

USRFONT / PUA 造字系統

技術手冊

STK 庫存管理系統 - Harbour GTWVT Console 版

Clipper 5.2 → Harbour 32-bit 轉換專案

文件版本: 1.0

日期: 2026-04-11

目錄: E:\ConvertFW\HBSTK\

目錄

1. 系統概述
2. 檔案架構
3. USRFONT.24M 造字檔
4. Unicode PUA 映射機制
 - 4.1 E000-E310 範圍 (標準 Big5 UDA)
 - 4.2 E311-F8FF 範圍 (華康擴展造字)
5. 動態映射與 .umap 持久化
6. GTWVT 渲染流程
7. WM_IME_CHAR 攔截機制
8. 修改檔案清單
9. 開發過程 - 錯誤與解決
10. 技術重點總結

1. 系統概述

本系統將 Clipper 5.2 庫存管理程式 (STK) 轉換為 Harbour 32-bit console 程式，使用 GTWVT 終端驅動在 Windows 視窗中模擬 DOS console。原始系統使用 USRFONT.24M 自訂造字檔來顯示特殊圖示符號 (如 Esc、F1-F12 按鍵圖示)，這些字元存在於華康楷書 W5A (DFKaiShu-SB-Estd-BF) 字型的 Unicode PUA (Private Use Area) 區域。

轉換挑戰: GTWVT 使用 Windows GDI 渲染文字，而非 DOS 的 bitmap 字型。需要一套完整的機制來：

- (1) 將 Big5 造字區碼 (FA40-FEFE) 映射到 Unicode PUA
- (2) 在 GTWVT 畫面上用正確的字型渲染這些 PUA 字元
- (3) 處理超出標準 Big5 UDA 映射範圍的華康擴展造字 (U+EC3F-EDBB)
- (4) 支援 Windows 內碼輸入法 ('Uxxxx) 直接輸入 PUA 字元

2. 檔案架構

2.1 核心檔案

compat/usrfont.prg

USRFONT 造字渲染引擎 (C + PRG 混合)

- Big5 codepage hooks (wcharGet/wcharPut/wcharLen)
- WndProc 子類別 (WM_PAINT/WM_TIMER/WM_IME_CHAR)
- USRFONT.24M bitmap 載入與渲染
- 動態 Unicode 映射 (DynAllocSlot)
- TrueType 字型渲染 (DrawDynUnicodeGlyph)

compat/unicode_ime.prg

Unicode/Big5 內碼輸入法介面

- ParseCodeInput(): Hex 碼解析 (Big5/Unicode 自動判斷)
- UNICODE_TO_BIG5(): Unicode → Big5 轉換 (C 函式)
- BIG5_TO_UNICODE(): Big5 → Unicode 轉換 (C 函式)

compat/clipper_compat.prg

Clipper 相容函式

- BoxChar(): 框線字元擷取
- WriteDebugLog(): 除錯日誌

2.2 資料檔案

build/USRFONT.24M

24x24 bitmap 造字檔 (CMEX 格式)

- 256 bytes 檔頭 + N 個 76 bytes 字形 slot
- 每個 slot: 4 bytes tag + 72 bytes bitmap (3 bytes/row x 24 rows)

build/USRFONT.umap

動態 Unicode 映射檔 (持久化)

- 785 x 2 = 1570 bytes (HB_WCHAR 陣列直接儲存)
- s_aDynUnicode[lookupIdx] = 原始 Unicode code point

3. USRFONT.24M 造字檔

USRFONT.24M 是 CMEX 格式的 24x24 bitmap 造字檔，包含原始 DOS 系統的自訂字形。檔案結構如下：

```
Offset 0x000: File Header (256 bytes)
Offset 0x100: Glyph Slot 0 (76 bytes)
  +0: Tag (4 bytes) = 01 80 xx xx (CMEX tag)
  +4: Bitmap (72 bytes) = 24 rows x 3 bytes/row
Offset 0x14C: Glyph Slot 1 (76 bytes)
...
```

CMEX Tag 轉換公式:

$$\text{CMEX_tag} = \text{serCode} + 0xA9C9$$
$$\text{serCode} = (\text{hi} - 0xA1) * 157 + \text{lo_part}$$
$$\text{lo_part} = \text{lo} - 0x40 + 1 \text{ (0x40-0x7E) 或 } \text{lo} - 0xA1 + 64 \text{ (0xA1-0xFE)}$$

載入時 (USRFONT_LOAD):

