SimMetrics library v 1.5 for .NET 2.0

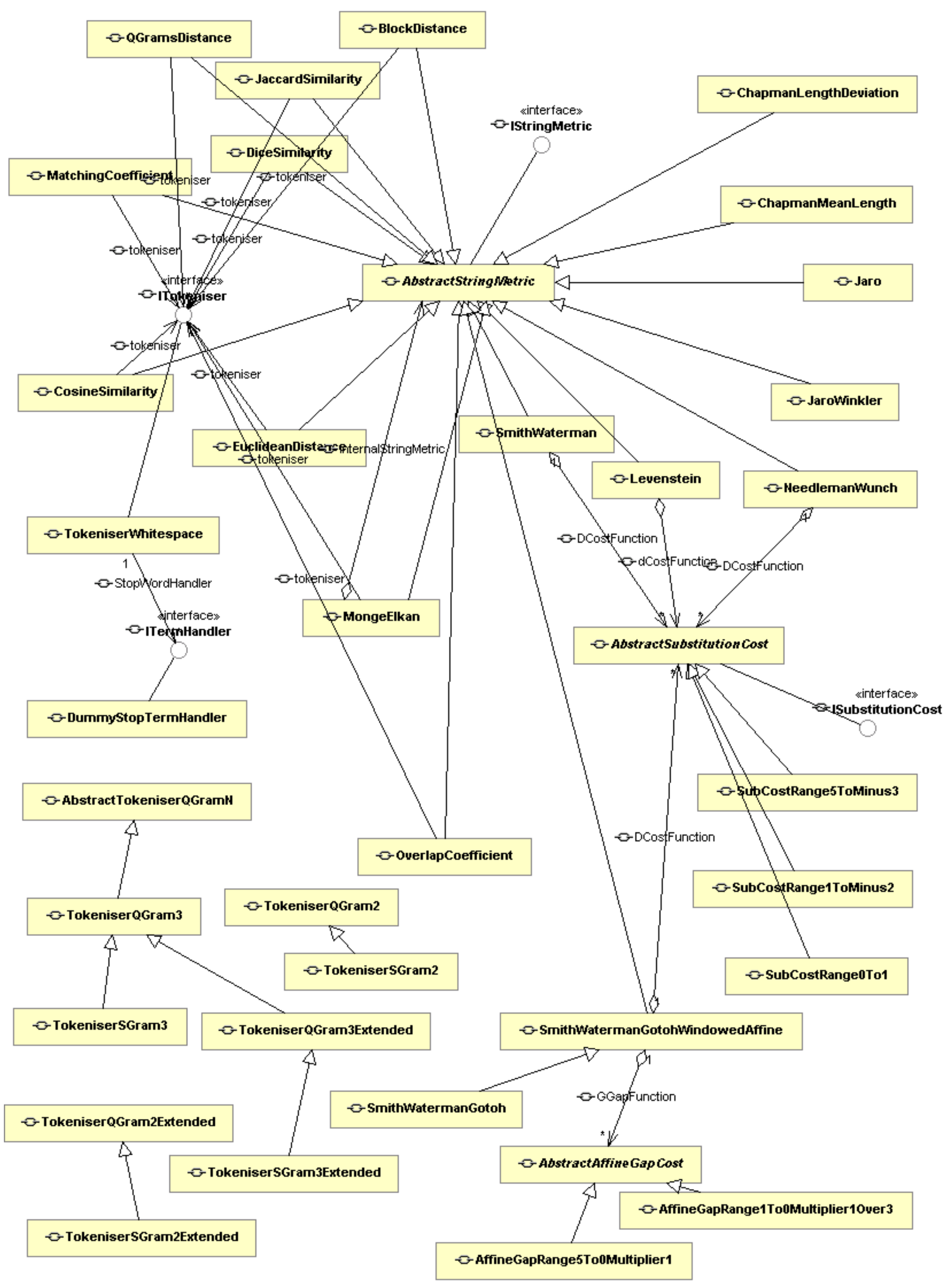
System and Reference Manual, created on 24/09/2006.

Table of Contents

SimMetrics library v 1.5 for .NET 2.0

1 Symbol Reference

Overall Classes and Interfaces



1.1 SimMetricsApi

This is namespace SimMetricsApi.

Classes

Class	Description
AbstractAffineGapCost (see page 4)	abstract class used as a base for all affine gap classes
AbstractStringMetric (see page 5)	base class which all metrics inherit from.
AbstractSubstitutionCost (see page 8)	AbstractSubstitutionCost implements a abstract class for substiution costs
AbstractTokeniserQGramN (see page 9)	Implements a QGram Tokeniser to cope with all gram sizes.

Interfaces

Interface	Description
IAffineGapCost (see page 12)	defines an Interface for AffineGapCost functions to be interchanged
IStringMetric (see page 13)	implements an interface for the string metrics
ISubstitutionCost (see page 15)	is an interface for a cost function d(i,j).
ITermHandler (see page 16)	defines an interface for stop word handlers.
ITokeniser (see page 17)	InterfaceTokeniser interface for a Tokeniser class.

1.1.1 AbstractAffineGapCost

abstract class used as a base for all affine gap classes

Class Hierarchy

```
SimMetricsApi.IAffineGapCost
    AbstractAffineGapCost
```

```
[Serializable]
public abstract class AbstractAffineGapCost : IAffineGapCost;
```

Methods

Method	Description
GetCost (see page 4)	get cost between characters.

Properties

Property	Description
MaxCost (see page 4)	returns the maximum possible cost.
MinCost (see page 4)	returns the minimum possible cost.
ShortDescriptionString (see page 4)	returns the name of the cost function.

AbstractAffineGapCost.MaxCost

returns the maximum possible cost.

```
public abstract double MaxCost;
```

AbstractAffineGapCost.MinCost

returns the minimum possible cost.

```
public abstract double MinCost;
```

AbstractAffineGapCost.ShortDescriptionString

returns the name of the cost function.

```
public abstract string ShortDescriptionString;
```

AbstractAffineGapCost.GetCost

get cost between characters.

```
public abstract double GetCost(string textToGap, int stringIndexStartGap, int stringIndexEndGap);
```

Returns

the cost of a Gap G

1.1.2 AbstractStringMetric

base class which all metrics inherit from.

Remarks

This class implemented a few basic methods and then leaves the others to be implemented by the similarity metric itself.

Class Hierarchy

```
SimMetricsApi.IStringMetric
    AbstractStringMetric
```

```
[Serializable]
public abstract class AbstractStringMetric : IStringMetric;
```

Methods

Method	Description
BatchCompareSet (see page 5)	does a batch comparison of the set of strings with the given comparator string returning an array of results equal in length to the size of the given set of strings to test.
BatchCompareSets (see page 6)	does a batch comparison of one set of strings against another set of strings returning an array of results equal in length to the minimum size of the given sets of strings to test.
GetSimilarity (see page 6)	gets the similarity measure of the metric for the given strings.
GetSimilarityExplained (see page 6)	gets a div class xhtml similarity explaining the operation of the metric.
GetSimilarityTimingActual (see page 6)	gets the actual time in milliseconds it takes to perform a similarity timing. This call takes as long as the similarity metric to perform so should not be done in normal circumstances.
GetSimilarityTimingEstimated (see page 6)	gets the estimated time in milliseconds it takes to perform a similarity timing.
GetUnnormalisedSimilarity (see page 7)	gets the un-normalised similarity measure of the metric for the given strings.

Properties

Property	Description
LongDescriptionString (see page 5)	reports the metric type.
ShortDescriptionString (see page 5)	reports the metric type.

AbstractStringMetric.LongDescriptionString

reports the metric type.

```
public abstract string LongDescriptionString;
```

AbstractStringMetric.ShortDescriptionString

reports the metric type.

```
public abstract string ShortDescriptionString;
```

AbstractStringMetric.BatchCompareSet

does a batch comparison of the set of strings with the given comparator string returning an array of results equal in length to the size of the given set of strings to test.

```
public double[] BatchCompareSet(string[] setRenamed, string comparator);
```

Returns

an array of results equal in length to the size of the given set of strings to test.

Body Source

```
public double[] BatchCompareSet(string[] setRenamed, string comparator) {
    if ((setRenamed != null) && (comparator != null)) {
        double[] results = new double[setRenamed.Length];
        for (int strNum = 0; strNum < setRenamed.Length; strNum++) {
            results[strNum] = GetSimilarity(setRenamed[strNum], comparator);
        }
        return results;
    }
    return null;
}
```

AbstractStringMetric.BatchCompareSets

does a batch comparison of one set of strings against another set of strings returning an array of results equal in length to the minimum size of the given sets of strings to test.

```
public double[] BatchCompareSets(string[] firstSet, string[] secondSet);
```

Returns

an array of results equal in length to the minimum size of the given sets of strings to test.

Body Source

```
public double[] BatchCompareSets(string[] firstSet, string[] secondSet) {
    if ((firstSet != null) && (secondSet != null)) {
        double[] results;
        if (firstSet.Length <= secondSet.Length) {
            results = new double[firstSet.Length];
        }
        else {
            results = new double[secondSet.Length];
        }
        for (int strNum = 0; strNum < results.Length; strNum++) {
            results[strNum] = GetSimilarity(firstSet[strNum], secondSet[strNum]);
        }
        return results;
    }
    return null;
}
```

AbstractStringMetric.GetSimilarity

gets the similarity measure of the metric for the given strings.

```
public abstract double GetSimilarity(string firstWord, string secondWord);
```

Returns

implemented version will return score between 0 and 1

AbstractStringMetric.GetSimilarityExplained

gets a div class xhtml similarity explaining the operation of the metric.

```
public abstract string GetSimilarityExplained(string firstWord, string secondWord);
```

Returns

a div class html section detailing the metric operation.

AbstractStringMetric.GetSimilarityTimingActual

gets the actual time in milliseconds it takes to perform a similarity timing. This call takes as long as the similarity metric to perform so should not be done in normal circumstances.

```
public long GetSimilarityTimingActual(string firstWord, string secondWord);
```

Returns

the actual time in milliseconds taken to perform the similarity measure

Body Source

```
public long GetSimilarityTimingActual(string firstWord, string secondWord) {
    long timeBefore = (DateTime.Now.Ticks - 621355968000000000) / 10000;
    GetSimilarity(firstWord, secondWord);
    long timeAfter = (DateTime.Now.Ticks - 621355968000000000) / 10000;
    return timeAfter - timeBefore;
}
```

AbstractStringMetric.GetSimilarityTimingEstimated

gets the estimated time in milliseconds it takes to perform a similarity timing.

```
public abstract double GetSimilarityTimingEstimated(string firstWord, string secondWord);
```

Returns

the estimated time in milliseconds taken to perform the similarity measure

AbstractStringMetric.GetUnnormalisedSimilarity

gets the un-normalised similarity measure of the metric for the given strings.

```
public abstract double GetUnnormalisedSimilarity(string firstWord, string secondWord);
```

Returns

returns the score of the similarity measure (un-normalised)

1.1.3 AbstractSubstitutionCost

AbstractSubstitutionCost implements a abstract class for substiution costs

Class Hierarchy

```
SimMetricsApi.ISubstitutionCost
    AbstractSubstitutionCost
```

```
[Serializable]
public abstract class AbstractSubstitutionCost : ISubstitutionCost;
```

Methods

Method	Description
GetCost (see page 8)	get cost between characters.

Properties

Property	Description
MaxCost (see page 8)	returns the maximum possible cost.
MinCost (see page 8)	returns the minimum possible cost.
ShortDescriptionString (see page 8)	returns the name of the cost function.

AbstractSubstitutionCost.MaxCost

returns the maximum possible cost.

```
public abstract double MaxCost;
```

AbstractSubstitutionCost.MinCost

returns the minimum possible cost.

```
public abstract double MinCost;
```

AbstractSubstitutionCost.ShortDescriptionString

returns the name of the cost function.

```
public abstract string ShortDescriptionString;
```

AbstractSubstitutionCost.GetCost

get cost between characters.

```
public abstract double GetCost(string firstWord, int firstWordIndex, string secondWord, int
secondWordIndex);
```

1.1.4 AbstractTokeniserQGramN

Implements a QGram Tokeniser to cope with all gram sizes.

Remarks

The cci value determines at what level the skip characters are gathered. This is a variation of the normal QGram analysis when character pairs are created having skipped characters in the words.

Class Hierarchy

```
SimMetricsApi.ITokeniser
    AbstractTokeniserQGramN
```

```
[Serializable]
public abstract class AbstractTokeniserQGramN : ITokeniser;
```

Methods

Method	Description
Tokenize (see page 10)	Return tokenized version of a string.
Tokenize (see page 10)	full version of Tokenise which allows for different token lengths as well as the characterCombinationIndexValue error level as well.
TokenizeToSet (see page 11)	Return tokenized set of a string.

Properties

Property	Description
CharacterCombinationIndex (see page 10)	CCI - error level used for the sgram analysis.
Delimiters (see page 10)	displays the delimiters used - ie none.
QGramLength (see page 10)	length of the qgram tokens to create
ShortDescriptionString (see page 10)	displays the tokenisation method.
StopWordHandler (see page 10)	the stop word handler used.
SuppliedWord (see page 10)	supplied word
TokenUtilities (see page 10)	class containing token utilities

AbstractTokeniserQGramN.characterCombinationIndex

This is characterCombinationIndex, a member of class AbstractTokeniserQGramN.

```
public int characterCombinationIndex;
```

AbstractTokeniserQGramN.defaultEndPadCharacter

This is defaultEndPadCharacter, a member of class AbstractTokeniserQGramN.

```
public const string defaultEndPadCharacter = "#";
```

AbstractTokeniserQGramN.defaultStartPadCharacter

This is defaultStartPadCharacter, a member of class AbstractTokeniserQGramN.

```
public const string defaultStartPadCharacter = "?";
```

AbstractTokeniserQGramN.qGramLength

This is qGramLength, a member of class AbstractTokeniserQGramN.

```
public int qGramLength;
```

AbstractTokeniserQGramN.stopWordHandler

This is stopWordHandler, a member of class AbstractTokeniserQGramN.

```
public ITermHandler stopWordHandler;
```

AbstractTokeniserQGramN.suppliedWord

This is suppliedWord, a member of class AbstractTokeniserQGramN.

```
public string suppliedWord;
```

AbstractTokeniserQGramN.tokenUtilities

This is tokenUtilities, a member of class AbstractTokeniserQGramN.

```
public TokeniserUtilities<string> tokenUtilities;
```

AbstractTokeniserQGramN.CharacterCombinationIndex

CCI - error level used for the sgram analysis.

```
public int CharacterCombinationIndex;
```

AbstractTokeniserQGramN.Delimiters

displays the delimiters used - ie none.

```
public string Delimiters;
```

AbstractTokeniserQGramN.QGramLength

length of the qgram tokens to create

```
public int QGramLength;
```

AbstractTokeniserQGramN.ShortDescriptionString

displays the tokenisation method.

```
public abstract string ShortDescriptionString;
```

AbstractTokeniserQGramN.StopWordHandler

the stop word handler used.

```
public ITermHandler StopWordHandler;
```

AbstractTokeniserQGramN.SuppliedWord

supplied word

```
public string SuppliedWord;
```

AbstractTokeniserQGramN.TokenUtilities

class containing token utilities

```
public TokeniserUtilities<string> TokenUtilities;
```

AbstractTokeniserQGramN.Tokenize

Return tokenized version of a string.

```
public abstract Collection<string> Tokenize(string word);
```

Returns

tokenized version of a string

AbstractTokeniserQGramN.Tokenize

full version of Tokenise which allows for different token lengths as well as the characterCombinationIndexValue error level as well.

```
public Collection<string> Tokenize(string word, bool extended, int tokenLength, int
characterCombinationIndexValue);
```

Returns

collection of tokens

Body Source

```
public Collection<string> Tokenize(string word, bool extended, int tokenLength, int characterCombinationIndexValue) {
    if (!String.IsNullOrEmpty(word)) {
        SuppliedWord = word;
        Collection<string> anArray = new Collection<string>();
        int wordLength = word.Length;
        int maxValue = 0;
        if (tokenLength > 0) {
            maxValue = (tokenLength - 1);
        }
        StringBuilder testword = new StringBuilder(wordLength + (2 * maxValue));
        if (extended) {
            testword.Insert(0, defaultStartPadCharacter, maxValue);
        }
        testword.Append(word);
        if (extended) {
            testword.Insert(testword.Length, defaultEndPadCharacter, maxValue);
        }
```

```
            // normal n-gram keys characterCombinationIndex = 0
            string testWordOne = testword.ToString();
            int maxLoop;
            if (extended) {
                maxLoop = wordLength + maxValue;
            }
            else {
                maxLoop = wordLength - tokenLength + 1;
            }
            for (int i = 0; i < maxLoop; i++) {
                string testWord = testWordOne.Substring(i, tokenLength);
                if (!stopWordHandler.IsWord(testWord)) {
                    anArray.Add(testWord);
                }
            }

            if (characterCombinationIndexValue != 0) {
                // special characterCombinationIndex n-gram keys
                testWordOne = testword.ToString();
                maxLoop -= 1; // have to reduce by 1 as we are skipping a letter
                for (int i = 0; i < maxLoop; i++) {
                    string testWord = testWordOne.Substring(i, maxValue) + testWordOne.Substring(i + tokenLength, 1);
                    if (!stopWordHandler.IsWord(testWord)) {
                        if (!anArray.Contains(testWord)) {
                            anArray.Add(testWord);
                        }
                    }
                }
            }
            return anArray;
        }
        return null;
    }
```

AbstractTokeniserQGramN.TokenizeToSet

Return tokenized set of a string.

```
public Collection<string> TokenizeToSet(string word);
```

Returns

tokenized version of a string as a set

Body Source

```
public Collection<string> TokenizeToSet(string word) {
    if (!String.IsNullOrEmpty(word)) {
        SuppliedWord = word;
        return TokenUtilities.CreateSet(Tokenize(word));
    }
    return null;
}
```

1.1.5 IAffineGapCost

defines an Interface for AffineGapCost functions to be interchanged

Class Hierarchy

```
IAffineGapCost
    SimMetricsApi.AbstractAffineGapCost
```

```
public interface IAffineGapCost;
```

Methods

Method	Description
GetCost (see page 12)	get cost between characters.

Properties

Property	Description
MaxCost (see page 12)	returns the maximum possible cost.
MinCost (see page 12)	returns the minimum possible cost.
ShortDescriptionString (see page 12)	returns the name of the affine gap cost function.

IAffineGapCost.MaxCost

returns the maximum possible cost.

```
double MaxCost;
```

IAffineGapCost.MinCost

returns the minimum possible cost.

```
double MinCost;
```

IAffineGapCost.ShortDescriptionString

returns the name of the affine gap cost function.

```
string ShortDescriptionString;
```

IAffineGapCost.GetCost

get cost between characters.

```
double GetCost(string textToGap, int stringIndexStartGap, int stringIndexEndGap);
```

1.1.6 IStringMetric

implements an interface for the string metrics

Class Hierarchy

```
IStringMetric
    SimMetricsApi.AbstractStringMetric
```

```
public interface IStringMetric;
```

Methods

Method	Description
GetSimilarity (see page 13)	returns a similarity measure of the string comparison.
GetSimilarityExplained (see page 13)	gets a div class xhtml similarity explaining the operation of the metric.
GetSimilarityTimingActual (see page 13)	gets the actual time in milliseconds it takes to perform a similarity timing.
GetSimilarityTimingEstimated (see page 14)	gets the estimated time in milliseconds it takes to perform a similarity timing.
GetUnnormalisedSimilarity (see page 14)	gets the un-normalised similarity measure of the metric for the given strings.

Properties

Property	Description
LongDescriptionString (see page 13)	returns a long string of the string metric description.
ShortDescriptionString (see page 13)	returns a string of the string metric name.

