**.nolist**; creates a 7k editor that has most if not all features needed for building assembler apps

```
include \masm32\include\masm32rt.inc

WinMain         PROTO:DWORD,:DWORD,:DWORD,:DWORD
SetSelFormat    PROTO:DWORD,:DWORD
WritePlain      PROTO:DWORD,:DWORD
MyFiles         PROTO:DWORD
MyFileOpen      PROTO:DWORD

AppX            = 30
AppY            = 40
AppWidth        = 700
AppHeight       = 660
ButMargin       = 3
ButWidth        = 54
ButHeight       = 24

m2m MACRO M1, M2
  pushd M2
  pop dword ptr M1
ENDM

movi MACRO M1, M2
LOCAL oa, num
  num = M2          ;; get rid of sizeof xxx syntax
  oa = (opattr num) and 127
  if oa ne 36
        echo <M2> is not an immediate
        .err
  endif
  if num le 127
        pushd M2
        pop dword ptr M1
  else
        mov M1, M2
  endif
ENDM

crtbuf MACRO var, BufLen
LOCAL lbl
.data?
align 4
  lbl LABEL byte
  ORG $+BufLen-1
  db ?
  .data
        var dd lbl              ;; define it in the data section
  .code
ENDM

SizeNormal      = 240       ; in twips
SizeLarge       = 280
SizeXXL         = 360
Font_RED        = 0000FFh
Font_BLUE       = 0FF0000h
BgYellow        = 0AAFFFFh
SelXxlSize      = 4096000

sm              equ invoke SendMessage, ; for faster typing ;-)
signed          equ sdword ptr
IdButton0       = 100       ; startpoint of buttons
IdMenuHelp      = 100
IdMenuOpen      = 101       ; ID menu item New
IdMenuSave      = 102       ; ID menu item Save
IdMenuSaveAs    = 103       ; ID menu item Save
IdMenuAssRun    = 104
IdMenuBold      = 105
IdMenuLarge     = 106
IdMenuEmRed     = 107
IdMenuBlue      = 108
```

```
IdMenuHilite        = 109
IdMenuFind          = 110
IdMenuReplace       = 111
idPrint             = 112
IdMenuFindNext      = 113
IdEscape            = 114       ; last item
IdEdit              = 120
IdStatic            = 121
IdButtonMax         = 14        ; number of buttons+1; see ButtonTable, ButtonJumps

.data
MyAccels            ACCEL <FCONTROL or FVIRTKEY, VK_O, IdMenuOpen>
                    ACCEL <FCONTROL or FVIRTKEY, VK_S, IdMenuSave>
                    ACCEL <FVIRTKEY, VK_F1, IdMenuHelp>
                    ACCEL <FVIRTKEY, VK_F3, IdMenuFindNext>
                    ACCEL <FVIRTKEY or FSHIFT, VK_F3, IdMenuFindNext>
                    ACCEL <FCONTROL or FVIRTKEY, VK_F, IdMenuFind>
                    ACCEL <FCONTROL or FVIRTKEY, VK_R, IdMenuReplace>
                    ACCEL <FVIRTKEY, VK_F6, IdMenuAssRun>
                    ACCEL <FCONTROL or FVIRTKEY, VK_B, IdMenuBold>
                    ACCEL <FCONTROL or FVIRTKEY, VK_L, IdMenuLarge>
                    ACCEL <FCONTROL or FVIRTKEY or FSHIFT, VK_L, IdMenuLarge>
                    ACCEL <FCONTROL or FVIRTKEY, VK_E, IdMenuEmRed>
                    ACCEL <FCONTROL or FVIRTKEY or FSHIFT, VK_B, IdMenuBlue>
                    ACCEL <FCONTROL or FVIRTKEY, VK_H, IdMenuHilite>
                    ACCEL <FCONTROL or FVIRTKEY, VK_P, idPrint>
LastAccel           ACCEL <FVIRTKEY, VK_ESCAPE, IdEscape>       ; quit
ButtonTable         dd txtButton0, txtButton1, txtButton2, txtButton3, txtButton4, txtButton5, txtButton6
                    dd txtButton7, txtButton8, txtButton9, txtButton10, txtButton11, txtButton12, 0
ButtonJumps         dd proButton0, proButton1, proButton2, proButton3, AssRun, proButton5, proButton6
                    dd proButton7, proButton8, proButton9, FindOnly, FindReplace, PrintRTF, FindNext,
proButtonEsc, 0
txtButton0          db "Help", 0
txtButton1          db "Open", 0
txtButton2          db "Save", 0
txtButton3          db "Save as", 0
txtButton4          db "Build", 0
txtButton5          db "Bold", 0
txtButton6          db "Large", 0
txtButton7          db "Red", 0
txtButton8          db "Blue", 0
txtButton9          db "Hilite", 0
txtButton10         db "Find", 0
txtButton11         db "Replace", 0
txtButton12         db "Print", 0

ClassAppWin         db "MyClass", 0
ClassButton         db "button", 0
ClassEdit           db "edit", 0
ClassStatic         db "static", 0
txStatic            db 0        ; we are stingy, no "Welcome" here
MyFont              db "Arial", 0
AppName             db "Tiny RTF Editor: ", 0
txFilter            db "Rich Text files", 0, "*.rtf", 0
                    db "Plain Asm", 0, "*.asm", 0, 0
txBatch             db "TinyTmp.bat ", 0
txBatchDef          db "TinyDef.bat ", 0

.data?
