```
include  \masm64\include64\masm64rt.inc
include ids.inc
include IID_Interfaces.inc
include Structures.inc
include Constants.inc

extern  malloc:proc
extern  free:proc

.data?
hInstance       dq ?
hWnd            dq ?
hIcon           dq ?
hCursor         dq ?
hBrush          dq ?
g_pFramework    dq ?
g_pApplication  dq ?
screenWidth     dd ?
screenHeight    dd ?
windowWidth     dd ?
windowHeight    dd ?
topOffset       dd ?
leftOffset      dd ?
wc  WNDCLASSEX    <?>
msg MSG           <?>

.data
AppRibbon_Vtbl  dq  AppQueryInterface
                dq  AppAddRef
                dq  AppRelease
                dq  AppOnViewChanged
                dq  AppOnCreateUICommand
                dq  AppOnDestroyUICommand
szApllRibbon    dw "T","E","M","P","L","A","T","E","6","4","_","R","I","B","B","O","N",0
classname       db "template64_class",0
caption         db "Template64",0

.code
;*****************************************************************;
; This is a standard Windows GUI initialization sequence, efficiently
; implemented in MASM64 assembly language.
; The code is well-commented, showing both the pure assembly and
; equivalent invoke-style pseudocode

entry_point             proc
```

```
                        xor     ecx, ecx
                        call    GetModuleHandleW
                        mov     hInstance,rax                   ; invoke GetModuleHandleW,0
; Initializes the COM library for use by the calling thread
                        xor     ecx, ecx                        ; 0
                        mov     edx, 2 or 4                     ; 4 = COINIT_DISABLE_OLE1DDE ;Disables DDE for OLE1 support.
                        call    CoInitializeEx                  ; 2 = COINIT_APARTMENTTHREADED
                        call    GetCommandLineW                 ; Return in rax the command-line string for the current
                          process.
                        call    WinMain                         ; invoke WinMain,hInstance,0,rax,SW_SHOWDEFAULT
                        call    msgloop                         ; start messages loop
;-----------------------------------------------------------------;
; Exit
                        call    CoUninitialize                  ; Closes the COM library on the current thread
                        xor     ecx, ecx
                        call    ExitProcess                     ; Ends the calling process and all its threads
                      ; ret
entry_point             endp
;-----------------------------------------------------------------;
; The code uses the MASM64 calling convention (parameters in RCX, RDX, R8, R9, rest on stack)
; It creates a centered window with light gray background
; The window will accept file drag-and-drop operations
; All resources (icon, cursor, brush) are properly loaded and stored
; The window will be immediately visible upon creation

WinMain                 proc

; Resource Loading
; Loads an icon resource with ID 200 from the application's resources
; Stores the handle in "hIcon" for later use in the window class

                        mov     edx, 200                        ; Icon ID from .rc  file
                        mov     rcx, hInstance
                        call    LoadIcon                        ; invoke LoadIcon,hInstance,200
                        mov     hIcon, rax

; Loads the standard arrow cursor (IDC_ARROW)
; Stores the handle in "hCursor"
                        mov     edx, IDC_ARROW
                        xor     ecx, ecx
                        call    LoadCursor                      ; invoke LoadCursor,0,IDC_ARROW
                        mov     hCursor, rax

; Creates a solid brush with light gray color (RGB 0xF0, 0xF0, 0xF0)
; Stores the brush handle in "hBrush"
```

```
 90                         mov     ecx, 0F0F0F0h                       ; Background color -> 0F0F0F0h
 91                         call    CreateSolidBrush                    ; invoke CreateSolidBrush,0F0F0F0h
 92                         mov     hBrush, rax
 93
 94 ; Window Class Registration (WNDCLASSEX)
 95 ; Initializes the WNDCLASSEX structure:
 96 ; Sets structure size
 97 ; Points to the window procedure ("WndProc")
 98 ; Sets class styles for byte-aligned windows
 99                         mov     wc.cbSize, SIZEOF WNDCLASSEX
100                         lea     rax, WndProc
101                         mov     wc.style,  CS_BYTEALIGNCLIENT or CS_BYTEALIGNWINDOW
102                         mov     wc.lpfnWndProc,rax
103                         mov     wc.cbClsExtra, 0
104                         mov     wc.cbWndExtra, 0
105                         mov     rax, hInstance
106                         mov     rcx, hIcon
107                         mov     r8,  hCursor
108                         mov     r9,  hBrush
109                         mov     wc.hInstance, rax
110                         mov     wc.hIcon,     rcx