IStringMetric.LongDescriptionString

returns a long string of the string metric description.

```
string LongDescriptionString;
```

IStringMetric.ShortDescriptionString

returns a string of the string metric name.

```
string ShortDescriptionString;
```

IStringMetric.GetSimilarity

returns a similarity measure of the string comparison.

```
double GetSimilarity(string firstWord, string secondWord);
```

Returns

a double between zero to one (zero = no similarity, one = matching strings)

IStringMetric.GetSimilarityExplained

gets a div class xhtml similarity explaining the operation of the metric.

```
string GetSimilarityExplained(string firstWord, string secondWord);
```

Returns

a div class html section detailing the metric operation.

IStringMetric.GetSimilarityTimingActual

gets the actual time in milliseconds it takes to perform a similarity timing.

Remarks

This call takes as long as the similarity metric to perform so should not be done in normal cercumstances.

```
long GetSimilarityTimingActual(string firstWord, string secondWord);
```

Returns

the actual time in milliseconds taken to perform the similarity measure

IStringMetric.GetSimilarityTimingEstimated

gets the estimated time in milliseconds it takes to perform a similarity timing.

```
double GetSimilarityTimingEstimated(string firstWord, string secondWord);
```

Returns

the estimated time in milliseconds taken to perform the similarity measure

IStringMetric.GetUnnormalisedSimilarity

gets the un-normalised similarity measure of the metric for the given strings.

```
double GetUnnormalisedSimilarity(string firstWord, string secondWord);
```

Returns

returns the score of the similarity measure (un-normalised)

1.1.7 ISubstitutionCost

is an interface for a cost function d(i,j).

Class Hierarchy

```
ISubstitutionCost
    SimMetricsApi.AbstractSubstitutionCost
```

```
public interface ISubstitutionCost;
```

Methods

Method	Description
GetCost (see page 15)	get cost between characters.

Properties

Property	Description
MaxCost (see page 15)	returns the maximum possible cost.
MinCost (see page 15)	returns the minimum possible cost.
ShortDescriptionString (see page 15)	returns the name of the cost function.

ISubstitutionCost.MaxCost

returns the maximum possible cost.

```
double MaxCost;
```

ISubstitutionCost.MinCost

returns the minimum possible cost.

```
double MinCost;
```

ISubstitutionCost.ShortDescriptionString

returns the name of the cost function.

```
string ShortDescriptionString;
```

ISubstitutionCost.GetCost

get cost between characters.

```
double GetCost(string firstWord, int firstWordIndex, string secondWord, int secondWordIndex);
```

1.1.8 ITermHandler

defines an interface for stop word handlers.

Class Hierarchy

ITermHandler

```
public interface ITermHandler;
```

Methods

Method	Description
AddWord (see page 16)	adds a Word to the interface.
IsWord (see page 16)	isStopWord determines if a given term is a word or not.
RemoveWord (see page 16)	removes the given word from the list.

Properties

Property	Description
NumberOfWords (see page 16)	gets the number of stopwords in the list.
ShortDescriptionString (see page 16)	gets the short description string of the stop word handler used.
WordsAsBuffer (see page 16)	gets the words as an output string buffer.

ITermHandler.NumberOfWords

gets the number of stopwords in the list.

```
int NumberOfWords;
```

ITermHandler.ShortDescriptionString

gets the short description string of the stop word handler used.

```
string ShortDescriptionString;
```

ITermHandler.WordsAsBuffer

gets the words as an output string buffer.

```
StringBuilder WordsAsBuffer;
```

ITermHandler.AddWord

adds a Word to the interface.

```
void AddWord(string termToAdd);
```

ITermHandler.IsWord

isStopWord determines if a given term is a word or not.

```
bool IsWord(string termToTest);
```

Returns

true if a stopword false otherwise.

ITermHandler.RemoveWord

removes the given word from the list.

```
void RemoveWord(string termToRemove);
```

1.1.9 ITokeniser

InterfaceTokeniser interface for a Tokeniser class.

Class Hierarchy

```
ITokeniser
    SimMetricsApi.AbstractTokeniserQGramN
```

```
public interface ITokeniser;
```

Methods

Method	Description
Tokenize (see page 17)	Return tokenized version of a string.
TokenizeToSet (see page 17)	Return tokenized version of a string as a set.

Properties

Property	Description
Delimiters (see page 17)	displays the delimitors used - (if applicable).
ShortDescriptionString (see page 17)	displays the tokenisation method.
StopWordHandler (see page 17)	gets the stop word handler used.

ITokeniser.Delimiters

displays the delimitors used - (if applicable).

```
string Delimiters;
```

ITokeniser.ShortDescriptionString

displays the tokenisation method.

```
string ShortDescriptionString;
```

ITokeniser.StopWordHandler

gets the stop word handler used.

```
ITermHandler StopWordHandler;
```

ITokeniser.Tokenize

Return tokenized version of a string.

```
Collection<string> Tokenize(string word);
```

Returns

tokenized version of a string

ITokeniser.TokenizeToSet

Return tokenized version of a string as a set.

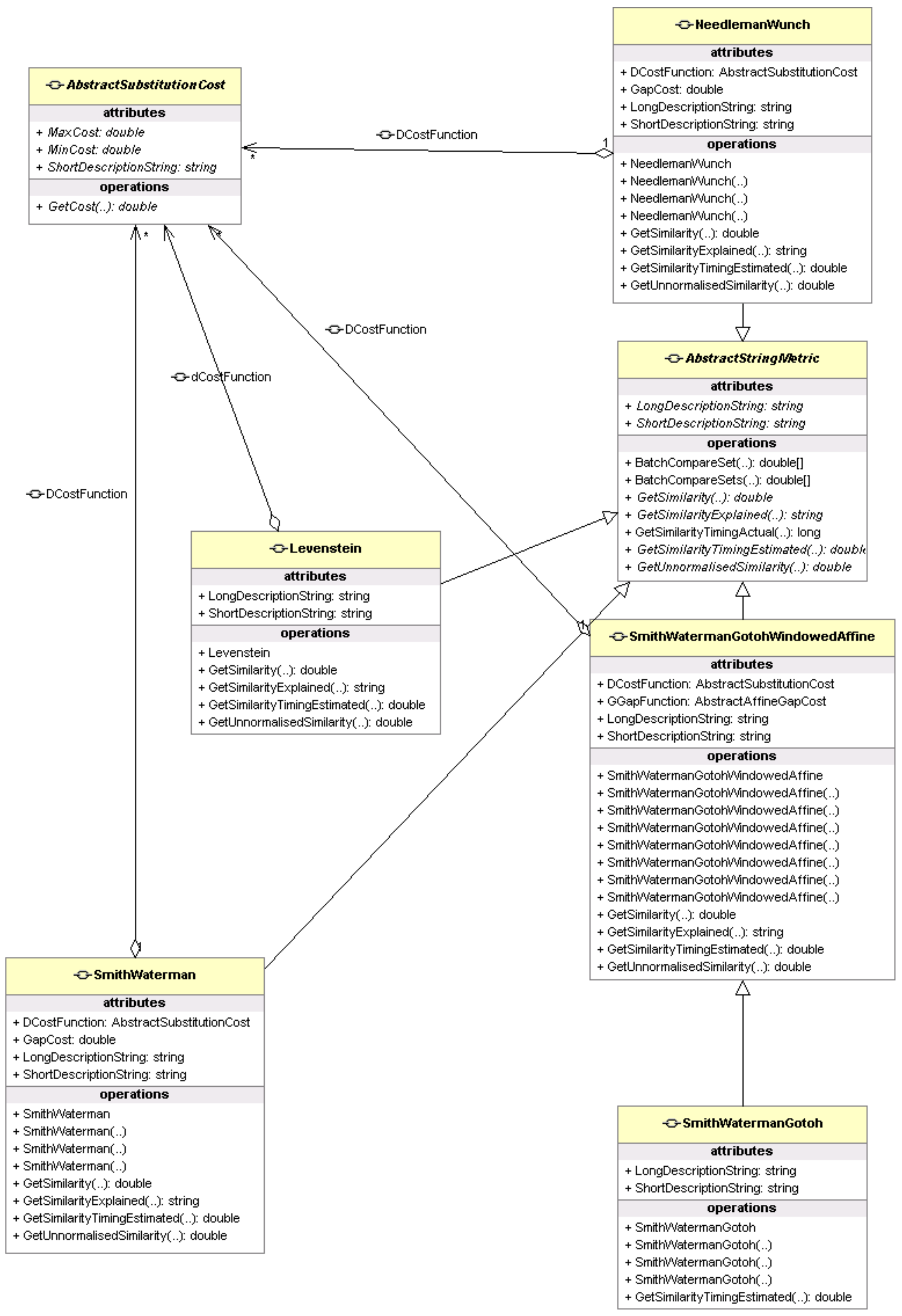
```
Collection<string> TokenizeToSet(string word);
```

Returns

tokenized version of a string as a set

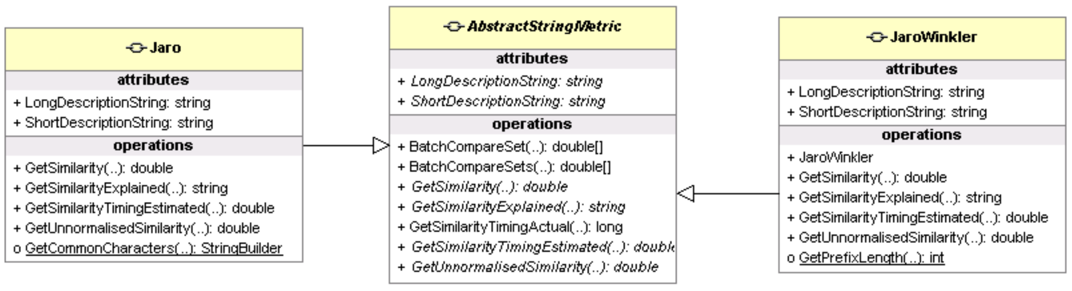
1.2 SimMetricsMetricUtilities

Edit Distance classes

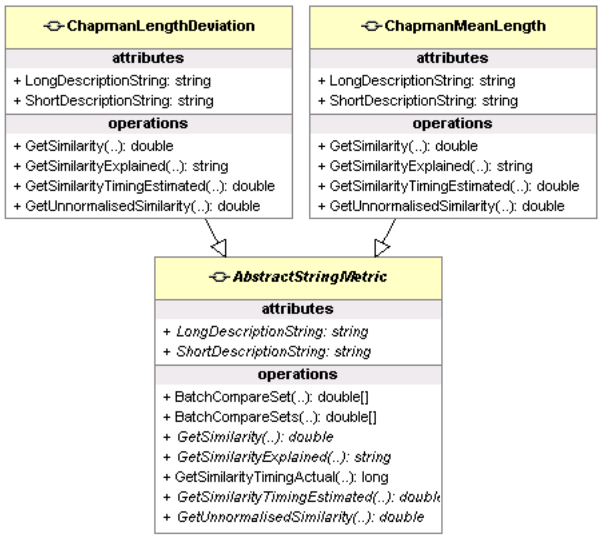

AbstractSubstitutionCost
attributes
+ MaxCost: double
+ MinCost: double
+ ShortDescriptionString: string
operations
+ GetCost(..): double
NeedlemanWunch
attributes
+ DCostFunction: AbstractSubstitutionCost
+ GapCost: double
+ LongDescriptionString: string
+ ShortDescriptionString: string
operations
+ NeedlemanWunch
+ NeedlemanWunch(..)
+ NeedlemanWunch(..)
+ NeedlemanWunch(..)
+ GetSimilarity(..): double
+ GetSimilarityExplained(..): string
+ GetSimilarityTimingEstimated(..): double
+ GetUnnormalisedSimilarity(..): double
DCostFunction
1
*
DCostFunction
dCostFunction
DCostFunction
*
AbstractStringMetric
attributes
+ LongDescriptionString: string
+ ShortDescriptionString: string
operations
+ BatchCompareSet(..): double[]
+ BatchCompareSets(..): double[]
+ GetSimilarity(..): double
+ GetSimilarityExplained(..): string
+ GetSimilarityTimingActual(..): long
+ GetSimilarityTimingEstimated(..): doubl
+ GetUnnormalisedSimilarity(..): double
Levenstein
attributes
+ LongDescriptionString: string
+ ShortDescriptionString: string
operations
+ Levenstein
+ GetSimilarity(..): double
+ GetSimilarityExplained(..): string
+ GetSimilarityTimingEstimated(..): double
+ GetUnnormalisedSimilarity(..): double
SmithWatermanGotohWindowedAffine
attributes
+ DCostFunction: AbstractSubstitutionCost
+ GGapFunction: AbstractAffineGapCost
+ LongDescriptionString: string
+ ShortDescriptionString: string
operations
+ SmithWatermanGotohWindowedAffine
+ SmithWatermanGotohWindowedAffine(..)
+ SmithWatermanGotohWindowedAffine(..)
+ SmithWatermanGotohWindowedAffine(..)
+ SmithWatermanGotohWindowedAffine(..)
+ SmithWatermanGotohWindowedAffine(..)
+ SmithWatermanGotohWindowedAffine(..)
+ SmithWatermanGotohWindowedAffine(..)
+ GetSimilarity(..): double
+ GetSimilarityExplained(..): string
+ GetSimilarityTimingEstimated(..): double
+ GetUnnormalisedSimilarity(..): double
SmithWaterman
attributes
+ DCostFunction: AbstractSubstitutionCost
+ GapCost: double
+ LongDescriptionString: string
+ ShortDescriptionString: string
operations
+ SmithWaterman
+ SmithWaterman(..)
+ SmithWaterman(..)
+ SmithWaterman(..)
+ GetSimilarity(..): double
+ GetSimilarityExplained(..): string
+ GetSimilarityTimingEstimated(..): double
+ GetUnnormalisedSimilarity(..): double
SmithWatermanGotoh
attributes
+ LongDescriptionString: string
+ ShortDescriptionString: string
operations
+ SmithWatermanGotoh
+ SmithWatermanGotoh(..)
+ SmithWatermanGotoh(..)
+ SmithWatermanGotoh(..)
+ GetSimilarityTimingEstimated(..): double

]

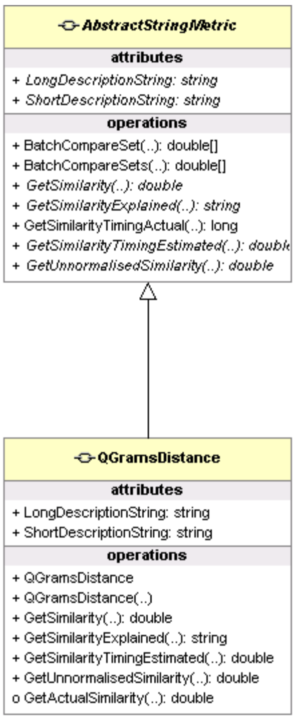
Jaro and JaroWinkler


Jaro
attributes
+ LongDescriptionString: string
+ ShortDescriptionString: string
operations
+ GetSimilarity(..): double
+ GetSimilarityExplained(..): string
+ GetSimilarityTimingEstimated(..): double
+ GetUnnormalisedSimilarity(..): double
o GetCommonCharacters(..): StringBuilder
AbstractStringMetric
attributes
+ LongDescriptionString: string
+ ShortDescriptionString: string
operations
+ BatchCompareSet(..): double[]
+ BatchCompareSets(..): double[]
+ GetSimilarity(..): double
+ GetSimilarityExplained(..): string
+ GetSimilarityTimingActual(..): long
+ GetSimilarityTimingEstimated(..): double
+ GetUnnormalisedSimilarity(..): double
JaroWinkler
attributes
+ LongDescriptionString: string
+ ShortDescriptionString: string
operations
+ JaroWinkler
+ GetSimilarity(..): double
+ GetSimilarityExplained(..): string
+ GetSimilarityTimingEstimated(..): double
+ GetUnnormalisedSimilarity(..): double
o GetPrefixLength(..): int

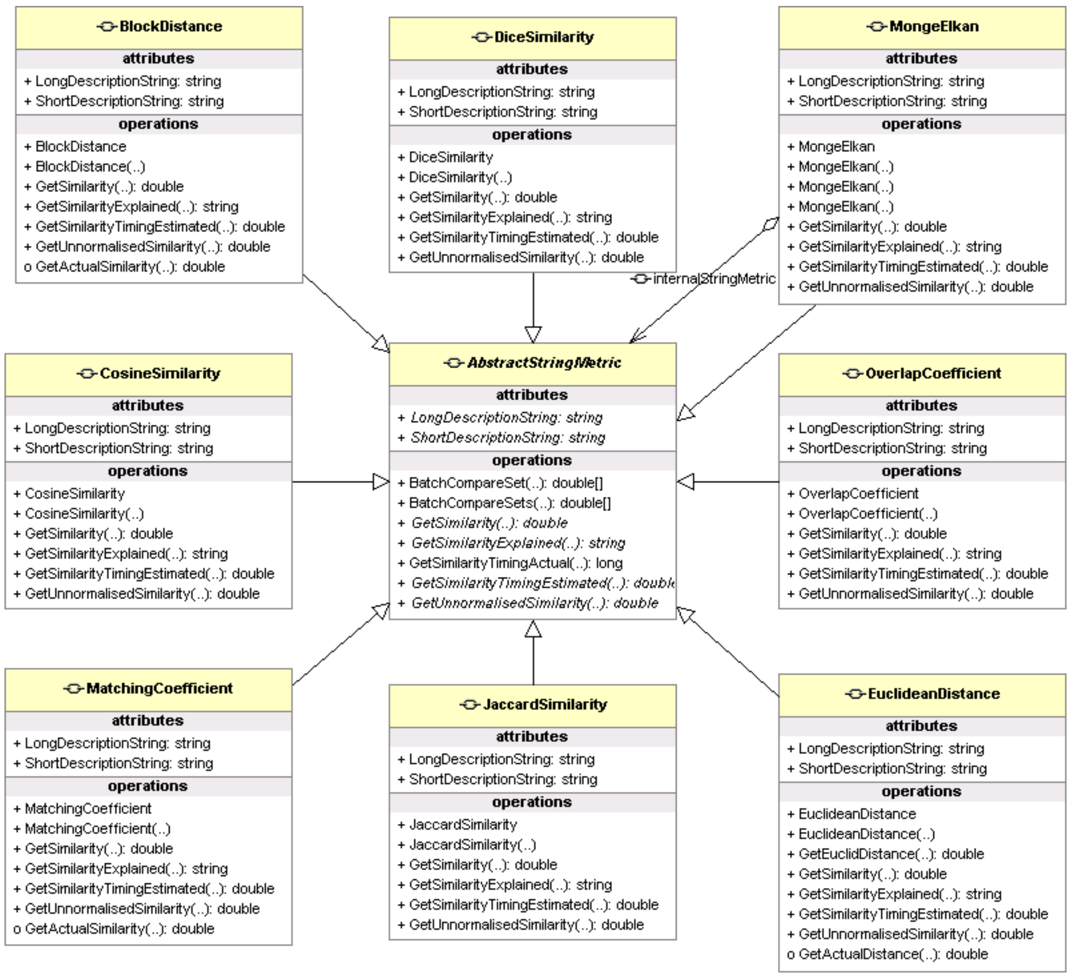
Length Based


ChapmanLengthDeviation
attributes
+ LongDescriptionString: string
+ ShortDescriptionString: string
operations
+ GetSimilarity(..): double
+ GetSimilarityExplained(..): string
+ GetSimilarityTimingEstimated(..): double
+ GetUnnormalisedSimilarity(..): double
ChapmanMeanLength
attributes
+ LongDescriptionString: string
+ ShortDescriptionString: string
operations
+ GetSimilarity(..): double
+ GetSimilarityExplained(..): string
+ GetSimilarityTimingEstimated(..): double
+ GetUnnormalisedSimilarity(..): double
AbstractStringMetric
attributes
+ LongDescriptionString: string
+ ShortDescriptionString: string
operations
+ BatchCompareSet(..): double[]
+ BatchCompareSets(..): double[]
+ GetSimilarity(..): double
+ GetSimilarityExplained(..): string
+ GetSimilarityTimingActual(..): long
+ GetSimilarityTimingEstimated(..): double
+ GetUnnormalisedSimilarity(..): double

