If the prototypes of DispatchMessageA and DispatchMessageW are identical, why have both?

devblogs.microsoft.com/oldnewthing/20181101-00

November 1, 2018

Raymond Chen

There are a number of functions, mostly in the window manager, which have both ANSI and Unicode variants, even though the prototypes are identical.

```
LRESULT WINAPI DispatchMessageW(const MSG*);
LRESULT WINAPI DispatchMessageA(const MSG*);

BOOL WINAPI TranslateMessageW(const MSG*);
BOOL WINAPI TranslateMessageA(const MSG*);

int WINAPI TranslateAcceleratorW(HWND, HACCEL, LPMSG);
int WINAPI TranslateAcceleratorA(HWND, HACCEL, LPMSG);

HACCEL WINAPI CreateAcceleratorTableW(LPACCEL, int);
HACCEL WINAPI CreateAcceleratorTableA(LPACCEL, int);
```

Why can't these pairs of functions be combined into a single function? Clearly there's no `CHAR` / `WCHAR` mismatch, seeing as the parameters are identical.

While it's true that there is no type mismatch, there is still a character set dependency.

For the `MSG`-based functions, the system needs to know whether the message was obtained via `GetMessageW` / `PeekMessageW` or via `GetMessageA` / `PeekMessageA`. If the message is `WM_CHAR`, then the meaning of the `WPARAM` changes depending on the character set of the function that obtained the `MSG`. If you used `GetMessageW` / `PeekMessageW`, then the `WPARAM` is a Unicode code unit, but if you used `GetMessageA` / `PeekMessageA`, then it's an ANSI code unit.

The case of `CreateAcceleratorTable` is more subtle. Even though the same `ACCEL` structure is used for both ANSI and Unicode, the meaning of one of the fields changes:

```
typedef struct tagACCEL {
  BYTE fVirt;
  WORD key;
  WORD cmd;
} ACCEL, *LPACCEL;
```

If the `FVIRTKEY` flag is not set in the `fVirt` member, then the `key` member contains a character code, and that's the place where a character set dependency sneaks in.

Raymond Chen

Follow