Simple data processor

1 Abstract

This document describes required functionality of the application. It defines class interfaces and sets guidelines which should be taken into consideration while writing code.

2 Functionality

First program loads these configuration values:

Key	**Value**	**Description**
type	String e.g.: "int"	Defines data type, which will be loaded ("int", "float", "string"
prefix	String: "prcss01_"	Prefix, which will be added to data
filePath	String e.g.: "c:\Import\data.txt"	Path to file with data

Application will load data from file and based on configuration values it will try to process them in following way:

- If (Type == int)
  - Adds 1000
  - Converts to string
  - Adds prefix before string
- If (Type == float)
  - Adds 0,5
  - Rounds to integer
  - Adds 1000
  - Converts to string
  - Adds prefix before string
  - Adds "FLOAT_" before string
- If (Type == string)
  - Sorts word in the string by first and second letter
  - Adds prefix before string
- Else
  - Throws Exception

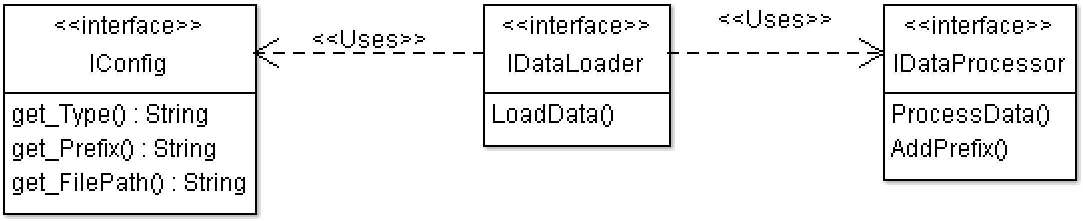
And finally it saves data.

Additionally application should log: start, successful finish (with type and number of records processed) and any errors encountered.

BONUS1: It saves result date into MS SQL or LiteDB database (http://www.litedb.org/).

BONUS2: Prepare application for parallel data processing.

3 INTERFACE



Getters in IConfig should be replaced with Get e.g.: string get_Type(); will be replaced with string Type { get; }.

4 GUIDELINES

4.1 DRY

https://en.wikipedia.org/wiki/Don%27t_repeat_yourself

4.2 SINGLE RESPONSIBILITY

https://en.wikipedia.org/wiki/Single_responsibility_principle

4.3 CODE COMPLEXITY

Function name – sufficiently descriptive English name. It should clearly describes what function does. After reading name once it should be 90% clear what it does. It is recommended not to unnecessarily increase number of words. If it's not possible to find right words there should be comment describing it.

Measures to ensure better code readability:

1. Use of switch should be limited, it is preferred to use virtual and abstract methods

2. It is preferred to use functions instead of nested loops

3. If will start with condition that we assume will be triggered more often, else should contain more rare cases. In case that both cases are equal (e.g.: moving left or right then order doesn't matter)