StreamMode          dd ?
IsHelp              dd ?
ChkEsp              dd ?
SaveAs              dd ?
ButTop              dd ?
lpWritten           dd ?
hFR                 dd ?
frs                 FINDREPLACE <>
WM_FindReplace      dd ?        ; FindR
BufFind             dd 50 dup (?)
BufRepl             dd 50 dup (?)
hInstance           HINSTANCE ?
hWin                HWND ?
```

```
hStatic         HWND ?
hEdit           HWND ?
hMenu           HANDLE ?
hLib            HANDLE ?
hM1             HANDLE ?
hM2             HANDLE ?
hButFnt         HANDLE ?
hAccT           HANDLE ?
ComLineBuffer   db MAX_PATH dup (?)
WinTitleBuffer  db MAX_PATH+40 dup (?)

.code
start:
    crtbuf  SelXXL$,    SelXxlSize  ; fat buffer for GetCurSel & Sel (409600)
    invoke GetModuleHandle, NULL
    mov hInstance, eax
    invoke GetCL, 1, addr ComLineBuffer
    .if eax!=1
        invoke lstrcpy, addr ComLineBuffer, chr$("TinyDemo.rtf")
    .endif
    invoke WinMain, hInstance, 0, 0, SW_SHOWDEFAULT

    ; MsgBox 0, "Bye", offset AppName, MB_OK        ; uncomment to test if your code exits properly
    invoke ExitProcess, eax

WinMain proc hInst:HINSTANCE, hPrevInst:HINSTANCE, CmdLine:LPSTR, CmdShow:DWORD
LOCAL wc:WNDCLASSEX
LOCAL msg:MSG
    call ClearLocVars                   ; set all local variables to zero
    movi wc.cbSize, SIZEOF WNDCLASSEX
    movi wc.style, CS_HREDRAW or CS_VREDRAW
    mov wc.lpfnWndProc, offset WndProc  ; m2m longer
;   mov wc.cbClsExtra, NULL             ; already zeroed
;   mov wc.cbWndExtra, NULL             ; by ClearLocVars
    m2m wc.hInstance, hInst
    movi wc.hbrBackground, COLOR_WINDOW                 ; ->GetSysColor
    m2m wc.lpszClassName, offset ClassAppWin
    mov wc.hIcon, rv(LoadIcon, hInst, IDI_APPLICATION)
    mov wc.hIconSm, eax                                 ; reuse eax as returned by rv
    mov wc.hCursor, rv(LoadCursor, NULL, IDC_ARROW)
    invoke RegisterClassEx, addr wc
    ; call InitCommonControls not needed
    movi ecx, AppX              ; lt 128, m2m shorter
    movi edx, AppWidth          ; ge 128, mov shorter
    mov eax, dword ptr ComLineBuffer
    .if eax=="yniT"             ; Tiny
        mov eax, dword ptr [ComLineBuffer+4]
        .if eax=="pleH"         ; Help
            lea ecx, [ecx+8*ecx+127]
            sub edx, 127
            inc IsHelp
        .endif
    .endif
    invoke CreateWindowEx, 0,
        addr ClassAppWin, 0,
        WS_OVERLAPPEDWINDOW or WS_CLIPCHILDREN or WS_VISIBLE,
        ecx, AppY, edx, AppHeight,
        NULL, NULL, hInst, NULL             ; sets hWin in WM_CREATE

    .Repeat
        invoke GetMessage, addr msg, 0, 0, 0
        .break .if eax==0
        invoke IsDialogMessage, hFR, addr msg
        .if eax==0
            invoke TranslateAccelerator, hWin, hAccT, addr msg
            .if eax==0
                invoke TranslateMessage, addr msg
                invoke DispatchMessage, addr msg
            .endif
        .endif
    .Until 0
```

```
        mov eax, msg.wParam     ; m2m longer
        ret
WinMain endp

WndProc proc hWnd:HWND, uMsg:UINT, wParam:WPARAM, lParam:LPARAM
LOCAL rc:RECT
        SWITCH uMsg
        CASE WM_CREATE
            ; ------------------- create keyboard shortcuts ----------------------------
            m2m hWin, hWnd      ; a global copy for MsgBox etc
            movi frs.lStructSize, sizeof FINDREPLACE
            m2m frs.hwndOwner, hWin
            movi frs.Flags, FR_DOWN             ; enter once, let user keep settings
            m2m frs.lpstrReplaceWith, offset BufRepl
            m2m frs.lpstrFindWhat, offset BufFind
            ; we need a dirty trick against Masm 9.0: bye bye to invoke, push by hand...!