1. 讀取檔案到記憶體 (s_pFontData)
2. 掃描每個 glyph slot 的 CMEX tag
3. 轉換 CMEX tag → Big5 碼 → lookupIdx
4. 建立 s_aBig5Lookup[lookupIdx] = glyph slot index
5. 載入 .umap 持久化映射 (LoadDynMap)

4. Unicode PUA 映射機制

4.1 E000-E310 範圍 (標準 Big5 UDA)

Windows CP950 (Big5) 標準映射:

Big5 FA40-FEFE $\leftrightarrow$ Unicode U+E000-E310

Big5 UDA (使用者造字區) 共 785 個碼位:

5 個 hi bytes (FA-FE) $\times$ 157 個 lo values = 785 entries

lo 值: 40-7E (63個) + A1-FE (94個) = 157個

lookupIdx 計算:

$\text{lookupIdx} = (\text{hi} - 0\text{xFA}) * 157 + \text{loOfs}$

loOfs = lo - 0x40 (for 40-7E) 或 63 + (lo - 0xA1) (for A1-FE)

此範圍的字元通過 CP950 標準轉換即可正常工作。GTWVT 收到 WM_CHAR/WM_IME_CHAR 時能正確轉換為 Big5 FA-FE，再經過我們的 wcharGet hook 映射到 PUA U+E000+lookupIdx。

4.2 E311-F8FF 範圍 (華康擴展造字)

華康字型的 PUA 字元分為兩個區域:

U+E000-E03B: 小圖示 (esc, f1-f12 等小寫版)

U+EC3F-EDBB: 大圖示 (ESC, F1-F12 等大寫版)

問題: U+EC3F-EDBB 超出 CP950 映射範圍 (E000-E310)。

Windows CP950 無法轉換這些 Unicode code point $\rightarrow$ Big5。

當使用者透過 Windows 內碼輸入法 ('UECE1) 輸入 U+ECE1 時,

GTWVT 的 CP950 轉換會失敗，字元被丟棄。

解法: DynAllocSlot 動態分配機制

1. 在 Big5 FA-FE 範圍中找一個沒被 USRFont bitmap 占用的空位
2. 將 U+ECE1 記錄在 $s_aDynUnicode[\text{slot}] = 0\text{xECE1}$
3. 將對應的 Big5 碼 (FA-FE hi/lo) 存入畫面緩衝區
4. wcharGet 讀取時回傳 PUA code (E000+slot)
5. ScanAndDrawUda 偵測到 PUA code，查 $s_aDynUnicode[\text{slot}]$ 得到 ECE1
6. DrawDynUnicodeGlyph 用華康 TrueType 字型渲染 U+ECE1 的正確字形

5. 動態映射與 .umap 持久化

動態映射系統確保超出 Big5 標準範圍的 Unicode 字元也能正確顯示和儲存。

5.1 DynAllocSlot 分配策略

分配邏輯 (DynAllocSlot 函式):

1. 先搜尋是否已分配過 (避免重複)
2. 從 BIG5_LOOKUP_SIZE-1 (=784) 往前找空位
(USRFONT bitmap 用 FB 區段在前面，動態分配從 FE 區段往前)
3. 空位條件: $s_aBig5Lookup[i] < 0$ 且 $s_aDynUnicode[i] == 0$
4. 分配後立即 SaveDynMap() 持久化到 .umap 檔

容量: 最多 785 個 slot，扣除 USRFONT bitmap 占用的數量

5.2 .umap 檔案格式

格式: 直接儲存 $s_aDynUnicode$ 陣列 (HB_WCHAR[785])

大小: $785 \times 2 = 1570$ bytes

載入時機: USRFONT_LOAD 完成後呼叫 LoadDynMap()

載入檢查: 只還原 $s_aBig5Lookup[i] < 0$ 的 slot (避免與 USRFONT 衝突)

儲存時機: 每次 DynAllocSlot 分配新 slot 或 wcharPut 註冊新映射時

6. GTWVT 渲染流程

6.1 Big5 Codepage Hooks

usrfont.prg 在 USRFONT_HOOKBIG5() 中攔截 Harbour 的 Big5 codepage，替換三個函式指標：

wcharGet (Big5 → Unicode):

攔截 FA-FE 範圍的 Big5 碼

→ 若有 s_aDynUnicode 映射，判斷是否為天然全形字 (IsNativeFullwidth)