QGram


AbstractStringMetric
attributes
+ LongDescriptionString: string
+ ShortDescriptionString: string
operations
+ BatchCompareSet(..): double[]
+ BatchCompareSets(..): double[]
+ GetSimilarity(..): double
+ GetSimilarityExplained(..): string
+ GetSimilarityTimingActual(..): long
+ GetSimilarityTimingEstimated(..): doubl
+ GetUnnormalisedSimilarity(..): double
QGramsDistance
attributes
+ LongDescriptionString: string
+ ShortDescriptionString: string
operations
+ QGramsDistance
+ QGramsDistance(..)
+ GetSimilarity(..): double
+ GetSimilarityExplained(..): string
+ GetSimilarityTimingEstimated(..): double
+ GetUnnormalisedSimilarity(..): double
o GetActualSimilarity(..): double

Token Based


BlockDistance
attributes
+ LongDescriptionString: string
+ ShortDescriptionString: string
operations
+ BlockDistance
+ BlockDistance(..)
+ GetSimilarity(..): double
+ GetSimilarityExplained(..): string
+ GetSimilarityTimingEstimated(..): double
+ GetUnnormalisedSimilarity(..): double
o GetActualSimilarity(..): double
DiceSimilarity
attributes
+ LongDescriptionString: string
+ ShortDescriptionString: string
operations
+ DiceSimilarity
+ DiceSimilarity(..)
+ GetSimilarity(..): double
+ GetSimilarityExplained(..): string
+ GetSimilarityTimingEstimated(..): double
+ GetUnnormalisedSimilarity(..): double
MongeElkan
attributes
+ LongDescriptionString: string
+ ShortDescriptionString: string
operations
+ MongeElkan
+ MongeElkan(..)
+ MongeElkan(..)
+ MongeElkan(..)
+ GetSimilarity(..): double
+ GetSimilarityExplained(..): string
+ GetSimilarityTimingEstimated(..): double
+ GetUnnormalisedSimilarity(..): double
internalStringMetric
AbstractStringMetric
attributes
+ LongDescriptionString: string
+ ShortDescriptionString: string
operations
+ BatchCompareSet(..): double[]
+ BatchCompareSets(..): double[]
+ GetSimilarity(..): double
+ GetSimilarityExplained(..): string
+ GetSimilarityTimingActual(..): long
+ GetSimilarityTimingEstimated(..): double
+ GetUnnormalisedSimilarity(..): double
CosineSimilarity
attributes
+ LongDescriptionString: string
+ ShortDescriptionString: string
operations
+ CosineSimilarity
+ CosineSimilarity(..)
+ GetSimilarity(..): double
+ GetSimilarityExplained(..): string
+ GetSimilarityTimingEstimated(..): double
+ GetUnnormalisedSimilarity(..): double
OverlapCoefficient
attributes
+ LongDescriptionString: string
+ ShortDescriptionString: string
operations
+ OverlapCoefficient
+ OverlapCoefficient(..)
+ GetSimilarity(..): double
+ GetSimilarityExplained(..): string
+ GetSimilarityTimingEstimated(..): double
+ GetUnnormalisedSimilarity(..): double
MatchingCoefficient
attributes
+ LongDescriptionString: string
+ ShortDescriptionString: string
operations
+ MatchingCoefficient
+ MatchingCoefficient(..)
+ GetSimilarity(..): double
+ GetSimilarityExplained(..): string
+ GetSimilarityTimingEstimated(..): double
+ GetUnnormalisedSimilarity(..): double
o GetActualSimilarity(..): double
JaccardSimilarity
attributes
+ LongDescriptionString: string
+ ShortDescriptionString: string
operations
+ JaccardSimilarity
+ JaccardSimilarity(..)
+ GetSimilarity(..): double
+ GetSimilarityExplained(..): string
+ GetSimilarityTimingEstimated(..): double
+ GetUnnormalisedSimilarity(..): double
EuclideanDistance
attributes
+ LongDescriptionString: string
+ ShortDescriptionString: string
operations
+ EuclideanDistance
+ EuclideanDistance(..)
+ GetEuclidDistance(..): double
+ GetSimilarity(..): double
+ GetSimilarityExplained(..): string
+ GetSimilarityTimingEstimated(..): double
+ GetUnnormalisedSimilarity(..): double
o GetActualDistance(..): double

Classes

Class	Description
BlockDistance (see page 22)	a block distance implementation metric
ChapmanLengthDeviation (see page 24)	implements a metric determined by the difference in string lengths
ChapmanMeanLength (see page 26)	implements Chapman Mean Length metric
CosineSimilarity (see page 28)	This is class SimMetricsMetricUtilities.CosineSimilarity.
DiceSimilarity (see page 30)	This is class SimMetricsMetricUtilities.DiceSimilarity.
EuclideanDistance (see page 32)	This is class SimMetricsMetricUtilities.EuclideanDistance.
JaccardSimilarity (see page 35)	This is class SimMetricsMetricUtilities.JaccardSimilarity.
Jaro (see page 37)	implements the Jaro string Metric.
JaroWinkler (see page 39)	implements the Jaro (see page 37) Winkler string metric
Levenstein (see page 41)	levenstein implements the levenstein distance function.
MatchingCoefficient (see page 44)	This is class SimMetricsMetricUtilities.MatchingCoefficient.
MongeElkan (see page 46)	This is class SimMetricsMetricUtilities.MongeElkan.
NeedlemanWunch (see page 49)	needlemanwunch implements an edit distance function
OverlapCoefficient (see page 52)	This is class SimMetricsMetricUtilities.OverlapCoefficient.
QGramsDistance (see page 54)	implements a QGram distance metric using supplied QGRam tokeniser
SmithWaterman (see page 56)	implements the Smith-Waterman edit distance function
SmithWatermanGotoh (see page 59)	implements the Gotoh extension of the smith waterman method incorporating affine gaps in the strings
SmithWatermanGotohWindowedAffine (see page 61)	implements the smith waterman with gotoh extension using a windowed affine gap.

1.2.1 BlockDistance

a block distance implementation metric

Class Hierarchy

```
AbstractStringMetric
    BlockDistance
```

```
[Serializable]
public sealed class BlockDistance : AbstractStringMetric;
```

Methods

Method	Description
BlockDistance (see page 22)	constructor - default (empty).
BlockDistance (see page 22)	constructor
GetSimilarity (see page 22)	gets the similarity of the two strings using BlockDistance.
GetSimilarityExplained (see page 23)	gets a div class xhtml similarity explaining the operation of the metric.
GetSimilarityTimingEstimated (see page 23)	gets the estimated time in milliseconds it takes to perform a similarity timing.
GetUnnormalisedSimilarity (see page 23)	gets the un-normalised similarity measure of the metric for the given strings.

Properties

Property	Description
LongDescriptionString (see page 22)	returns the long string identifier for the metric.
ShortDescriptionString (see page 22)	returns the string identifier for the metric.

BlockDistance.LongDescriptionString

returns the long string identifier for the metric.

```
public override string LongDescriptionString;
```

BlockDistance.ShortDescriptionString

returns the string identifier for the metric.

```
public override string ShortDescriptionString;
```

BlockDistance.BlockDistance

constructor - default (empty).

```
public BlockDistance();
```

Body Source

```
public BlockDistance() : this(new TokeniserWhitespace()) {}
```

BlockDistance.BlockDistance

constructor

```
public BlockDistance(ITokeniser tokeniserToUse);
```

Body Source

```
public BlockDistance(ITokeniser tokeniserToUse) {
    tokeniser = tokeniserToUse;
    tokenUtilities = new TokeniserUtilities<string>();
}
```

BlockDistance.GetSimilarity

gets the similarity of the two strings using BlockDistance (see page 22).

```
public override double GetSimilarity(string firstWord, string secondWord);
```

Returns

a 0-1 similarity score

Body Source

```
public override double GetSimilarity(string firstWord, string secondWord) {
    Collection<string> firstTokens = tokeniser.Tokenize(firstWord);
    Collection<string> secondTokens = tokeniser.Tokenize(secondWord);
```

```
        int totalPossible = firstTokens.Count + secondTokens.Count;
        double totalDistance = GetActualSimilarity(firstTokens, secondTokens);
        return (totalPossible - totalDistance) / totalPossible;
    }
```

BlockDistance.GetSimilarityExplained

gets a div class xhtml similarity explaining the operation of the metric.

```
public override string GetSimilarityExplained(string firstWord, string secondWord);
```

Returns

a div class html section detailing the metric operation.

Body Source

```
    public override string GetSimilarityExplained(string firstWord, string secondWord) {
        throw new NotImplementedException();
    }
```

BlockDistance.GetSimilarityTimingEstimated

gets the estimated time in milliseconds it takes to perform a similarity timing.

```
public override double GetSimilarityTimingEstimated(string firstWord, string secondWord);
```

Returns

the estimated time in milliseconds taken to perform the similarity measure

Body Source

```
    public override double GetSimilarityTimingEstimated(string firstWord, string secondWord) {
        double firstTokens = tokeniser.Tokenize(firstWord).Count;
        double secondTokens = tokeniser.Tokenize(secondWord).Count;
        return
            ((firstTokens + secondTokens) * firstTokens + (firstTokens + secondTokens) * secondTokens) *
            estimatedTimingConstant;
    }
```

BlockDistance.GetUnnormalisedSimilarity

gets the un-normalised similarity measure of the metric for the given strings.

```
public override double GetUnnormalisedSimilarity(string firstWord, string secondWord);
```

Returns

returns the score of the similarity measure (un-normalised)

Body Source

```
    public override double GetUnnormalisedSimilarity(string firstWord, string secondWord) {
        Collection<string> firstTokens = tokeniser.Tokenize(firstWord);
        Collection<string> secondTokens = tokeniser.Tokenize(secondWord);
        return GetActualSimilarity(firstTokens, secondTokens);
    }
```

1.2.2 ChapmanLengthDeviation

implements a metric determined by the difference in string lengths

Class Hierarchy

```
AbstractStringMetric
    ChapmanLengthDeviation
```

```
[Serializable]
public sealed class ChapmanLengthDeviation : AbstractStringMetric;
```

Methods

Method	Description
GetSimilarity (see page 24)	gets the similarity of the two strings using ChapmanLengthDeviation
GetSimilarityExplained (see page 24)	gets a div class xhtml similarity explaining the operation of the metric.
GetSimilarityTimingEstimated (see page 25)	gets the estimated time in milliseconds it takes to perform a similarity timing.
GetUnnormalisedSimilarity (see page 25)	gets the un-normalised similarity measure of the metric for the given strings.

Properties

Property	Description
LongDescriptionString (see page 24)	returns the long string identifier for the metric.
ShortDescriptionString (see page 24)	returns the string identifier for the metric.

ChapmanLengthDeviation.LongDescriptionString

returns the long string identifier for the metric.

```
public override string LongDescriptionString;
```

ChapmanLengthDeviation.ShortDescriptionString

returns the string identifier for the metric.

```
public override string ShortDescriptionString;
```

ChapmanLengthDeviation.GetSimilarity

gets the similarity of the two strings using ChapmanLengthDeviation (see page 24)

Remarks

this is simply a ratio of difference in string lengths between those compared.

```
public override double GetSimilarity(string firstWord, string secondWord);
```

Returns

a value between 0-1 of the similarity

Body Source

```
public override double GetSimilarity(string firstWord, string secondWord) {
    if ((firstWord != null) && (secondWord != null)) {
        double firstLength = firstWord.Length;
        double secondLength = secondWord.Length;
        if (firstLength >= secondLength) {
            return secondLength / firstLength;
        }
        else {
            return firstLength / secondLength;
        }
    }
    return 0.0;
}
```

ChapmanLengthDeviation.GetSimilarityExplained

gets a div class xhtml similarity explaining the operation of the metric.

```
public override string GetSimilarityExplained(string firstWord, string secondWord);
```

Returns

a div class html section detailing the metric operation.

Body Source

```
public override string GetSimilarityExplained(string firstWord, string secondWord) {
    throw new NotImplementedException();
}
```

ChapmanLengthDeviation.GetSimilarityTimingEstimated

gets the estimated time in milliseconds it takes to perform a similarity timing.

```
public override double GetSimilarityTimingEstimated(string firstWord, string secondWord);
```

Returns

the estimated time in milliseconds taken to perform the similarity measure

Body Source

```
public override double GetSimilarityTimingEstimated(string firstWord, string secondWord) {
    return 0.0;
}
```

ChapmanLengthDeviation.GetUnnormalisedSimilarity

gets the un-normalised similarity measure of the metric for the given strings.

```
public override double GetUnnormalisedSimilarity(string firstWord, string secondWord);
```

Returns

returns the score of the similarity measure (un-normalised)

Body Source

```
public override double GetUnnormalisedSimilarity(string firstWord, string secondWord) {
    return GetSimilarity(firstWord, secondWord);
}
```

1.2.3 ChapmanMeanLength

implements Chapman Mean Length metric

Class Hierarchy

```
AbstractStringMetric
    ChapmanMeanLength
```

```
[Serializable]
public sealed class ChapmanMeanLength : AbstractStringMetric;
```

Methods

Method	Description
GetSimilarity (see page 26)	gets the similarity of the two strings using ChapmanMeanLength
GetSimilarityExplained (see page 27)	gets a div class xhtml similarity explaining the operation of the metric.
GetSimilarityTimingEstimated (see page 27)	gets the estimated time in milliseconds it takes to perform a similarity timing.
GetUnnormalisedSimilarity (see page 27)	gets the un-normalised similarity measure of the metric for the given strings.

Properties

Property	Description
LongDescriptionString (see page 26)	returns the long string identifier for the metric.
ShortDescriptionString (see page 26)	returns the string identifier for the metric.

ChapmanMeanLength.defaultMismatchScore

This is defaultMismatchScore, a member of class ChapmanMeanLength.

```
public const double defaultMismatchScore = 0.0;
```

ChapmanMeanLength.defaultPerfectScore

This is defaultPerfectScore, a member of class ChapmanMeanLength.

```
public const double defaultPerfectScore = 1.0;
```

ChapmanMeanLength.LongDescriptionString

returns the long string identifier for the metric.

```
public override string LongDescriptionString;
```

ChapmanMeanLength.ShortDescriptionString

returns the string identifier for the metric.

```
public override string ShortDescriptionString;
```

ChapmanMeanLength.GetSimilarity

gets the similarity of the two strings using ChapmanMeanLength (see page 26)

```
public override double GetSimilarity(string firstWord, string secondWord);
```

Returns

a value between 0-1 of the similarity

Body Source

```
public override double GetSimilarity(string firstWord, string secondWord) {
    if ((firstWord != null) && (secondWord != null)) {
        double bothLengths = secondWord.Length + firstWord.Length;
        if (bothLengths > chapmanMeanLengthMaxString) {
            return defaultPerfectScore;
        }
        else {
            double oneMinusBothScaled = (chapmanMeanLengthMaxString - bothLengths) / chapmanMeanLengthMaxString;
            return
                defaultPerfectScore - oneMinusBothScaled * oneMinusBothScaled * oneMinusBothScaled *
oneMinusBothScaled;
        }
    }
    return defaultMismatchScore;
}
```

ChapmanMeanLength.GetSimilarityExplained

gets a div class xhtml similarity explaining the operation of the metric.

```
public override string GetSimilarityExplained(string firstWord, string secondWord);
```

Returns

a div class html section detailing the metric operation.

Body Source

```
public override string GetSimilarityExplained(string firstWord, string secondWord) {
    throw new NotImplementedException();
}
```

ChapmanMeanLength.GetSimilarityTimingEstimated

gets the estimated time in milliseconds it takes to perform a similarity timing.

```
public override double GetSimilarityTimingEstimated(string firstWord, string secondWord);
```

Returns

the estimated time in milliseconds taken to perform the similarity measure

Body Source

```
public override double GetSimilarityTimingEstimated(string firstWord, string secondWord) {
    return 0.0;
}
```

ChapmanMeanLength.GetUnnormalisedSimilarity

gets the un-normalised similarity measure of the metric for the given strings.

```
public override double GetUnnormalisedSimilarity(string firstWord, string secondWord);
```

Returns

returns the score of the similarity measure (un-normalised)

Body Source

```
public override double GetUnnormalisedSimilarity(string firstWord, string secondWord) {
    return GetSimilarity(firstWord, secondWord);
}
```

1.2.4 CosineSimilarity

This is class SimMetricsMetricUtilities.CosineSimilarity.

Class Hierarchy

```
AbstractStringMetric
    CosineSimilarity
```

```
[Serializable]
public sealed class CosineSimilarity : AbstractStringMetric;
```

Methods

Method	Description
CosineSimilarity (see page 28)	constructor
CosineSimilarity (see page 28)	constructor
GetSimilarity (see page 28)	gets the similarity of the two strings using CosineSimilarity.
GetSimilarityExplained (see page 29)	gets a div class xhtml similarity explaining the operation of the metric.
GetSimilarityTimingEstimated (see page 29)	gets the estimated time in milliseconds it takes to perform a similarity timing.
GetUnnormalisedSimilarity (see page 29)	gets the un-normalised similarity measure of the metric for the given strings.

Properties

Property	Description
LongDescriptionString (see page 28)	returns the long string identifier for the metric.
ShortDescriptionString (see page 28)	returns the string identifier for the metric.

CosineSimilarity.LongDescriptionString

returns the long string identifier for the metric.

```
public override string LongDescriptionString;
```

CosineSimilarity.ShortDescriptionString

returns the string identifier for the metric.

```
public override string ShortDescriptionString;
```

CosineSimilarity.CosineSimilarity

constructor

```
public CosineSimilarity();
```

Body Source

```
public CosineSimilarity() : this(new TokeniserWhitespace()) {}
```

CosineSimilarity.CosineSimilarity

constructor

```
public CosineSimilarity(ITokeniser tokeniserToUse);
```

Body Source

```
public CosineSimilarity(ITokeniser tokeniserToUse) {
    tokeniser = tokeniserToUse;
    tokenUtilities = new TokeniserUtilities<string>();
}
```

CosineSimilarity.GetSimilarity

gets the similarity of the two strings using CosineSimilarity (see page 28).

```
public override double GetSimilarity(string firstWord, string secondWord);
```

Returns

a value between 0-1 of the similarity

Body Source

```
public override double GetSimilarity(string firstWord, string secondWord) {
    if ((firstWord != null) && (secondWord != null)) {
        if (tokenUtilities.CreateMergedSet(tokeniser.Tokenize(firstWord), tokeniser.Tokenize(secondWord)).Count > 0) {
            return
                tokenUtilities.CommonSetTerms() /
                (Math.Pow(tokenUtilities.FirstSetTokenCount, 0.5) * Math.Pow(tokenUtilities.SecondSetTokenCount,
0.5));
        }
    }
    return 0.0;
}
```

CosineSimilarity.GetSimilarityExplained

gets a div class xhtml similarity explaining the operation of the metric.

```
public override string GetSimilarityExplained(string firstWord, string secondWord);
```

Returns

a div class html section detailing the metric operation.

Body Source

```
public override string GetSimilarityExplained(string firstWord, string secondWord) {
    throw new NotImplementedException();
}
```

CosineSimilarity.GetSimilarityTimingEstimated

gets the estimated time in milliseconds it takes to perform a similarity timing.

```
public override double GetSimilarityTimingEstimated(string firstWord, string secondWord);
```

Returns

the estimated time in milliseconds taken to perform the similarity measure

Body Source

```
public override double GetSimilarityTimingEstimated(string firstWord, string secondWord) {
    if ((firstWord != null) && (secondWord != null)) {
        double firstLength = firstWord.Length;
        double secondLength = secondWord.Length;
        return (firstLength + secondLength) * ((firstLength + secondLength) * estimatedTimingConstant);
    }
    return 0.0;
}
```

CosineSimilarity.GetUnnormalisedSimilarity

gets the un-normalised similarity measure of the metric for the given strings.

```
public override double GetUnnormalisedSimilarity(string firstWord, string secondWord);
```

Returns

returns the score of the similarity measure (un-normalised)

Body Source

```
public override double GetUnnormalisedSimilarity(string firstWord, string secondWord) {
    return GetSimilarity(firstWord, secondWord);
}
```

1.2.5 DiceSimilarity

This is class SimMetricsMetricUtilities.DiceSimilarity.

