

BCACHE 離線備援系統完整參考手冊

日期：2026-04-26 專案：HBSTK (Harbour GTWVT Console Mode)

目錄

- 一、系統概述
- 二、字形檔案說明
- 三、字形渲染流程
- 四、BCACHE 保護機制
- 五、EXE 打包與啟動流程
- 六、System.ini 設定詳解
- 七、BCACHE 生成方式
- 八、多站部署
- 九、BCC ICE 問題與解決
- 十、C Code 分離架構 (脫離 BEGINDUMP)
- 十一、BCacheThreshold 即時銳化
- 十二、關鍵經驗教訓
- 十三、相關檔案清單

一、系統概述

STK 庫存管理系統使用華康楷書W5A 的造字區 (Big5 UDA FA40-FEFE) 顯示自定義字形。當程式部署到沒有華康字型的電腦時，造字無法顯示。

BCACHE (Binary Cache) 離線備援系統的解決方案：

- 在有華康字型的開發機預先渲染所有造字字形
- 將渲染結果儲存為 BCACHE 檔案，打包進 EXE
- 部署到任何電腦都能顯示造字，無需安裝華康字型
- 針對 10 種常見螢幕解析度預備多套 BCACHE
- 工作站自動成長：遇到新造字自動收集存入本機 BCACHE

二、字形檔案說明

2.1 造字相關檔案

檔案	大小	用途	共用/獨立
USRFONT.24M	6 KB	84 個造字原始點陣 (24x24 1-bit)	共用
USRFONT.UMAP	1.5 KB	785 格 Big5 UDA -> Unicode PUA 對照	共用

檔案	大小	用途	共用/獨立
USRFONT.BCAC HE	~190 KB	造字 alpha mask 快取 (8-bit)	各站獨立
ASCFONT.15	100 KB	13 個控制字元原始字形	共用
ASCFONT.UMAP	512 B	256 格 ASCII -> Unicode 對照表	共用
ASCFONT.BCAC HE	~10 KB	控制字元 alpha mask 快取	各站獨立

2.2 BCACHE 檔案格式

Header (10 bytes):

[0-3] Magic: "UFNT" (USRFONT) / "ASCF" (ASCFONT)

[4-5] Width: cell width (WORD, Little-Endian)

[6-7] Height: cell height (WORD, Little-Endian)

[8-9] Count: glyph count (WORD, Little-Endian)

Entries:

USRFONT: [2 bytes idx] + [W * H bytes alpha]

ASCFONT: [1 byte asc] + [W * H bytes alpha]

2.3 多套 BCACHE 解析度對照表 (80x25 模式)

螢幕解析度	USRFONT (fullW x H)	ASCFONT (halfW x H)	生成方式
1024x768	24x27	12x27	Native
1280x720	30x25	15x25	Native
1280x1024	30x37	15x37	Native
1366x768	32x27	16x27	Native
1440x900	34x32	17x32	Native
1536x864	38x32	19x32	Native
1600x900	38x32	19x32	Native (=1536x864)
1920x1080	46x40	23x40	Native (最大化)
1920x1080	38x39	19x39	Dev (非最大化)
1920x1200	48x46	24x46	Native

檔名格式：USRFONT_{fullW}x{cellH}_{螢幕寬}x{螢幕高}.BCACHE

fullW = cellW x 2 (CJK 全形) · cellH = GTWVT 實際 cell 高度

三、字形渲染流程

3.1 三種字形來源

類型	數量	來源	渲染方式	需 BCACHE
.24M 點陣	84	USRFONT.24M (24x24 1-bit)	ScaleGlyph1bit 即時縮放	不需要
動態字形	47+	華康楷書W5A TrueType	CaptureGlyphAlpha 即時渲染	需要
ASCFONT	13	華康楷書W5A 控制字元	DrawAscFontDirect/Fro mCache	需要

3.2 ScanAndDrawUda 渲染流程

WM_PAINT:

1. scan screen cells
2. UDA char (U+E000~U+E30F):
 - a. s_aUfntAlpha[idx] has cache -> DrawUdaAlphaBlend
 - b. s_aBig5Lookup[idx] >= 0 -> ScaleGlyph1bit (.24M)
 - c. s_aDynUnicode[idx] != 0 -> CaptureGlyphAlpha (TrueType)
3. control char (< 0x20 or 0x7F) -> DrawAscFontGlyph
4. s_nUfntDirty=1 -> SaveUsrFontUmap (write BCACHE)

3.3 LoadUsrFontUmap 載入流程

LoadUsrFontUmap(nW, nH):

1. open file, verify header (do NOT clear cache first) [FIX-2]
2. W/H match -> load glyphs (skip .24M glyphs) [Step 1]
3. W/H differ -> SCALE from BCACHE [FIX-2 SCALE]
4. file missing -> return FALSE -> EnsureUfntAlpha

四、BCACHE 保護機制 (6 道防線 + 字形數保護)

代號	位置	問題	修正
FIX-1	USRFONT_LOA D	載入.24M時清空 alpha cache	不清空，保留離線備援
FIX-2	LoadUsrFontU map	先清空再驗證，失敗時 cache 已空	先驗證再清空 + 不符就縮放
FIX-3	SaveUsrFontU map	無條件覆寫，少量字形蓋掉 完整版	新 count < 舊 count 不覆寫
FIX-4	USRFONT_LOA D	載入.24M時清空 ASCFONT cache	不清空 ASCFONT
FIX-5	LoadAscFontU map	尺寸不符時重新生成(需華康)	從 BCACHE 縮放載入
overwrite= 0	STK_EXTRACT_ RES	BCACHE每次啟動被EXE資源 覆寫	BCACHE 改為不覆寫

代號	位置	問題	修正
字形數保護	STK_SELECT_B CACHE	bcache/ 版本覆蓋收集的新字形	本機字形數 >= bcache/ 就跳過

五、EXE 打包與啟動流程

5.1 資源覆寫規則

資源	overwrite	說明
USRFONT.24M	1 (覆寫)	造字定義隨 EXE 更新
USRFONT.UMAP	1 (覆寫)	對照表隨 EXE 更新
USRFONT.BCACHE	0 (不覆寫)	保留工作站收集的字形
ASCFONT.15	1 (覆寫)	ASCFONT 定義隨 EXE 更新
ASCFONT.UMAP	1 (覆寫)	ASCFONT 對照表隨 EXE 更新
ASCFONT.BCACHE	0 (不覆寫)	保留工作站 ASCFONT 快取
bcache/*.BCACHE	1 (覆寫)	多套 BCACHE 隨 EXE 更新
stk.ini	0 (不覆寫)	保留使用者設定
system.ini	0 (不覆寫)	保留使用者設定

5.2 啟動流程

Stk32.exe startup:

1. STK_EXTRACT_RES() extract EXE resources
2. STK_EXTRACT_MULTI_BCACHE() extract multi-BCACHE to bcache/
3. LoadIni(System.ini/stk.ini) load settings
4. Font setup hb_gtInfo(FONTNAME/SIZE/WIDTH)
5. SetMode(Rows, Cols) set screen size
6. STK_SELECT_BCACHE() match best BCACHE by actual cell size
7. FontDirect check 0=auto 1=force font 2=force BCACHE
8. UsrFontInit("USRFONT.24M") init font engine (no cache clear)
9. USRFONT_SETTHRESHOLD(n) set BCacheThreshold from ini
10. First WM_PAINT ScanAndDrawUda starts rendering