→ 全形字 (CJK 漢字): 回傳原始 Unicode，讓 GTWVT 正常渲染

→ 非全形 (PUA 符號): 回傳 E000+lookupIdx，讓 DrawDynUnicodeGlyph 拉伸

→ 若有 USRFONT bitmap: 回傳 E000+lookupIdx

→ 若都沒有: 查 Windows CP950 註冊映射，然後回傳 E000+lookupIdx

wcharPut (Unicode → Big5):

E000-E310 範圍: 直接轉回 FA-FE Big5 碼

E311-F8FF 範圍: DynAllocSlot 分配 FA-FE 空位

其他 Unicode: 先嘗試原始 Big5 轉換，失敗則 DynAllocSlot

wcharLen (字元寬度):

E000-F8FF 範圍: 回傳 2 (全形)

s_aDynUnicode 中有映射的: 回傳 2

其他: 交給原始 handler

6.2 WndProc 子類別

UsrFontWndProc 子類別 GTWVT 視窗，處理:

WM_PAINT:

1. 殘影偵測: 檢查上次畫 bitmap 的 cell 內容是否改變

→ 有殘影: InvalidateRect 全視窗，讓 GTWVT 完整重繪

2. 呼叫 GTWVT 原始 WndProc 繪圖

3. ScanAndDrawUda: 掃描畫面，用 bitmap/TrueType 覆蓋 PUA 字元

WM_TIMER:

1. 游標閃爍切換

2. 呼叫 GTWVT 原始 WndProc

3. RedrawCursorUda: 只重繪游標所在的 UDA cell

WM_IME_CHAR / WM_CHAR (關鍵!):

攔截 PUA E311-F8FF 範圍的字元

→ DynAllocSlot 分配 FA-FE 空位

→ 替換 wParam 為 E000+slot

→ GTWVT 正常處理 (CP950 可映射 E000-E310)

6.3 ScanAndDrawUda 渲染

掃描整個畫面緩衝區 (hb_gtGetChar) · 對每個 cell:

E000-E310 範圍 (inLookup):

→ 有 USRFont bitmap (s_aBig5Lookup >= 0):

DrawOneGlyph: 24x24 bitmap → StretchBlt 到螢幕

→ 有動態映射 (s_aDynUnicode != 0):

DrawDynUnicodeGlyph: 用 TrueType 字型渲染到 48x48 高解析度 buffer

→ HALFTONE 縮小到目標大小 (抗鋸齒)

E311-F8FF 範圍 (超出 lookup):

→ DrawDynUnicodeGlyph 直接用 usChar 渲染

記錄每個畫過的 cell 位置 (殘影偵測用)

7. WM_IME_CHAR 攔截機制 (關鍵突破)

這是整個 PUA 造字系統最關鍵的技術突破，解決了華康擴展造字區 (U+EC3F-EDBB) 無法透過 Windows 內碼輸入法輸入的問題。

7.1 問題描述

使用者在中文注音輸入法中輸入 'UECE1'，Windows IME 產生 U+ECE1。但 GTWVT 畫面顯示亂碼 (Pt@) 而非華康的 ESC 大圖示。

U+E000-E03B (小圖示) 輸入正常。

U+EC3F-EDBB (大圖示) 全部失敗。

7.2 根本原因分析

Windows CP950 (Big5) 只映射 U+E000-E310 Big5 FA40-FEFE。

U+ECE1 超出此範圍，CP950 無法轉換。

字元流程：

1. Windows IME → WM_IME_CHAR (wParam=0xECE1)
2. GTWVT 收到 WM_IME_CHAR
3. GTWVT 嘗試 CP950 轉換 → 失敗
4. 字元被丟棄或變成替代字元
5. 畫面緩衝區 (hb_gtGetChar) 永遠看不到 U+ECE1
6. wcharPut hook 不被呼叫
7. ScanAndDrawUda 找不到 PUA 字元

所有在 wcharPut/wcharGet/ScanAndDrawUda 層面的修改都無效，因為字元在 GTWVT 的 CP950 轉換階段就已經被丟棄了。

7.3 解決方案

在 UsrFontWndProc 中攔截 WM_IME_CHAR 和 WM_CHAR。

在 GTWVT 處理之前將 E311-F8FF 範圍的字元替換為 E000+slot。

```
if( msg == WM_CHAR || msg == WM_IME_CHAR )
{
    HB_WCHAR wc = (HB_WCHAR) wParam;
    if( wc > (UDA_UNICODE_BASE + BIG5_LOOKUP_SIZE - 1)
        && wc <= 0xF8FF )
    {
        int slot = DynAllocSlot( wc );
        if( slot >= 0 )
            wParam = (WPARAM)(UDA_UNICODE_BASE + slot);
    }
}
```