Class Hierarchy

```
AbstractStringMetric
    DiceSimilarity
```

```
[Serializable]
public sealed class DiceSimilarity : AbstractStringMetric;
```

Methods

Method	Description
DiceSimilarity (see page 30)	constructor
DiceSimilarity (see page 30)	constructor
GetSimilarity (see page 30)	gets the similarity of the two strings using DiceSimilarity
GetSimilarityExplained (see page 31)	gets a div class xhtml similarity explaining the operation of the metric.
GetSimilarityTimingEstimated (see page 31)	gets the estimated time in milliseconds it takes to perform a similarity timing.
GetUnnormalisedSimilarity (see page 31)	gets the un-normalised similarity measure of the metric for the given strings.

Properties

Property	Description
LongDescriptionString (see page 30)	returns the long string identifier for the metric.
ShortDescriptionString (see page 30)	returns the string identifier for the metric.

DiceSimilarity.LongDescriptionString

returns the long string identifier for the metric.

```
public override string LongDescriptionString;
```

DiceSimilarity.ShortDescriptionString

returns the string identifier for the metric.

```
public override string ShortDescriptionString;
```

DiceSimilarity.DiceSimilarity

constructor

```
public DiceSimilarity();
```

Body Source

```
public DiceSimilarity() : this(new TokeniserWhitespace()) {}
```

DiceSimilarity.DiceSimilarity

constructor

```
public DiceSimilarity(ITokeniser tokeniserToUse);
```

Body Source

```
public DiceSimilarity(ITokeniser tokeniserToUse) {
    tokeniser = tokeniserToUse;
    tokenUtilities = new TokeniserUtilities<string>();
}
```

DiceSimilarity.GetSimilarity

gets the similarity of the two strings using DiceSimilarity (see page 30)

Remarks

Dices coefficient = (2*Common Terms) / (Number of terms in String1 + Number of terms in String2).

```
public override double GetSimilarity(string firstWord, string secondWord);
```

Returns

a value between 0-1 of the similarity

Body Source

```
public override double GetSimilarity(string firstWord, string secondWord) {
    if ((firstWord != null) && (secondWord != null)) {
        if (tokenUtilities.CreateMergedSet(tokeniser.Tokenize(firstWord), tokeniser.Tokenize(secondWord)).Count > 0) {
            return
                (2.0 * tokenUtilities.CommonSetTerms()) /
                (tokenUtilities.FirstSetTokenCount + tokenUtilities.SecondSetTokenCount);
        }
    }
    return 0.0;
}
```

DiceSimilarity.GetSimilarityExplained

gets a div class xhtml similarity explaining the operation of the metric.

```
public override string GetSimilarityExplained(string firstWord, string secondWord);
```

Returns

a div class html section detailing the metric operation.

Body Source

```
public override string GetSimilarityExplained(string firstWord, string secondWord) {
    throw new NotImplementedException();
}
```

DiceSimilarity.GetSimilarityTimingEstimated

gets the estimated time in milliseconds it takes to perform a similarity timing.

```
public override double GetSimilarityTimingEstimated(string firstWord, string secondWord);
```

Returns

the estimated time in milliseconds taken to perform the similarity measure

Body Source

```
public override double GetSimilarityTimingEstimated(string firstWord, string secondWord) {
    if ((firstWord != null) && (secondWord != null)) {
        double firstLength = firstWord.Length;
        double secondLength = secondWord.Length;
        return (firstLength + secondLength) * ((firstLength + secondLength) * estimatedTimingConstant);
    }
    return 0.0;
}
```

DiceSimilarity.GetUnnormalisedSimilarity

gets the un-normalised similarity measure of the metric for the given strings.

```
public override double GetUnnormalisedSimilarity(string firstWord, string secondWord);
```

Returns

returns the score of the similarity measure (un-normalised)

Body Source

```
public override double GetUnnormalisedSimilarity(string firstWord, string secondWord) {
    return GetSimilarity(firstWord, secondWord);
}
```

1.2.6 EuclideanDistance

This is class SimMetricsMetricUtilities.EuclideanDistance.

Class Hierarchy

```
AbstractStringMetric
    EuclideanDistance
```

```
[Serializable]
public sealed class EuclideanDistance : AbstractStringMetric;
```

Methods

Method	Description
EuclideanDistance (see page 32)	constructor
EuclideanDistance (see page 32)	constructor
GetEuclidDistance (see page 32)	gets the actual euclidean distance ie not the value between 0-1.
GetSimilarity (see page 33)	gets the similarity of the two strings using EuclideanDistance
GetSimilarityExplained (see page 33)	gets a div class xhtml similarity explaining the operation of the metric.
GetSimilarityTimingEstimated (see page 33)	gets the estimated time in milliseconds it takes to perform a similarity timing.
GetUnnormalisedSimilarity (see page 33)	gets the un-normalised similarity measure of the metric for the given strings.

Properties

Property	Description
LongDescriptionString (see page 32)	returns the long string identifier for the metric.
ShortDescriptionString (see page 32)	returns the string identifier for the metric.

EuclideanDistance.defaultMismatchScore

This is defaultMismatchScore, a member of class EuclideanDistance.

```
public const double defaultMismatchScore = 0.0;
```

EuclideanDistance.LongDescriptionString

returns the long string identifier for the metric.

```
public override string LongDescriptionString;
```

EuclideanDistance.ShortDescriptionString

returns the string identifier for the metric.

```
public override string ShortDescriptionString;
```

EuclideanDistance.EuclideanDistance

constructor

```
public EuclideanDistance();
```

Body Source

```
public EuclideanDistance() : this(new TokeniserWhitespace()) {}
```

EuclideanDistance.EuclideanDistance

constructor

```
public EuclideanDistance(ITokeniser tokeniserToUse);
```

Body Source

```
public EuclideanDistance(ITokeniser tokeniserToUse) {
    tokeniser = tokeniserToUse;
    tokenUtilities = new TokeniserUtilities<string>();
}
```

EuclideanDistance.GetEuclidDistance

gets the actual euclidean distance ie not the value between 0-1.

```
public double GetEuclidDistance(string firstWord, string secondWord);
```

Returns

the actual euclidean distance

Body Source

```
public double GetEuclidDistance(string firstWord, string secondWord) {
    if ((firstWord != null) && (secondWord != null)) {
        Collection<string> firstTokens = tokeniser.Tokenize(firstWord);
        Collection<string> secondTokens = tokeniser.Tokenize(secondWord);

        return GetActualDistance(firstTokens, secondTokens);
    }
    return defaultMismatchScore;
}
```

EuclideanDistance.GetSimilarity

gets the similarity of the two strings using EuclideanDistance (see page 32)

Remarks

the 0-1 return is calcualted from the maximum possible Euclidean distance between the strings from the number of terms within them.

```
public override double GetSimilarity(string firstWord, string secondWord);
```

Returns

a value between 0-1 of the similarity 1.0 identical

Body Source

```
public override double GetSimilarity(string firstWord, string secondWord) {
    if ((firstWord != null) && (secondWord != null)) {
        double difference = GetUnnormalisedSimilarity(firstWord, secondWord);
        double totalPossible = Math.Sqrt(tokenUtilities.FirstTokenCount + tokenUtilities.SecondTokenCount);
        return (totalPossible - difference) / totalPossible;
    }
    return defaultMismatchScore;
}
```

EuclideanDistance.GetSimilarityExplained

gets a div class xhtml similarity explaining the operation of the metric.

```
public override string GetSimilarityExplained(string firstWord, string secondWord);
```

Returns

a div class html section detailing the metric operation.

Body Source

```
public override string GetSimilarityExplained(string firstWord, string secondWord) {
    throw new NotImplementedException();
}
```

EuclideanDistance.GetSimilarityTimingEstimated

gets the estimated time in milliseconds it takes to perform a similarity timing.

```
public override double GetSimilarityTimingEstimated(string firstWord, string secondWord);
```

Returns

the estimated time in milliseconds taken to perform the similarity measure

Body Source

```
public override double GetSimilarityTimingEstimated(string firstWord, string secondWord) {
    if ((firstWord != null) && (secondWord != null)) {
        double firstTokens = tokeniser.Tokenize(firstWord).Count;
        double secondTokens = tokeniser.Tokenize(secondWord).Count;
        return
            ((firstTokens + secondTokens) * firstTokens + (firstTokens + secondTokens) * secondTokens) *
            estimatedTimingConstant;
    }
    return 0.0;
}
```

EuclideanDistance.GetUnnormalisedSimilarity

gets the un-normalised similarity measure of the metric for the given strings.

```
public override double GetUnnormalisedSimilarity(string firstWord, string secondWord);
```

Returns

returns the score of the similarity measure (un-normalised)

Body Source

```
public override double GetUnnormalisedSimilarity(string firstWord, string secondWord) {
    return GetEuclidDistance(firstWord, secondWord);
}
```

1.2.7 JaccardSimilarity

This is class SimMetricsMetricUtilities.JaccardSimilarity.

Class Hierarchy

```
AbstractStringMetric
    JaccardSimilarity
```

```
[Serializable]
public sealed class JaccardSimilarity : AbstractStringMetric;
```

Methods

Method	Description
GetSimilarity (see page 35)	gets the similarity of the two strings using JaccardSimilarity.
GetSimilarityExplained (see page 36)	gets a div class xhtml similarity explaining the operation of the metric.
GetSimilarityTimingEstimated (see page 36)	gets the estimated time in milliseconds it takes to perform a similarity timing.
GetUnnormalisedSimilarity (see page 36)	gets the un-normalised similarity measure of the metric for the given strings.
JaccardSimilarity (see page 36)	This is JaccardSimilarity, a member of class JaccardSimilarity.
JaccardSimilarity (see page 36)	the tokeniser to use should a different tokeniser be required

Properties

Property	Description
LongDescriptionString (see page 35)	returns the long string identifier for the metric.
ShortDescriptionString (see page 35)	returns the string identifier for the metric .

JaccardSimilarity.defaultMismatchScore

This is defaultMismatchScore, a member of class JaccardSimilarity.

```
public const double defaultMismatchScore = 0.0;
```

JaccardSimilarity.LongDescriptionString

returns the long string identifier for the metric.

```
public override string LongDescriptionString;
```

JaccardSimilarity.ShortDescriptionString

returns the string identifier for the metric .

```
public override string ShortDescriptionString;
```

JaccardSimilarity.GetSimilarity

gets the similarity of the two strings using JaccardSimilarity (see page 35).

Remarks

Each instance is represented as a Jaccard vector similarity function. The Jaccard between two vectors X and Y is (X*Y) / (|X||Y|-(X*Y)) where (X*Y) is the inner product of X and Y, and |X| = (X*X)^1/2, i.e. the Euclidean norm of X. This can more easily be described as (|X and Y|) / (| X or Y |)

```
public override double GetSimilarity(string firstWord, string secondWord);
```

Returns

a value between 0-1 of the similarity

Body Source

```
public override double GetSimilarity(string firstWord, string secondWord) {
    if ((firstWord != null) && (secondWord != null)) {
        Collection<string> allTokens =
            tokenUtilities.CreateMergedSet(tokeniser.Tokenize(firstWord), tokeniser.Tokenize(secondWord));
        if (allTokens.Count > 0) {
            return (double)tokenUtilities.CommonSetTerms() / (double)allTokens.Count;
        }
    }
    return defaultMismatchScore;
}
```

JaccardSimilarity.GetSimilarityExplained

gets a div class xhtml similarity explaining the operation of the metric.

```
public override string GetSimilarityExplained(string firstWord, string secondWord);
```

Returns

a div class html section detailing the metric operation.

Body Source

```
public override string GetSimilarityExplained(string firstWord, string secondWord) {
    throw new NotImplementedException();
}
```

JaccardSimilarity.GetSimilarityTimingEstimated

gets the estimated time in milliseconds it takes to perform a similarity timing.

```
public override double GetSimilarityTimingEstimated(string firstWord, string secondWord);
```

Returns

the estimated time in milliseconds taken to perform the similarity measure

Body Source

```
public override double GetSimilarityTimingEstimated(string firstWord, string secondWord) {
    if ((firstWord != null) && (secondWord != null)) {
        double firstTokens = tokeniser.Tokenize(firstWord).Count;
        double secondTokens = tokeniser.Tokenize(secondWord).Count;
        return firstTokens * secondTokens * estimatedTimingConstant;
    }
    return defaultMismatchScore;
}
```

JaccardSimilarity.GetUnnormalisedSimilarity

gets the un-normalised similarity measure of the metric for the given strings.

```
public override double GetUnnormalisedSimilarity(string firstWord, string secondWord);
```

Returns

returns the score of the similarity measure (un-normalised)

Body Source

```
public override double GetUnnormalisedSimilarity(string firstWord, string secondWord) {
    return GetSimilarity(firstWord, secondWord);
}
```

JaccardSimilarity.JaccardSimilarity

This is JaccardSimilarity, a member of class JaccardSimilarity.

```
public JaccardSimilarity();
```

Body Source

```
public JaccardSimilarity() : this(new TokeniserWhitespace()) {}
```

JaccardSimilarity.JaccardSimilarity

the tokeniser to use should a different tokeniser be required

```
public JaccardSimilarity(ITokeniser tokeniserToUse);
```

Body Source

```
public JaccardSimilarity(ITokeniser tokeniserToUse) {
    tokeniser = tokeniserToUse;
    tokenUtilities = new TokeniserUtilities<string>();
}
```

1.2.8 Jaro

implements the Jaro string Metric.

Class Hierarchy

```
AbstractStringMetric
    Jaro
```

```
[Serializable]
public sealed class Jaro : AbstractStringMetric;
```

Methods

Method	Description
GetSimilarity (see page 37)	gets the similarity of the two strings using Jaro distance.
GetSimilarityExplained (see page 38)	gets a div class xhtml similarity explaining the operation of the metric.
GetSimilarityTimingEstimated (see page 38)	gets the estimated time in milliseconds it takes to perform a similarity timing.
GetUnnormalisedSimilarity (see page 38)	gets the un-normalised similarity measure of the metric for the given strings.

Properties

Property	Description
LongDescriptionString (see page 37)	returns the long string identifier for the metric.
ShortDescriptionString (see page 37)	returns the string identifier for the metric.

Jaro.LongDescriptionString

returns the long string identifier for the metric.

```
public override string LongDescriptionString;
```

Jaro.ShortDescriptionString

returns the string identifier for the metric.

```
public override string ShortDescriptionString;
```

Jaro.GetSimilarity

gets the similarity of the two strings using Jaro (see page 37) distance.

```
public override double GetSimilarity(string firstWord, string secondWord);
```

Returns

a value between 0-1 of the similarity

Body Source

```
public override double GetSimilarity(string firstWord, string secondWord) {
    if ((firstWord != null) && (secondWord != null)) {
        //get half the length of the string rounded up - (this is the distance used for acceptable transpositions)
        int halflen = Math.Min(firstWord.Length, secondWord.Length) / 2 + 1;
        //get common characters
        StringBuilder common1 = GetCommonCharacters(firstWord, secondWord, halflen);
        int commonMatches = common1.Length;
        //check for zero in common
        if (commonMatches == 0) {
            return defaultMismatchScore;
        }
        StringBuilder common2 = GetCommonCharacters(secondWord, firstWord, halflen);
        //check for same length common strings returning 0.0f is not the same
        if (commonMatches != common2.Length) {
            return defaultMismatchScore;
        }
        //get the number of transpositions
        int transpositions = 0;
        for (int i = 0; i < commonMatches; i++) {
            if (common1[i] != common2[i]) {
                transpositions++;
            }
        }

        //calculate jaro metric
        transpositions /= 2;
        double tmp1;
        tmp1 = commonMatches / (3.0 * firstWord.Length) + commonMatches / (3.0 * secondWord.Length) +
               (commonMatches - transpositions) / (3.0 * commonMatches);
        return tmp1;
    }
    return defaultMismatchScore;
}
```

Jaro.GetSimilarityExplained

gets a div class xhtml similarity explaining the operation of the metric.

```
public override string GetSimilarityExplained(string firstWord, string secondWord);
```

Returns

a div class html section detailing the metric operation.

Body Source

```
public override string GetSimilarityExplained(string firstWord, string secondWord) {
    throw new NotImplementedException();
}
```

Jaro.GetSimilarityTimingEstimated

gets the estimated time in milliseconds it takes to perform a similarity timing.

```
public override double GetSimilarityTimingEstimated(string firstWord, string secondWord);
```

Returns

the estimated time in milliseconds taken to perform the similarity measure

Body Source

```
public override double GetSimilarityTimingEstimated(string firstWord, string secondWord) {
    if ((firstWord != null) && (secondWord != null)) {
        double firstLength = firstWord.Length;
        double secondLength = secondWord.Length;
        return firstLength * secondLength * estimatedTimingConstant;
    }
    return 0.0;
}
```

Jaro.GetUnnormalisedSimilarity

gets the un-normalised similarity measure of the metric for the given strings.

```
public override double GetUnnormalisedSimilarity(string firstWord, string secondWord);
```

Returns

returns the score of the similarity measure (un-normalised)

Body Source

```
public override double GetUnnormalisedSimilarity(string firstWord, string secondWord) {
    return GetSimilarity(firstWord, secondWord);
}
```

1.2.9 JaroWinkler

implements the Jaro (see page 37) Winkler string metric

Class Hierarchy

```
AbstractStringMetric
    JaroWinkler
```

```
[Serializable]
public sealed class JaroWinkler : AbstractStringMetric;
```

Methods

Method	Description
GetSimilarity (see page 39)	gets the similarity measure of the JaroWinkler metric for the given strings.
GetSimilarityExplained (see page 39)	gets a div class xhtml similarity explaining the operation of the metric.
GetSimilarityTimingEstimated (see page 40)	gets the estimated time in milliseconds it takes to perform a similarity timing.
GetUnnormalisedSimilarity (see page 40)	gets the un-normalised similarity measure of the metric for the given strings.
JaroWinkler (see page 40)	constructor

Properties

Property	Description
LongDescriptionString (see page 39)	returns the long string identifier for the metric.
ShortDescriptionString (see page 39)	returns the string identifier for the metric.

JaroWinkler.LongDescriptionString

returns the long string identifier for the metric.

```
public override string LongDescriptionString;
```

JaroWinkler.ShortDescriptionString

returns the string identifier for the metric.

```
public override string ShortDescriptionString;
```

JaroWinkler.GetSimilarity

gets the similarity measure of the JaroWinkler (see page 39) metric for the given strings.

```
public override double GetSimilarity(string firstWord, string secondWord);
```

Returns

0-1 similarity measure of the JaroWinkler (see page 39) metric

Body Source

```
public override double GetSimilarity(string firstWord, string secondWord) {
    if ((firstWord != null) && (secondWord != null)) {
        double dist = jaroStringMetric.GetSimilarity(firstWord, secondWord);
        int prefixLength = GetPrefixLength(firstWord, secondWord);
        return dist + prefixLength * prefixAdustmentScale * (1.0 - dist);
    }
    return 0.0;
}
```

JaroWinkler.GetSimilarityExplained

gets a div class xhtml similarity explaining the operation of the metric.

```
public override string GetSimilarityExplained(string firstWord, string secondWord);
```

Returns

a div class html section detailing the metric operation.