            push dword ptr ((LastAccel-MyAccels)/(SIZEOF ACCEL)+1)
            push offset MyAccels
            call CreateAcceleratorTable
            mov hAccT, eax
            if 0

            ; ------------------- create menus and sub-menus --------------------
            mov hMenu, rv(CreateMenu)                   ; create the main menu
            mov hM1, rv(CreateMenu)                     ; plus two
            invoke AppendMenu,hMenu, MF_POPUP, hM1,chr$("&File")
            invoke AppendMenu, hM1, MF_STRING, IdMenuOpen, chr$("&Open",9,"Ctrl+O")
            invoke AppendMenu, hM1, MF_STRING, IdMenuSaveAs, chr$("&Save as",9,"Ctrl+S")
            if menu2
              mov hM2, rv(CreateMenu)                   ; sub menus
              invoke AppendMenu,hMenu, MF_POPUP, hM2,chr$("&Edit")
              invoke AppendMenu, hM2, MF_STRING, IdMenuCopy, chr$("&Copy",9,"Ctrl+C")
            endif
            invoke SetMenu, hWnd, hMenu                 ; attach menu to main window
            endif

            ; ------------------- create controls: buttons, edit window, static ----------

            invoke GetStockObject, ANSI_VAR_FONT    ; get a cute little font for the controls
            mov hButFnt, eax
            ; invoke LoadLibrary, chr$("MSFTEDIT.dll")  ; requires Win XP
            invoke LoadLibrary, chr$("RichEd20.dll")
            mov hLib, eax
            MyStyle = WS_EX_CLIENTEDGE or WS_EX_TRANSPARENT         ; Test this one
            MyStyle = WS_EX_CLIENTEDGE
            invoke CreateWindowEx, MyStyle, chr$("RichEdit20A"), NULL,      ; 50W slightly better...
              WS_CHILD or WS_VISIBLE or WS_BORDER\
              or ES_LEFT or ES_AUTOVSCROLL or ES_AUTOHSCROLL or ES_MULTILINE or WS_VSCROLL, 0,
0, 0, 0,
              hWnd, IdEdit, hInstance, NULL
            mov hEdit, eax                              ; we have created a RichEdit control
            sm hEdit, EM_EXLIMITTEXT, 0, -1  ; text limit (default 64K
            sm hEdit, EM_SETMARGINS, EC_LEFTMARGIN or EC_RIGHTMARGIN, 4
            .if IsHelp==0
                call CreateButtons
            .endif
            invoke RegisterWindowMessage, chr$("commdlg_FindReplace")
            mov WM_FindReplace, eax
            sm hEdit, EM_SETBKGNDCOLOR, 0, 0F8FFEEh     ; BGR
            invoke MyFiles, offset ComLineBuffer

        CASE WM_COMMAND                         ; react here to menus and controls
            mov ecx, wParam                     ; notification code in hiword of wParam
            shr ecx, 16
            movzx eax, word ptr wParam          ; the Ids are in the loword of wParam
            sub eax, IdButton0
;           .if ecx==BN_CLICKED
;               int 3
;           .endif
```

```
            .if eax>=0 && eax<=IdButtonMax && (ecx==BN_CLICKED || ecx==1)        ; ecx=1: Accel
                ; mov ChkEsp, esp
                call [ButtonJumps+4*eax]
;               mov eax, ChkEsp
;               .if esp!=ChkEsp
;                   MsgBox 0, "Stack corruption!", "Hi", MB_OK
;               .endif
            .endif

        CASE WM_FindReplace
            mov eax, lParam
            call DoFR

        CASE WM_SIZE                ; resize controls
            invoke GetClientRect, hWnd, addr rc
            mov eax, rc.right
            sub eax, ButWidth+3*ButMargin   ; editbox = window width minus button width
            mov ecx, rc.bottom
            sub ecx, 2*ButMargin
            movi edx, ButWidth+2*ButMargin
            .if IsHelp
                sub edx, ButWidth+ButMargin
                add eax, ButWidth+1
            .endif
            invoke MoveWindow, hEdit, edx, ButMargin, eax, ecx, 1

        CASE WM_ACTIVATE
            movzx eax, word ptr wParam ; the fActive flag
            sub eax, WA_INACTIVE
            .if !Zero?