修改後流程:

1. Windows IME → WM_IME_CHAR (wParam=0xECE1)
2. UsrFontWndProc 攔截 → DynAllocSlot(0xECE1) → slot=774
3. wParam 替換為 0xE306 (= E000 + 774)
4. GTWVT 收到 0xE306 → CP950 轉換成功 (Big5 FE+xx)
5. Big5 碼存入畫面緩衝區
6. wcharGet 讀取 → s_aDynUnicode[774] = 0xECE1
7. ScanAndDrawUda 偵測 → DrawDynUnicodeGlyph(0xECE1)
8. 華康 TrueType 渲染正確的 ESC 大圖示

7.4 關鍵發現

透過 debug log 確認:

- 字元是透過 WM_IME_CHAR (msg=0x0286) 進入，不是 WM_CHAR
- 視窗是 Unicode 視窗 (IsWindowUnicode=1)
- wParam 完整攜帶 16-bit Unicode code point (0xECE1)
- DynAllocSlot 成功分配 slot 774，映射為 U+E306

8. 修改檔案清單

8.1 新建檔案

- compat/usrfont.prg - USRFONT 造字渲染引擎 (約 1400 行 PRG + C)
- compat/unicode_ime.prg - Unicode/Big5 內碼輸入介面 (約 250 行)

8.2 修改檔案

- compat/clipper_compat.prg - 新增 BoxChar(), WriteDebugLog()
- source/main.prg - 新增 UsrFontInit() 呼叫、AddFontResourceEx 載入華康字型
- source/P11.PRG - RUN() → WIN_RUNCMD_HIDDEN() (7處)
- source/P82.PRG - BIG5 編碼修復
- source/P84.PRG - RUN() → WIN_RUNCMD_HIDDEN() (1處)
- source/SYSSETUP.PRG - RUN() → WIN_RUNCMD_HIDDEN() (1處)
- compat/dos_stubs.prg - BIG5 編碼修復
- compat/funcky_compat.prg - BIG5 編碼修復
- build/stk32.hbp - 新增 usrfont.prg, unicode_ime.prg 編譯

8.3 資料檔案

- build/USRFONT.24M - 24x24 bitmap 造字檔 (從原始 DOS 系統複製)
- build/USRFONT.umap - 動態 Unicode 映射檔 (程式自動產生/更新)
- build/DFKaiSB.ttf - 華康楷書 W5A TrueType 字型 (渲染動態 Unicode 用)

9. 開發過程 - 錯誤與解決

9.1 BIG5 編碼損毀

問題: Claude Code 的 Edit 工具在修改 PRG 檔案時，將 Big5 編碼的中文註解自動轉換為 UTF-8，導致程式執行時中文顯示為亂碼。

影響檔案: P11.PRG, P82.PRG, P84.PRG, SYSSETUP.PRG, dos_stubs.prg, funky_compat.prg

解決: 從 .bak 備份檔還原原始 Big5 編碼，再用 perl (二進位安全) 重新套用程式碼修改。

教訓: Big5 編碼的 PRG 檔案不能使用 Edit 工具修改，必須使用 perl 或 Python 做二進位安全的替換。usrfont.prg 因為已經是 UTF-8 編碼 (C 語言區段)，所以可以使用 Edit 工具。

9.2 ParseCodeInput Big5/Unicode 誤判

問題: UnicodeIME() 的 ParseCodeInput 函式將 ECE1 誤判為 Big5 碼。

原因: nHi=0xEC (在 81-FE 範圍) 且 nLo=0xE1 (在 A1-FE 範圍)，符合 Big5 格式。

解決: 加入 PUA 排除條件:

.NOT. (nVal >= 0xE000 .AND. nVal <= 0xF8FF)

後續發現: UnicodeIME() 函式在此情境中根本不會被呼叫。使用者使用的是 Windows 內碼輸入法 ('Uxxxx)，不是我們的 UnicodeIME() 對話框。這段修改雖然正確但對解決問題無幫助，屬於冤枉路。

