

École Polytechnique de l'Université de Tours  
64, Avenue Jean Portalis  
37200 TOURS, FRANCE  
Tél. +33 (0)2 47 36 14 14  
[www.polytech.univ-tours.fr](http://www.polytech.univ-tours.fr)

**Département Informatique**  
**4e année**  
**2016 - 2017**

# Developer & User Guide

# Contents

## Developer guide

Installation.....	2
Structure.....	2
Testing .....	3

## User guide

Installation.....	4
Accidents happens .....	4
Getting started .....	5
MatchingExecutor methods.....	5
Packages.....	5
Algorithms.....	6
Existing algorithms .....	6
Creating correspondence algorithms .....	6
Parameters.....	7
Parsers .....	7
Existing parsers.....	7
CSV format .....	7
EXT format .....	8
XML format .....	8
Command lines.....	9
Command line Application .....	9
Create commands .....	9

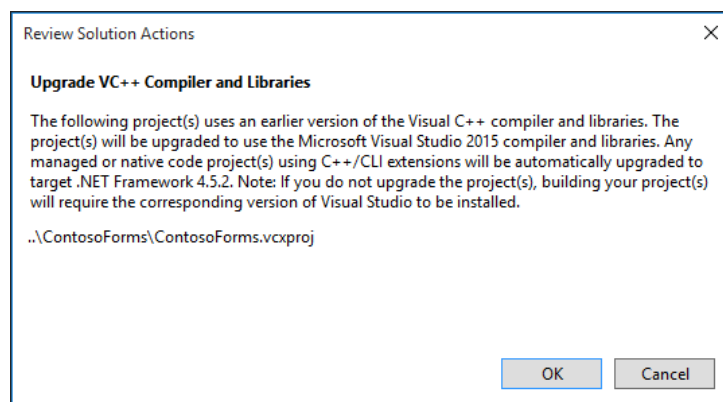
This guide is divided into two parts, the first one is addressed to the developers who will continue working on this project, the second one is about how to use the library.

# Developer guide

## Installation

The project was developed using [Visual Studio 2015](#), but it can be opened generally with any version of VS, some problems may arise depending on the version you're using but most of them are discussed below on how to fix them.

When you open the solution in VS a warning might show up “**Upgrade VC++ Compiler and libraries**” if pressing OK didn't fix the problem then it is fairly simple to fix manually:



**Figure 1 – upgrade dialog**

You need to go to Properties of each project in the solution then follow this:

Configuration properties > General

You need to change “Platform Toolset” to whatever version of VS you're using.

## Structure

When you successfully open the project you'll see at the right panel in Solution Explorer 3 projects inside the solution, each project is responsible for a different aspect:

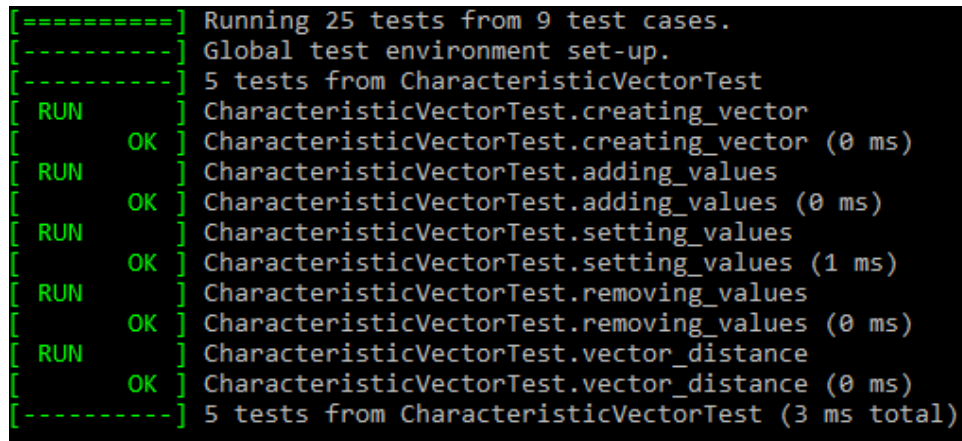
- **MatchingLibrary:** in this project where you can find the library classes, it's not a Console application so there is no executable and instead when building it produces a .lib file.
- **MatchingToolBox:** This project uses the library and as opposed to the first one, it's a console application so it has an executable.
- **MatchingToolBoxTest:** this project is used only to test the library; it uses the google test framework which makes the job very easy to perform unit tests.

