Grandez

21-12-2021



**Command Line Parser User Manual**

Indice

1 Introducción 2

2 Conceptos 3

2.1 Definiciones 3

3 Modos de funcionamiento 4

4 Tipos de datos 2

4.1 Rangos 2

4.2 Conjuntos 3

5 Herencia 4

6 Verbos 5

7 Libreria cmdline 6

7.1 Introduccion 6

7.2 Enumeraciones 6

7.2.1 Type 6

7.2.2 Source 6

7.3 Parametros 6

7.4 Flags 6

8 Excepciones 7

9 Caracteristicas 1

10 Clases 2

11 Excepciones 3

12 Linea de comandos 4

12.1 Taxonomia de la línea de comandos 4

12.1.1 Programa ejemplo 4

12.2 Analisis 6

12.3 Analisis 7

Disclaimer

As my mother tongue is Spanish and not forgetting that it is one of the most widely spoken languages ​​in the world but for sack of accuracy this document has been written is Spanish Castillian.

However, if someone is interested in the job or wants more information, I will be happy to translate the documents or help as much as possible.

In any case, for readability and maintainability reasons, I use English inside the code.

# Introducción

Aunque este documento se ha titulado “manual de usuario” no deja de ser un manual de uso de una librería C++, y por lo tanto, un manual técnico.

# Conceptos

## Definiciones

# Modos de funcionamiento

Existen métodos que definen el funcionamiento:

* Sensibilidad: Los parámetros son sensibles a mayúsculas y minúsculas. Por defecto, si la plataforma es MS Windows, no es sensible
* Estricto: Es posible especifi
* Global
* Parámetros con nombre

Sensibilidad

Hace referencia a sí en los parámetros, no los valores, se distinguen mayúsculas de minúsculas; es decir, si “help” se considera distinto de “Help” o no.

En MS Windows no es sensible por defecto de acuerdo con los criterios de la plataforma.

Estricto

Define el comportamiento cuando un mismo parámetro aparece mas de una vez en la línea de comandos.

Si se define como estricto, cada parámetro solo puede aparecer una vez, independientemente de que acepte múltiples valores o no.

Por defecto si un parámetro aparece mas de una vez en la línea de comandos se actúa de la siguiente manera:

1. Si el parámetro no acepta múltiples valores se aplica el último indicado en la línea de comandos
2. Si el parámetro acepta múltiples valores entonces se añaden

Global

Suele ser una práctica común procesar primero todos los parámetros antes de procesar los argumentos independientemente del orden que se hayan introducido; ese es el modo de funcionamiento por defecto.

En algunas ocasiones puede que sea significativo procesar los parámetros y argumentos en el orden en el que se han introducido.

Ejemplo:

XXXX

Parámetros con nombre

Como se ha indicado, una posible opción de indicar los parámetros puede ser mediante el par clave=valor, esta es una posible opción pero puede dar problemas si el programa no tiene contemplado esa posibilidad, supongamos el siguiente comando:

programa /parm1 value1 argument1 argument2 parm1=value2

En el caso de que se admitan los parámetros con nombre, no se indica de manera explicita si el parámetro es un flag, una opción o una definición para el programa y es posible , aunque no recomendable, que un mismo nombre se use para un flag y una opción; en esta situación se aplica el siguiente criterio de prioridades:

1. Flag
2. Opción
3. Definición

Definiciones

Ciertos programas pueden aceptar “definiciones”; es decir, parámetros que no están definidos en el programa pero que pueden ser utilizados por él en tiempo de ejecución.

