ANSI and UNICODE ENCODING

TCHAR, LPSTR, LPCSTR, LPWSTR, LPCWSTR, LPTSTR, and LPCTSTR

[(17) ANSI and Unicode encoding, TCHAR LPSTR LPCSTR LPWSTR LPCWSTR LPTSTR LPCTSTR - YouTube]

Windows natively supports Unicode strings for UI elements, file names, and so forth. Unicode is the preferred character encoding, because it supports all character sets and languages. Windows represents Unicode characters using UTF-16 encoding, in which each character is encoded as a 16-bit value. UTF-16 characters are called *wide* characters, to distinguish them from 8-bit ANSI characters. The Visual C++ compiler supports the built-in data type **wchar_t** for wide characters. The header file WinNT.h also defines the following **typedef**.

C++ Copy

```
typedef wchar_t WCHAR;
```

You will see both versions in MSDN example code. To declare a wide-character literal or a wide-character string literal, put **L** before the literal.

C++ Copy

```
wchar_t a = L'a';
wchar_t *str = L"hello";
```

we know**strlen**The definition is as follows:

```
size_t strlen(const char*);
```

It can also be written as

```
size_t strlen(LPCSTR);
```

and so

```
// Simplified
typedef const char* LPCSTR;
```

Its meaning is as follows

- **LP:** Long Pointer
- **C:** Constant
- **STR:** String

Long Pointer has the same meaning as Pointer.

Here are some other string-related typedefs that you will see:

Typedef	Definition
CHAR	`char`
PSTR or LPSTR	`char*`
PCSTR or LPCSTR	`const char*`
PWSTR or LPWSTR	`wchar_t*`
PCWSTR or LPCWSTR	`const wchar_t*`

By analogy, for Unicode characters, we have:

```
size_t wcslen(const wchar_t* szString); // Or WCHAR*
size_t wcslen(LPCWSTR szString);
```

Here **LPCWSTR**representative

```
typedef const WCHAR* LPCWSTR;
```

Its meaning is as follows

- **LP** - Pointer
- **C** - Constant
- **WSTR** - Wide character String

Going further, there is **LPCTSTR**

- LP - Pointer
- C - Constant
- **T = TCHAR**
- STR = String

to sum up:

- **TCHAR-char / wchar_t (depending on character set)**
-
- **LPSTR - char***
- **LPCSTR - const char***
- **LPWSTR - wchar_t***
- **LPCWSTR - const wchar_t***
-
- **LPTSTR - TCHAR***
- **LPCTSTR - const TCHAR***

Unicode and ANSI Functions

When Microsoft introduced Unicode support to Windows, it eased the transition by providing two parallel sets of APIs, one for ANSI strings and the other for Unicode strings. For example, there are two functions to set the text of a window's title bar:

- **SetWindowTextA** takes an ANSI string.
- **SetWindowTextW** takes a Unicode string.

Internally, the ANSI version translates the string to Unicode. The Windows headers also define a macro that resolves to the Unicode version when the preprocessor symbol `UNICODE` is defined or the ANSI version otherwise.

C++ Copy

```
#ifdef UNICODE
#define SetWindowText  SetWindowTextW
#else
#define SetWindowText  SetWindowTextA
#endif
```

TCHARs

Back when applications needed to support both Windows NT as well as Windows 95, Windows 98, and Windows Me, it was useful to compile the same code for either ANSI or Unicode strings, depending on the target platform. To this end, the Windows SDK provides macros that map strings to Unicode or ANSI, depending on the platform.

Macro	Unicode	ANSI
TCHAR	`wchar_t`	`char`
TEXT("x")	`L"x"`	`"x"`

For example, the following code:

C++ Copy

```
SetWindowText(TEXT("My Application"));
```

resolves to one of the following:

C++ Copy

```
SetWindowTextW(L"My Application"); // Unicode function with wide-character string.

SetWindowTextA("My Application");  // ANSI function.
```

The **TEXT** and **TCHAR** macros are less useful today, because all applications should use Unicode. However, you might see them in older code and in some of the MSDN code examples.

The headers for the Microsoft C run-time libraries define a similar set of macros. For example, **_tcslen** resolves to **strlen** if `_UNICODE` is undefined; otherwise it resolves to **wcslen**, which is the wide-character version of **strlen**.

C++ Copy

```
#ifdef _UNICODE
#define _tcslen     wcslen
#else
#define _tcslen     strlen
#endif
```

Be careful: Some headers use the preprocessor symbol `UNICODE`, others use `_UNICODE` with an underscore prefix. Always define both symbols. Visual C++ sets them both by default when you create a new project.

In windows, the general prefix**T** It means that it can adapt to different character sets.

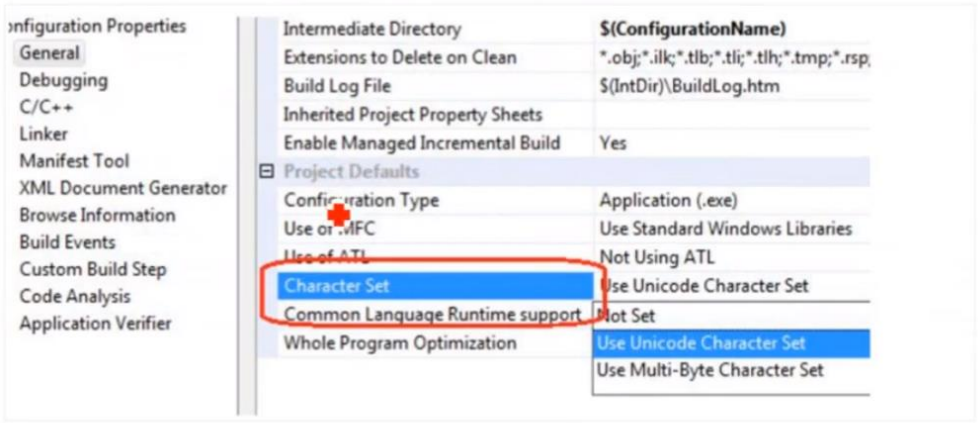
such as:**strcpy** , **strlen** , **strcat**(Including security suffix _s) represents the ANSI version;

wcscpy , **wcslen** , **wcscat**(Including the security suffix _s), which represents the Unicode version, where WC stands for Wide Character;

_tcscpy , **_tcslen** , **_tcscat**It depends on the situation:

```
size_t strlen(const char*); //ANSI
size_t wcslen(const wchar_t* ); //Unicdoe
size_t _tcslen(const TCHAR* ); //ANSI or Unicode
```

TCHARIs based on the selected character set to determine whether to translate into**char**still is**wchar_t**, The character set is as follows:



and soTCHARIs defined as follows:

```
#ifdef _UNICODE
typedef wchar_t TCHAR;
#else
typedef char TCHAR;
#endif
```