- To switch between the projects, right click on a project and press “Set as StartUp Project”

# Testing

The project MatchingToolBoxTest uses Google test framework which provide a comprehensive support for running unit tests.

The framework requires an executable that's why the project is a console application, it print its output in the console and even makes them nice and colorful.

A screenshot of a terminal window showing the output of a Google Test run. The output is color-coded: green for 'RUN' and 'OK', and yellow for the test names. It shows 25 tests from 9 test cases. The tests are grouped by test case, with a summary line for each group. The tests are: CharacteristicVectorTest.creating\_vector (0 ms), CharacteristicVectorTest.adding\_values (0 ms), CharacteristicVectorTest.setting\_values (1 ms), and CharacteristicVectorTest.removing\_values (0 ms). The total time for the tests is 3 ms.

```
[=====] Running 25 tests from 9 test cases.
[-----] Global test environment set-up.
[-----] 5 tests from CharacteristicVectorTest
[ RUN    ] CharacteristicVectorTest.creating_vector
[       OK ] CharacteristicVectorTest.creating_vector (0 ms)
[ RUN    ] CharacteristicVectorTest.adding_values
[       OK ] CharacteristicVectorTest.adding_values (0 ms)
[ RUN    ] CharacteristicVectorTest.setting_values
[       OK ] CharacteristicVectorTest.setting_values (1 ms)
[ RUN    ] CharacteristicVectorTest.removing_values
[       OK ] CharacteristicVectorTest.removing_values (0 ms)
[ RUN    ] CharacteristicVectorTest.vector_distance
[       OK ] CharacteristicVectorTest.vector_distance (0 ms)
[-----] 5 tests from CharacteristicVectorTest (3 ms total)
```

Figure 2 – Google test output

All you need to test a class is:

- Include gtest header

```
#include "gtest/gtest.h"
```

- Fill your TEST macro, example:

```
TEST(NumericTest, setting_getting_value) {
    model::Numeric n(5);
    ASSERT_EQ(n.getValue(), 5);

    n.setValue(4);
    ASSERT_EQ(n.getValue(), 4);
}
```

- In order to run your tests, you need to execute this code in the main:

```
int main(int argc, char *argv[])
{
    ::testing::InitGoogleTest(&argc, argv);
    return RUN_ALL_TESTS();
}
```

The project already contains test units of these packages:

- Model package
- Inout package
- CorrespondenceTools package
- Commandline package

For more information visit this link : <https://github.com/google/googletest>

# User guide

## Installation

Enclosed with this document 2 folders:

- include: contains the headers.
- lib: contains the library, debug version and a release version.

In order to use the Matching library, you need to go through 3 steps:

1 – Add an additional include directory in the project properties:

Configuration properties > C/C++ > General > Additional Include Directories

And then add the path to “include” folder of the library.

2 – Add an additional library directory in the project properties:

Configuration properties > Linker > General > Additional Library Directories

And then add the path to “lib” folder of the library.

3 – Add dependency:

Configuration properties > Linker > Input > Additional Dependencies

In **Debug** configuration add: MatchingLibrary\_d.lib;

In **Release** configuration add: MatchingLibrary.lib;

That's all to set up the library.

## Accidents happens

An error might occur if the Runtime Library in your project is using dynamic value, to solve this:

Go to your project properties and change the value of Runtime Library, you'll find it in:

Configuration properties > C/C++ > Code Generation > Runtime Library

In Debug configuration change it to: Multi-threaded Debug

In Release configuration change it to: Multi-threaded

Compile the project again and let's hope it works!

# Getting started

To match between 2 files, you'll need to use MatchingExecutor class and a Correspondence implementation like Levenshtein, DTW, CDP...

