突	gtwvt.c 和 usrfont_c.c 有同名 static 變數	各自為獨立編譯單元，static 不衝突
初始化順序	usrfont 需在 GTWVT 視窗建立後才能 hook	保持 UsrFontInit() 在 Harbour idle 時呼叫
向後相容	舊版 stk32.hbp 仍包含 usrfont_c.c 行	移除該行即可，不影響其他專案
Timer 9988 衝突	gtwvt 有自己的 timer，加入 usrfont timer 後 ID 衝突	Timer ID 9988 與 gtwvt 的 blink timer 不同，無衝突
Debug_Type.txt	usrfont WriteDebugLogC 與 Harbour WriteDebugLog 並存	兩者獨立（C 層 vs PRG 層），不衝突

六、預期成果

合併完成後，使用者的 .hbp 只需要：

```
-gtwvt
mycompat\usrfont.prg
```

不需要再列出 usrfont_c.c。gtwvt.lib 包含：

- GTWVT 終端驅動（含 CJK 修正、TRUE double buffer）
- USRFONT 造字渲染（24x24 bitmap 縮放 + alpha blend）
- ASCFONT 框線/符號渲染（快取 alpha blend + TextOutW fallback）
- 動態 Unicode PUA 分配（FA-FE slot）
- WndProc 攔截（WM_PAINT 疊加 + WM_CHAR PUA 處理）

七、結論

Phase 1（方案 A）可立即實作，預計 30 分鐘內完成。改動僅限 build 腳本和 hbp 設定，零程式碼修改，零風險。

Phase 2（方案 C）建議下次做，需要修改 gtwvt.c 加入 callback 機制，但改動不大（~20 行）。完成後可移除 SetWindowLongPtrW，讓 hook 由 gtwvt 內部控制。

Phase 3（extern → pWVT）是理想目標，但工程量大。建議在系統穩定後再考慮。