En el caso de que el programa acepte parámetros con nombre

# Tipos de datos

La librería soporta los siguientes tipos de datos:

|  |  |  |
| --- | --- | --- |
| Grupo | Tipo | Descripción |
| Cadenas | |  |
|  | CHAR | Un carácter |
|  | STRING | Una cadena |
| Números | |  |
|  | BYTE | Un número en los rangos [0,255] o [-127,127] |
|  | SHORT | Un número en los rangos [0,65535] o [-32768, 32767] |
|  | NATURAL | Un número positivo hasta 4294967295 |
|  | INTEGER | Un número entero dependiendo de la arquitectura |
|  | LONG | Un número entero dependiendo de la arquitectura |
|  | DECIMAL | Un número con decimales |
| Fechas y horas | |  |
|  | DATE | Una fecha de acuerdo con el formato del país |
|  | TIME | Una hora en formato hh:mm:ss |
|  | DATETIME | Fecha y hora en formato “YYYY/MM/DD hh:mm:ss” |
|  | TMS | Timestamp |
| Almacenamiento | |  |
|  | DIR | Una ruta que pueda intepretarse como un directorio |
|  | DIR\_EXISTS | Un ruta a un directorio existente |
|  | FILE | Una ruta que pueda intepretarse como un fichero |
|  | FILE\_EXISTS | Una ruta a un fichero existente |
| Lógico | |  |
|  | BOOL | Una secuencia que pueda interpretarse como un valor lógico:  TRUE / FALSE  ON / OFF  YES / NO  -1 / 0 / 1 |
| Configuración | |  |
|  | CONFIG | Un fichero de configuración |
|  | CONFIG\_RELAXED | Un fichero de configuración |

Notas:

* CONFIG/CONFIG\_RELAXED: Si el fichero de configuración contiene claves que no están definidas provocará una excepción si el tipo es CONFIG

## Rangos

Los tipos de datos numéricos soportan opcionalmente un rango de valores válidos en el rango [min, max]

## Conjuntos

Los tipos de datos STRING y CHAR soportan opcionalmente un conjunto de valores validos

# Herencia

Como se puede observar, la librería maneja XXXX

La idea subyacente es que cada programa genere su propia clase especializada abstrayendo la clase base y haciéndola mas amigable al desarrollador mediante wrappers and friendly names

Ejemplo:

Supongamos un programa que acepta:

* verbose como flag
* dirOut como directorio de salida único

Una posible implementación sería:

#include “cmdline.hpp”

class MyCLP : protected CmdLine {

public:

MyCLP() = delete;

MyCLP(int argc, const char\*\* argv);

bool isVerbose();

string getDir();

}

myCLP.hpp

#include “cmdline.hpp”

Parameters parms = {

ParmFlag ("verbose", true)

ParmOption("dirOut", Type::DIR)

};

MyCLP::MyCLP(int argc, const char\*\* argv): CmdLine(argc, argv, parms){}

bool MyCLP::isVerbose() { return hasFlag(“verbose”); }

string MyCLP::getDir() { return getOption(“dirOut”); }

myCLP.cpp

#include “myCLP.hpp”

int main (int argc, const char \*\*argv) {

MyCLP myCLP(argc, argv);

if (myCLP.isVerbose()) ….

…

Return 0;

}

main.cpp

# Verbos

Por definicion los parametros son inmutables, lo cual no implica que se puedan leer.

* getXXX: Obtiene una referencia al parametro
* letXXX: Obtieneuna copia del parametro
* hasXXX: Indica siexiste o no
* addXXX: Permite incluir parametros

# Libreria cmdline

Placeholder

## Introduccion

## Enumeraciones

### Type

### Source

## Parametros

## Flags

# Excepciones

Si se produce un error durante el proceso de análisis de la línea de comandos o se produce un error en tiempo de ejecución se genera una excepción **CmdLineException**

Distinguimos los siguientes tipos:

CmdLineParameterException;El parámetro es erroneo CmdLineValueException;el valor asociado alparámetro no es valido de acuerdo con el tipo indicado

CmdLineNotFoundException;runtime\_error;Se ha solicitado un parámetro que no existe CmdLineInvalidTypeException;Se ha solicitado una conversion no valida de un parámetro

Además, para gestionar la solicitud de ayuda, se generan las siguientes excepciones:

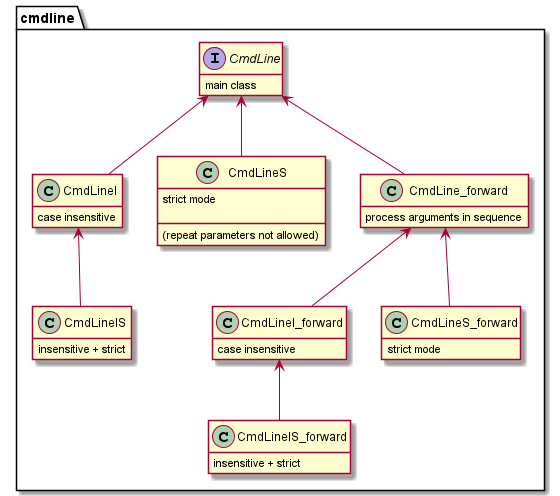
* HelpRequested
* HelpDetailedRequested

Libreria

# Caracteristicas

# Clases

La clase base de la librería se denomina CmdLine y se implementa como singleton



Todas las instancias devuelve un objeto CmdLine

|  |  |  |
| --- | --- | --- |
|  |  |  |
| CmdLine |  |  |
| CmdLineI |  |  |
| CmdLineIS |  |  |
|  |  |  |
| CmdLineS |  |  |

# Excepciones

De acuerdo con los criterios de C++ los posibles errores generan excepciones.

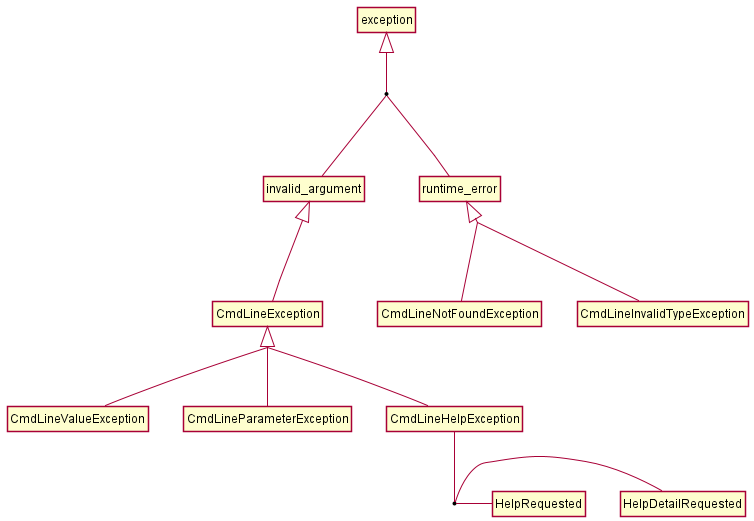
Considerando que la librería analiza la línea de comandos distinguimos dos tipos de excepciones:

* Errores en tiempo de desarrollo
* Entrada invalida de datos

Los primeros se consideran que solo pueden ocurrir en tiempo de desarrollo por que ocurren cuando se intenta acceder a un parámetro que no existe o no está definido; es decir, dado que el acceso se produce de manera programática solo puede darse en el caso de que se haya tecleado o definido de manera errónea ese valor en el programa.

Los segundos se producen en función de los datos introducidos por el usuario: nombre del parámetro incorrecto, valores erróneos, etc.

La estructura de las excepciones se muestra en el siguiente diagrama:



# Linea de comandos

La línea de comandos es la forma de ejecutar cualquier programa en un ordenador, ya sea de forma directa, mediante un terminal, o a través de iconos y otras maneras mas o menos amigables. Esta línea de comandos está compuesta por, el nombre y ubicación del programa a ejecutar junto con un conjunto de parámetros que permiten adecuar la ejecución concreta del programa a las condiciones particulares esa ejecución, además de indicar sobe que objeto u objetos debe operar el programa.

Por poner únicamente un ejemplo, el procesador de textos de Microsoft®, WinWord® 365 admite 17 parámetros diferentes en la línea de comandos.

## Taxonomia de la línea de comandos

Como se ha indicado, a continuación del nombre del programa se pueden indicar un conjunto de datos, a los que denominaremos de manera general parámetros o argumentos que modifican o definen el comportamiento predefinido del programa para esa ejecución, estos parámetros solo pueden ser de los siguientes tipos:

1. **Argumentos**: Objetos que se deben procesar
2. **Flags**: Indicadores que activan o desactivan alguna funcionalidad del programa
3. **Opciones**: Datos que cambian o definen determinados componentes predefinidos del programa
4. **Definiciones**: Información que se añade al programa para su posible uso pero que no están predefinidos en el mismo

### Programa ejemplo

Pongamos por ejemplo un programa, no muy útil a efectos prácticos, al que llamaremos collector .