111                         mov     wc.hCursor,   r8
112                         mov     wc.hbrBackground, r9
113
114 ; Continues filling the WNDCLASSEX structure:
115 ; No extra class/window memory
116 ; Application instance handle
117 ; Icon, cursor, and background brush from earlier
118 ; Small icon uses same as main icon
119 ; Sets the class name (pointer to string in "classname")
120 ; No menu specified
121                         lea     r8,  classname
122                         mov     wc.lpszMenuName,  0
123                         mov     wc.lpszClassName, r8
124                         mov     wc.hIconSm, rcx
125
126 ; Registers the window class
127                         lea     rcx, wc                                 ; invoke RegisterClassEx, addr wc
128                         call    RegisterClassEx
129
130 ; Window Size and Positioning:
131 ; Sets desired main window dimensions to 920×450 pixels
132                         mov     windowWidth, 920
133                         mov     windowHeight, 450
134 ;*****************************************************************;
```

```
135 ; Retrieves screen dimensions using GetSystemMetrics
136                         mov     ecx, SM_CXSCREEN
137                         call    GetSystemMetrics                ; invoke GetSystemMetrics,SM_CXSCREEN
138                         mov     screenWidth, eax
139                         mov     ecx, SM_CYSCREEN
140                         call    GetSystemMetrics                ; invoke GetSystemMetrics,SM_CYSCREEN
141                         mov     screenHeight, eax
142
143 ; Calculates centered position for the window:
144 ; (screen width - window width) / 2 ? left offset
145 ; (screen height - window height) / 2 ? top offset
146                         mov     eax, screenWidth                ; calculate offset from left side
147                         sub     eax, windowWidth
148                         shr     eax, 1
149                         mov     leftOffset, eax
150
151                         mov     eax, screenHeight               ; calculate offset from top edge
152                         sub     eax, windowHeight
153                         shr     eax, 1
154                         mov     topOffset, eax
155 ;*****************************************************************;
156 ; Window Creation:
157 ; Prepares parameters for CreateWindowEx:
158 ; Left and top positions (centered)
159                         xor     edx, edx
160                         mov     ecx, leftOffset
161                         mov     eax, topOffset
162                         mov     [rsp+4*8], rcx
163                         mov     [rsp+5*8], rax
164
165 ; Window width and height parameters
166                         mov     ecx, windowWidth
167                         mov     eax, windowHeight
168                         mov     [rsp+6*8], rcx
169
170 ; Additional parameters:
171 ; Parent and menu handles (NULL)
172 ; Application instance
173 ; Creation parameters (NULL)
174                         mov     rcx, hInstance
175                         mov     [rsp+7*8], rax
176                         mov     [rsp+8*8], rdx
177                         mov     [rsp+9*8], rdx
178                         mov     [rsp+10*8],rcx
179                         mov     [rsp+11*8],rdx
```

```
180
181 ; Creates the window with:
182 ; Extended style: Accepts file drag-and-drop (WS_EX_ACCEPTFILES)
183 ; Standard overlapped window style
184 ; Visible from creation
185 ; Class name and window caption
186 ; Stores the returned window handle in "hWnd"
187                         mov     r9d, WS_OVERLAPPEDWINDOW or WS_VISIBLE
188                         lea     r8,  caption
189                         lea     rdx, classname
190                         mov     ecx, WS_EX_LEFT or WS_EX_ACCEPTFILES
191                         call    CreateWindowEx                      ; invoke  CreateWindowEx,WS_EX_LEFT or WS_EX_ACCEPTFILES, \
192                                                                     ; ADDR classname,ADDR caption, WS_OVERLAPPEDWINDOW or \
193                                                                     ; WS_VISIBLE,  leftOffset,topOffset,windowWidth,windowHeight,\
194                                                                     ; 0,0,hInstance,0
195                         mov     hWnd, rax                           ; hWnd = Main window
196                         ret
197 WinMain                 endp
198 ;*************************************************************;
199 ; This is the standard Windows message loop that:
200 ; Jumps directly to Get_msg initially to start processing messages
201 ; Uses GetMessage to retrieve messages from the thread's message queue
202 ; Parameters are cleared (set to 0) using XOR before the call
203 ; rcx points to the msg structure
204 ; If GetMessage returns a non-zero value (indicating there's a message), it:
205 ; Calls TranslateMessage (for keyboard message translation)
206 ; Calls DispatchMessage (to send the message to the window procedure)
207 ; The loop continues until GetMessage returns 0 (typically indicating WM_QUIT)
208 ; The align 16 directive ensures the loop starts on a 16-byte boundary for potential performance benefits
209 msgloop                 proc
210                         jmp     Get_msg
211 align 16
212 @@loop:
213                         lea     rcx, msg
214                         call    TranslateMessage                    ; invoke    TranslateMessage, addr msg
215                         lea     rcx, msg
216                         call    DispatchMessage                     ; invoke    DispatchMessageW,  addr msg
217 Get_msg:
218                         xor     r9d, r9d
219                         xor     r8d, r8d
220                         xor     edx, edx
221                         lea     rcx, msg
222                         call    GetMessage                          ; invoke  GetMessage, addr msg, 0, 0, 0
223                         test    eax, eax
224                         jne     @@loop
```

```
225                         ret
226 msgloop                 endp
227 ;*************************************************************;
228 ; This is the main window procedure that handles messages sent to the main window("hWnd").
229 ; The message handling is fairly standard Windows programming but optimized in assembly
230 ; The code uses some interesting optimizations:
231 ; - Tail-call optimization for DefWindowProc
232 ; - Register saving/restoring for the window procedure parameters
233 ; - Direct jumps instead of procedure calls where possible
234
235 WndProc                 proc
236 ; Prologue:
237 ; Saves the incoming parameters (which are in registers in x64 calling convention) to the stack frame.
238 ; Save 4 registers                                            ; rbp+8=Return address
239                         mov     [rbp+16], rcx                 ; Save HWND (window handle)
240                         mov     [rbp+24], rdx                 ; Save message (e.g., WM_CREATE)
241                         mov     [rbp+32], r8                  ; Save wParam
242                         mov     [rbp+40], r9                  ; Save lParam
243
244                         cmp     rdx, WM_CREATE
245                         je      It_is_WM_CREATE
246                         cmp     rdx, WM_SYSCOMMAND
247                         je      It_is_WM_SYSCOMMAND
248                         cmp     rdx, WM_CLOSE
249                         je      It_is_WM_CLOSE
250                         cmp     rdx, WM_DESTROY
251                         je      It_is_WM_DESTROY
252 @Ret:
253 ; For unhandled messages, this:
254 ; Restores the original parameters
255 ; Jumps to DefWindowProc (using a tail call for efficiency)
256 ; The jmp is used instead of call/ret to avoid growing the stack
257 ;
258 ; Restore stack pointer
259                         lea     rsp,[rbp+8]                   ; rsp=Return address
260 ; Get address of default window proc
261                         lea     rax,DefWindowProc
262 ; Restore parameters in registers
263                         mov     r9, [rsp+32]
264                         mov     r8, [rsp+24]
265                         mov     rdx,[rsp+16]
266                         mov     rcx,[rsp+8]
267                         jmp     qword ptr [rax]
268 ;*************************************************************;
269 ; Saves the window handle from RCX to hWnd
```

```
; Initializes the Ribbon framework
; Returns 0 to indicate successful creation
It_is_WM_CREATE:
                        mov     hWnd, rcx
                        call    InitializeRibbonFramework
@Ret_0:
                        xor     eax, eax
                        ret
;*****************************************************************;
; Specifically checks for the SC_CLOSE system command
; Calls DestroyWindow if it's a close command
It_is_WM_SYSCOMMAND:
                        cmp     r8, SC_CLOSE
                        jne     @Ret

; Calls DestroyWindow when the window receives a close message
It_is_WM_CLOSE:
                        mov     rcx, hWnd
                        call    DestroyWindow
                        jmp     @Ret
;*****************************************************************;
; Tears down the Ribbon framework
; Posts WM_QUIT with exit code 0 to terminate the message loop
It_is_WM_DESTROY:
                        call    DestroyFramework
                        xor     ecx, ecx
                        call    PostQuitMessage
                        jmp     @Ret
WndProc                 endp
;*****************************************************************;
; This code is an implementation of initializing and destroying the Windows Ribbon Framework, which
; is a UI framework introduced in Windows 7 for creating modern,; Fluent-style user interfaces.
