A demo project for race simulation

# Introduction

The application needs to use the following technologies:

* Microsoft .Net (4.7.2)
* C#
* WPF
* Linq
* XML, Data structure design
* …

# Description

The application handles a race from the initialization (selection of participants out of a list of provided persons) to the computation of the final classification.

## Glossary

|  |  |
| --- | --- |
| **Persons / Athletes** | A physical person is susceptible to participating in one race of a given discipline. It is defined by an identifier and a name. |
| **Competitor** | It is a person/athlete participating in a race. It’s defined as it usually as soon as it has been added to a Start List. |
| **BIB** | It is a numerical value identifying the competitor within the race. It isn’t related to the person's identifier. |
| **IRM** | An event that happens within a race that influences the result of a competitor. |
| **DNS** | IRM meaning that the competitor didn’t start the race. It wasn’t present at the start line. |
| **DNF** | IRM meaning the competitor didn’t end the race; this IRM usually follows an abandon. |
| **DSQ** | IRM meaning the competitor has been received a jury notification requiring from him to end its race. |
| **Day Time** | The day time (with precision depending on the timing device) associated with a given race event (Start, Intermediate split, Finish…)  This is supplied as DateTime .Net object |
| **Net Time** | Is a difference between two day time rounded according to the sport’s rules |
| **Mass Start** | All competitors start at the same time (eg. Swimming; athletic’s 100m…) |
| **Individual Start** | Each competitor has its own start time (eg. Alpine Skiing…) |

# Provided parts

**Initial data**; an xml-formatted list of persons

**Race simulator**; an engine that once initialized simulates a race and its events. More information is given inside the dedicated section.

## Initial data

The XML document contains a variable list of persons; each person has been assigned a unique identifier and a name. Those are the single information provided as input for the competitors selection.

## Race Simulator

This Class provided within the SwissTiming.Timing.Simulator.dll assembly has to be initialized with a list of Competitor instances. Those competitors will then compete for a race once the simulator is started. The Race Simulator offers three .Net events to monitor the race’s progress;

|  |  |
| --- | --- |
| **RaceStarted** | Raised when the simulator starts the Race; depending on the start mode one or many competitors might be affected.  This event is timed using a day time. |
| **CompetitorChanged** | Raised when the simulator raises an IRM for a given competitor. |
| **RaceCompleted** | Raised each time a competitor ends its race by triggering the finish contacts.  This event is timed using a day time. |

The sample use of the RaceSimulator is provided with the assembly.

## Rules

#### Start List ordering

The start list order defines the order of start from the competitors. The competitor’s with the smallest identifier starts the first, the competitors with the highest value starts as last.

At most 10 competitors can compete within the same race.

Although in Mass-Start mode all competitors start at together the start list still needs to be sorted accordingly.

#### Net Time calculation

Net time is the difference between the finish event’s day time and the start event’s day time. The difference is rounded as follows.

* Mathematically rounded to the nearest 1/10 second, where;  
    
   *9.87 => 9.9 12.149 => 12.1*

#### Final Ranking

The winner of the race is the competitor with the smallest net time value. It’ll be ranked first.

Competitors having received an IRM will be listed without any rank at the end of the result list in the following order.

DNF, DSQ and then DNS

Tied are divided using their start list position; first in the start list, winner of the tie.

#### Post-competition IRM

This feature only applies if the software is supporting the operation.

A given competitor having completed the race might be disqualified after the end of the race due to an infraction to the sport’s rules.

In this situation, the present competitor will be marked as DSQ, ranked as last and its performance will be erased. The full ranking is then moved by one to the top of the race to replace the missing rank.

# Requirements

You are required to implement the following functionalities:

* Using a-WPF user interface in C# (The MVVM pattern is preferred)
* Create the Start List with a selection of the provided people list.
* Initiate the RaceSimulator with the selected list of competitors.
* Start the RaceSimulator in the Mass-Start mode and handle race events
* Calculate using the defined rules the Net Time for competitors having ended the race.
* At the end of the race display the competitors’ final ranking within the result list.
* Exportation of the race’s data (Start List, Result List) to an XML file
* Use WPF ItemTemplates to customize the presentation from People, Competitors and Results.
* Use different colors to differentiate competitors in the Result List based on their IRM (DNS displayed in orange, DNF in yellow…).
* Populate the result list while the race is running. Not only when all competitors have finished the race. This provides a temporary live-updated ranking of competitors.
* [Low-priority requirement] Use the Individual Start and differentiates competitors having started their race from the other within the start list.
* [Low-priority requirement] Provide a user interface to edit the competition’s results. A sample change can be a disqualification from a competitor. In this case, the result must be updated.
* [Low-priority requirement] Add a multilingual user interface (i18n) for English and your favorite language.



Controls to be present on the User Interface  
The layout is a suggestion

# Deliverable

It is required to deliver the following files/solutions/documentation within a single ZIP archive.

* The Visual Studio Solution implementing the required functionalities
* Documentation describing the use of the application like a user guide
* A technical documentation explaining
  + The technology and version used for the implementation
  + How the solution has been implemented
  + The list of required libraries (aside .Net standards) such as custom framework
* An executable version of the application

# Attachments

* SwissTiming.Timing.Simulator assembly
* SwissTiming.Timing.Simulator developer documentation
* Sample C# program that uses the Simulator.

# Evaluation

You need to implement the best and optimized approach for **requirements/solving the problem (design)** as well as its **development (coding)** and **testing (**preferablyTDD**)**. It is also important to document the code following code documentation standards.