Body Source

```
public override string GetSimilarityExplained(string firstWord, string secondWord) {
    throw new NotImplementedException();
}
```

JaroWinkler.GetSimilarityTimingEstimated

gets the estimated time in milliseconds it takes to perform a similarity timing.

```
public override double GetSimilarityTimingEstimated(string firstWord, string secondWord);
```

Returns

the estimated time in milliseconds taken to perform the similarity measure

Body Source

```
public override double GetSimilarityTimingEstimated(string firstWord, string secondWord) {
    if ((firstWord != null) && (secondWord != null)) {
        double firstLength = firstWord.Length;
        double secondLength = secondWord.Length;
        return firstLength * secondLength * estimatedTimingConstant;
    }
    return 0.0;
}
```

JaroWinkler.GetUnnormalisedSimilarity

gets the un-normalised similarity measure of the metric for the given strings.

```
public override double GetUnnormalisedSimilarity(string firstWord, string secondWord);
```

Returns

returns the score of the similarity measure (un-normalised)

Body Source

```
public override double GetUnnormalisedSimilarity(string firstWord, string secondWord) {
    return GetSimilarity(firstWord, secondWord);
}
```

JaroWinkler.JaroWinkler

constructor

```
public JaroWinkler();
```

Body Source

```
public JaroWinkler() {
    jaroStringMetric = new Jaro();
}
```

1.2.10 Levenstein

levenstein implements the levenstein distance function.

Class Hierarchy

```
AbstractStringMetric
    Levenstein
```

```
[Serializable]
public sealed class Levenstein : AbstractStringMetric;
```

Methods

Method	Description
GetSimilarity (see page 41)	gets the similarity of the two strings using levenstein distance.
GetSimilarityExplained (see page 41)	gets a div class xhtml similarity explaining the operation of the metric.
GetSimilarityTimingEstimated (see page 42)	gets the estimated time in milliseconds it takes to perform a similarity timing.
GetUnnormalisedSimilarity (see page 42)	gets the un-normalised similarity measure of the metric for the given strings.
Levenstein (see page 43)	constructor to load dummy Java converter classes only

Properties

Property	Description
LongDescriptionString (see page 41)	returns the long string identifier for the metric.
ShortDescriptionString (see page 41)	returns the string identifier for the metric.

Levenstein.LongDescriptionString

returns the long string identifier for the metric.

```
public override string LongDescriptionString;
```

Levenstein.ShortDescriptionString

returns the string identifier for the metric.

```
public override string ShortDescriptionString;
```

Levenstein.GetSimilarity

gets the similarity of the two strings using levenstein distance.

```
public override double GetSimilarity(string firstWord, string secondWord);
```

Returns

a value between 0-1 of the similarity

Body Source

```
public override double GetSimilarity(string firstWord, string secondWord) {
    if ((firstWord != null) && (secondWord != null)) {
        double levensteinDistance = GetUnnormalisedSimilarity(firstWord, secondWord);
        double maxLen = firstWord.Length;
        if (maxLen < secondWord.Length) {
            maxLen = secondWord.Length;
        }
        if (maxLen == defaultMismatchScore) {
            return defaultPerfectMatchScore;
        }
        else {
            return defaultPerfectMatchScore - levensteinDistance / maxLen;
        }
    }
    return defaultMismatchScore;
}
```

Levenstein.GetSimilarityExplained

gets a div class xhtml similarity explaining the operation of the metric.

```
public override string GetSimilarityExplained(string firstWord, string secondWord);
```

Returns

a div class html section detailing the metric operation.

Body Source

```
public override string GetSimilarityExplained(string firstWord, string secondWord) {
    throw new NotImplementedException();
}
```

Levenstein.GetSimilarityTimingEstimated

gets the estimated time in milliseconds it takes to perform a similarity timing.

```
public override double GetSimilarityTimingEstimated(string firstWord, string secondWord);
```

Returns

the estimated time in milliseconds taken to perform the similarity measure

Body Source

```
public override double GetSimilarityTimingEstimated(string firstWord, string secondWord) {
    if ((firstWord != null) && (secondWord != null)) {
        double firstLength = firstWord.Length;
        double secondLength = secondWord.Length;
        return firstLength * secondLength * estimatedTimingConstant;
    }
    return defaultMismatchScore;
}
```

Levenstein.GetUnnormalisedSimilarity

gets the un-normalised similarity measure of the metric for the given strings.

Remarks

Copy character from string1 over to string2 (cost 0) Delete a character in string1 (cost 1) Insert a character in string2 (cost 1) Substitute one character for another (cost 1)

D(i-1,j-1) + d(si,tj) //subst/copy D(i,j) = min D(i-1,j)+1 //insert D(i,j-1)+1 //delete

d(i,j) is a function whereby d(c,d)=0 if c=d, 1 else.

```
public override double GetUnnormalisedSimilarity(string firstWord, string secondWord);
```

Returns

returns the score of the similarity measure (un-normalised)

Body Source

```
public override double GetUnnormalisedSimilarity(string firstWord, string secondWord) {
    if ((firstWord != null) && (secondWord != null)) {
        // Step 1
        int n = firstWord.Length;
        int m = secondWord.Length;
        if (n == 0) {
            return m;
        }
        if (m == 0) {
            return n;
        }

        double[][] d = new double[n + 1][];
        for (int i = 0; i < n + 1; i++) {
            d[i] = new double[m + 1];
        }

        // Step 2
        for (int i = 0; i <= n; i++) {
            d[i][0] = i;
        }
        for (int j = 0; j <= m; j++) {
            d[0][j] = j;
        }

        // Step 3
        for (int i = 1; i <= n; i++) {
            // Step 4
            for (int j = 1; j <= m; j++) {
                // Step 5
                double cost = dCostFunction.GetCost(firstWord, i - 1, secondWord, j - 1);
                // Step 6
                d[i][j] = MathFunctions.MinOf3(d[i - 1][j] + 1.0, d[i][j - 1] + 1.0, d[i - 1][j - 1] + cost);
            }
        }

        // Step 7
        return d[n][m];
    }
```

```
        return 0.0;
    }
```

Levenstein.Levenstein

constructor to load dummy Java converter classes only

```
public Levenstein();
```

Body Source

```
public Levenstein() {
    dCostFunction = new SubCostRange0To1();
}
```

1.2.11 MatchingCoefficient

This is class SimMetricsMetricUtilities.MatchingCoefficient.

Class Hierarchy

```
AbstractStringMetric
    MatchingCoefficient
```

```
[Serializable]
public sealed class MatchingCoefficient : AbstractStringMetric;
```

Methods

Method	Description
GetSimilarity (see page 44)	gets the similarity of the two strings using MatchingCoefficient.
GetSimilarityExplained (see page 44)	gets a div class xhtml similarity explaining the operation of the metric.
GetSimilarityTimingEstimated (see page 45)	gets the estimated time in milliseconds it takes to perform a similarity timing.
GetUnnormalisedSimilarity (see page 45)	gets the un-normalised similarity measure of the metric for the given strings.
MatchingCoefficient (see page 45)	This is MatchingCoefficient, a member of class MatchingCoefficient.
MatchingCoefficient (see page 45)	the tokeniser to use should a different tokeniser be required

Properties

Property	Description
LongDescriptionString (see page 44)	returns the long string identifier for the metric.
ShortDescriptionString (see page 44)	returns the string identifier for the metric .

MatchingCoefficient.LongDescriptionString

returns the long string identifier for the metric.

```
public override string LongDescriptionString;
```

MatchingCoefficient.ShortDescriptionString

returns the string identifier for the metric .

```
public override string ShortDescriptionString;
```

MatchingCoefficient.GetSimilarity

gets the similarity of the two strings using MatchingCoefficient (see page 44).

```
public override double GetSimilarity(string firstWord, string secondWord);
```

Returns

a value between 0-1 of the similarity

Body Source

```
public override double GetSimilarity(string firstWord, string secondWord) {
    if ((firstWord != null) && (secondWord != null)) {
        double totalFound = GetUnnormalisedSimilarity(firstWord, secondWord);
        int totalPossible = Math.Max(tokenUtilities.FirstTokenCount, tokenUtilities.SecondTokenCount);
        return totalFound / totalPossible;
    }
    return defaultMismatchScore;
}
```

MatchingCoefficient.GetSimilarityExplained

gets a div class xhtml similarity explaining the operation of the metric.

```
public override string GetSimilarityExplained(string firstWord, string secondWord);
```

Returns

a div class html section detailing the metric operation.

Body Source

```
public override string GetSimilarityExplained(string firstWord, string secondWord) {
    throw new NotImplementedException();
}
```

MatchingCoefficient.GetSimilarityTimingEstimated

gets the estimated time in milliseconds it takes to perform a similarity timing.

```
public override double GetSimilarityTimingEstimated(string firstWord, string secondWord);
```

Returns

the estimated time in milliseconds taken to perform the similarity measure

Body Source

```
public override double GetSimilarityTimingEstimated(string firstWord, string secondWord) {
    if ((firstWord != null) && (secondWord != null)) {
        double firstTokens = tokeniser.Tokenize(firstWord).Count;
        double secondTokens = tokeniser.Tokenize(secondWord).Count;
        return secondTokens * firstTokens * estimatedTimingConstant;
    }
    return 0.0;
}
```

MatchingCoefficient.GetUnnormalisedSimilarity

gets the un-normalised similarity measure of the metric for the given strings.

```
public override double GetUnnormalisedSimilarity(string firstWord, string secondWord);
```

Returns

returns the score of the similarity measure (un-normalised)

Body Source

```
public override double GetUnnormalisedSimilarity(string firstWord, string secondWord) {
    Collection<string> firstTokens = tokeniser.Tokenize(firstWord);
    Collection<string> secondTokens = tokeniser.Tokenize(secondWord);
    return GetActualSimilarity(firstTokens, secondTokens);
}
```

MatchingCoefficient.MatchingCoefficient

This is MatchingCoefficient, a member of class MatchingCoefficient.

```
public MatchingCoefficient();
```

Body Source

```
public MatchingCoefficient() : this(new TokeniserWhitespace()) {}
```

MatchingCoefficient.MatchingCoefficient

the tokeniser to use should a different tokeniser be required

```
public MatchingCoefficient(ITokeniser tokeniserToUse);
```

Body Source

```
public MatchingCoefficient(ITokeniser tokeniserToUse) {
    tokeniser = tokeniserToUse;
    tokenUtilities = new TokeniserUtilities<string>();
}
```

1.2.12 MongeElkan

This is class SimMetricsMetricUtilities.MongeElkan.

Class Hierarchy

```
AbstractStringMetric
    MongeElkan
```

```
[Serializable]
public class MongeElkan : AbstractStringMetric;
```

Methods

Method	Description
GetSimilarity (see page 46)	gets the similarity of the two strings using Monge Elkan.
GetSimilarityExplained (see page 47)	gets a div class xhtml similarity explaining the operation of the metric.
GetSimilarityTimingEstimated (see page 47)	gets the estimated time in milliseconds it takes to perform a similarity timing.
GetUnnormalisedSimilarity (see page 47)	gets the un-normalised similarity measure of the metric for the given strings.
MongeElkan (see page 47)	basic constructor
MongeElkan (see page 47)	constructor taking metric to use
MongeElkan (see page 47)	constructor taking a tokeniser to use
MongeElkan (see page 48)	constructor taking a tokeniser and string metric to use

Properties

Property	Description
LongDescriptionString (see page 46)	returns the long string identifier for the metric.
ShortDescriptionString (see page 46)	returns the string identifier for the metric.

MongeElkan.LongDescriptionString

returns the long string identifier for the metric.

```
public override string LongDescriptionString;
```

MongeElkan.ShortDescriptionString

returns the string identifier for the metric.

```
public override string ShortDescriptionString;
```

MongeElkan.GetSimilarity

gets the similarity of the two strings using Monge Elkan.

```
public override double GetSimilarity(string firstWord, string secondWord);
```

Returns

a value between 0-1 of the similarity

Body Source

```
public override double GetSimilarity(string firstWord, string secondWord) {
    if ((firstWord != null) && (secondWord != null)) {
        Collection<string> firstTokens = tokeniser.Tokenize(firstWord);
        Collection<string> secondTokens = tokeniser.Tokenize(secondWord);

        double sumMatches = 0.0;
        for (int i = 0; i < firstTokens.Count; i++) {
            string sToken = firstTokens[i];
            double maxFound = 0.0;
            for (int j = 0; j < secondTokens.Count; j++) {
                string tToken = secondTokens[j];
                double found = internalStringMetric.GetSimilarity(sToken, tToken);
                if (found > maxFound) {
                    maxFound = found;
                }
            }
            sumMatches += maxFound;
        }
        return sumMatches / firstTokens.Count;
    }
    return defaultMismatchScore;
}
```

MongeElkan.GetSimilarityExplained

gets a div class xhtml similarity explaining the operation of the metric.

```
public override string GetSimilarityExplained(string firstWord, string secondWord);
```

Returns

a div class html section detailing the metric operation.

Body Source

```
public override string GetSimilarityExplained(string firstWord, string secondWord) {
    throw new NotImplementedException();
}
```

MongeElkan.GetSimilarityTimingEstimated

gets the estimated time in milliseconds it takes to perform a similarity timing.

```
public override double GetSimilarityTimingEstimated(string firstWord, string secondWord);
```

Returns

the estimated time in milliseconds taken to perform the similarity measure

Body Source

```
public override double GetSimilarityTimingEstimated(string firstWord, string secondWord) {
    if ((firstWord != null) && (secondWord != null)) {
        double firstTokens = tokeniser.Tokenize(firstWord).Count;
        double secondTokens = tokeniser.Tokenize(secondWord).Count;
        return
            ((firstTokens + secondTokens) * firstTokens + (firstTokens + secondTokens) * secondTokens) *
            estimatedTimingConstant;
    }
    return 0.0;
}
```

MongeElkan.GetUnnormalisedSimilarity

gets the un-normalised similarity measure of the metric for the given strings.

```
public override double GetUnnormalisedSimilarity(string firstWord, string secondWord);
```

Returns

returns the score of the similarity measure (un-normalised)

Body Source

```
public override double GetUnnormalisedSimilarity(string firstWord, string secondWord) {
    // todo check this is valid before use mail sam@dcs.shef.ac.uk if problematic
    return GetSimilarity(firstWord, secondWord);
}
```

MongeElkan.MongeElkan

basic constructor

```
public MongeElkan();
```

Body Source

```
public MongeElkan() : this(new TokeniserWhitespace()) {}
```

MongeElkan.MongeElkan

constructor taking metric to use

```
public MongeElkan(AbstractStringMetric metricToUse);
```

Body Source

```
public MongeElkan(AbstractStringMetric metricToUse) {
    tokeniser = new TokeniserWhitespace();
    internalStringMetric = metricToUse;
}
```

MongeElkan.MongeElkan

constructor taking a tokeniser to use

```
public MongeElkan(ITokeniser tokeniserToUse);
```

Body Source

```
public MongeElkan(ITokeniser tokeniserToUse) {
    tokeniser = tokeniserToUse;
    internalStringMetric = new SmithWatermanGotoh();
}
```

MongeElkan.MongeElkan

constructor taking a tokeniser and string metric to use

```
public MongeElkan(ITokeniser tokeniserToUse, AbstractStringMetric metricToUse);
```

Body Source

```
public MongeElkan(ITokeniser tokeniserToUse, AbstractStringMetric metricToUse) {
    tokeniser = tokeniserToUse;
    internalStringMetric = metricToUse;
}
```

1.2.13 NeedlemanWunch

needlemanwunch implements an edit distance function

Class Hierarchy

```
AbstractStringMetric
    NeedlemanWunch
```

```
[Serializable]
public sealed class NeedlemanWunch : AbstractStringMetric;
```

Methods

Method	Description
GetSimilarity (see page 49)	gets the similarity of the two strings using Needleman Wunch distance.
GetSimilarityExplained (see page 50)	gets a div class xhtml similarity explaining the operation of the metric.
GetSimilarityTimingEstimated (see page 50)	gets the estimated time in milliseconds it takes to perform a similarity timing.
GetUnnormalisedSimilarity (see page 50)	gets the un-normalised similarity measure of the metric for the given strings.
NeedlemanWunch (see page 51)	constructor
NeedlemanWunch (see page 51)	constructor
NeedlemanWunch (see page 51)	constructor
NeedlemanWunch (see page 51)	constructor

Properties

Property	Description
DCostFunction (see page 49)	set/get the d(i,j) cost function.
GapCost (see page 49)	sets/gets the gap cost for the distance function.
LongDescriptionString (see page 49)	returns the long string identifier for the metric.
ShortDescriptionString (see page 49)	returns the string identifier for the metric.

NeedlemanWunch.DCostFunction

set/get the d(i,j) cost function.

```
public AbstractSubstitutionCost DCostFunction;
```

NeedlemanWunch.GapCost

sets/gets the gap cost for the distance function.

```
public double GapCost;
```

NeedlemanWunch.LongDescriptionString

returns the long string identifier for the metric.

```
public override string LongDescriptionString;
```

NeedlemanWunch.ShortDescriptionString

returns the string identifier for the metric.

```
public override string ShortDescriptionString;
```

NeedlemanWunch.GetSimilarity

gets the similarity of the two strings using Needleman Wunch distance.

```
public override double GetSimilarity(string firstWord, string secondWord);
```

Returns

a value between 0-1 of the similarity

Body Source

```
public override double GetSimilarity(string firstWord, string secondWord) {
    if ((firstWord != null) && (secondWord != null)) {
        double needlemanWunch = GetUnnormalisedSimilarity(firstWord, secondWord);
        double maxValue = Math.Max(firstWord.Length, secondWord.Length);
        double minValue = maxValue;
        if (dCostFunction.MaxCost > gapCost) {
            maxValue *= dCostFunction.MaxCost;
        }
```

```
            else {
                maxValue *= gapCost;
            }
            if (dCostFunction.MinCost < gapCost) {
                minValue *= dCostFunction.MinCost;
            }
            else {
                minValue *= gapCost;
            }
            if (minValue < defaultMismatchScore) {
                maxValue -= minValue;
                needlemanWunch -= minValue;
            }
            if (maxValue == defaultMismatchScore) {
                return defaultPerfectMatchScore;
            }
            else {
                return defaultPerfectMatchScore - needlemanWunch / maxValue;
            }
        }
        return defaultMismatchScore;
    }
```

NeedlemanWunch.GetSimilarityExplained

gets a div class xhtml similarity explaining the operation of the metric.

```
public override string GetSimilarityExplained(string firstWord, string secondWord);
```

Returns

a div class html section detailing the metric operation.

Body Source

```
public override string GetSimilarityExplained(string firstWord, string secondWord) {
    throw new NotImplementedException();
}
```

NeedlemanWunch.GetSimilarityTimingEstimated

gets the estimated time in milliseconds it takes to perform a similarity timing.

```
public override double GetSimilarityTimingEstimated(string firstWord, string secondWord);
```

Returns

the estimated time in milliseconds taken to perform the similarity measure

Body Source

```
public override double GetSimilarityTimingEstimated(string firstWord, string secondWord) {
    if ((firstWord != null) && (secondWord != null)) {
        double firstLength = firstWord.Length;
        double secondLength = secondWord.Length;
        return firstLength * secondLength * estimatedTimingConstant;
    }
    return defaultMismatchScore;
}
```

NeedlemanWunch.GetUnnormalisedSimilarity

gets the un-normalised similarity measure of the metric for the given strings.

```
public override double GetUnnormalisedSimilarity(string firstWord, string secondWord);
```

Returns

returns the score of the similarity measure (un-normalised)

Body Source

```
public override double GetUnnormalisedSimilarity(string firstWord, string secondWord) {
    if ((firstWord != null) && (secondWord != null)) {
        int n = firstWord.Length;
        int m = secondWord.Length;
        if (n == 0) {
            return m;
        }
        if (m == 0) {
            return n;
        }
        double[][] d = new double[n + 1][];
        for (int i = 0; i < n + 1; i++) {
            d[i] = new double[m + 1];
        }
        for (int i = 0; i <= n; i++) {
            d[i][0] = i;
        }
```

```
            for (int j = 0; j <= m; j++) {
                d[0][j] = j;
            }

            for (int i = 1; i <= n; i++) {
                for (int j = 1; j <= m; j++) {
                    double cost = dCostFunction.GetCost(firstWord, i - 1, secondWord, j - 1);
                    d[i][j] = MathFunctions.MinOf3(d[i - 1][j] + gapCost, d[i][j - 1] + gapCost, d[i - 1][j - 1] + cost);
                }
            }

            return d[n][m];
        }
        return 0.0;
    }
```

NeedlemanWunch.NeedlemanWunch

constructor

`public NeedlemanWunch();`

Body Source

```
public NeedlemanWunch() : this(defaultGapCost, new SubCostRange0To1()) {}
```

NeedlemanWunch.NeedlemanWunch

constructor

`public NeedlemanWunch(AbstractSubstitutionCost costFunction);`

Body Source

```
public NeedlemanWunch(AbstractSubstitutionCost costFunction) : this(defaultGapCost, costFunction) {}
```

NeedlemanWunch.NeedlemanWunch

constructor

`public NeedlemanWunch(double costG);`

Body Source

```
public NeedlemanWunch(double costG) : this(costG, new SubCostRange0To1()) {}
```

NeedlemanWunch.NeedlemanWunch

constructor

`public NeedlemanWunch(double costG, AbstractSubstitutionCost costFunction);`

Body Source

```
public NeedlemanWunch(double costG, AbstractSubstitutionCost costFunction) {
    gapCost = costG;
    dCostFunction = costFunction;
}
```

1.2.14 OverlapCoefficient

This is class SimMetricsMetricUtilities.OverlapCoefficient.