;
; The code follows standard COM programming patterns:
; Object creation via CoCreateInstance
; Reference counting via AddRef/Release
; Interface method calls through vtables
; Implements the required IUIApplication interface for Ribbon callbacks
; Uses the standard Ribbon initialization sequence
; Manually allocates memory for the application object
; Properly cleans up both framework and application objects
; Basic error checking after CoCreateInstance
; Graceful cleanup in destructor
; Direct vtable access via fixed offsets
; Manual memory management
```

```
315
316 InitializeRibbonFramework       proc
317 ;*************************************************************;
318 ; Calling CoCreateInstance to create an instance of the Ribbon framework.
319 ; Parameters:
320 ; CLSID_UIRibbonFramework: The class ID of the Ribbon framework
321 ; IID_IUIFRAMEWORK: The interface ID for the IUIFramework interface
322 ; CLSCTX_INPROC_SERVER: Context specifying an in-process server
323 ; The result is stored in g_pFramework
324                         lea     rax, g_pFramework               ; Result: IUIFramework interface
325                         lea     r9,  IID_IUIFRAMEWORK
326                         mov     [rsp+4*8], rax                  ; Result: IUIFramework interface
327                         mov     r8d, CLSCTX_INPROC_SERVER       ; CLSCTX_INPROC_SERVER = 1
328                         xor     edx, edx
329                         lea     rcx, CLSID_UIRibbonFramework
330                         call    CoCreateInstance                ; invoke CoCreateInstance, addr CLSID_UIRibbonFramework,0, \
331                                                                 ; CLSCTX_INPROC_SERVER, addr IID_IUIFRAMEWORK, addr
                                 g_pFramework
332 ; Checks the return value (in eax) and jumps to Ret_0 if there was an error (non-zero return value)
333                         test    eax, eax
334                         jne     Ret_0                           ; return FALSE
335
336 ; Next, we create the application object (IUIApplication) and call the framework Initialize method,
337 ; passing the application object and the host "hWnd" that the Ribbon will attach itself to.
338 ; Application Object Creation: Create -> g_pApplication
339 ; Allocates memory for the application object (24 bytes = 3*8) using malloc
340 ; Sets up a vtable (virtual function table) for the application object:
341 ; First entry points to AppRibbon_Vtbl of IUIApplication interface
342 ; Second entry is initialized to 1. It is a reference count. Third entry is initialized to 0
343                         mov     ecx, 3*8                        ; 24 = 3*8 -> address 0f MyVtabl, 1,0
344                         call    malloc
345                         lea     rcx, AppRibbon_Vtbl             ; vtbl of IUIApplication interface
346                         mov     g_pApplication, rax             ; IUIApplication interface
347                         mov     [rax], rcx
348                         mov     qword ptr [rax+1*8], 1
349                         mov     qword ptr [rax+2*8],0
350
351 ; Framework Initialization:
352 ; Calls the Initialize method of the IUIFramework interface (offset +3*8 in the vtable)
353 ; Parameters:
354 ; hWnd: The host window handle
355 ; g_pApplication: Result-> The application object implementing IUIApplication interface
356                         mov     rcx, g_pFramework               ; IUIFramework interface
357                         mov     rdx, hWnd                       ; main window
358                         mov     rax, [rcx]                      ; vtable of IUIFramework interface
```

```
359                         mov     r8,  g_pApplication                 ; Result: IUIApplication interface
360                         call    qword ptr [rax+3*8]                 ; call IUIFramework::Initialize method
361
362 ; Gets the module handle for the current process
363 ; Calls LoadUI method from offset 40 in the vtable of IUIFramework interface to:
364 ; - Load our compiled binary markup from "ribbonmarkup.xml" resource file
365 ; - Initiate callbacks to IUIApplication
366 ; - Bind command handlers
367 ; Uses the resource name "TEMPLATE64_RIBBON" (stored in szApllRibbon)
368                         xor     ecx, ecx
