

STK32 功能提示系統

TopMessage / BottomMessage / WindowMessage
AddHintLabel + 防汙染機制

撰寫日期：2026-05-16

專案：HBSTK (Harbour GTWVT Console Mode)

一、訴求與現象

Stk32 從 Clipper 移植到 Harbour GTWWT

後，原本只有鍵盤操作的「畫面功能提示」（如 [ESC] 離開、[F1] 說明、[PgDn] 翻頁等），需要在保留鍵盤操作的同時，**讓滑鼠也能直接點擊提示文字觸發對應功能**。

具體需求：

- 系統頂部標題列：日期、標題、公司名、ESC/PgUp/PgDn 提示
- 全螢幕底部：可變的功能鍵提示列
- 視窗內部底部：限定範圍的功能列
- 表身狀態列：單一熱區（如 Alt+H 功能提示鈕）
- 跨進入點不可互相汙染（按了 A 模組的鍵卻觸發 B 模組）

二、整體架構（三層設計）

整套提示系統建立在 MsHandler 類別 + 兩個熱區陣列 之上。所有的 TopMessage / BottomMessage / WindowMessage / AddHintLabel 最終都走相同的三層流程：

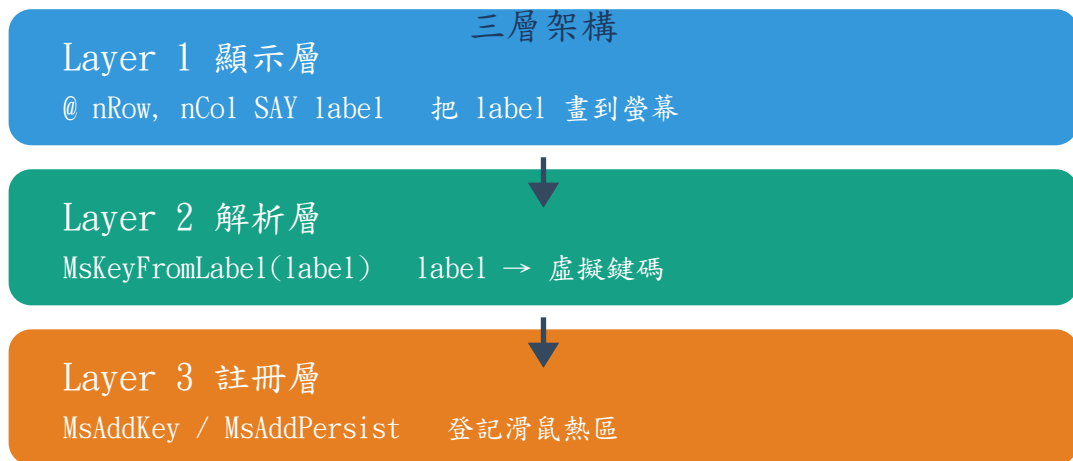


圖 1 三層架構

兩個熱區陣列的差異：

陣列	型別	生命週期	用途
aMsReg	MEMVAR PRIVATE	跨 PRIVATE 自動消失	區域提示 (BottomMessage / WindowMessage / AddHintLabel)
aTopReg	MsHandler DATA	跨 PRIVATE 持久存在	持久提示 (TopMessage / 視窗框按鈕)

三、完整實作流程（5 Step）

從使用者看到提示、滑鼠點擊、到系統收到 InKey 的完整鏈路：



圖 2 從畫面到 InKey 的完整資料流

關鍵零件 MsKeyFromLabel

位於 mouse_handler.prg:638，是「label 文字 → 虛擬鍵碼」的核心翻譯器。重點是用 CUSTOMCHAR.ch 的 KY_* #define 比對，不可硬寫 hex code（因為華康造字 glyph 的編碼會隨字型而變）。

```
/// 範例：把 KY_ESC 解析成 K_ESC，KY_ALT+KY_H 解析成 K_ALT_H
FUNCTION MsKeyFromLabel( cItem )
    cShort := Left( cItem, At(":", cItem) - 1 )
    DO CASE
        CASE cShort == KY_ESC ; RETURN K_ESC
        CASE cShort == KY_F1 ; RETURN K_F1
        ...
    ENDCASE
```