Class Hierarchy

```
AbstractStringMetric
    OverlapCoefficient
```

```
[Serializable]
public sealed class OverlapCoefficient : AbstractStringMetric;
```

Methods

Method	Description
GetSimilarity (see page 52)	gets the similarity of the two strings using OverlapCoefficient
GetSimilarityExplained (see page 52)	gets a div class xhtml similarity explaining the operation of the metric.
GetSimilarityTimingEstimated (see page 53)	gets the estimated time in milliseconds it takes to perform a similarity timing.
GetUnnormalisedSimilarity (see page 53)	gets the un-normalised similarity measure of the metric for the given strings.
OverlapCoefficient (see page 53)	constructor
OverlapCoefficient (see page 53)	Constructor

Properties

Property	Description
LongDescriptionString (see page 52)	returns the long string identifier for the metric.
ShortDescriptionString (see page 52)	returns the string identifier for the metric.

OverlapCoefficient.LongDescriptionString

returns the long string identifier for the metric.

```
public override string LongDescriptionString;
```

OverlapCoefficient.ShortDescriptionString

returns the string identifier for the metric.

```
public override string ShortDescriptionString;
```

OverlapCoefficient.GetSimilarity

gets the similarity of the two strings using OverlapCoefficient (see page 52)

Remarks

overlap_coefficient(q,r) = (| q and r |) / min{ | q | , | r | }.

```
public override double GetSimilarity(string firstWord, string secondWord);
```

Returns

a value between 0-1 of the similarity

Body Source

```
public override double GetSimilarity(string firstWord, string secondWord) {
    if ((firstWord != null) && (secondWord != null)) {
        //Collection<string> allTokens =
        tokenUtilities.CreateMergedSet(tokeniser.Tokenize(firstWord), tokeniser.Tokenize(secondWord));
        return
            tokenUtilities.CommonSetTerms() /
            (double)Math.Min(tokenUtilities.FirstSetTokenCount, tokenUtilities.SecondSetTokenCount);
    }
    return defaultMismatchScore;
}
```

OverlapCoefficient.GetSimilarityExplained

gets a div class xhtml similarity explaining the operation of the metric.

```
public override string GetSimilarityExplained(string firstWord, string secondWord);
```

Returns

a div class html section detailing the metric operation.

Body Source

```
public override string GetSimilarityExplained(string firstWord, string secondWord) {
    throw new NotImplementedException();
}
```

OverlapCoefficient.GetSimilarityTimingEstimated

gets the estimated time in milliseconds it takes to perform a similarity timing.

```
public override double GetSimilarityTimingEstimated(string firstWord, string secondWord);
```

Returns

the estimated time in milliseconds taken to perform the similarity measure

Body Source

```
public override double GetSimilarityTimingEstimated(string firstWord, string secondWord) {
    if ((firstWord != null) && (secondWord != null)) {
        double firstTokens = tokeniser.Tokenize(firstWord).Count;
        double secondTokens = tokeniser.Tokenize(secondWord).Count;
        return firstTokens * secondTokens * estimatedTimingConstant;
    }
    return 0.0;
}
```

OverlapCoefficient.GetUnnormalisedSimilarity

gets the un-normalised similarity measure of the metric for the given strings.

```
public override double GetUnnormalisedSimilarity(string firstWord, string secondWord);
```

Returns

returns the score of the similarity measure (un-normalised)

Body Source

```
public override double GetUnnormalisedSimilarity(string firstWord, string secondWord) {
    return GetSimilarity(firstWord, secondWord);
}
```

OverlapCoefficient.OverlapCoefficient

constructor

```
public OverlapCoefficient();
```

Body Source

```
public OverlapCoefficient() : this(new TokeniserWhitespace()) {}
```

OverlapCoefficient.OverlapCoefficient

Constructor

```
public OverlapCoefficient(ITokeniser tokeniserToUse);
```

Body Source

```
public OverlapCoefficient(ITokeniser tokeniserToUse) {
    tokeniser = tokeniserToUse;
    tokenUtilities = new TokeniserUtilities<string>();
}
```

1.2.15 QGramsDistance

implements a QGram distance metric using supplied QGRam tokeniser

Class Hierarchy

```
AbstractStringMetric
    QGramsDistance
```

```
[Serializable]
public sealed class QGramsDistance : AbstractStringMetric;
```

Methods

Method	Description
GetSimilarity (see page 54)	gets the similarity of the two strings using QGramsDistance.
GetSimilarityExplained (see page 54)	gets a div class xhtml similarity explaining the operation of the metric.
GetSimilarityTimingEstimated (see page 55)	gets the estimated time in milliseconds it takes to perform a similarity timing.
GetUnnormalisedSimilarity (see page 55)	gets the un-normalised similarity measure of the metric for the given strings.
QGramsDistance (see page 55)	constructor - default (empty).
QGramsDistance (see page 55)	the tokeniser to use; should a different tokeniser be required

Properties

Property	Description
LongDescriptionString (see page 54)	returns the long string identifier for the metric.
ShortDescriptionString (see page 54)	returns the string identifier for the metric.

QGramsDistance.LongDescriptionString

returns the long string identifier for the metric.

```
public override string LongDescriptionString;
```

QGramsDistance.ShortDescriptionString

returns the string identifier for the metric.

```
public override string ShortDescriptionString;
```

QGramsDistance.GetSimilarity

gets the similarity of the two strings using QGramsDistance (see page 54).

```
public override double GetSimilarity(string firstWord, string secondWord);
```

Returns

a value between 0-1 of the similarity

Body Source

```
public override double GetSimilarity(string firstWord, string secondWord) {
    if ((firstWord != null) && (secondWord != null)) {
        double difference = GetUnnormalisedSimilarity(firstWord, secondWord);
        int maxQGramsMatching = tokenUtilities.FirstTokenCount + tokenUtilities.SecondTokenCount;

        return (maxQGramsMatching == 0) ? defaultMismatchScore : ((maxQGramsMatching - difference) /
maxQGramsMatching);
    }
    return defaultMismatchScore;
}
```

QGramsDistance.GetSimilarityExplained

gets a div class xhtml similarity explaining the operation of the metric.

```
public override string GetSimilarityExplained(string firstWord, string secondWord);
```

Returns

a div class html section detailing the metric operation.

Body Source

```
public override string GetSimilarityExplained(string firstWord, string secondWord) {
    throw new NotImplementedException();
}
```

QGramsDistance.GetSimilarityTimingEstimated

gets the estimated time in milliseconds it takes to perform a similarity timing.

```
public override double GetSimilarityTimingEstimated(string firstWord, string secondWord);
```

Returns

the estimated time in milliseconds taken to perform the similarity measure

Body Source

```
public override double GetSimilarityTimingEstimated(string firstWord, string secondWord) {
    if ((firstWord != null) && (secondWord != null)) {
        double firstLength = firstWord.Length;
        double secondLength = secondWord.Length;
        return firstLength * secondLength * estimatedTimingConstant;
    }
    return 0.0;
}
```

QGramsDistance.GetUnnormalisedSimilarity

gets the un-normalised similarity measure of the metric for the given strings.

```
public override double GetUnnormalisedSimilarity(string firstWord, string secondWord);
```

Returns

returns the score of the similarity measure (un-normalised)

Body Source

```
public override double GetUnnormalisedSimilarity(string firstWord, string secondWord) {
    Collection<string> firstTokens = tokeniser.Tokenize(firstWord);
    Collection<string> secondTokens = tokeniser.Tokenize(secondWord);
    tokenUtilities.CreateMergedList(firstTokens, secondTokens);
    return GetActualSimilarity(firstTokens, secondTokens);
}
```

QGramsDistance.QGramsDistance

constructor - default (empty).

```
public QGramsDistance();
```

Body Source

```
public QGramsDistance() : this(new TokeniserQGram3Extended()) {}
```

QGramsDistance.QGramsDistance

the tokeniser to use; should a different tokeniser be required

```
public QGramsDistance(ITokeniser tokeniserToUse);
```

Body Source

```
public QGramsDistance(ITokeniser tokeniserToUse) {
    tokeniser = tokeniserToUse;
    tokenUtilities = new TokeniserUtilities<string>();
}
```

1.2.16 SmithWaterman

implements the Smith-Waterman edit distance function

Class Hierarchy

```
AbstractStringMetric
    SmithWaterman
```

```
[Serializable]
public sealed class SmithWaterman : AbstractStringMetric;
```

Methods

Method	Description
GetSimilarity (see page 56)	gets the similarity of the two strings using Smith Waterman distance.
GetSimilarityExplained (see page 57)	gets a div class xhtml similarity explaining the operation of the metric.
GetSimilarityTimingEstimated (see page 57)	gets the estimated time in milliseconds it takes to perform a similarity timing.
GetUnnormalisedSimilarity (see page 57)	gets the un-normalised similarity measure of the metric for the given strings.
SmithWaterman (see page 58)	constructor - default (empty).
SmithWaterman (see page 58)	constructor
SmithWaterman (see page 58)	constructor
SmithWaterman (see page 58)	constructor

Properties

Property	Description
DCostFunction (see page 56)	get the d(i,j) cost function.
GapCost (see page 56)	the gap cost for the distance function.
LongDescriptionString (see page 56)	returns the long string identifier for the metric.
ShortDescriptionString (see page 56)	returns the string identifier for the metric .

SmithWaterman.DCostFunction

get the d(i,j) cost function.

```
public AbstractSubstitutionCost DCostFunction;
```

SmithWaterman.GapCost

the gap cost for the distance function.

```
public double GapCost;
```

SmithWaterman.LongDescriptionString

returns the long string identifier for the metric.

```
public override string LongDescriptionString;
```

SmithWaterman.ShortDescriptionString

returns the string identifier for the metric .

```
public override string ShortDescriptionString;
```

SmithWaterman.GetSimilarity

gets the similarity of the two strings using Smith Waterman distance.

```
public override double GetSimilarity(string firstWord, string secondWord);
```

Returns

a value between 0-1 of the similarity

Body Source

```
public override double GetSimilarity(string firstWord, string secondWord) {
    if ((firstWord != null) && (secondWord != null)) {
        double smithWaterman = GetUnnormalisedSimilarity(firstWord, secondWord);
        double maxValue = Math.Min(firstWord.Length, secondWord.Length);
        if (dCostFunction.MaxCost > -gapCost) {
            maxValue *= dCostFunction.MaxCost;
        }
        else {
            maxValue *= (-gapCost);
        }
        if (maxValue == defaultMismatchScore) {
            return defaultPerfectMatchScore;
        }
        else {
            return smithWaterman / maxValue;
        }
    }
    return defaultMismatchScore;
}
```

SmithWaterman.GetSimilarityExplained

gets a div class xhtml similarity explaining the operation of the metric.

```
public override string GetSimilarityExplained(string firstWord, string secondWord);
```

Returns

a div class html section detailing the metric operation.

Body Source

```
public override string GetSimilarityExplained(string firstWord, string secondWord) {
    throw new NotImplementedException();
}
```

SmithWaterman.GetSimilarityTimingEstimated

gets the estimated time in milliseconds it takes to perform a similarity timing.

```
public override double GetSimilarityTimingEstimated(string firstWord, string secondWord);
```

Returns

the estimated time in milliseconds taken to perform the similarity measure

Body Source

```
public override double GetSimilarityTimingEstimated(string firstWord, string secondWord) {
    if ((firstWord != null) && (secondWord != null)) {
        double firstLength = firstWord.Length;
        double secondLength = secondWord.Length;
        return (firstLength * secondLength + firstLength + secondLength) * estimatedTimingConstant;
    }
    return 0.0;
}
```

SmithWaterman.GetUnnormalisedSimilarity

gets the un-normalised similarity measure of the metric for the given strings.

```
public override double GetUnnormalisedSimilarity(string firstWord, string secondWord);
```

Returns

returns the score of the similarity measure (un-normalised)

Body Source

```
public override double GetUnnormalisedSimilarity(string firstWord, string secondWord) {
    if ((firstWord != null) && (secondWord != null)) {
        int n = firstWord.Length;
        int m = secondWord.Length;
        if (n == 0) {
            return m;
        }
        if (m == 0) {
            return n;
        }
        double[][] d = new double[n][];
        for (int i = 0; i < n; i++) {
            d[i] = new double[m];
        }
        double maxSoFar = defaultMismatchScore;
        for (int i = 0; i < n; i++) {
            double cost = dCostFunction.GetCost(firstWord, i, secondWord, 0);
            if (i == 0) {
                d[0][0] = MathFunctions.MaxOf3(defaultMismatchScore, -gapCost, cost);
            }
            else {
                d[i][0] = MathFunctions.MaxOf3(defaultMismatchScore, d[i - 1][0] - gapCost, cost);
            }
            if (d[i][0] > maxSoFar) {
                maxSoFar = d[i][0];
            }
        }
```

```
            for (int j = 0; j < m; j++) {
                double cost = dCostFunction.GetCost(firstWord, 0, secondWord, j);
                if (j == 0) {
                    d[0][0] = MathFunctions.MaxOf3(defaultMismatchScore, -gapCost, cost);
                }
                else {
                    d[0][j] = MathFunctions.MaxOf3(defaultMismatchScore, d[0][j - 1] - gapCost, cost);
                }
                if (d[0][j] > maxSoFar) {
                    maxSoFar = d[0][j];
                }
            }

            for (int i = 1; i < n; i++) {
                for (int j = 1; j < m; j++) {
                    double cost = dCostFunction.GetCost(firstWord, i, secondWord, j);
                    d[i][j] =
                        MathFunctions.MaxOf4(defaultMismatchScore, d[i - 1][j] - gapCost, d[i][j - 1] - gapCost,
                                             d[i - 1][j - 1] + cost);
                    if (d[i][j] > maxSoFar) {
                        maxSoFar = d[i][j];
                    }
                }
            }

            return maxSoFar;
        }
        return 0.0;
    }
```

SmithWaterman.SmithWaterman

constructor - default (empty).

```
public SmithWaterman();
```

Body Source

```
public SmithWaterman() : this(defaultGapCost, new SubCostRange1ToMinus2()) {}
```

SmithWaterman.SmithWaterman

constructor

```
public SmithWaterman(AbstractSubstitutionCost costFunction);
```

Body Source

```
public SmithWaterman(AbstractSubstitutionCost costFunction) : this(defaultGapCost, costFunction) {}
```

SmithWaterman.SmithWaterman

constructor

```
public SmithWaterman(double costG);
```

Body Source

```
public SmithWaterman(double costG) : this(costG, new SubCostRange1ToMinus2()) {}
```

SmithWaterman.SmithWaterman

constructor

```
public SmithWaterman(double costG, AbstractSubstitutionCost costFunction);
```

Body Source

```
public SmithWaterman(double costG, AbstractSubstitutionCost costFunction) {
    gapCost = costG;
    dCostFunction = costFunction;
}
```

1.2.17 SmithWatermanGotoh

implements the Gotoh extension of the smith waterman method incorporating affine gaps in the strings

Class Hierarchy

```
AbstractStringMetric
    SimMetricsMetricUtilities.SmithWatermanGotohWindowedAffine
        SmithWatermanGotoh
```

```
[Serializable]
public sealed class SmithWatermanGotoh : SmithWatermanGotohWindowedAffine;
```

Methods

Method	Description
GetSimilarityTimingEstimated (see page 59)	gets the estimated time in milliseconds it takes to perform a similarity timing.
SmithWatermanGotoh (see page 59)	constructor - default (empty).
SmithWatermanGotoh (see page 59)	constructor
SmithWatermanGotoh (see page 60)	constructor
SmithWatermanGotoh (see page 60)	constructor

Properties

Property	Description
LongDescriptionString (see page 59)	returns the long string identifier for the metric.
ShortDescriptionString (see page 59)	returns the string identifier for the metric.

SmithWatermanGotoh.LongDescriptionString

returns the long string identifier for the metric.

```
public override string LongDescriptionString;
```

SmithWatermanGotoh.ShortDescriptionString

returns the string identifier for the metric.

```
public override string ShortDescriptionString;
```

SmithWatermanGotoh.GetSimilarityTimingEstimated

gets the estimated time in milliseconds it takes to perform a similarity timing.

```
public override double GetSimilarityTimingEstimated(string firstWord, string secondWord);
```

Returns

the estimated time in milliseconds taken to perform the similarity measure

Body Source

```
public override double GetSimilarityTimingEstimated(string firstWord, string secondWord) {
    if ((firstWord != null) && (secondWord != null)) {
        double firstLength = firstWord.Length;
        double secondLength = secondWord.Length;
        return
            (firstLength * secondLength * firstLength + firstLength * secondLength * secondLength) *
            estimatedTimingConstant;
    }
    return 0.0;
}
```

SmithWatermanGotoh.SmithWatermanGotoh

constructor - default (empty).

```
public SmithWatermanGotoh();
```

Body Source

```
public SmithWatermanGotoh()
: base(new AffineGapRange5To0Multiplier1(), new SubCostRange5ToMinus3(), affineGapWindowSize) {}
```

SmithWatermanGotoh.SmithWatermanGotoh

constructor

```
public SmithWatermanGotoh(AbstractAffineGapCost gapCostFunction);
```

Body Source

```
public SmithWatermanGotoh(AbstractAffineGapCost gapCostFunction)
: base(gapCostFunction, new SubCostRange5ToMinus3(), affineGapWindowSize) {}
```

SmithWatermanGotoh.SmithWatermanGotoh

constructor

```
public SmithWatermanGotoh(AbstractAffineGapCost gapCostFunction, AbstractSubstitutionCost
costFunction);
```

Body Source

```
public SmithWatermanGotoh(AbstractAffineGapCost gapCostFunction, AbstractSubstitutionCost costFunction)
: base(gapCostFunction, costFunction, affineGapWindowSize) {}
```

SmithWatermanGotoh.SmithWatermanGotoh

constructor

```
public SmithWatermanGotoh(AbstractSubstitutionCost costFunction);
```

Body Source

```
public SmithWatermanGotoh(AbstractSubstitutionCost costFunction)
: base(new AffineGapRange5To0Multiplier1(), costFunction, affineGapWindowSize) {}
```

1.2.18 SmithWatermanGotohWindowedAffine

implements the smith waterman with gotoh extension using a windowed affine gap.