六、System.ini 設定詳解

6.1 [Screen] 螢幕/字型

Key	預設	說明
FontName	DFKai-S B	GTWVT 字型名稱 (標楷體) 。必須支援 Big5
Rows	0	螢幕列數。0=自動 (80x25 為主)
Cols	0	螢幕欄數。0=自動
FontSize	0	字型大小 px。0=依螢幕自動計算
pad_x	2	字元格水平額外空白 pixel
pad_y	0	字元格垂直額外空白 pixel

6.2 [FontDirect] 造字渲染模式

值	模式	說明	使用場景
0	自動 (預設)	偵測華康：有 -> 用字型，無 -> 用BCACHE	所有電腦 (推薦)
1	強制字型	清除BCACHE，華康即時渲染	確定有華康時
2	強制BCACHE	不偵測字型，強制用BCACHE	測試BCACHE效果

FontDirect=1 時如果華康未安裝，造字會全部空白！

6.3 [BCACHE] 銳化設定

Key	預設	說明
BCacheThreshold	80	銳化閾值 0~255。僅影響 SCALE 路徑 (縮放載入時)

0=不銳化 (保留原始灰階)，80=預設中等銳化，120=更銳利，255=純黑白。

此值在啟動時由 USRFONT_SETTHRESHOLD() 載入，影響 usrfont_c.c 中 ApplyThreshold() 的行為。

詳見「十一、BCacheThreshold 即時銳化」。

6.4 [UI] 功能 Key

Key	預設	說明
alt_copy	true	啟用 Alt-C 複製 / Alt-V 貼上
close_window	false	是否允許右上角 [X] 關閉程式

6.5 [GTWVT] 擴展全形

Key	預設	說明
ExtWideSymbo l	1	啟用 BIG5 擴展全形符號 (CP950 U+2200-26FF)

6.6 [Print] 列印設定

Key	預設	說明
PRNMETHOD	1	列印方式。0=舊式(dospntgo)・1=新式(PrnAsk)
PRINTER	Epson LQ-1170C	印表機名稱
PRN_SKIPASK	0	跳過列印詢問。0=每次問・1=自動略過
PRN_AUTOFORMAT	1	自動格式。1=列印 2=PDF 3=Excel 4=固定
PRN_PAPER	9x5.5	紙張大小。A4/A3/B4/B5/B6/LA4/LA3/LB4/LB5
PRN_SEND_PRINTER	0	是否直接送印表機
PRN_KEEP_PRN	1	是否保留 .PRN 檔
PRN_LOGRESULT	0	記錄列印結果
PRN_OUTDIR		輸出目錄 (空=預設)

七、BCACHE 生成方式

方案	做法	品質	限制
A: 原生生成	P83 -> GenBCacheAll 自動切換螢幕解析度	最佳	螢幕不支援的解析度無法生成
B: Python 縮放	gen_bcache_sets.py 從高解析度 BCACHE 縮放	次佳	需要 BCacheThreshold 調整
混合 (推薦)	A 能生成的用 A · A 不能的用 B 補齊	最佳混合	

7.1 .24M 字形 vs BCACHE 字形

載入時跳過 .24M [Step 1] : LoadUsrFontUmap 載入 BCACHE 時 · s_aBig5Lookup[idx] >= 0 的字形 (共 84 個) 跳過。 .24M 有原始 24x24 點陣 · ScaleGlyph1bit 任何尺寸都完美 · 不需要 BCACHE。

存檔時只存動態 [Step 2] : SaveUsrFontUmap 只計算 s_aBig5Lookup[i] < 0 的字形。 84 個 .24M 字形不存 BCACHE · 檔案更小。

7.2 GenBCacheAll 自動生成 (P83.PRГ)

GenBCacheAll():

for each resolution in 10 presets:

1. USRFONT_RESETCACHE() clear alpha cache
2. STK_CHANGERESOLUTION(W, H) switch screen resolution
3. wait for GTWVT to repaint
4. read actual cell size from hb_gtInfo()
5. render all glyphs via ScanAndDrawUda
6. SaveUsrFontUmap writes BCACHE
7. copy to bcache/ with resolution-specific name