四、三大函式比較

項目	TopMessage	BottomMessage	WindowMessage
原始檔位置	TLIB. PRG:329	TLIB. PRG:1620	WINDOW. PRG:1111
顯示位置	畫面頂部（預設 24 列）	GetMaxRow() 最底列	視窗底部 nWindowBottom-1
參數	cTitle, nRow, lHintMode	xString, cColor	nWindowTop/Left/Bottom/Right, xString, cColor
熱區陣列	aTopReg（持久）	aMsReg（區域）	aMsReg（區域）
顏色變數	cTopMsgColor	cHintBarColor + cHelpHintColor	cHintBarColor + cHelpHintColor
回傳值	NIL	aLocal（熱區清單）	aLocal（熱區清單）
邊界裁切	整列覆蓋	依 MaxCol 截斷	依 nWindowRight 截斷 + 補色

TopMessage 特色（最複雜）

1. lHintMode 參數：.F. 顯示完整 ESC/PgUp/PgDn 熱區；.T. 進入表身編輯時只留公司名，避免誤觸。
2. 標題本身也是熱區：點擊標題會呼叫 ShowHotKeys()（透過第 7 參數 xAction）。
3. MsClearPersist(nRow)：先清掉該列舊熱區，避免疊加。

BottomMessage / WindowMessage 共用迴圈

兩者用幾乎一樣的 FOR 迴圈，畫＋解析＋註冊綁在一起：

```

FOR nI := 1 TO Len(xString)
  nColSav := nCol
  @ nRow, nCol SAY xString[nI] // 畫
  nMsWinKey := MsKeyFromLabel( xString[nI] ) // 解析
  IF nMsWinKey != 0
    MsAddKey( nRow, nColSav, COL()-1, nKey, ... ) // 註冊
  ENDIF
  nCol := COL() + 1
NEXT

```

五、防汙染機制（核心議題）

汙染問題的本質

aTopReg（持久熱區）跨 PRIVATE 不消失。如果直接從 MainMenu 進入 P32，上層的 MainMenu 熱區會殘留 — 使用者在 P32 畫面上點到「對應 MainMenu 座標的位置」就會誤觸功能。反向地，P32 自己 AddPersist 的熱區若沒清，回到 MainMenu 後也會反汙染。

解決方案：進入隔離 + 離開還原（三明治）



圖 3 P32 防汙染三明治結構

MenuEnterIsolate() (TLIB.PRG:283) 會將上層的所有熱區、SayReg、第 0 列畫面整包搬走（不是刪除）回傳給呼叫者；MenuLeaveRestore(aSaved) (TLIB.PRG:295) 負責將整包搬回來。

時機	沒有 isolate	有 isolate
進入 P32	殘留 MainMenu 熱區 → 誤觸	全清空，P32 重新建立乾淨環境

時機	沒有 isolate	有 isolate
退出 P32	P32 熱區洩漏到上層 → 反汙染	整包還原回上層原始狀態

P32 標準三明治寫法

```

PROCEDURE P32( cCallPrg, aData )
  SAVE SCREEN TO cBuffer
  aSavedIsolate := MenuEnterIsolate()  // 進入隔離
  nLastCursor := SetCursor(SC_NONE)
  ...
  PayRece( ProcName(), oBrowse, nType, ... )  // 業務邏輯
  ...
  MenuLeaveRestore( aSavedIsolate )  // 離開還原
  RESTORE SCREEN FROM cBuffer
  SetCursor(nLastCursor)

```

六、PAYRECE / LineMessage 怎麼快速加提示

如果每次加單一熱區都要寫「SAY → MsKeyFromLabel → MsAddKey」三步，整個系統會充滿重複樣板。為此封裝了 AddHintLabel() (mouse_handler.prg:1427)，讓單一熱區註冊可以一行搞定。