Class Hierarchy

```
AbstractStringMetric
    SmithWatermanGotohWindowedAffine
        SimMetricsMetricUtilities.SmithWatermanGotoh
```

```
[Serializable]
public class SmithWatermanGotohWindowedAffine : AbstractStringMetric;
```

Methods

Method	Description
GetSimilarity (see page 61)	gets the similarity of the two strings using Smith-Waterman-Gotoh distance.
GetSimilarityExplained (see page 62)	gets a div class xhtml similarity explaining the operation of the metric.
GetSimilarityTimingEstimated (see page 62)	gets the estimated time in milliseconds it takes to perform a similarity timing.
GetUnnormalisedSimilarity (see page 62)	gets the un-normalised similarity measure of the metric for the given strings.
SmithWatermanGotohWindowedAffine (see page 64)	constructor - default (empty).
SmithWatermanGotohWindowedAffine (see page 64)	constructor
SmithWatermanGotohWindowedAffine (see page 64)	constructor
SmithWatermanGotohWindowedAffine (see page 64)	constructor
SmithWatermanGotohWindowedAffine (see page 64)	constructor
SmithWatermanGotohWindowedAffine (see page 64)	constructor
SmithWatermanGotohWindowedAffine (see page 64)	constructor
SmithWatermanGotohWindowedAffine (see page 64)	constructor

Properties

Property	Description
DCostFunction (see page 61)	get the d(i,j) cost function.
GGapFunction (see page 61)	get the g gap cost function.
LongDescriptionString (see page 61)	returns the long string identifier for the metric.
ShortDescriptionString (see page 61)	returns the string identifier for the metric.

SmithWatermanGotohWindowedAffine.DCostFunction

get the d(i,j) cost function.

```
public AbstractSubstitutionCost DCostFunction;
```

SmithWatermanGotohWindowedAffine.GGapFunction

get the g gap cost function.

```
public AbstractAffineGapCost GGapFunction;
```

SmithWatermanGotohWindowedAffine.LongDescriptionString

returns the long string identifier for the metric.

```
public override string LongDescriptionString;
```

SmithWatermanGotohWindowedAffine.ShortDescriptionString

returns the string identifier for the metric.

```
public override string ShortDescriptionString;
```

SmithWatermanGotohWindowedAffine.GetSimilarity

gets the similarity of the two strings using Smith-Waterman-Gotoh distance.

```
public override double GetSimilarity(string firstWord, string secondWord);
```

Returns

a value between 0-1 of the similarity

Body Source

```
public override double GetSimilarity(string firstWord, string secondWord) {
    if ((firstWord != null) && (secondWord != null)) {
        double smithWatermanGotoh = GetUnnormalisedSimilarity(firstWord, secondWord);
        double maxValue = Math.Min(firstWord.Length, secondWord.Length);
        if (dCostFunction.MaxCost > -gGapFunction.MaxCost) {
            maxValue *= dCostFunction.MaxCost;
        }
        else {
            maxValue *= (-gGapFunction.MaxCost);
        }
        if (maxValue == defaultMismatchScore) {
            return defaultPerfectScore;
        }
        else {
            return smithWatermanGotoh / maxValue;
        }
    }
    return defaultMismatchScore;
}
```

SmithWatermanGotohWindowedAffine.GetSimilarityExplained

gets a div class xhtml similarity explaining the operation of the metric.

```
public override string GetSimilarityExplained(string firstWord, string secondWord);
```

Returns

a div class html section detailing the metric operation.

Body Source

```
public override string GetSimilarityExplained(string firstWord, string secondWord) {
    throw new NotImplementedException();
}
```

SmithWatermanGotohWindowedAffine.GetSimilarityTimingEstimated

gets the estimated time in milliseconds it takes to perform a similarity timing.

```
public override double GetSimilarityTimingEstimated(string firstWord, string secondWord);
```

Returns

the estimated time in milliseconds taken to perform the similarity measure

Body Source

```
public override double GetSimilarityTimingEstimated(string firstWord, string secondWord) {
    if ((firstWord != null) && (secondWord != null)) {
        double firstLength = firstWord.Length;
        double secondLength = secondWord.Length;
        return
            (firstLength * secondLength * windowSize + firstLength * secondLength * windowSize) *
            estimatedTimingConstant;
    }
    return 0.0;
}
```

SmithWatermanGotohWindowedAffine.GetUnnormalisedSimilarity

gets the un-normalised similarity measure of the metric for the given strings.

```
public override double GetUnnormalisedSimilarity(string firstWord, string secondWord);
```

Returns

returns the score of the similarity measure (un-normalised)

Body Source

```
public override double GetUnnormalisedSimilarity(string firstWord, string secondWord) {
    if ((firstWord != null) && (secondWord != null)) {
        int n = firstWord.Length;
        int m = secondWord.Length;
        // check for zero length input
        if (n == 0) {
            return m;
        }
        if (m == 0) {
            return n;
        }
        double[][] d = new double[n][];
        for (int i = 0; i < n; i++) {
            d[i] = new double[m];
        }
        //process first row and column first as no need to consider previous rows/columns
        double maxSoFar = 0.0;
        for (int i = 0; i < n; i++) {
            // get the substution cost
            double cost = dCostFunction.GetCost(firstWord, i, secondWord, 0);
            if (i == 0) {
                d[0][0] = Math.Max(defaultMismatchScore, cost);
            }
            else {
                double maxGapCost = defaultMismatchScore;
                int windowStart = i - windowSize;
                if (windowStart < 1) {
                    windowStart = 1;
                }
                for (int k = windowStart; k < i; k++) {
                    maxGapCost = Math.Max(maxGapCost, d[i - k][0] - gGapFunction.GetCost(firstWord, i - k, i));
                }

                d[i][0] = MathFunctions.MaxOf3(defaultMismatchScore, maxGapCost, cost);
            }
            //update max possible if available
            if (d[i][0] > maxSoFar) {
                maxSoFar = d[i][0];
            }
        }

        for (int j = 0; j < m; j++) {
            // get the substution cost
            double cost = dCostFunction.GetCost(firstWord, 0, secondWord, j);
            if (j == 0) {
                d[0][0] = Math.Max(defaultMismatchScore, cost);
            }
            else {
                double maxGapCost = defaultMismatchScore;
                int windowStart = j - windowSize;
                if (windowStart < 1) {
                    windowStart = 1;
                }
                for (int k = windowStart; k < j; k++) {
                    maxGapCost = Math.Max(maxGapCost, d[0][j - k] - gGapFunction.GetCost(secondWord, j - k, j));
                }

                d[0][j] = MathFunctions.MaxOf3(defaultMismatchScore, maxGapCost, cost);
            }
            //update max possible if available
            if (d[0][j] > maxSoFar) {
                maxSoFar = d[0][j];
            }
        }

        // cycle through rest of table filling values from the lowest cost value of the three part cost function
        for (int i = 1; i < n; i++) {
            for (int j = 1; j < m; j++) {
                // get the substution cost
                double cost = dCostFunction.GetCost(firstWord, i, secondWord, j);
                // find lowest cost at point from three possible
                double maxGapCost1 = defaultMismatchScore;
                double maxGapCost2 = defaultMismatchScore;
                int windowStart = i - windowSize;
                if (windowStart < 1) {
                    windowStart = 1;
                }
                for (int k = windowStart; k < i; k++) {
                    maxGapCost1 = Math.Max(maxGapCost1, d[i - k][j] - gGapFunction.GetCost(firstWord, i - k, i));
                }

                windowStart = j - windowSize;
                if (windowStart < 1) {
                    windowStart = 1;
                }
                for (int k = windowStart; k < j; k++) {
                    maxGapCost2 = Math.Max(maxGapCost2, d[i][j - k] - gGapFunction.GetCost(secondWord, j - k, j));
                }

                d[i][j] = MathFunctions.MaxOf4(defaultMismatchScore, maxGapCost1, maxGapCost2, d[i - 1][j - 1] +
cost);
                if (d[i][j] > maxSoFar) {
                    maxSoFar = d[i][j];
                }
            }
        }

        // return max value within matrix as holds the maximum edit score
        return maxSoFar;
    }
    return 0.0;
}
```

SmithWatermanGotohWindowedAffine.SmithWatermanGotohWindowedAffine

constructor - default (empty).

```
public SmithWatermanGotohWindowedAffine();
```

Body Source

```
public SmithWatermanGotohWindowedAffine()
: this(new AffineGapRange5To0Multiplier1(), new SubCostRange5ToMinus3(), defaultWindowSize) {}
```

SmithWatermanGotohWindowedAffine.SmithWatermanGotohWindowedAffine

constructor

```
public SmithWatermanGotohWindowedAffine(AbstractAffineGapCost gapCostFunction);
```

Body Source

```
public SmithWatermanGotohWindowedAffine(AbstractAffineGapCost gapCostFunction)
: this(gapCostFunction, new SubCostRange5ToMinus3(), defaultWindowSize) {}
```

SmithWatermanGotohWindowedAffine.SmithWatermanGotohWindowedAffine

constructor

```
public SmithWatermanGotohWindowedAffine(AbstractAffineGapCost gapCostFunction,
AbstractSubstitutionCost costFunction);
```

Body Source

```
public SmithWatermanGotohWindowedAffine(AbstractAffineGapCost gapCostFunction, AbstractSubstitutionCost costFunction)
: this(gapCostFunction, costFunction, defaultWindowSize) {}
```

SmithWatermanGotohWindowedAffine.SmithWatermanGotohWindowedAffine

constructor

```
public SmithWatermanGotohWindowedAffine(AbstractAffineGapCost gapCostFunction,
AbstractSubstitutionCost costFunction, int affineGapWindowSize);
```

Body Source

```
public SmithWatermanGotohWindowedAffine(AbstractAffineGapCost gapCostFunction, AbstractSubstitutionCost costFunction,
                                        int affineGapWindowSize) {
    gGapFunction = gapCostFunction;
    dCostFunction = costFunction;
    windowSize = affineGapWindowSize;
}
```

SmithWatermanGotohWindowedAffine.SmithWatermanGotohWindowedAffine

constructor

```
public SmithWatermanGotohWindowedAffine(AbstractAffineGapCost gapCostFunction, int
affineGapWindowSize);
```

Body Source

```
public SmithWatermanGotohWindowedAffine(AbstractAffineGapCost gapCostFunction, int affineGapWindowSize)
: this(gapCostFunction, new SubCostRange5ToMinus3(), affineGapWindowSize) {}
```

SmithWatermanGotohWindowedAffine.SmithWatermanGotohWindowedAffine

constructor

```
public SmithWatermanGotohWindowedAffine(AbstractSubstitutionCost costFunction);
```

Body Source

```
public SmithWatermanGotohWindowedAffine(AbstractSubstitutionCost costFunction)
: this(new AffineGapRange5To0Multiplier1(), costFunction, defaultWindowSize) {}
```

SmithWatermanGotohWindowedAffine.SmithWatermanGotohWindowedAffine

constructor

```
public SmithWatermanGotohWindowedAffine(AbstractSubstitutionCost costFunction, int
affineGapWindowSize);
```

Body Source

```
public SmithWatermanGotohWindowedAffine(AbstractSubstitutionCost costFunction, int affineGapWindowSize)
: this(new AffineGapRange5To0Multiplier1(), costFunction, affineGapWindowSize) {}
```

SmithWatermanGotohWindowedAffine.SmithWatermanGotohWindowedAffine

constructor

```
public SmithWatermanGotohWindowedAffine(int affineGapWindowSize);
```

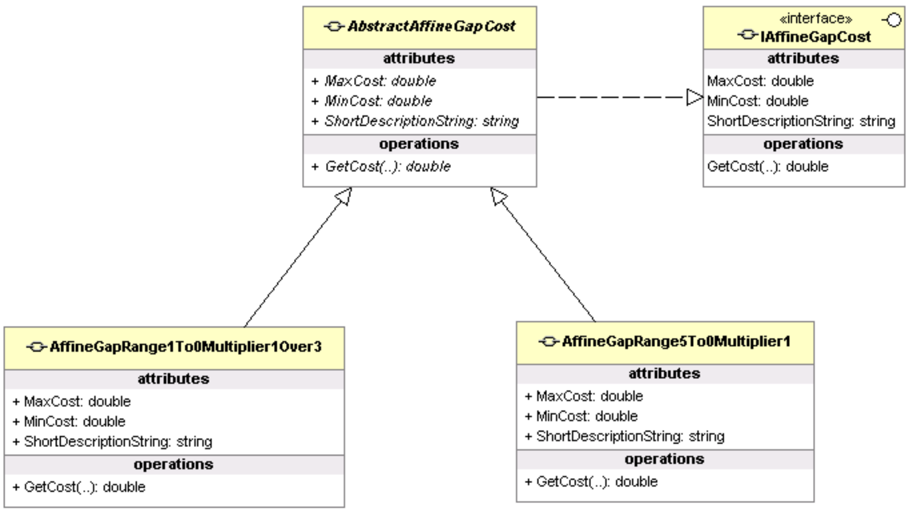
Body Source

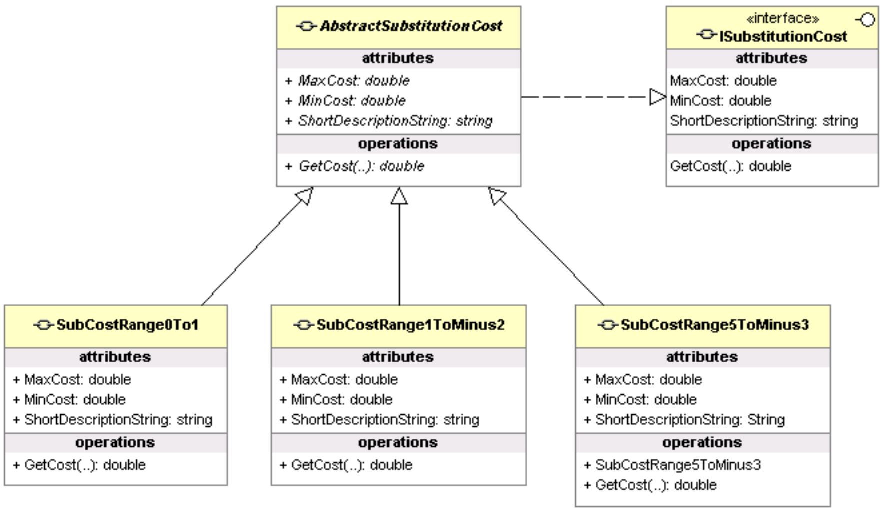
```
public SmithWatermanGotohWindowedAffine(int affineGapWindowSize)
: this(new AffineGapRange5To0Multiplier1(), new SubCostRange5ToMinus3(), affineGapWindowSize) {}
```

1.3 SimMetricsUtilities

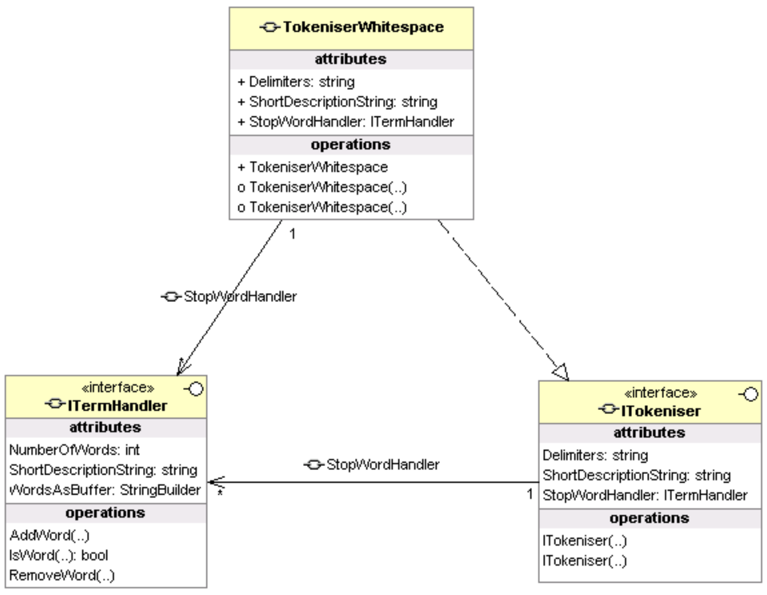
Utility Classes

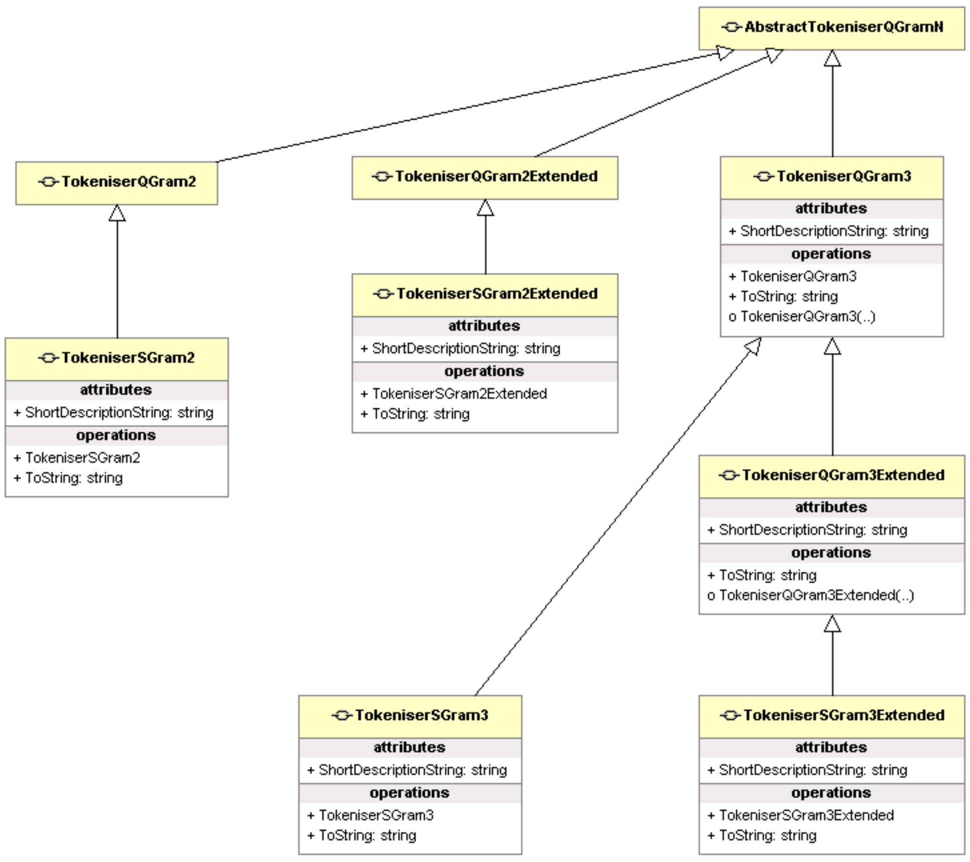
Cost Functions





Tokenisers





Classes

Class	Description
AffineGapRange1To0Multiplier1Over3 (see page 69)	implements a Affine Gap cost function.
AffineGapRange5To0Multiplier1 (see page 70)	implements a affine gap cost function.
DummyStopTermHandler (see page 71)	DummyStopTermHandler implements a dummy stop word handling function whereby no stopwords are considered.

MathFunctions (see page 72)	MathFuncs implements a number of handy maths functions.
SubCostRange0To1 (see page 74)	implements a substitution cost function where d(i,j) = 1 if idoes not equal j, 0 if i equals j.
SubCostRange1ToMinus2 (see page 75)	implements a substitution cost function where d(i,j) = 1 if i does not equal j, -2 if i equals j.
SubCostRange5ToMinus3 (see page 76)	SubCostRange5ToMinus3 implements a cost function as used in Monge Elkan where by an exact match no match or an approximate match whereby a set of characters are in an approximate range. for pairings in {dt} {gj} {lr} {mn} {bpv} {aeiou} {,.}
TokeniserQGram2 (see page 78)	implementaton of the Bigram tokeniser
TokeniserQGram2Extended (see page 80)	implementation of a Bigram tokeniser using extended logic
TokeniserQGram3 (see page 81)	implementaton of the Bigram tokeniser
TokeniserQGram3Extended (see page 83)	implementation of a Bigram tokeniser using extended logic
TokeniserSGram2 (see page 84)	implementaton of the Sgram tokeniser
TokeniserSGram2Extended (see page 85)	implementation of a SGram tokeniser using extended logic
TokeniserSGram3 (see page 86)	implementaton of the Sgram tokeniser
TokeniserSGram3Extended (see page 87)	implementation of a SGram tokeniser using extended logic
TokeniserUtilities (see page 88)	class containing utility functions for the tokenisers to use. these are in two main version collections or sets a collection can contain the same value multiple times ad set can only have the value once.
TokeniserWhitespace (see page 89)	implements a simple whitespace tokeniser.