First of all include these 2 headers: "CorrespondenceTools.h" and "Support.h"

```
tools::MatchingExecutor executor("target_vectors_1.csv", "reference_vectors_2.csv",
new support::DTWCorrespondence());
executor.execute();
```

The two files need to be in CSV format and contains sequence of vectors, we'll talk more about format and types of sequences below.

The execute() method will go through reading the files and extracting the sequences, it'll also match between them using the provided method in the case it's DTW. The result of course will be stored by default in the same directory of the first file.

## MatchingExecutor methods

You can change the directory where the result should be stored using the method:

```
void setResultDirectory(std::string dir)
```

Sometimes you'll need to customize the parameters of the algorithm:

```
void setParameters(Parameters* par)
```

There are 3 types of sequence a file can have (CHARACTER, NUMERIC, VECTOR):

```
void setType(inout::SEQUENCE_TYPE type)
```

There 3 parsers you can choose from are CSVParser, EXTParser, XMLParser (Default is CSVParser):

```
void setParser(inout::SequenceParser* p)
```

## Packages

The classes are in separate packages/namespaces and in order to use any of them you need to know in which package they belong.

The first package is called "model" and it contains these classes:

- Element, Numeric, Character, CharacteristicVector, Sequence

Package "tools" which has the tools to create matching algorithms contains:

- Correspondence, Parameters, MatchingExecutor, ResultCorrespondence

Package "inout" is used to read and parse sequence files or parameters file and it contains:

- SequenceParser, XMLParser, CSVParser, EXTParser, ParamParser

Package "commandline" to read commandlines and execute matching algorithms based on the arguments and it contains:

- CommandLine, CommandLineApplication

# Algorithms

## Existing algorithms

There are 7 algorithms in the library that can be used to match between sequences and can be parameterized:

- Levenshtein
  - Parameters: LEVENSHTein\_ADD\_COST, LEVENSHTein\_DEL\_COST, LEVENSHTein\_TRANS\_COST
- Longest Common SubSequences (LCS)
- Dynamic Time Warping (DTW)
- Minimum Variance Matching (MVM)
  - Parameters: MVM\_ELASTICITY
- Continuous Dynamic Programming (CDP)
  - Parameters: CDP\_THRESHOLD
- Flexible Sequence Matching (FSM)
  - Parameters: FSM\_WEIGHT, FSM\_SMALL\_SKIP\_COST, FSM\_SKIP\_COST, FSM\_TYPE\_RESULT, FSM\_NUMBER\_ELEMENTS\_BY\_LINE, FSM\_FIRST\_COL\_FOR\_RESULT, FSM\_DEFAULT\_SKIP\_COST, FSM\_STANDARD\_DEVIATION\_NUMBER.
- Exemplary Sequence Cardinality (ESC)

Some of the algorithms use default values for the parameters and they are modifiable using the Parameters class.

## Creating correspondence algorithms

To create your own matching algorithm, the library provides a set of tools to help you do that.

First of all, your class should extend Correspondence class which is an abstract class, then implement this method:

```
vector<ResultCorrespondence> *match(model::Sequence *sTarget, model::Sequence *sRef)
```

The function takes two sequences and it returns a vector of results, a result is a structure that defined by the distance between the sequences and also indexes of the matches between them.

```
typedef struct {  
    vector<int> *correspondanceT1;  
    vector<int> *correspondanceT2;  
    float distance;  
} ResultCorrespondence;
```

- distance: represents the distance between the two sequences and its value differs depending on the algorithm.
- correspondanceT1: contains the index of target which match with reference.
- correspondanceT2: contains the index of reference which match with target.

It should be in mind the sequences that match() method take as parameters can be sequences of:

- Character
- Numeric
- CharacteristicVector

There is a default implementation of how the result are presented using the format function but you can override it to change how you want to show the results:

```
std::string format(model::Sequence *sTarget, model::Sequence *sRef,
ResultCorrespondence * result)
```

## Parameters

Using the class Parameters, you can add or modify parameters that your correspondence algorithm or the other already existing algorithms might use.