                invoke SetTimer, hWnd, 4444, 100, 0
            .endif

        CASE WM_TIMER
            invoke KillTimer, hWnd, 4444
            invoke SetFocus, hEdit

         CASE WM_CLOSE
            call ChkMods
            .if eax
                xor eax, eax
                ret             ; return 0
            .endif
        ;   no difference in behaviour
        ;       invoke DestroyWindow, hWnd          ; will destroy all its children, too
        ;       invoke FreeLibrary, hLib                    ; get rid of RichEd20.dll       See here
        ;       invoke DestroyAcceleratorTable, hAccT       ; and the accelerators

        ; CASE WM_QUIT
        ;   MsgBox 0, "Quit", "Hi", MB_OK                   ; never seen

        CASE WM_DESTROY
            ; MsgBox 0, "Destroy", "Hi", MB_OK
            ; no good here:
            ; invoke FreeLibrary, hLib
            ; invoke DestroyAcceleratorTable, hAccT
            invoke PostQuitMessage, NULL
            ; 77D194A8         CD 2B               int 2B      ; unload msftedit.dll - stops here if
no pq mess sent
            ; 77D1861F         FF15 1C13D177       call near dword ptr
[<&KERNEL32.InterlockedIncrement>]
            ; 77D18854         FF15 3414D177       call near dword ptr
[<&ntdll.RtlDeactivateActivationContextUnsafeFast>]
            ; return 0
        ENDSW
        invoke DefWindowProc, hWnd, uMsg, wParam, lParam
        ret
WndProc endp

OPTION PROLOGUE:NONE
OPTION EPILOGUE:NONE
```

```
CreateButtons proc uses edi esi ebx
  mov esi, offset ButtonTable       ; with offset, mov is shorter than m2m
  movi ebx, IdButton0
  movi ButTop, ButMargin+1          ; left margin = top margin (m2m 8 bytes, mov 10)
  .Repeat
        lodsd
        .break .if eax==0           ; or eax, eax - very destructive but efficient
        invoke CreateWindowEx, NULL, offset ClassButton, eax,       ; eax=address txButtonN
          WS_CHILD or WS_VISIBLE or BS_PUSHBUTTON, ButMargin, ButTop, ButWidth, ButHeight,
          hWin, ebx, hInstance, NULL
        sm eax, WM_SETFONT, hButFnt, 1; give it the small font
        add ButTop, ButHeight+ButMargin
        inc ebx
  .Until 0
  invoke CreateWindowEx, NULL, addr ClassStatic, addr txStatic,
    WS_CHILD or WS_VISIBLE or ES_LEFT, ButMargin, ButTop, ButWidth, ButHeight*9,
    hWin, IdStatic, hInstance, NULL
  mov hStatic, eax                  ; static window under buttons created
  sm eax, WM_SETFONT, hButFnt, 1  ; give it the small font
  ret
CreateButtons endp

ChkMods proc
  sm hEdit, EM_GETMODIFY, 0, 0
  .if eax
        MsgBox hWin, chr$("You made changes.", 13, "Save now?"), addr AppName, MB_YESNOCANCEL
        .if eax==IDYES
              invoke MyFiles, 0     ; save current
              xor eax, eax      ; flag exit
        .else
              sub eax, IDNO         ; ret zero for IDNO
        .endif
  .endif
  ret
ChkMods endp

proButton0 proc ; Help
  .if IsHelp
        jmp proButtonEsc
  .endif
  invoke WinExec, chr$("TinyRTF.exe TinyHelp.rtf"), SW_RESTORE
  ret
proButton0 endp

proButtonEsc proc
  sm hWin, WM_CLOSE, 0, 0     ; Escape: exit without asking (unless your text was modified)
  ret
proButtonEsc endp

proButton1 proc ; Open
  call ChkMods
  .if eax==0
        invoke MyFiles, 1
  .endif
  ret
proButton1 endp

proButton2 proc ; Save
  invoke MyFiles, 0
  ret
proButton2 endp

proButton3 proc ; Save as
  or SaveAs, -1 ; flag
  invoke MyFiles, 0
  ret
proButton3 endp

proButton5 proc ; Bold
  invoke SetSelFormat, CFE_BOLD, CFM_BOLD; effect, mask
  ret
proButton5 endp
```

```
proButton6 proc ; Large
  invoke GetKeyState, VK_SHIFT        ; GetKeyState returns a WORD (?)
  .if sword ptr ax>=0
        invoke SetSelFormat, SizeLarge, CFM_SIZE                ; effect, mask
 .else
        invoke SetSelFormat, SizeXXL, CFM_SIZE                  ; effect, mask
  .endif
  ret
proButton6 endp

proButton7 proc ; Red
  invoke SetSelFormat, Font_RED, CFM_COLOR          ; effect, mask
  ret
proButton7 endp

proButton8 proc ; Blue
  invoke SetSelFormat, Font_BLUE, CFM_COLOR         ; effect, mask
  ret
proButton8 endp

proButton9 proc ; Hilite
  invoke SetSelFormat, BgYellow, CFM_BACKCOLOR      ; effect, mask
  ret
proButton9 endp

Susy proc
  sm hEdit, EM_FINDTEXT, FR_DOWN or FR_MATCHCASE, edi
  .if signed eax>=0
        xor eax, eax
  .endif
  ret
Susy endp
```

## FindNext:

```
        or eax, -1
        jmp FindReplace
```

## FindOnly:

```
        xor eax, eax      ; flag it's find only, then fall through

FindReplace proc                ; Replace
LOCAL repl:DWORD, txrg:TEXTRANGE
  mov repl, eax
  if 0                    ; not here
  movi frs.lStructSize, sizeof FINDREPLACE
  m2m frs.hwndOwner, hWin
  movi frs.Flags, FR_DOWN                           ; enter once, let user keep settings
  m2m frs.lpstrReplaceWith,   offset BufRepl
  m2m frs.lpstrFindWhat, offset BufFind
  endif
  mov txrg.lpstrText, offset BufFind
  sm hEdit, EM_EXGETSEL, 0, addr txrg
  mov eax, txrg.chrg.cpMax
  sub eax, txrg.chrg.cpMin
  .if eax<20
        sm hEdit, EM_GETTEXTRANGE, 0, addr txrg             ; preload with current selection
        .if repl && repl!=-1
              mov eax, txrg.chrg.cpMin
              dec eax
              .if !Sign?