STK_RESTORERESOLUTION() restore original

八、多站部署

W:\ network drive (mapped to server)
S001\ server
Stk32.exe
System.ini
USRFONT.24M from EXE (overwrite=1)
USRFONT.BCACHE local (overwrite=0, auto-grow)
bcache/ multi-resolution BCACHE sets
Dbf/ shared database
S002\ workstation #2
Stk32.exe
USRFONT.BCACHE local (may differ by resolution)
bcache/
S003~S012\ workstations #3~12 (same layout)

8.1 造字更新

修改 USRFONT.24M 後，用 bat 複製到所有站，或 rebuild EXE (.24M 打包在 EXE，overwrite=1 自動更新)。

8.2 BCACHE 自動成長

工作站遇到新造字 -> EnsureUfntAlpha 生成 -> SaveUsrFontUmap 存入 BCACHE。下次啟動 STK_SELECT_BCACHE 檢查：本機字形數 $\geq$ bcache/ 版本 -> 跳過 (保留收集的)。

九、BCC ICE 問題與解決

9.1 什麼是 BCC ICE

ICE = Internal Compiler Error。BCC 7.73 (Embarcadero 版 Borland C++ Compiler) 在編譯大型 C 檔案時，可能發生內部崩潰，錯誤計數器增加但不顯示具體錯誤行。

9.2 我們遇到的三種 ICE 原因

原因一：dinkumware C++ STL headers (佔 90% 的案例)

bcc32.cfg 內容：

```
-I@..\include\windows\crtl  
-I@..\include\windows\sdk  
-I@..\include\dinkumware <-- C++ STL headers !  
-w!  
-w-inl  
-w-pro
```

dinkumware 是 C++ STL 實作 (<vector>、<string> 等)。純 C 專案不需要。但 bcc32.cfg 預設 -I 包含它，某些 Windows SDK headers 間接 include 到 C++ header 時，就會觸發 ICE。

解法：將 E:\BCC7732\Include\dinkumware 目錄改名為 dinkumware_DISABLED。

改名後不影響純 C 編譯。如果未來需要編譯 C++ 專案，改回原名即可。

原因二：bcc32.cfg 的 -w! flag (Warning 升級為 Error)

bcc32.cfg 有 -w! 設定，意思是「所有 Warning 視為 Error」。這導致正常的 Warning (例如 W8071 型別轉換) 被升級為編譯錯誤。

BCC 輸出只顯示 "Warning W8071" 但錯誤計數為 "**** 1 errors in Compile ****"。看起來像是 ICE (有 error 但沒有 Error 行)，其實只是 warning 被升級。

症狀 (看起來像 ICE)：

```
..\c_code\threshold_c.c:  
Warning W8071 threshold_c.c 12: Conversion may lose significant digits  
*** 1 errors in Compile ***
```

真正原因：bcc32.cfg 的 -w! 把 W8071 升級為 error

解法：修正 C 程式碼消除 warning。例如加上明確的 (BYTE) 型別轉換。

遇到 ICE 時第一步：用 bcc32 -v flag 手動編譯，看完整輸出。很多時候不是真正的 ICE。

原因三：真正的 ICE — C 檔案太大

usrfont_c.c 有 137KB、3700+ 行 C 程式碼。BCC 7.73

在處理這種大檔時，如果再加入更多函式，可能真的觸發 ICE (記憶體或狀態機溢出)。

解法：將小型獨立函式拆到另一個 .c 檔 (例如 threshold_c.c)。詳見「十、C Code 分離架構」。

9.3 ICE 排查 SOP

Step 1: 用 -v flag 手動編譯

```
bcc32 -c -v -IE:\BCC7732\INCLUDE\windows\crtl \  
-IE:\BCC7732\INCLUDE\windows\sdk \  
-IE:\Harbour32\include \  
file.c
```

