# GlobalX Coding Assessment — Name Sorter

# Technical Specification Document

### 2nd May 2019

### Simon Le Serve

# Introduction

This document describes the code implemented here as a solution for the GlobalX recruiting assessment.

## Name Sorting Problem

The problem to be solved is to sort a list of names by Family Name and then any of one to three possible Given Names. Input to the program is a list of full names, each of the form:

[first given name] [second given name] [third given name] [family name]

The second and third given names are optional.

## The Solution

The solution provided solves the problem by reordering the parts of the full name so that a string comparison can be invoked on the full name where the family name appears before the given names. This is the simplest solution to the problem.

## Problems and Limitations

One problem with the program rests in the fact that the list of family and given names are *space* separated, but some family names have a space such as “von Neumann”. This semantic feature cannot be represented in the input format chosen. We ignore this limitation.

## Sorting Algorithm

Two sorting methods are available in the program. The default sorting method uses the .NET array sort method. An additional method is included with uses the classic Quick Sort algorithm. Code for this Quick Sort is located in the static generic class QuickSort<T>.

## Execution Options

The default sorting algorithm used is the .NET Array Sort algorithm. However an additional custom QuickSort algorithm is supplied in a static class called QuickSort. The QuickSort algorithm can be invoked by adding an additional command option “quicksort” as a second optional command argument.

Usage

name-sorter <filename> [<sorting method>]

Optional Algorithms

quicksort Quick sort algorithm

arraysort .NET array sort algorithm (default)

## Output

The output is written to the console and also to a file named sorted-names-list.txt in the execution directory.

## Input

The input is supplied as a file containing a list of full-names. Each full name must consist of 1 to 3 given names followed by a single family name.

For example:

Janet Parsons

Vaughn Lewis

Adonis Julius Archer

Shelby Nathan Yoder

Marin Alvarez

London Lindsey

Beau Tristan Bentley

Leo Gardner

Hunter Uriah Mathew Clarke

Mikayla Lopez

Frankie Conner Ritter

## Solution Design

The solution is named GlobalX.Coding.Assessment.

The solution consists of 7 classes including the main Program class:

Program

OrderedName

NameSorter

ArrayNameSorter

QuickNameSorter

QuickSort

NameSorterFactory

Plus an Enum:

SortMethod

The main Program class simply validates command arguments and prints usage info when invalid. If the command line is valid, the Program class creates either the ArrayNameSorter class or the QuickNameSorter class using the NameSorterFactory passing in the chosen, or default, sort method from the command line arguments.

The OrderedName class implements the IComparable interface and the CompareTo method for this interface allows sorting methods to invoke a comparison based on a reordering of FullName, so that Family Name is the first name to be ordered by, then first given name, second given name etc. This reordering of the full name is exposed as a method called ToOrderedSpelling and this ordered spelling of the full name is what makes it possible to sort an array of Name objects based on the stipulated sorting criteria.

The Enum SortMethod represents which sort method has been selected: either ArraySort or QuickSort. The NameSorterFactory is a static class with one method ‘Create’ that returns an instance of a NameSort type dependant upon the SortMethod Enum provided as an argument.

The Program and QuickSort classes are located in the root project directory.

The OrderedName model class is located in a subfolder Models.

The rest of the types are located in a subfolder NameSorter.

## Tests

There are 6 tests targeting the comparison operation and constructor of the Name class, plus a test of the correctness of the classic quick sort algorithm in the generic class QuickSort<T>.

1. Name\_Construction\_AurgumentException\_1\_Name
2. Name\_Construction\_AurgumentException\_5\_Names
3. Name\_CompareTo\_2\_And\_3\_Names
4. Name\_CompareTo\_Different\_Given\_Names
5. Name\_CompareTo\_Multiple\_Spaces
6. QuickSortTest

The Name constructor should throw an ArgumentException when the full name consists of less than 2 or more than 4 names. The constructor should also deal with names that have multiple spaces by replacing multiple spaces with a single space.

The CompareTo operation on the Name class should return -1 when the first operand alphabetically precedes the second, 0 when they are equal and 1 when the second operand alphabetically precedes the first.

Finally the QuickSortTest tests the cogency of the Quick Sort algorithm by testing that the sorting behavior is correct on an array of integers.