La funcionalidad de este programa es recolectar un conjunto de ficheros de un conjunto de directorios y copiarlos en otro directorio, durante el proceso además, el programa busca dentro de los ficheros ciertas palabras clave, y si existen, las cambia por otras definidas por el usuario. Además, si no se le indica ningún directorio de entrada, y no se indica ninguna lista de ficheros, lee los datos que se introduzcan por consola y si no se le indica un directorio de salida, el resultado lo escribe, con un nombre predeterminado en el directorio actual.

El programa se puede configurar con los siguientes parámetros:

|  |  |  |
| --- | --- | --- |
| Parámetro | Defecto | Descripción |
| In | Consola | Indica el/los directorios donde buscar ficheros |
| Out | Directorio actual | Indica donde guardar las copias |
| file | Collector.txt | Nombre del fichero a generar |
| Overwrite | Si | Sobrescribir un fichero si ya existe en el directorio de salida |
| Pattern | \*.\* | Mascara de ficheros a procesar |
| Verbose | Si | Mostrar o no información de progreso del programa |
| Summary | Si | Mostrar resumen del resultado de la ejecución |
| Define | N/A | Conjunto de pares palabra clave/valor que se deben sustituir |

De acuerdo con las convenciones actuales la definición del programa podría ser parecido a esta:

Método de parámetros largos \*nix (--nombre):

Collector [–-overwrite -–verbose --summary --out dir\_output --file file\_name

[--in dir\_input]\* [--pattern pattern]\* [--define clave=valor]\* ...

Método de parámetros cortos \*nix (-letra)

Collector [–svw -o dir\_output -f file\_name [-i dir\_input]\* -p pattern]\* [-d clave=valor]\* ...

Método de parámetros Windows:

Collector [/s/v/W /o dir\_output /f:file\_name [/i dir\_input]\* [/p pattern]\* [/d:clave=valor]\* ...

## Analisis

Tomando como referencia el programa anterior, tenemos

Argumentos: Los puntos suspensivos

Flags: overwrite, summary, verbose

Opciones: in, out, pattern

Definiciones: define

Veamos las diferencias entre cada uno de estos tipos:

* Los argumentos son los objetos que debe procesar el programa, por lo tanto son dependientes de cada programa.
* Los flags únicamente indican la activación/desactivación de la funcionalidad correspondiente
* Las opciones necesitan información adicional, pero también están definidas en el programa
* Las definiciones también necesitan información adicional, indicada por el signo igual (=) pero a diferencia de las opciones, no están definidas en el programa

Es decir, si ejecutamos alguno de los siguientes comandos:

Collector -x

Collector -v -o

Collector /j opcion

El programa deberá emitir algún mensaje de error indicando que el parámetro no existe o que le faltan datos. Sin embargo, el siguiente comando:

collector -d autor=yo\_mismo

El programa no puede identificar, a priori, si ‘autor’ será útil o no en el programa.

Por otro lado, los siguientes comandos son equivalentes:

collector –-verbose -w

collector –-overwrite

collector

Puesto que los flags verbose y overwrite están activados por defecto

Esto obliga a incluir otras opciones en el programa, las que inhabilitan las funcionalidades activas por defecto en el programa, en nuestro caso asumiremos que son:

-m | --mute para no mostrar la información de progreso

-n | --nooverwrite para evitar sobreescribir ficheros existentes

-r | --nosummary para no emitir el resumen

Con lo que el programa completo se define ahora como:

Collector [–mnrsvw -o dir\_output -f file\_name [-i dir\_input]\* -p pattern]\* [-d clave=valor]\* ...

## Analisis

Tomando como referencia el programa anterior, tenemos

1. En función del sistema operativo el modo de indicar los parámetros difiere
2. No se puede identificar que es un flag o una opción
3. No se puede conocer directamente si se está activando o desactivando un flag
4. El uso de parámetros largos es proclive a errores de escritura

Ademas hay que tener en cuenta que el programa debe