Lamey™ Command Line Utility

The Lamey™ Command Line Utility is a tool used to enable the fast and easy creation of settings classes that interface directly with the input from the command line. The Command Line Utility performs all the parsing and conversion operations required to turn command line input into class property values. Using the utility, your settings classes will be simple, easy to read and understand, and require very little coding to configure.

This document outlines the features of the Command Line Utility, their intended purposes, and how to implement them in your own project.

# Terminology

Throughout this document, whenever any of the following terms are printed in **bold**, refer to this section for its meaning.

|  |  |
| --- | --- |
| Term | Description |
| Utility | The Lamey™ Command Line Utility |
| Collection | An array (System.Array) or a type inheriting from System.Collections.ICollection. |
| Convertible Type | A type that can be converted to by the methods inherited from the System.IConvertible interface, or an enum type. |
| Convertible Collection | A **collection** whose elements are of a **convertible type**. |
| GIA | Global Indexed Argument |
| GUA | Global Unconsumed Argument |

|  |
| --- |
| **Important Note:** The **Utility** has built-in validation functionality to ensure that your settings class is designed properly. However, this validation will only run when a debugger is attached to the process, so as not to negatively impact the performance of your application’s production release. Ensure that you test your project with a debugger attached to make sure it is designed properly. If you want to run this settings class validation manually, simply call the CommandLineParser.ValidateSettingsClass method. |

# Ingredients

The **utility**’s ingredients consist of switches, their arguments, global indexed arguments, global unconsumed arguments, and argument validation methods.

## Switches

Switches are used to turn features of your application on or off. On the command line, they’re generally indicated by a dash (“-“) or a forward slash (“/”) followed by the name of the switch, with no space between them. A common use is for silent functionality. If an application should run with no UI or user input, it’s said to be running silently, and many applications support a switch that turns silent mode on. Example: *myapp.exe -silent* or *myapp.exe -s*.

Occasionally, switches accept arguments immediately following the switch. An example would be an application that outputs to a file specified on the command line: *myapp.exe -out “C:\myfile.txt”*.

## Global Indexed Arguments

Arguments are essential to many applications, providing a fast and easy way to give the application input. Global Indexed Arguments (GIA’s) are arguments that are accepted as input without the need of a preceding switch, they are generally used when the application *requires* that argument’s input to function correctly.

Since these arguments belong to the command and not a switch, we refer to them as “global”. These global arguments are usually required to follow a certain order, so that the application knows how to use each input. For this reason, these global arguments are indexed numerically, hence its name. A common use is the specification of an output file, same as before except without the switch (implying that this information is now somewhat more vital): *myapp.exe “C:\myfile.txt”*.

## Global Unconsumed Arguments

# Switches

* The property scope must be public.
* The property must be an instance property, not static.
* Property type can be any of the following types:
  + A **convertible type**
  + A **collection** whose elements are any of the above types.
* If the property is not a collection and it allows arguments, then restrict it to one argument.
* If SetValue is null, the default value for the switch’s property type will be used.
  + Note: Default values can be found at <http://msdn.microsoft.com/en-us/library/83fhsxwc.aspx>
  + Note: For enum type MyEnum, the default value is the equivalent of “(MyEnum)0”. So ensure that your enum types include a value whose underlying value is 0.

# Global Indexed Arguments

* The property scope must be public.
* The property must be an instance member, not static.
* Property type must be a **convertible type**,and not a **collection**.

# Global Unconsumed Arguments

* The GUA’s property is defined in the Lamey.Tools.CommandLine.ISettings interface.
* The property is scoped as public.
* The property is an instance member, not static.
* The property type is an array of type System.String.

# Argument Validation Methods

* The method scope must be public.
* The method must be an instance member, not static.
* The method must have one parameter.
* Parameter modifiers “out” and “ref” are not allowed. [TODO: or should they be?]

## Switch Arguments

* The method’s parameter must be one of the following types:
  + System.String
  + The same type or an ancestor type of the associated switch’s property’s type.
  + A **collection** whose element type is System.String.
  + A **collection** whose element type is the same type or an ancestor type of the associated switch’s property’s type.
    - “params” are allowed, this should not make a difference to the Utility
  + Parameter modifiers “out” and “ref” are not allowed. [TODO: or should they be?]
* The method must have the following return type:
  + System.Int32, if the parameter is a **collection**.
  + System.Boolean, if the parameter is not a **collection**.

## Global Indexed Arguments

* The method’s parameter must be one of the following types:
  + System.String
  + The same type or an ancestor type of the associated argument’s property’s type
* The method must have the return type System.Boolean.

## Global Unconsumed Arguments

* The method’s parameter must be a **collection** of type System.String.
* The method must have the return type System. Boolean.