Parsing C command-line arguments

Article • 12/11/2021 • 2 minutes to read • 7 contributors

In this article

Microsoft Specific

Microsoft C startup code uses the following rules when interpreting arguments given on the operating system command line:

- Arguments are delimited by whitespace characters, which are either spaces or tabs.
- The first argument (`argv[0]`) is treated specially. It represents the program name. Because it must be a valid pathname, parts surrounded by double quote marks (`"`) are allowed. The double quote marks aren't included in the `argv[0]` output. The parts surrounded by double quote marks prevent interpretation of a space or tab character as the end of the argument. The later rules in this list don't apply.
- A string surrounded by double quote marks is interpreted as a single argument, whether it contains whitespace characters or not. A quoted string can be embedded in an argument. The caret (`^`) isn't recognized as an escape character or delimiter. Within a quoted string, a pair of double quote marks is interpreted as a single escaped double quote mark. If the command line ends before a closing double quote mark is found, then all the characters read so far are output as the last argument.
- A double quote mark preceded by a backslash (`\"`) is interpreted as a literal double quote mark (`"`).
- Backslashes are interpreted literally, unless they immediately precede a double quote mark.
- If an even number of backslashes is followed by a double quote mark, then one backslash (`\`) is placed in the `argv` array for every pair of backslashes (`\\`), and the double quote mark (`"`) is interpreted as a string delimiter.

- If an odd number of backslashes is followed by a double quote mark, then one backslash (`\`) is placed in the `argv` array for every pair of backslashes (`\\`). The double quote mark is interpreted as an escape sequence by the remaining backslash, causing a literal double quote mark (`"`) to be placed in `argv`.

This list illustrates the rules above by showing the interpreted result passed to `argv` for several examples of command-line arguments. The output listed in the second, third, and fourth columns is from the ARGS.C program that follows the list.

Command-line input	argv[1]	argv[2]	argv[3]
`"a b c" d e`	`a b c`	`d`	`e`
`"ab\"c" "\\" d`	`ab"c`	`\`	`d`
`a\\\b d"e f"g h`	`a\\\b`	`de fg`	`h`
`a\\\"b c d`	`a\"b`	`c`	`d`
`a\\\\"b c" d e`	`a\\b c`	`d`	`e`
`a"b"" c d`	`ab" c d`		

Example

Code

C Copy

```c
// ARGS.C illustrates the following variables used for accessing
// command-line arguments and environment variables:
// argc  argv  envp
//

#include <stdio.h>

int main( int argc, // Number of strings in array argv
char *argv[],      // Array of command-line argument strings
char **envp )      // Array of environment variable strings
{
    int count;

    // Display each command-line argument.
    printf_s( "\nCommand-line arguments:\n" );
    for( count = 0; count < argc; count++ )
```

```
        printf_s( "  argv[%d]   %s\n", count, argv[count] );

    // Display each environment variable.
    printf_s( "\nEnvironment variables:\n" );
    while( *envp != NULL )
        printf_s( "  %s\n", *(envp++) );

    return;
}
```

One example of output from this program is:

```
Command-line arguments:
  argv[0]   C:\MSC\ARGS.EXE

Environment variables:
  COMSPEC=C:\NT\SYSTEM32\CMD.EXE

PATH=c:\nt;c:\binb;c:\binr;c:\nt\system32;c:\word;c:\help;c:\msc;c:\;
  PROMPT=[$p]
  TEMP=c:\tmp
  TMP=c:\tmp
  EDITORS=c:\binr
  WINDIR=c:\nt
```

END Microsoft Specific

See also

main function and program execution

Recommended content

/P (Preprocess to a File)

Learn more about: /P (Preprocess to a File)

Pragma directives and the __pragma and _Pragma keywords

Describes the pragma directives available in Microsoft Visual C and C++ (MSVC)

VCCLCompilerTool.AdditionalOptions Property (Microsoft.VisualStudio.VCProjectEngine)

Gets or sets options to add to the end of the command line immediately before the file name(s). An example is if an option is not supported in the object model.

/Ox (Enable Most Speed Optimizations)

The MSVC /Ox option combines some of the compiler optimization options for speed into a single option.

/O options (Optimize code)

The MSVC /O compiler options specify the compiler optimizations to use.

__func__

Learn more about: __func__

String and character literals (C++)

How to declare and define string and character literals in C++.

/w, /W0, /W1, /W2, /W3, /W4, /w1, /w2, /w3, /w4, /Wall, /wd, /we, /wo, /Wv, /WX (Warning level)

Reference for the Microsoft C/C++ compiler options: /w, /W0, /W1, /W2, /W3, /W4, /w1, /w2, /w3, /w4, /Wall, /wd, /we, /wo, /Wv, and /WX.

Show more