                    mov txrg.chrg.cpMin, eax
                    mov txrg.chrg.cpMax, eax
                    sm hEdit, EM_EXSETSEL, 0, addr txrg
              .endif
        .endif
  .endif
  movi frs.wFindWhatLen, sizeof BufFind
  movi frs.wReplaceWithLen,   sizeof BufRepl
  push offset frs                 ; one longword
  cmp repl, -1
  je DoFR_F3                      ; fall through
  .if repl
        call ReplaceText          ; expects one dword on stack
```

```
  .else
        call FindText               ; expects one dword on stack
  .endif
    mov hFR, eax                    ; save the handle
  ret
FindReplace endp
```

## DoFR_F3:

```
  movi frs.Flags, FR_FINDNEXT
  invoke GetKeyState, VK_SHIFT      ; GetKeyState returns a WORD (?)
  .if sword ptr ax>=0
        or frs.Flags, FR_DOWN
  .endif
  pop eax                           ; correct stack and pass pointer to frs

OPTION PROLOGUE:PrologueDef
OPTION EPILOGUE:EpilogueDef
DoFR proc
LOCAL ft:FINDTEXTEX
;   CHARRANGE chrg;      // range to search
;   LPSTR lpstrText;     // null-terminated string to find
;   CHARRANGE chrgText;  // range in which text is found
  push ebx
  push esi
  mov ebx, [eax.FINDREPLACE.Flags]
  .if ebx & FR_DIALOGTERM
        and hFR, 0                                  ; flag handle no longer valid
        and [eax.FINDREPLACE.Flags], 0 ; and this flag must be reset by hand!

  .elseif ebx & (FR_FINDNEXT or FR_REPLACE or FR_REPLACEALL)
        m2m ft.lpstrText, [eax.FINDREPLACE.lpstrFindWhat]
        sm hEdit, EM_EXGETSEL, 0, addr ft.chrg
        or esi, -1        ; default: from current sel to end of doc
        .if ebx & (FR_REPLACE or FR_REPLACEALL)
              mov eax, ft.chrg.cpMax
              sub eax, ft.chrg.cpMin
              .if eax>20
                    mov esi, ft.chrg.cpMax
              .endif
        .endif
        .Repeat
              inc ft.chrg.cpMin
              m2m ft.chrg.cpMax, esi
              sm hEdit, EM_FINDTEXTEX, ebx, addr ft
              .break .if signed eax<0
              sm hEdit,  EM_EXSETSEL, 0, addr ft.chrgText
              .if ebx & (FR_REPLACE or FR_REPLACEALL)
                    sm hEdit,  EM_REPLACESEL, 1, offset BufRepl
              .endif
              invoke SetActiveWindow, hWin          ; needed to show the selection
              sm hEdit,  EM_HIDESELECTION, 0, 0
        .Until !(ebx & FR_REPLACEALL)
  .endif
  pop esi
  pop ebx
  ret
DoFR endp
```

## MyFiles proc IsRead:DWORD

```
LOCAL ofn:OPENFILENAME
LOCAL hFile:DWORD, Read4:DWORD, dwBytesRead:DWORD
LOCAL editstream:EDITSTREAM
LOCAL LocBuf[MAX_PATH]:BYTE
  call ClearLocVars
  push esi
  mov esi, offset ComLineBuffer
  cmp IsRead, esi
  .if Zero?
        dec eax
  .else
        lea esi, LocBuf
        movi ofn.lStructSize, sizeof OPENFILENAME
        m2m ofn.hWndOwner, hWin
```

```
        m2m ofn.hInstance, hInstance
        mov ofn.lpstrFilter, offset txFilter
        movi ofn.nMaxFile, MAX_PATH
        mov ofn.lpstrDefExt, chr$("rtf")            ; an offset, mov is shorter
        ; set the current folder:
        invoke GetModuleFileName, 0, esi, MAX_PATH ; returns # of chars

@@:     dec eax
        ; jns @F ; GMF returns D:\masm32\RichMasm\TinyRTF.exe
        cmp byte ptr [esi+eax],"\"
        jne @B
        mov byte ptr [esi+eax], ah    ; ah is zero

@@:     mov ofn.lpstrInitialDir, esi
        mov esi, offset ComLineBuffer
        mov ofn.lpstrFile, esi    ; mov shorter than m2m
        .if IsRead
            mov ofn.lpstrTitle, chr$("Open:")
            movi ofn.Flags, OFN_EXPLORER or OFN_LONGNAMES or OFN_PATHMUSTEXIST        ; m2m
longer
            invoke GetOpenFileName, addr ofn
        .elseif SaveAs
            invoke lstrcpy, esi, chr$("MyFile.rtf")
            mov ofn.lpstrTitle, chr$("Save my text:")
            movi ofn.Flags, OFN_EXPLORER or OFN_LONGNAMES or OFN_OVERWRITEPROMPT
            invoke GetSaveFileName, addr ofn
        .endif
        and SaveAs, 0
    .endif
    .if eax
        movi ecx, GENERIC_WRITE
        movi edx, CREATE_ALWAYS
        .if IsRead
            movi ecx, GENERIC_READ
            movi edx, OPEN_EXISTING
        .endif