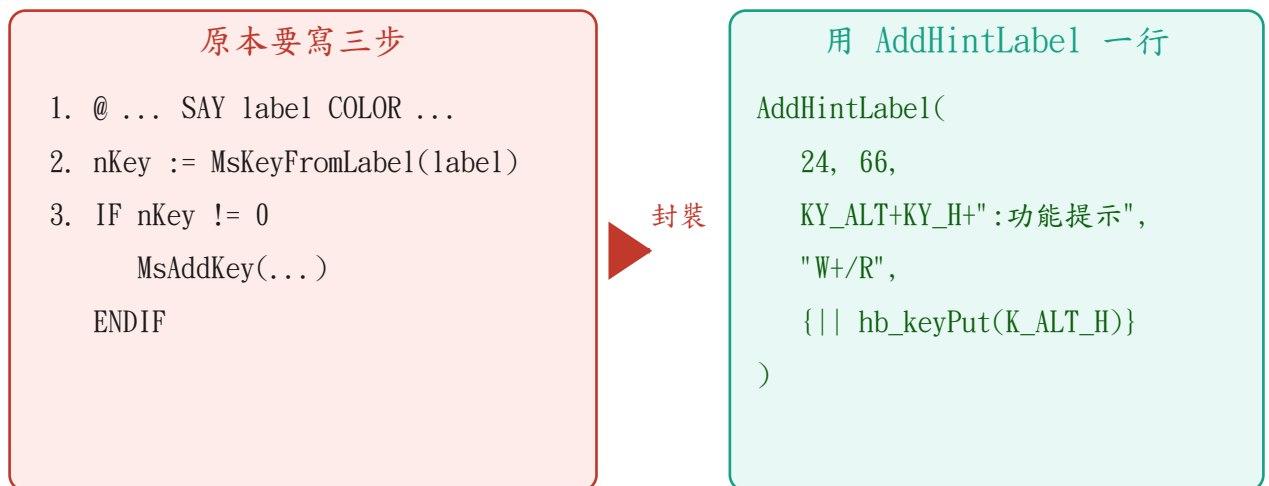


圖 4 三步合一：AddHintLabel 封裝對照

AddHintLabel 兩種模式

第 5 參數 bAction	行為	適用情境
省略	自動 MsKeyFromLabel 解析 KY_* → K_*	純功能鍵，例如 KY_ESC、KY_F1
給 codeblock	點擊時直接 Eval(block) 不走鍵碼	自訂動作，例如 hb_keyPut(K_ALT_H)

PAYRECE.PRG:1227 LineMessage 實際用法

```
STATIC PROCEDURE LineMessage( nSerialNo, nTotalLine )
    Block(22, 65, 24, 79, .F.)
    DevPos(23, 71)
    DevOut( StrZero(nSerialNo,3)+"／"+StrZero(nTotalLine,3),
MEMVAR->cColorNorm )
    AddHintLabel( 24, 66, KY_ALT+KY_H+":功能提示", "W+/R", {||
hb_keyPut(K_ALT_H)} ) )
    RETURN
```

每次表身重畫狀態時，這一行就自動補上 Alt+H 熱區。點擊 → Eval({||
hb_keyPut(K_ALT_H)}) → 主迴圈收到 K_ALT_H → 觸發功能提示視窗。

七、使用情境決策表

依「要加在哪裡」選用對應函式，不要混用：

情境	用什麼	範例
系統頂部標題列	TopMessage()	TopMessage("買賣作業", 24, .F.)
全螢幕底部一整列功能列	BottomMessage()	BottomMessage({KY_ESC, KY_F1, KY_F10})
視窗內部底部一整列	WindowMessage()	WindowMessage(nT, nL, nB, nR, aLabels)
視窗框邊／表身單一提示	AddHintLabel()	AddHintLabel(24, 66, KY_ALT+KY_H+":...")
新進入點（非從主選單來）	MenuEnterIsolate / MenuLeaveRestore	見 P32.PRG 三明治寫法