## Public GIT Repository

The GIT repository for the solution can be found at

<https://github.com/CognitiveFeedback/globalx.coding.assessment>

or by simply cloning the repository using

<https://github.com/CognitiveFeedback/globalx.coding.assessment.git>

## Build Pipeline in Travis CI

The Travis CI build pipeline can be found at

<https://travis-ci.org/CognitiveFeedback>

The build pipeline executes unit tests and will fail if not all unit tests pass.

See figure 1 for the build history page in Travis CI.

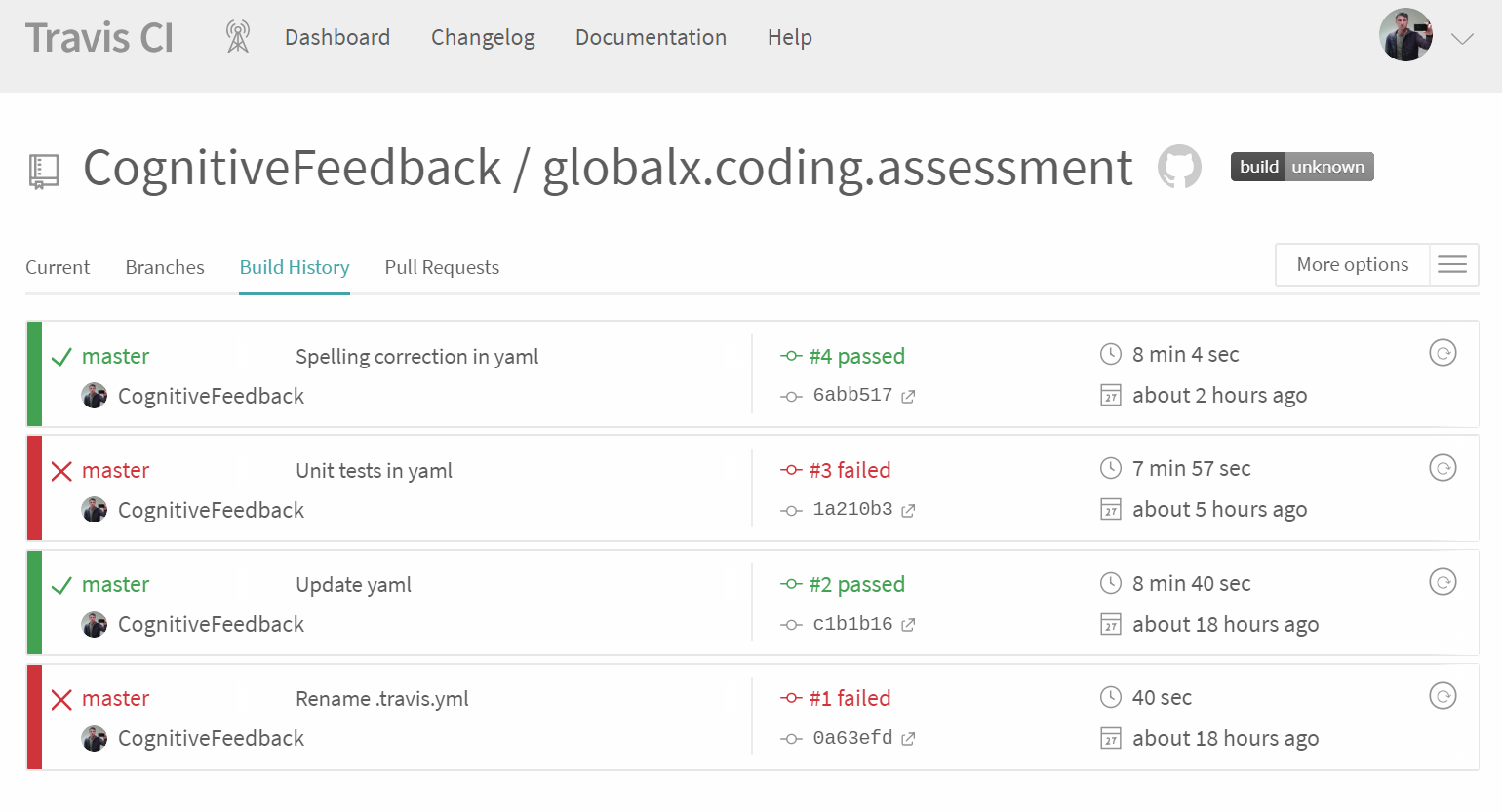


Figure 1: Build history in Travis CI

See figure 2 for an example build output from Travis CI showing test run summary.