Step 2: 檢查是否有 Warning 被 -w! 升級

-> 看到 Warning 但 error count > 0 => 修正 Warning

Step 3: 檢查 dinkumware path

-> "Unable to open include file" for C++ header

-> rename dinkumware -> dinkumware_DISABLED

Step 4: 真正的 ICE (無任何錯誤訊息但 error count > 0)

-> 檔案太大，拆分函式到新 .c 檔

-> 或檢查是否有 C89 語法錯誤 (BCC 不支援 C99)

十、C Code 分離架構 (脫離 BEGINDDUMP)

10.1 傳統做法：BEGINDDUMP

Harbour 的 BEGINDDUMP...ENDDUMP 指令可以在 PRG 檔案中嵌入 C 程式碼。Harbour 編譯器會將這段 C 程式碼原封不動地放入生成的 .c 檔中。

```
// myfile.prg
FUNCTION MyPrgFunc()
    RETURN MY_C_FUNC()

#pragma BEGINDDUMP
#include "hbapi.h"
HB_FUNC( MY_C_FUNC ) {
    hb_retni(42);
}
#pragma ENDDUMP
```

問題：當 C 程式碼很長 (usrfont 有 3700 行)，PRG 檔案變得巨大且難以維護。更嚴重的是，BCC 編譯 Harbour 生成的 .c 檔 (包含 PRG 轉譯碼 + BEGINDDUMP C 碼) 可能超過編譯器的大小限制觸發 ICE。

10.2 新做法：獨立 .c 檔

將 BEGINDDUMP 中的 C 程式碼提取到獨立的 .c 檔案，直接在 .hbp 中引用。

步驟 1：建立 c_code/ 目錄

```
E:\ConvertFW\HBSTK\
c_code/
    usrfont_c.c    <-- 從 usrfont.prg BEGINDDUMP 提取 ( 137KB )
    threshold_c.c <-- 獨立的小型 C 函式 ( 避免 ICE )
```

步驟 2：從 PRG 移除 BEGINDDUMP

```
// usrfont.prg - 移除前
FUNCTION UsrFontInit( cFile )
    ...
#pragma BEGINDDUMP
// ... 3700 lines of C code ...
#pragma ENDDUMP

// usrfont.prg - 移除後
EXTERNAL USRFONT_SETTHRESHOLD    <-- 告訴 Harbour 此函式在外部

FUNCTION UsrFontInit( cFile )
    ...
```

步驟 3：在 .hbp 中引用 .c 檔

```
# stk32.hbp
# =====
# C Code (extracted from PRG BEGINDDUMP)
# =====
```

```
..\c_code\usrfont_c.c
..\c_code\threshold_c.c
```

步驟 4：PRG 中使用 EXTERNAL 宣告

當 HB_FUNC() 定義在外部 .c 檔時，PRG 必須用 EXTERNAL 告訴 Harbour 連結器。

```
// usrfont.prg
EXTERNAL USRFONT_SETTHRESHOLD // threshold_c.c 中定義

FUNCTION UsrFontInit( cFile )
...
    USRFONT_SETTHRESHOLD( Val( GETE("BCacheThreshold")) )
...
```

沒有 EXTERNAL 宣告，Harbour linker 會報 "Unresolved external" 錯誤！

10.3 HB_FUNC() 與 extern 的關係

```
// threshold_c.c (外部 .c 檔)
#include "hbapi.h"

int s_nBCacheThreshold = 0;    // variable 定義

HB_FUNC(USRFONT_SETTHRESHOLD) {    // Harbour callable
    s_nBCacheThreshold = hb_parni(1);
}

BYTE ApplyThreshold(BYTE v) {    // pure C callable
    if (s_nBCacheThreshold > 0)
        return (BYTE)((v > (BYTE)s_nBCacheThreshold) ? 255 : 0);
    return v;
}

// usrfont_c.c (使用端)
extern int s_nBCacheThreshold;    // 引用 threshold_c.c 的變數
extern BYTE ApplyThreshold(BYTE v); // 引用 threshold_c.c 的函式

// 在 SCALE 路徑中使用：
_pn[y*nW+x] = ApplyThreshold(_po[sy*wW+sx]);
```