369                         call    GetModuleHandleW                    ; rax= module handle for the current process
370                         mov     rcx, g_pFramework                   ; IUIFramework interface
371                         mov     rdx, rax                            ; rax= module handle for the current process
372                         mov     rax, [rcx]                          ; rax= vtable of the IUIFramework interface
373                         lea     r8,  szApllRibbon                   ; L"TEMPLATE64_RIBBON"
374                         call    qword ptr [rax+40]                  ; call IUIFramework::LoadUI method
375                         xor     eax, eax                            ; Normal OK exit
376 Ret_0:
377                         ret
378 InitializeRibbonFramework       endp
379 ;*****************************************************************;
380 ; Framework Cleanup:
381 ; Checks if g_pFramework is non-zero
382 ; If exists:
383 ; Calls IUIFramework interface-> Destroy method (offset +4*8)
384 ; Calls IUIFramework interface-> Release method (offset +2*8) to decrement reference count
385 ; Sets g_pFramework to 0
386
387 DestroyFramework        proc
388                         cmp     g_pFramework,0
389                         jz      @f
390 ;g_pFramework->Destroy();
391                         mov     rcx, g_pFramework
392                         mov     rax, [rcx]
393                         call    qword ptr [rax+4*8]
394 ;g_pFramework->Release();
395                         mov     rcx, g_pFramework
396                         mov     rax, [rcx]
397                         call    qword ptr [rax+2*8]
398                         mov     g_pFramework,0
399 @@:
400 ;Application Object Cleanup:
401 ; Checks if g_pApplication -> IUIApplication interface is non-zero
402 ; If exists:
403 ; Calls IUIApplication interface-> Release method (offset +2*8) in vtable
```

```
404 ; Sets g_pApplication to 0
405                         cmp     g_pApplication,0    ; IUIApplication interface
406                         jz      @f
407 ;g_pApplication->Release();
408                         mov     rcx, g_pApplication
409                         mov     rax, [rcx]
410                         call    qword ptr [rax+2*8]
411                         mov     g_pApplication,0
412 @@:
413                         ret
414 DestroyFramework        endp
415 ;------------------------------------------------------------------;
416 AppQueryInterface       proc    gApplication:QWORD, iid:QWORD,ppv:QWORD
417                         mov     eax, -2147467263                    ; return E_NOTIMPL = ffffffff80004001H
418                         ret
419 AppQueryInterface       endp
420 ;------------------------------------------------------------------;
421 AppAddRef               proc
422                         mov     eax, -2147467263                    ; return E_NOTIMPL = ffffffff80004001H
423                         ret
424 AppAddRef               endp
425 ;------------------------------------------------------------------;
426 AppRelease              proc    gApplication:QWORD
427                         mov     eax, -2147467263                    ; return E_NOTIMPL = ffffffff80004001H
428                         ret
429 AppRelease              endp
430 ;------------------------------------------------------------------;
431 AppOnViewChanged        proc    gApplication:QWORD,viewId:QWORD, typeId:QWORD, pView:QWORD, verba:QWORD,  ReasonCode:QWORD
432                         mov     eax, -2147467263                    ; return E_NOTIMPL = ffffffff80004001H
433                         ret
434 AppOnViewChanged        endp
435 ;------------------------------------------------------------------;
436 AppOnCreateUICommand    proc    gApplication:QWORD, nCmdID:QWORD, typeID:QWORD, ppCommandHandler:QWORD
437                         mov     eax, -2147467263                    ; return E_NOTIMPL = ffffffff80004001H
438                         ret
439 AppOnCreateUICommand    endp
440 ;------------------------------------------------------------------;
441 AppOnDestroyUICommand   proc
442                         mov     eax, -2147467263                    ; return E_NOTIMPL = ffffffff80004001H
443                         ret
444 AppOnDestroyUICommand   endp
445 ;------------------------------------------------------------------;
446 end
```