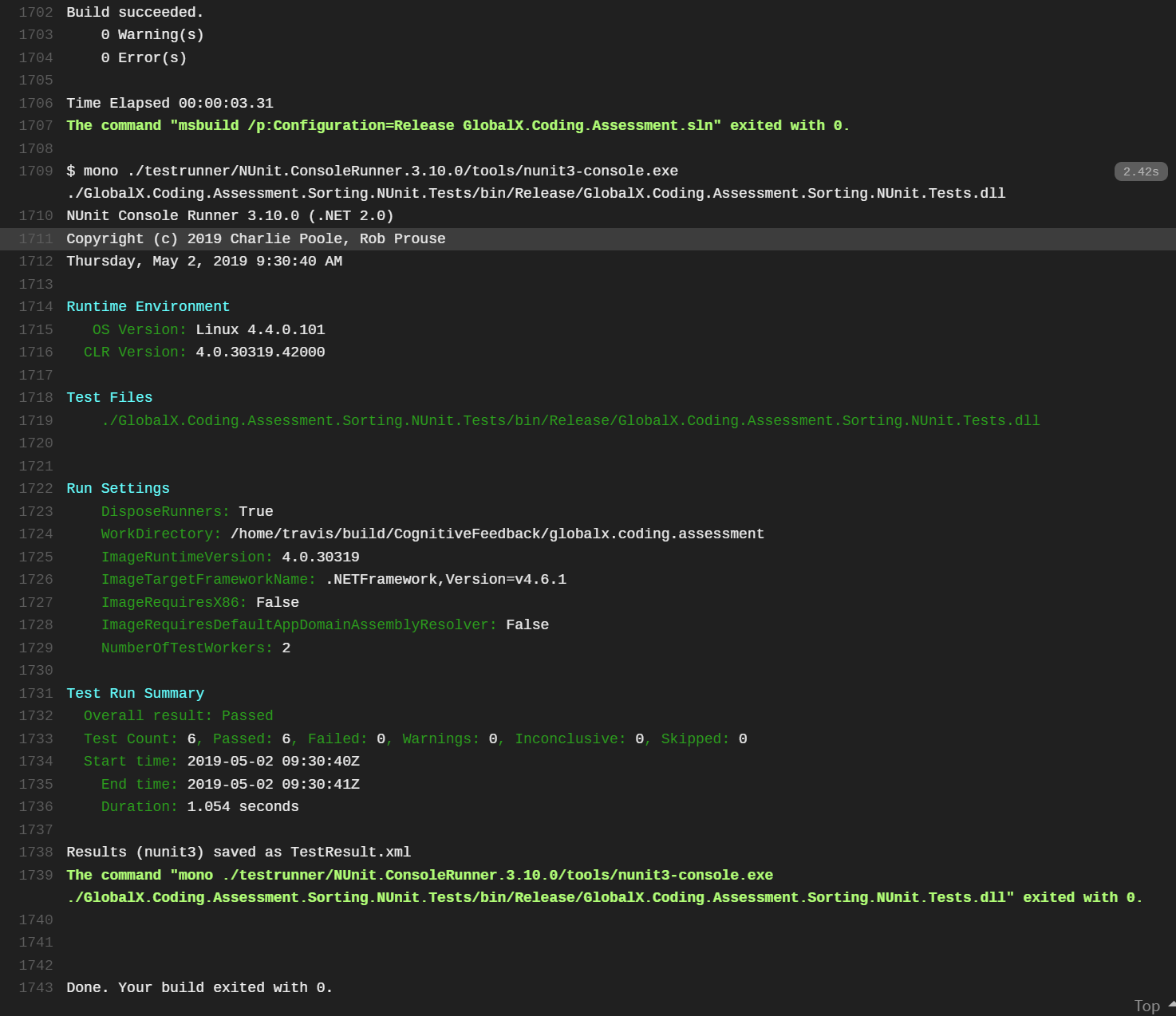


Figure 2: Example build output from Travis CI showing build results and successful test execution