        invoke MyFileOpen, esi
        ; invoke CreateFile, esi, ecx, FILE_SHARE_READ, 0, edx, FILE_ATTRIBUTE_NORMAL, 0
        .if eax!=INVALID_HANDLE_VALUE
            mov hFile, eax
            mov editstream.dwCookie, eax
            push edi
            mov edi, offset WinTitleBuffer
            invoke lstrcpy, edi, offset AppName
            invoke lstrcat, edi, esi
            sm hWin, WM_SETTEXT, 0, edi
            pop edi
            movi ecx, EM_STREAMOUT
            and StreamMode, 0
            movi esi, SF_RTF
            .if IsRead
                inc StreamMode
                invoke SetFocus, hEdit                    ; give the focus to the edit window
                sm hEdit, EM_SETTARGETDEVICE, 0, 0
                invoke ReadFile, hFile, addr Read4, 4, ADDR dwBytesRead, 0
                invoke SetFilePointer, hFile, 0, 0, FILE_BEGIN
                mov eax, Read4
                or eax, 20200000h          ; force lowercase for tr
                .if eax!="tr\{"
                    mov esi, SF_TEXT      ; rtf if you find the magic string
                .endif
                movi ecx, EM_STREAMIN
            .endif
            mov editstream.pfnCallback, StreamRTF
            sm hEdit, ecx, esi, addr editstream
            invoke CloseHandle, hFile
            sm hEdit, EM_SETMODIFY, 0, 0
        .endif
    .endif
    pop esi
    ret
MyFiles endp
```

```
StreamRTF proc hFile:DWORD,pBuffer:DWORD, NumBytes:DWORD, pBytes:DWORD
  push 0
  push pBytes
  push NumBytes
  push pBuffer
  push hFile
  .if StreamMode
        call ReadFile
  .else
        call WriteFile
  .endif
  xor eax, 1
  ret
StreamRTF endp

SetSelFormat proc CharFt:DWORD, CharFtMask:DWORD
  LOCAL charfmt:CHARFORMAT2
  call ClearLocVars
        movi charfmt.cbSize, sizeof CHARFORMAT2
        mrm charfmt.dwMask, CharFtMask
        mrm charfmt.crTextColor, CharFt
        sm hEdit, EM_GETCHARFORMAT, SCF_SELECTION, ADDR charfmt
        mov eax, CharFt

        .if CharFtMask==CFM_COLOR
          .if charfmt.crTextColor==eax
              invoke GetSysColor, COLOR_WINDOWTEXT
          .endif
          mov charfmt.crTextColor, eax
          movi eax, CFE_AUTOCOLOR
          or charfmt.dwEffects, eax ; get rid of CFE_AUTOCOLOR
          xor charfmt.dwEffects, eax

        .elseif CharFtMask==CFM_BACKCOLOR
          .if charfmt.crBackColor==eax
              invoke GetSysColor, COLOR_WINDOW
          .endif
          mov charfmt.crBackColor, eax
          movi eax, CFE_AUTOBACKCOLOR
          or charfmt.dwEffects, eax ; get rid of CFE_AUTOBACKCOLOR
          xor charfmt.dwEffects, eax

        .elseif CharFtMask==CFM_SIZE
          or CharFtMask, CFM_FACE
          mov ecx, offset MyFont    ; sizing not possible with SysFont
          .if charfmt.yHeight==eax
              mov eax, SizeNormal
              ; mov ecx, offset SysFont
          .endif
          mov charfmt.yHeight, eax
          invoke lstrcpy, addr charfmt.szFaceName, ecx    ; max 32 chars
        .else
          xor charfmt.dwEffects, eax      ; sending twice toggles... in theory!
        .endif

        sm hEdit, EM_SETCHARFORMAT, SCF_SELECTION, ADDR charfmt
  ret
SetSelFormat endp

AssRun proc
LOCAL ft:FINDTEXT, IsCon:SDWORD
LOCAL txrg:TEXTRANGE, txrgASM:TEXTRANGE
LOCAL LocBuf[1600]:BYTE
call ClearLocVars
  push edi
  push ebx
        or ebx, -1
        mov ft.chrg.cpMax, ebx          ; -1, end of doc
        and ft.chrg.cpMin, 0            ; 0, start of doc
        lea edi, ft
        mov ft.lpstrText, chr$("pri","nt")
        call Susy
```

```
        add eax, eax
        add IsCon, eax
        mov ft.lpstrText, chr$("SendMes")
        call Susy
        sub IsCon, eax
        mov ft.lpstrText, chr$("WM_")
        call Susy
        sub IsCon, eax
  lea edi, LocBuf
  mrm ft.lpstrText, chr$("BATCH", "$")
  .Repeat
        sm hEdit, EM_FINDTEXT, FR_DOWN or FR_MATCHCASE, addr ft                    ; no FR_DOWN = FR_UP
        .if ebx     ; ==1
            dec eax
            mov txrgASM.chrg.cpMax, eax
            add eax, 8
            mov txrg.chrg.cpMin, eax
            mov ft.chrg.cpMin, eax
        .endif
        inc ebx
  .Until !Zero?