八、踩坑與陷阱（歷史教訓）

坑	症狀	解法
MsKeyFromLabel 硬寫 hex	改字型後對映全錯	必須用 CUSTOMCHAR.ch 的 KY_* #define 比對
BottomMessage 缺 F1/F2/F5-F8	點不到對應功能鍵	補齊 KeyFromLabel 的 DO CASE 分支
熱區沒清就疊加	同一列多份重疊，誤觸	TopMessage 先 MsClearPersist(nRow)；BottomMessage 自動掃 aMsReg
新進入點沒包 Isolate	上層熱區汙染／下層洩漏	頭尾加 MenuEnterIsolate / MenuLeaveRestore 三明治
全形字寬度算錯	熱區範圍錯位	用 COL()-1 抓 SAY 後位置，不要自己算字長

九、展望：還可加強的方向

A. 統一三函式的顏色 API

TopMessage 只用 cTopMsgColor，Bottom/Window 同時用 cHintBarColor + cHelpHintColor。應該統一成一組「背景／前景／hover」三色變數，或封裝成 GetHintColors(scope) 回傳統一結構。

B. 修補 WindowMessage MEMVAR->lHelp=.T. 的熱區缺漏

WindowMessage 在 lHelp=.T.

的兩段顯示分支 (WINDOW.PR:1152-1157) 沒有呼叫 MsAddKey，導致幫助模式下視窗內提示無法點擊。應補上與下方主要分支相同的 MsKeyFromLabel + MsAddKey 邏輯。

C. 封裝陣列鋪排成共用 helper

BottomMessage 和 WindowMessage 的 FOR

迴圈邏輯重複度極高（只差邊界裁切）。依「封裝共用邏輯」原則（記憶 feedback_encapsulate.md），應抽出 LayoutHintRow(nRow, nLeft, nRight, aLabels, cColor)，讓兩者只負責計算 left/right 邊界，內部共用。

D. AddHintLabel 自動偵測 isolate scope

目前 AddHintLabel 只用

MsAddKey（區域），無法用於需要持久的場景。可加第 6 參數

lPersist，或自動偵測呼叫者是否在 isolate scope 內，決定要寫入 aMsReg 還是 aTopReg。

E. 統一 Isolate scope 入口

目前各 P3X 進入點都自己寫三明治 (SAVE SCREEN + MenuEnterIsolate + 業務 + MenuLeaveRestore + RESTORE SCREEN)。應封裝成：

RunIsolated(bBlock) — 自動處理頭尾，業務邏輯包成 codeblock 傳入。

F. 加 hover 視覺回饋的統一顏色策略

目前 HitTest flash 用寫死的 N/GR、W+/R。應改為從色彩主題 (aColor) 讀，讓深淺主題下都能保持「黃金對比 5~7:1」（記憶 feedback_color_theme_principles.md）。

G. 熱區重疊偵測（除錯工具）

提供一個 Ctrl+F12 模式：把 aTopReg + aMsReg
所有熱區用半透明色塊疊到畫面上，並偵測「同列同 col range
是否被重複註冊」。對於診斷汙染問題極有幫助。

H. 把陣列模式參數做型別安全

目前 xString 可以是字串或陣列，靠 VALTYPE 分流。未來可加
IsHintArray() 驗證陣列每個元素都有 ":" 分隔的 label 格式，對缺少
KY_* prefix 的元素提早給出 debug log，避免靜默失敗。

結語

整套提示系統的設計核心是「畫 + 解析 +
註冊綁在一起」與「三明治隔離」。前者讓座標計算永遠跟著 BIG5 全形字
寬走、不用呼叫者自己算；後者讓任意進入點都能與上層完全解耦。AddHint
Label 是這個體系演化到第二階段的代表作 —
把「最常用的單一熱區註冊」從 5 行壓縮到 1
行，這正是「封裝共用邏輯」原則的最佳示範。