The constraint of this class is the parameters need to be a numerical value that means it takes only float, integer values.

```
void putValue(std::string key, float value);
float getValue(std::string key);
```

The correspondence class accepts Parameters in constructor or to be modified using a setter.

## Parsers

### Existing parsers

There are 3 parsers that can be used to read and extract sequences from files depending on the format of the sequences.

Each parser is in charge of parsing a certain format:

- CSVParser > CSV format
- EXTParser > EXT format
- XMLParser > XML format

### CSV format

Example:

```
0.25835,0.025862,0.034483,0.93966,0.017241,0.1,0.051724,0
0.26771,0.034483,0.025862,0.93966,0.025862,0.1,0.047414,0
0.28583,0.043103,0.017241,0.93966,0.034483,0.1,0.043103,0
0.30078,0.051724,0.017241,0.93103,0.043103,0.1,0.047414,0
0.31479,0.11207,0.017241,0.31034,0.66379,0.2,0.33621,0
0.3252,0.11207,0.017241,0.31034,0.66379,0.2,0.33621,0
```

Numbers are separated by a comma and each line represents either a vector or sequence.

If you set the type of parser to “Numeric” or “Character”, it will parse each line as a separate sequence.

If you set the type of parser to “Vector” then each line is a CharacteristicVector and the whole file is a sequence.

To change the type of a parser, call this method:

```
void setType(inout::SEQUENCE_TYPE t)
```



## EXT format

```
1 2 8 8
5 1 1 2
1 3 8 8
```

Numbers are separated by double space ' '.

If you set the type of parser to "Numeric" or "Character", it will parse each line as a separate sequence.

If you set the type of parser to "Vector" then each line is a CharacteristicVector and the whole file is a sequence.

## XML format

```
<Sequences></Sequences>
<Type id="0 | 1 | 2" />
<Sequence idS="string">
  <!-- if type id = 0 -->
  <Element id="string" value="character" />
  <!-- if type id = 1 -->
  <Element id="string" value="float" />
  <!-- if type id = 2 -->
  <Element id="string">
    <VectorElement value="float" />
    ... <!-- more vector elements -->
  </Element>
  ... <!-- more elements of the same type -->
</Sequence>
... <!-- more sequences -->
```

The file should start with an empty tag of sequences

The type tag represents the type of sequences in the file:

- 0 > Character , 1 > Numeric , 2 > Vector

Each sequence tag represents one sequence with an ID.

Inside the sequence tag you can write as many Element tags as you'd like, you need to make sure that all of their values are of the same type.

**PS:** the word inside the value represents the type of value it should have.

# Command lines

## Command line Application

The class `CommandLineApplication` is able to read this command to match between sequences of two files:

```
MatchingToolBox.exe [--help |  
-sequences target reference -method {lvn | lcs | dtw | mvm | cdp | fsm | esc}  
[ --costs <addCost> <delCost> <transCost> ]  
[ -param <parameters.xml>]  
[ -result </result/path> ]  
[ -parser {csv|ext|xml} ]  
[ -type {character|numeric|vector} ] ]
```

- `--help`: prints the arguments of the command and their description.
- `-sequences`: Specifies target and a reference file.
- `-method`: Defines the algorithm used in sequence matching and it should be after `-sequences`.
- `--costs`: costs that are used with Levenshtein algorithm.
- `-param`: Specifies the parameters used for the algorithm.
- `-result`: Defines the directory where the output will be saved.
- `-parser`: Specifies the parser that would be used to parse the files.
- `-type`: Specifies the type of sequence in the file.

## Create commands

To create your own command, you'll need to extend `CommandLine` class which offer a set of tools that can help you read better the command and interpret them easily.

Tools that the class offers:

- You can assign expected arguments and values using `expectArgument()` and `expectValue()` methods.
- It detects entered arguments and their values in a command line.

Implement `run()` method where you read the command, to get the values of an argument you can use :

```
getValue(argument, position);
```

Of course in the `run()` method is where you should execute the command.