9.3 ScanAndDrawUda 範圍不足

問題: ScanAndDrawUda 原本只檢查 E000-E310 範圍 (inLookup)。

E311-F8FF 範圍的字元即使出現在畫面緩衝區也不會被渲染。

解決: 新增 else-if 分支處理 E311-F8FF 範圍，直接用 DrawDynUnicodeGlyph 渲染。

同樣的問題也存在於 RedrawCursorUda (游標重繪) 和游標指示 (PatBlit)，一併修正。

結果: 這段修改是正確的但不夠——字元根本不會出現在畫面緩衝區，因為更上游的 GTWVT CP950 轉換就已經失敗了。

9.4 wcharPut DynAllocSlot 路徑 (正確但無用)

問題: 在 wcharPut 中加入 E311-F8FF 的 DynAllocSlot 處理路徑。

理論上 wcharPut 會在 Unicode → Big5 轉換時被呼叫。

結果: wcharPut 永遠不會收到 U+ECE1，因為字元在 GTWVT 內部的 CP950 轉換階段就已經被丟棄了。wcharPut 只會被 Harbour 的字串操作呼叫，不會被 GTWVT 的鍵盤輸入處理呼叫。

這段修改本身是正確的 (用於其他路徑如 DBF 存取)，但無法解決輸入問題。

9.5 真正的突破: WM_IME_CHAR 攔截

經過多輪 debug log 追蹤，發現：

- pua_debug.log 中 ScanUda 從未記錄到 E311+ 範圍的字元
- wcharPut ENTRY log 也從未記錄到 ECE1
- 字元在 GTWVT 內部就被丟棄了

關鍵洞察: 必須在 WM_CHAR/WM_IME_CHAR 層面攔截，在 GTWVT 處理之前替換字元。

第一次嘗試: 在 UsrFontWndProc 中攔截 WM_CHAR (msg == 0x0102)。

加入 debug log 後發現: 完全沒有 WM_CHAR 記錄。

原因: 只記錄了 wc >= 0x80 的 WM_CHAR，但實際上 WM_CHAR 根本沒有攜帶 ECE1。

改為記錄所有 WM_CHAR 後發現: 字元是透過 WM_IME_CHAR (msg=0x0286) 進入的。

最終修正: 同時攔截 WM_CHAR 和 WM_IME_CHAR，成功解決問題。

10. 技術重點總結

10.1 架構層次

```
Windows IME Input
|
v
WM_IME_CHAR (wParam = Unicode)
|
v
UsrFontWndProc [攔截 E311-F8FF -> E000+slot]
|
v
GTWVT WndProc [CP950: E000+slot -> Big5 FA-FE]
|
v
hb_gt Screen Buffer [Big5 FA-FE bytes]
|
v
wcharGet [FA-FE -> PUA E000+lookupIdx]
|
v
ScanAndDrawUda [PUA -> s_aDynUnicode[slot]]
|
v
DrawDynUnicodeGlyph [TrueType render U+ECE1]
```

10.2 核心技術

1. Big5 Codepage Hook: 攔截 Harbour 的 wcharGet/wcharPut/wcharLen，實現自訂 Big5 Unicode 映射
2. WndProc 子類別: SetWindowLongPtrW 攔截 GTWVT 的 WM_PAINT/WM_TIMER/WM_IME_CHAR
3. DynAllocSlot: 動態分配 FA-FE 空位來映射超出 CP950 範圍的 Unicode 字元
4. .umap 持久化: 映射關係存檔，程式重啟後自動載入，不需重新分配
5. 雙重渲染: USRFONT bitmap (DrawOneGlyph) + TrueType (DrawDynUnicodeGlyph)
6. 高解析度縮放: 4x 放大渲染 (48x48) 再 HALFTONE 縮小到目標尺寸，品質遠優於放大
7. 殘影偵測: 記錄每次畫 bitmap 的 cell 位置和內容，下次 WM_PAINT 前比對是否改變
8. IsNativeFullwidth: 區分天然全形字 (CJK 漢字) 和需要拉伸的半形字 (PUA 符號)

10.3 注意事項

1. PRG 編碼: Big5 編碼的 PRG 不能用 Edit 工具修改，必須用二進位安全的方法 (perl/Python)
2. 字型名稱: Windows GDI 註冊的字型名稱是簡體 (華康楷體W5-A)，Big5 顯示名稱是繁體 (華康楷書W5A)
3. CP950 限制: 只映射 E000-E310，超出範圍的 PUA 必須在 WM_IME_CHAR 層攔截
4. WM_IME_CHAR vs WM_CHAR: Windows 內碼輸入法的字元是透過 WM_IME_CHAR (0x0286) 傳送
5. slot 分配方向: 從 FE 區段往前分配，避免與 FB 區段的 USRFONT bitmap 衝突