  mov ebx, offset txBatchDef        ; default □s TinyDef.bat if no batch$ pair found
  .if signed eax>0
        mov ebx, offset txBatch
        mov txrg.chrg.cpMax, eax
        mov txrg.lpstrText, edi
        invoke WritePlain, offset txBatch, addr txrg
  .endif
  mrm txrgASM.lpstrText, SelXXL$                                    ; mrm 8, m2m 9 bytes
  invoke WritePlain, chr$("TinyTmp.asm"), addr txrgASM ; SelXXL$
  invoke ShowWindow, hWin, SW_HIDE
  invoke GetModuleFileName, 0, edi, 260    ; eax returns # of bytes copied not incl. null
  .While signed eax>=0
        mov cl, [edi+eax]
        .break .if cl=="\"
        dec eax
  .Endw
  mov byte ptr [edi+eax+1], 0        ; path to exe delimited with zero byte
  invoke lstrcat, edi, ebx
  invoke lstrcat, edi, offset ComLineBuffer
  .if IsCon>=0
        invoke lstrcat, edi, chr$(" SuSyCo")
  .endif
  invoke WinExec, edi, SW_MAXIMIZE
  .if eax<=32
        MsgBox 0, edi, chr$("Could not launch:"), MB_OK
  .endif
  invoke ShowWindow, hWin, SW_SHOWNA
  m2m txrg.chrg.cpMax, txrg.chrg.cpMin
  invoke SendMessage, hEdit, EM_EXSETSEL, 0, addr txrg
  pop ebx
  pop edi
  ret
AssRun endp

WritePlain proc fname, sel     ; EM_STREAMOUT
LOCAL hFile
LOCAL editstream:EDITSTREAM
        movi ecx, GENERIC_WRITE
        movi edx, CREATE_ALWAYS
        invoke MyFileOpen, fname
        .if eax!=INVALID_HANDLE_VALUE
            mov hFile, eax
            mov editstream.dwCookie, eax
            and StreamMode, 0
            mov editstream.pfnCallback, StreamRTF
            sm hEdit, EM_EXSETSEL, 0, sel
            sm hEdit, EM_STREAMOUT, SF_TEXT or SFF_SELECTION, addr editstream
            invoke CloseHandle, hFile
        .endif
        ret
WritePlain endp
```

```
MyFileOpen proc fname
  invoke CreateFile, fname, ecx, FILE_SHARE_READ, 0, edx, FILE_ATTRIBUTE_NORMAL, 0
  .if eax==INVALID_HANDLE_VALUE
        invoke MessageBox, hWin, fname, chr$("Could not open this file:"), MB_OK
  .endif
  ret
MyFileOpen endp

twips MACRO arg
  push arg
  call twipsP
  EXITM <eax>
ENDM
;       567 twips per cm: A4 = 21 X 29,7 cm
;       twips = 21*567 X 29.7X567 = 11907 * 16839,9

PrintRTF proc
LOCAL fSuccess:SDWORD
LOCAL hPrDC:DWORD
LOCAL OldSel:CHARRANGE
LOCAL psd:PAGESETUPDLG
LOCAL docInfo:DOCINFO
LOCAL fr:FORMATRANGE
  call ClearLocVars                                     ; clear all structure elements
  push edi
  push esi
  mov psd.lStructSize, sizeof PAGESETUPDLG
.data?
  prtMargins          dd 4 dup(?)
.code
  mov esi, offset prtMargins
  lea edi, psd.rtMargin
  m2m ecx, 4
  push ecx
  push esi
  push edi
  .if dword ptr [esi+8]             ; take the right margin as flag
                      mov psd.Flags, PSD_INHUNDREDTHSOFMILLIMETERS or PSD_MARGINS
                      rep movsd
  .else
                      mov psd.Flags, PSD_INHUNDREDTHSOFMILLIMETERS or PSD_DEFAULTMINMARGINS
  .endif
  invoke PageSetupDlg, addr psd                                 ; get a printer device context
  pop esi             ; the unchanged order is
  pop edi             ; intentional: we swap the pointers
  pop ecx
  .if eax==0
        invoke CommDlgExtendedError
        test eax, eax
        jne PriError
  .else
        rep movsd
        mov esi, rv(GlobalLock, psd.hDevNames)
        push rv(GlobalLock, psd.hDevMode)
        mov edx, esi
        movzx eax, word ptr [esi.DEVNAMES.wOutputOffset]
        add edx, eax
        push edx
        mov edx, esi
        movzx eax, word ptr [esi.DEVNAMES.wDeviceOffset]
        add edx, eax
        push edx
        mov edx, esi
        movzx eax, word ptr [esi.DEVNAMES.wDriverOffset]
        add edx, eax
        push edx
        call CreateDC     ;  hPrDC=CreateDC(lpszDriver, lpszDevice, lpszOutput, pDeviceMode)
        mov hPrDC, eax
        push eax
        invoke GlobalUnlock, psd.hDevNames
        invoke GlobalUnlock, psd.hDevMode
```

```
        pop eax
        mov docInfo.cbSize, sizeof DOCINFO
        mov docInfo.lpszDocName, chr$("TinyRtf")
        invoke StartDoc, hPrDC, addr docInfo            ; start a print job
        .if eax==SP_ERROR
              invoke DeleteDC, hPrDC
              jmp PriError
        .endif
;       invoke SendMessage, hEdit, EM_SETTARGETDEVICE, hPrDC, cxPhys          ; not needed
        m2m fr.hdc, hPrDC
        m2m fr.hdcTarget, hPrDC

;       mov cxPhys, rv(GetDeviceCaps, hPrDC, PHYSICALWIDTH) yields 4958, a factor 2.x too small
;       mov cyPhys, rv(GetDeviceCaps, hPrDC, PHYSICALHEIGHT) yields 7017 - expected 16840 for A4

