***JArgs2 – A Simple Command Line Parser***

***What Is It?***

I occasionally write simple C# console applications, and have become frustrated at having to write code to handle command line arguments each time. So, I decided to create a library that can be reused. These notes detail this library.

***How To Use It***

Three ways:

1. Download the dll and copy it to a folder. From your code, right-click on the project name in Visual Studio’s Solution Explorer and choose ‘Add’ then ‘Reference’. Navigate to the dll, tick the box, then ‘OK’. Add ‘using JArgs2’ to the ‘using’ section of your code. Use the parser as explained below.

2. Download the source files. Study the code and verify that it is not malicious. Use Solution Explorer to add them to a new ‘Class Library’ project in Visual Studio and compile a dll yourself. Use it as above.

3. Download the source files and drag them into your project in Solution Explorer. Edit the namespace code in each file. You can then call the parser directly from your code and the classes will be compiled directly into your .exe. No dll needed.

***Classes Of Command Line Arguments***

I decided that I would allow the parser to look for two different types of command-line argument.

|  |  |
| --- | --- |
| Switches: | Used for Boolean items. For example, /R to recurse.  -R and --R are also recognised as is any substring of the full name, for example /Rec  The user either includes these on the command line or omits them. |
| Values: | Used for passing in values. For example, /delay=10 or /password=^f\*g$%gd  As above, -del=10 and --d=10 are also recognised.  Values may be configured as mandatory or optional.  The user is also permitted to enter values on the command line without explicitly associating them with expected items. For example, instead of entering /path=c:\\*.tmp he could simply enter c:\\*.tmp These are then associated with values by the order in which they occur and the order the items were added to the parser. This happens only after all other arguments have been processed. For this reason, you should add optional values items to the parser *after* all of the mandatory ones. |

Two switches are added automatically: /? And /help They both bring up help text.

***Reading Arguments From A File***

By pointing the ArgFile property (see below) to a text file, the command line can be bypassed. If any command line argument is present, the argument file is ignored. If no command line arguments are entered, then the argument file is read.

The argument file should have once argument per line. Lines beginning with // or # are ignored.

***Classes In the DLL***

There is just one externally available class: Parser. You will need to instantiate a Parser in your code.

***Public Properties***

|  |  |  |
| --- | --- | --- |
| public string Version | Read only. | Reports the version of the Parser class that you are using. |
| public string ClientVersion | Read / write | Use this to tell the parser the name of the application using it. So that it can display this within help information. |
| public string ClientVersion | Read / write | Use this to tell the parser the version of the application using it. So that it can display this within help information. |
| public string HelpText | Read / write | Use it to set help text that will be displayed if the arguments are incorrect or the user enters /? Or /help on the command line. |
| public string ArgFile | Read / write | Use this to point the parser to an argument file instead of using the command line. |

***Public Methods***

|  |  |
| --- | --- |
| public Parser(  string ClientName,  string ClientVersion,  string HelpText,  bool IgnoreCase = true ) | Constructor.  You must set the name and version of the client app, enter any help text to be displayed if an error occurs, and specify whether or not to ignore case when matching command line arguments to the items expected by the parser. |
| public bool AddSwitch(  string Name,  string Description) | Used to tell the parser to expect a switch.  Specify the name of the switch e.g. Recurse and a description e.g. “Process sub-directories”. |
| public bool AddValue(  string Name,  string Description,  bool Mandatory) | Used to tell the parser to expect a ‘value’ item.  Specify the name – e.g. Password and a description e.g. “Enter password”.  You must also say whether it is mandatory that the user enter this value. |
| public bool Parse() | Call this only after all of the expected items have been added to the parser. It causes the command line to be read and arguments to be matched to the expected items added to the parser.. |
| public bool IsItemPresent(  string Name) | Returns true if an argument matching the named item is present on the command line. |
| public string GetValue(  string Name) | Returns the value entered on the command line corresponding to this item name. Throws an exception if there was no argument entered which matches the item with this name. So check with IsItemPresent() before calling. |
| public override string ToString() | Returns the current state of the parser (items added and arguments read) as a string. |

***Example Usage***

**using** System**;**

**using** JArgs2**;**

**namespace** Demo2

**{**

class Program

**{**

static void Main**(**string**[]** args**)**

**{**

Parser P **=** **new** Parser**(**"Demo2.exe"**,** "1.0"**,**

"Help Text goes here."**);**

P**.**AddSwitch**(**"Recurse"**,**

"Should program recursively access subfolders?"**);**

P**.**AddValue**(**"Retry"**,**

"How many times should the program retry to read a file?"**,**

**true);**

P**.**AddValue**(**"Wait"**,**

"How long should we wait if a folder cannot be accessed?"**,**

**false);**

P**.**AddValue**(**"Root"**,**

"Path to the directory tree to process."**,** **true);**

P**.**AddValue**(**"Log"**,** "Path to a log file."**,** **false);**

P**.**AddValue**(**"ErrorLog"**,** "Path to an error log."**,** **false);**

**if** **(!**P**.**Parse**())**

**{**

Console**.**WriteLine**(**"Processing command line failed."**);**

Console**.**ReadKey**();**

**return;**

**}**

**if** **(**P**.**IsItemPresent**(**"Recurse"**))**

Console**.**WriteLine**(**"Recurse selected"**);**

// Retry must be present because it was a mandatory item

// and P.Parse returned true.

Console**.**WriteLine**(**"Retry " **+** P**.**GetValue**(**"Retry"**)** **+** " times"**);**

**if** **(**P**.**IsItemPresent**(**"Log"**))**

Console**.**WriteLine**(**"Log file path is: " **+**

P**.**GetValue**(**"Log"**));**

Console**.**ReadKey**();**

**}**

**}**

**}**