10.4 為什麼需要拆分

情境	問題	解法
BEGINDUMP 太長	PRG + C = 超大 .c 檔 -> BCC ICE	提取到獨立 .c 檔
獨立 .c 檔太大	單一 .c 137KB -> 加函式就 ICE	拆到第二個 .c 檔
C code 有 Warning	-w! 升級為 Error -> 編譯失敗	修正 C 碼消除 Warning
HB_FUNC 在外部	Harbour linker 找不到符號	PRG 加 EXTERNAL 宣告

10.5 注意事項

- .c 檔中的 HB_FUNC() 巨集展開為 HB_FUN_funcname()，PRG 通過 EXTERNAL 連結
- 純 C 函式 (如 ApplyThreshold) 不需要 EXTERNAL，link 階段自動解析
- bcc32.cfg 的 -w! 會升級所有 Warning，寫 C code 時必須消除所有 Warning
- BCC 7.73 只支援 C89 語法：變數宣告必須在 block 開頭，不可在 for() 內宣告
- hbm2 會將 .hbp 中列出的 .c 檔與 PRG 生成的 .c 檔一起編譯和連結

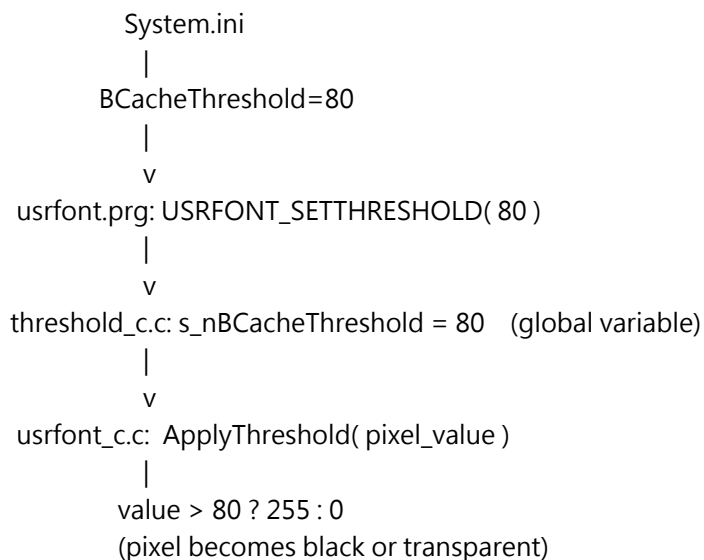
十一、BCacheThreshold 即時銳化

11.1 問題背景

BCACHE 儲存的是 8-bit alpha 值 (0~255 灰階) 。當 SCALE 路徑從不同尺寸的 BCACHE 縮放字形時，灰階邊緣可能造成字形模糊。

BCacheThreshold 機制：將灰階值以閾值二值化，增加字形銳度。

11.2 技術架構



11.3 檔案結構

檔案	內容	角色
threshold_c.c	s_nBCacheThreshold 變數定義 H B_FUNC(USRFONT_SETTHRESH OLD) ApplyThreshold() 函式	定義端
usrfont_c.c	extern s_nBCacheThreshold extern ApplyThreshold() 在 SCALE 路徑呼叫	使用端
usrfont.prg	EXTERNAL USRFONT_SETTHRESHOLD UsrFontInit() 中設值	PRG 介面
System.ini	BCacheThreshold=80	使用者設定

11.4 ApplyThreshold 函式

```
BYTE ApplyThreshold(BYTE v)
{
    if (s_nBCacheThreshold > 0)
        return (BYTE)((v > (BYTE)s_nBCacheThreshold) ? 255 : 0);
    return v; // threshold=0: no change, keep original alpha
}
```