        mov fr.rc.left, twips(psd.rtMargin.left)   ; psd:PAGESETUPDLG
        neg eax
        mov fr.rc.right, eax
        add fr.rc.right, twips(psd.ptPaperSize.x)
        sub fr.rc.right, twips(psd.rtMargin.right)

        mov fr.rc.top, twips(psd.rtMargin.top)
        neg eax
        mov fr.rc.bottom, eax
        add fr.rc.bottom, twips(psd.ptPaperSize.y)
        sub fr.rc.bottom, twips(psd.rtMargin.bottom)

;       Get the current selection into a CHARRANGE
        invoke SendMessage, hEdit, EM_EXGETSEL, 0, addr fr.chrg
        mov eax, fr.chrg.cpMax
        mov edx, fr.chrg.cpMin
        mov OldSel.cpMax, eax
        mov OldSel.cpMin, edx
        sub eax, edx
        .if sdword ptr eax<=127         ; User has not selected a lot of text, therefore print all pages
              invoke SendMessage, hEdit, EM_SETSEL, 0, -1
              invoke SendMessage, hEdit, EM_EXGETSEL, 0, addr fr.chrg
        .endif

;       Use GDI to print successive pages
        .Repeat
              invoke StartPage, hPrDC
              mov fSuccess, eax
              .Break .if sdword ptr eax<=0
              push fr.rc.bottom
              invoke SendMessage, hEdit, EM_FORMATRANGE, 1, addr fr
              pop fr.rc.bottom
;              The rc.bottom member may be changed after the message is sent. If it is changed, it must indicate
the
;              largest rectangle that can fit within the bounds of the original rectangle and still contain the
specified
;              text without printing partial lines. It may be necessary to reset this value after each page is printed.
;              These dimensions are given in TWIPS.  (MS Support)
;             deb 1, "Loop", eax, fr.chrg.cpMin, fr.chrg.cpMax
              .Break .if eax<=fr.chrg.cpMin
              .Break .if eax>=fr.chrg.cpMax
              mov fr.chrg.cpMin, eax
              invoke EndPage, hPrDC
              mov fSuccess, eax
        .Until sdword ptr eax<=0
        invoke SendMessage, hEdit, EM_FORMATRANGE, 0, 0          ; free the cache, important
        .if fSuccess>0
              invoke EndDoc, hPrDC
        .else
              invoke AbortDoc, hPrDC
        .endif
        invoke DeleteDC, hPrDC
        invoke SendMessage, hEdit, EM_EXSETSEL, 0, addr OldSel              ; restore old selection
  .endif
;  mov eax, fSuccess
@@:
  pop esi
  pop edi
```

```
  ret
PriError:
  MsgBox 0, "Printing problem", 0, MB_OK
  jmp @B
PrintRTF endp


twipsP proc
.data
  tw2cm    REAL4 0.567
.code
  ffree st(7)
  ffree st(7)
  fld tw2cm
  fild dword ptr [esp+4]
  fmul
  fistp dword ptr [esp+4]
  pop edx
  pop eax
  jmp edx
twipsP endp

ClearLocVars proc       ; put "call ClearLocals" as first instruction after LOCALS - eax unchanged on exit
  push eax              ; do not use with uses esi etc - push them manually behind the call!
  lea eax, [esp+8]      ; pushed eax and ret address
  mov esp, ebp          ; base page of calling procedure
  align 4               ; 74 instead of 123 cycles on Celeron M, no effect on P4
@@:
  push 0                ; 120 bytes: 196 cycles on P4
  cmp esp, eax          ; rep stosd??
  ja @B
  sub esp, 8            ; 19 bytes with align 4
  pop eax
  ret
ClearLocVars endp

if 0
FillData    dd 1, 2, 3, 4, 5, 6, 7, 8, 9, 10    ; code size is 6144-4*48 = 5952 bytes
            dd 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
            dd 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
            dd 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
            dd 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
            dd 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
            dd 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
            dd 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
            dd 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
endif
end start

RichMasm options
OPT_Susy        Windows
OxPT_Icon       Smiley      ; see \Masm32\RichMasm\Icons
OxPT_Linker     link        ; default is polink (if found), otherwise link.exe
OPT_DebugL      /merge:.text=.data
OPT_Tmp2Asm     1
OPT_Arg1        \masm32\RichMasm\TinyDemo.rtf
OxPT_Arg1       \masm32\RichMasm\TinyRTF.asc
; delete the x if you prefer \masm32\bin\link.exe
```

## FreeLibrary

The reference count is decremented each time the FreeLibrary or FreeLibraryAndExitThread function is called for the module. When a module's reference count reaches zero **or the process terminates**, the system unloads the module from the address space of the process.