此函式在 usrfont_c.c 的兩個 SCALE 路徑中被呼叫：

- LoadUsrFontUmap SCALE (USRFONT BCACHE 尺寸不符時的縮放載入)
- LoadAscFontUmap SCALE (ASCFONT BCACHE 尺寸不符時的縮放載入)

11.5 閾值效果

BCacheThreshold	效果	適用場景
0	不銳化。保留原始 8-bit alpha 灰階	高解析度、原生 BCACHE
80	預設。較暗的灰階透明化，較亮的變白	一般使用
120	更銳利。只有較亮的像素保留	模糊嚴重時
255	純黑白。只有完全不透明(255)的像素保留	極端除錯

11.6 -w! 陷阱

ApplyThreshold 最初寫為三元運算子一行：

```
// 原始寫法 -> W8071 Warning -> -w! 升級為 Error -> 編譯失敗！
return (s_nBCacheThreshold > 0) ? ((v > s_nBCacheThreshold) ? 255 : 0) : v;
```

```
// 修正寫法 -> 明確 (BYTE) 轉型 -> 無 Warning -> 編譯成功
if (s_nBCacheThreshold > 0)
    return (BYTE)((v > (BYTE)s_nBCacheThreshold) ? 255 : 0);
return v;
```

這就是整個 build 失敗的根因。BCC 的 -w! flag 將型別轉換 warning 升級為 error，但錯誤訊息只顯示 "Warning"，導致看起來像 ICE (有 error count 但沒有 Error 行)。

十二、關鍵經驗教訓

12.1 FreezeWindow 連續性

問題在邏輯，不在 API。FreezeWindow(.F.) 放在錯誤位置造成「裸奔」區間。用盡天下的 API 也沒用，要先追蹤完整流程找邏輯缺口。

12.2 逐行流程追蹤法

Debug SOP:

1. list complete execution order
2. annotate key state at each line
3. mark state transitions with *
4. logic gap becomes visible

12.3 BCC ICE 排查三步驟

- 第一步：用 -v flag 手動編譯看完整輸出 (90% 的 ICE 其實是路徑/語法錯誤)
- 第二步：檢查 bcc32.cfg 的 -w! 是否把 warning 升級為 error
- 第三步：真正 ICE -> 拆分大檔案，或移除 dinkumware include path

12.4 C Code 分離原則

- 大量 C code 不要放在 BEGINDUMP，提取到獨立 .c 檔
- 獨立 .c 檔中的 HB_FUNC 需要 PRG 端用 EXTERNAL 宣告
- BCC 寫 C code 必須消除所有 Warning (因為 -w!)
- BCC 只支援 C89，不可在 for() 內宣告變數

十三、相關檔案清單

檔案	內容
compat/usrfont.prg	USRFONT/ASCFONT PRG 主程式 (BEGINDUMP 已移除)
c_code/usrfont_c.c	造字渲染 C 引擎 (137KB，從 BEGINDUMP 提取)
c_code/threshold_c.c	BCacheThreshold 變數 + ApplyThreshold 函式
source/main.prg	啟動流程、字型設定、STK_SELECT_BCACHE 呼叫
source/TLIB.PRG	FreezeWindow、EXTRACT_MULTI、SELECT_BCACHE (BEGINDUMP)
source/P83.PRG	ShowUDA、ShowASCFONT、GenBCacheAll/One
build/stk32.hbp	專案定義，引用 .c 檔
build/stk32.rc	EXE 資源定義 (RCDATA 101-311)
build/System.ini	螢幕/字型/FontDirect/BCacheThreshold 設定
build/bcache/	10 套多解析度 BCACHE 檔案
E:\BCC7732\bin\bcc32.cfg	BCC 編譯器預設 flags (含 -w!)
E:\BCC7732\Include\dinkum ware_DISABLED	C++ STL headers (已停用)