1.3.1 AffineGapRange1To0Multiplier1Over3

implements a Affine Gap cost function.

Class Hierarchy

```
AbstractAffineGapCost
    AffineGapRange1To0Multiplier1Over3
```

```
[Serializable]
public sealed class AffineGapRange1To0Multiplier1Over3 : AbstractAffineGapCost;
```

Methods

Method	Description
GetCost (see page 69)	get cost between characters.

Properties

Property	Description
MaxCost (see page 69)	returns the maximum possible cost.
MinCost (see page 69)	returns the minimum possible cost.
ShortDescriptionString (see page 69)	returns the name of the cost function.

AffineGapRange1To0Multiplier1Over3.MaxCost

returns the maximum possible cost.

```
public override double MaxCost;
```

AffineGapRange1To0Multiplier1Over3.MinCost

returns the minimum possible cost.

```
public override double MinCost;
```

AffineGapRange1To0Multiplier1Over3.ShortDescriptionString

returns the name of the cost function.

```
public override string ShortDescriptionString;
```

AffineGapRange1To0Multiplier1Over3.GetCost

get cost between characters.

```
public override double GetCost(string textToGap, int stringIndexStartGap, int stringIndexEndGap);
```

Returns

the cost of a Gap G

Body Source

```
public override double GetCost(string textToGap, int stringIndexStartGap, int stringIndexEndGap) {
    if (stringIndexStartGap >= stringIndexEndGap) {
        return charMismatchMatchScore;
    }
    else {
        return charExactMatchScore + (stringIndexEndGap - 1 - stringIndexStartGap) * 0.3333333F;
    }
}
```

1.3.2 AffineGapRange5To0Multiplier1

implements a affine gap cost function.

Class Hierarchy

```
AbstractAffineGapCost
    AffineGapRange5To0Multiplier1
```

```
[Serializable]
public sealed class AffineGapRange5To0Multiplier1 : AbstractAffineGapCost;
```

Methods

Method	Description
GetCost (see page 70)	get cost between characters.

Properties

Property	Description
MaxCost (see page 70)	returns the maximum possible cost.
MinCost (see page 70)	returns the minimum possible cost.
ShortDescriptionString (see page 70)	returns the name of the cost function.

AffineGapRange5To0Multiplier1.MaxCost

returns the maximum possible cost.

```
public override double MaxCost;
```

AffineGapRange5To0Multiplier1.MinCost

returns the minimum possible cost.

```
public override double MinCost;
```

AffineGapRange5To0Multiplier1.ShortDescriptionString

returns the name of the cost function.

```
public override string ShortDescriptionString;
```

AffineGapRange5To0Multiplier1.GetCost

get cost between characters.

```
public override double GetCost(string textToGap, int stringIndexStartGap, int stringIndexEndGap);
```

Returns

the cost of a Gap G

Body Source

```
public override double GetCost(string textToGap, int stringIndexStartGap, int stringIndexEndGap) {
    if (stringIndexStartGap >= stringIndexEndGap) {
        return charMismatchMatchScore;
    }
    else {
        return charExactMatchScore + (stringIndexEndGap - 1 - stringIndexStartGap);
    }
}
```

1.3.3 DummyStopTermHandler

DummyStopTermHandler implements a dummy stop word handling function whereby no stopwords are considered.

Class Hierarchy

```
ITermHandler
    DummyStopTermHandler
```

```
public sealed class DummyStopTermHandler : ITermHandler;
```

Methods

Method	Description
AddWord (see page 71)	adds a word to the intewrface.
IsWord (see page 71)	isStopWord determines if a given term is a stop word or not.
RemoveWord (see page 71)	removes the given stopword from the list.

Properties

Property	Description
NumberOfWords (see page 71)	gets the number of stopwords in the list.
ShortDescriptionString (see page 71)	displays the stopWordHandler method.
WordsAsBuffer (see page 71)	gets the stopwords as a stringBuffer.

DummyStopTermHandler.NumberOfWords

gets the number of stopwords in the list.

```
public int NumberOfWords;
```

DummyStopTermHandler.ShortDescriptionString

displays the stopWordHandler method.

```
public string ShortDescriptionString;
```

DummyStopTermHandler.WordsAsBuffer

gets the stopwords as a stringBuffer.

```
public StringBuilder WordsAsBuffer;
```

DummyStopTermHandler.AddWord

adds a word to the intewrface.

```
public void AddWord(string termToAdd);
```

Body Source

```
public void AddWord(string termToAdd) {}
```

DummyStopTermHandler.IsWord

isStopWord determines if a given term is a stop word or not.

```
public bool IsWord(string termToTest);
```

Returns

always returns false.

Body Source

```
public bool IsWord(string termToTest) {
    return false;
}
```

DummyStopTermHandler.RemoveWord

removes the given stopword from the list.

```
public void RemoveWord(string termToRemove);
```

Body Source

```
public void RemoveWord(string termToRemove) {}
```

1.3.4 MathFunctions

MathFuncs implements a number of handy maths functions.

Class Hierarchy

MathFunctions

```
public static class MathFunctions;
```

Methods

Method	Description
MaxOf3 (see page 72)	returns the max of three numbers.
MaxOf3 (see page 72)	returns the max of three numbers.
MaxOf4 (see page 72)	returns the max of four numbers.
MinOf3 (see page 72)	returns the min of three numbers.
MinOf3 (see page 73)	returns the min of three numbers.

MathFunctions.MaxOf3

returns the max of three numbers.

```
public static double MaxOf3(double firstNumber, double secondNumber, double thirdNumber);
```

Returns

the max of three numbers.

Body Source

```
static public double MaxOf3(double firstNumber, double secondNumber, double thirdNumber) {
    return Math.Max(firstNumber, Math.Max(secondNumber, thirdNumber));
}
```

MathFunctions.MaxOf3

returns the max of three numbers.

```
public static int MaxOf3(int firstNumber, int secondNumber, int thirdNumber);
```

Returns

the max of three numbers.

Body Source

```
static public int MaxOf3(int firstNumber, int secondNumber, int thirdNumber) {
    return Math.Max(firstNumber, Math.Max(secondNumber, thirdNumber));
}
```

MathFunctions.MaxOf4

returns the max of four numbers.

```
public static double MaxOf4(double firstNumber, double secondNumber, double thirdNumber, double
fourthNumber);
```

Returns

the max of four numbers.

Body Source

```
static public double MaxOf4(double firstNumber, double secondNumber, double thirdNumber, double fourthNumber) {
    return Math.Max(Math.Max(firstNumber, secondNumber), Math.Max(thirdNumber, fourthNumber));
}
```

MathFunctions.MinOf3

returns the min of three numbers.

```
public static double MinOf3(double firstNumber, double secondNumber, double thirdNumber);
```

Returns

the min of three numbers.

Body Source

```
static public double MinOf3(double firstNumber, double secondNumber, double thirdNumber) {
    return Math.Min(firstNumber, Math.Min(secondNumber, thirdNumber));
}
```

MathFunctions.MinOf3

returns the min of three numbers.

```
public static int MinOf3(int firstNumber, int secondNumber, int thirdNumber);
```

Returns

the min of three numbers.

Body Source

```
static public int MinOf3(int firstNumber, int secondNumber, int thirdNumber) {
    return Math.Min(firstNumber, Math.Min(secondNumber, thirdNumber));
}
```

1.3.5 SubCostRange0To1

implements a substitution cost function where d(i,j) = 1 if idoes not equal j, 0 if i equals j.

Class Hierarchy

```
AbstractSubstitutionCost
    SubCostRange0To1
```

```
[Serializable]
public sealed class SubCostRange0To1 : AbstractSubstitutionCost;
```

Methods

Method	Description
GetCost (see page 74)	get cost between characters where d(i,j) = 1 if i does not equals j, 0 if i equals j.

Properties

Property	Description
MaxCost (see page 74)	returns the maximum possible cost.
MinCost (see page 74)	returns the minimum possible cost.
ShortDescriptionString (see page 74)	returns the name of the cost function.

SubCostRange0To1.MaxCost

returns the maximum possible cost.

```
public override double MaxCost;
```

SubCostRange0To1.MinCost

returns the minimum possible cost.

```
public override double MinCost;
```

SubCostRange0To1.ShortDescriptionString

returns the name of the cost function.

```
public override string ShortDescriptionString;
```

SubCostRange0To1.GetCost

get cost between characters where d(i,j) = 1 if i does not equals j, 0 if i equals j.

```
public override double GetCost(string firstWord, int firstWordIndex, string secondWord, int
secondWordIndex);
```

Returns

the cost of a given subsitution d(i,j) where d(i,j) = 1 if i!=j, 0 if i==j

Body Source

```
public override double GetCost(string firstWord, int firstWordIndex, string secondWord, int secondWordIndex) {
    if ((firstWord != null) && (secondWord != null)) {
        return firstWord[firstWordIndex] != secondWord[secondWordIndex] ? charExactMatchScore :
charMismatchMatchScore;
    }
    return 0.0;
}
```

1.3.6 SubCostRange1ToMinus2

implements a substitution cost function where d(i,j) = 1 if i does not equal j, -2 if i equals j.

Class Hierarchy

```
AbstractSubstitutionCost
    SubCostRange1ToMinus2
```

```
[Serializable]
public sealed class SubCostRange1ToMinus2 : AbstractSubstitutionCost;
```

Methods

Method	Description
GetCost (see page 75)	get cost between characters where d(i,j) = 1 if i does not equal j, -2 if i equals j.

Properties

Property	Description
MaxCost (see page 75)	returns the maximum possible cost.
MinCost (see page 75)	returns the minimum possible cost.
ShortDescriptionString (see page 75)	returns the name of the cost function.

SubCostRange1ToMinus2.MaxCost

returns the maximum possible cost.

```
public override double MaxCost;
```

SubCostRange1ToMinus2.MinCost

returns the minimum possible cost.

```
public override double MinCost;
```

SubCostRange1ToMinus2.ShortDescriptionString

returns the name of the cost function.

```
public override string ShortDescriptionString;
```

SubCostRange1ToMinus2.GetCost

get cost between characters where d(i,j) = 1 if i does not equal j, -2 if i equals j.

```
public override double GetCost(string firstWord, int firstWordIndex, string secondWord, int
secondWordIndex);
```

Returns

the cost of a given subsitution d(i,j) where d(i,j) = 1 if i!=j, -2 if i==j

Body Source

```
public override double GetCost(string firstWord, int firstWordIndex, string secondWord, int secondWordIndex) {
    if ((firstWord != null) && (secondWord != null)) {
        if (firstWord.Length <= firstWordIndex || firstWordIndex < 0) {
            return charMismatchMatchScore;
        }
        if (secondWord.Length <= secondWordIndex || secondWordIndex < 0) {
            return charMismatchMatchScore;
        }
        return firstWord[firstWordIndex] != secondWord[secondWordIndex] ? charMismatchMatchScore :
charExactMatchScore;
    }
    return charMismatchMatchScore;
}
```

1.3.7 SubCostRange5ToMinus3

SubCostRange5ToMinus3 implements a cost function as used in Monge Elkan where by an exact match no match or an approximate match whereby a set of characters are in an approximate range. for pairings in {dt} {gj} {lr} {mn} {bpv} {aeiou} {,.}

Class Hierarchy

```
AbstractSubstitutionCost
    SubCostRange5ToMinus3
```

```
[Serializable]
public sealed class SubCostRange5ToMinus3 : AbstractSubstitutionCost;
```

Methods

Method	Description
GetCost (see page 76)	get cost between characters where d(i,j) = charExactMatchScore if i equals j, charApproximateMatchScore if i approximately equals j or charMismatchMatchScore if i does not equal j.
SubCostRange5ToMinus3 (see page 77)	constructor Sets up the matching sets approximate match = +3, for pairings in {dt} {gj} {lr} {mn} {bpv} {aeiou} {,.}.

Properties

Property	Description
MaxCost (see page 76)	returns the maximum possible cost.
MinCost (see page 76)	returns the minimum possible cost.
ShortDescriptionString (see page 76)	returns the name of the cost function.

SubCostRange5ToMinus3.MaxCost

returns the maximum possible cost.

```
public override double MaxCost;
```

SubCostRange5ToMinus3.MinCost

returns the minimum possible cost.

```
public override double MinCost;
```

SubCostRange5ToMinus3.ShortDescriptionString

returns the name of the cost function.

```
public override String ShortDescriptionString;
```

SubCostRange5ToMinus3.GetCost

get cost between characters where d(i,j) = charExactMatchScore if i equals j, charApproximateMatchScore if i approximately equals j or charMismatchMatchScore if i does not equal j.

```
public override double GetCost(String firstWord, int firstWordIndex, String secondWord, int
secondWordIndex);
```

Returns

the cost of a given subsitution d(i,j) as defined above

Body Source

```
public override double GetCost(String firstWord, int firstWordIndex, String secondWord, int secondWordIndex) {
    if ((firstWord != null) && (secondWord != null)) {
        if (firstWord.Length <= firstWordIndex || firstWordIndex < 0) {
            return charMismatchMatchScore;
        }
        if (secondWord.Length <= secondWordIndex || secondWordIndex < 0) {
            return charMismatchMatchScore;
        }
        if (firstWord[firstWordIndex] == secondWord[secondWordIndex]) {
            return charExactMatchScore;
        }

        string si = firstWord[firstWordIndex].ToString().ToLowerInvariant();
        string ti = secondWord[secondWordIndex].ToString().ToLowerInvariant();
        for (int i = 0; i < approx.Length; i++) {
            if (approx[i].Contains(si) && approx[i].Contains(ti)) {
                return charApproximateMatchScore;
            }
        }
    }
    return charMismatchMatchScore;
}
```

SubCostRange5ToMinus3.SubCostRange5ToMinus3

constructor Sets up the matching sets approximate match = +3, for pairings in {dt} {gj} {lr} {mn} {bpv} {aeiou} {,.}.

```
public SubCostRange5ToMinus3();
```

Body Source

```
public SubCostRange5ToMinus3() {
    {
        approx = new Collection<string>[7];
        approx[0] = new Collection<string>();
        approx[0].Add("d");
        approx[0].Add("t");
        approx[1] = new Collection<string>();
        approx[1].Add("g");
        approx[1].Add("j");
        approx[2] = new Collection<string>();
        approx[2].Add("l");
        approx[2].Add("r");
        approx[3] = new Collection<string>();
        approx[3].Add("m");
        approx[3].Add("n");
        approx[4] = new Collection<string>();
        approx[4].Add("b");
        approx[4].Add("p");
        approx[4].Add("v");
        approx[5] = new Collection<string>();
        approx[5].Add("a");
        approx[5].Add("e");
        approx[5].Add("i");
        approx[5].Add("o");
        approx[5].Add("u");
        approx[6] = new Collection<string>();
        approx[6].Add(",");
        approx[6].Add(".");
    }
}
```

1.3.8 TokeniserQGram2

implementaton of the Bigram tokeniser

Class Hierarchy

```
AbstractTokeniserQGramN
    TokeniserQGram2
        SimMetricsUtilities.TokeniserQGram2Extended
        SimMetricsUtilities.TokeniserSGram2
```

```
public class TokeniserQGram2 : AbstractTokeniserQGramN;
```

Methods

Method	Description
TokeniserQGram2 (see page 78)	constructor
Tokenize (see page 78)	Return tokenized version of a string.
ToString (see page 78)	override the ToString method to give accurate information on current settings

Properties

Property	Description
ShortDescriptionString (see page 78)	displays the tokenisation method.

TokeniserQGram2.ShortDescriptionString

displays the tokenisation method.

```
public override string ShortDescriptionString;
```

TokeniserQGram2.TokeniserQGram2

constructor

```
public TokeniserQGram2();
```

Body Source

```
public TokeniserQGram2() {
    StopWordHandler = new DummyStopTermHandler();
    TokenUtilities = new TokeniserUtilities<string>();
    CharacterCombinationIndex = 0;
    QGramLength = 2;
}
```

TokeniserQGram2.Tokenize

Return tokenized version of a string.

```
public override Collection<string> Tokenize(string word);
```

Returns

tokenized version of a string

Body Source

```
public override Collection<string> Tokenize(string word) {
    return Tokenize(word, false, QGramLength, CharacterCombinationIndex);
}
```

TokeniserQGram2.ToString

override the ToString method to give accurate information on current settings

```
public override string ToString();
```

Returns

details of current tokeniser

Body Source

```
public override string ToString() {
    if (String.IsNullOrEmpty(SuppliedWord)) {
        return string.Format("{0} : not word passed for tokenising yet.", ShortDescriptionString);
    }
    else {
        return
```

```
            string.Format("{0} - currently holding : {1}.{2}The method is using a QGram length of {3}.",
                          ShortDescriptionString, SuppliedWord, Environment.NewLine, Convert.ToInt32(QGramLength));
        }
    }
```

1.3.9 TokeniserQGram2Extended

implementation of a Bigram tokeniser using extended logic

Class Hierarchy

```
AbstractTokeniserQGramN
    SimMetricsUtilities.TokeniserQGram2
        TokeniserQGram2Extended
            SimMetricsUtilities.TokeniserSGram2Extended
```

```
public class TokeniserQGram2Extended : TokeniserQGram2;
```

Methods

Method	Description
Tokenize (see page 80)	Return tokenized version of a string.
ToString (see page 80)	override the ToString method to give accurate information on current settings

Properties

Property	Description
ShortDescriptionString (see page 80)	displays the tokenisation method.

TokeniserQGram2Extended.ShortDescriptionString

displays the tokenisation method.

```
public override string ShortDescriptionString;
```

TokeniserQGram2Extended.Tokenize

Return tokenized version of a string.

```
public override Collection<string> Tokenize(string word);
```

Returns

tokenized version of a string

Body Source

```
public override Collection<string> Tokenize(string word) {
    return Tokenize(word, true, QGramLength, CharacterCombinationIndex);
}
```

TokeniserQGram2Extended.ToString

override the ToString method to give accurate information on current settings

```
public override string ToString();
```

Returns

details of current tokeniser

Body Source

```
public override string ToString() {
    if (String.IsNullOrEmpty(SuppliedWord)) {
        return string.Format("{0} : not word passed for tokenising yet.", ShortDescriptionString);
    }
    else {
        return
            string.Format("{0} - currently holding : {1}.{2}The method is using a QGram length of {3}.",
                          ShortDescriptionString, SuppliedWord, Environment.NewLine, Convert.ToInt32(QGramLength));
    }
}
```

1.3.10 TokeniserQGram3

implementaton of the Bigram tokeniser

Class Hierarchy

```
AbstractTokeniserQGramN
    TokeniserQGram3
        SimMetricsUtilities.TokeniserQGram3Extended
        SimMetricsUtilities.TokeniserSGram3
```

```
public class TokeniserQGram3 : AbstractTokeniserQGramN;
```

Methods

Method	Description
TokeniserQGram3 (see page 81)	constructor
Tokenize (see page 81)	Return tokenized version of a string.
ToString (see page 81)	override the ToString method to give accurate information on current settings

Properties

Property	Description
ShortDescriptionString (see page 81)	displays the tokenisation method.

TokeniserQGram3.ShortDescriptionString

displays the tokenisation method.

```
public override string ShortDescriptionString;
```

TokeniserQGram3.TokeniserQGram3

constructor

```
public TokeniserQGram3();
```

Body Source

```
public TokeniserQGram3() {
    StopWordHandler = new DummyStopTermHandler();
    TokenUtilities = new TokeniserUtilities<string>();
    CharacterCombinationIndex = 0;
    QGramLength = 3;
}
```

TokeniserQGram3.Tokenize

Return tokenized version of a string.

```
public override Collection<string> Tokenize(string word);
```

Returns

tokenized version of a string

Body Source

```
public override Collection<string> Tokenize(string word) {
    return Tokenize(word, false, QGramLength, CharacterCombinationIndex);
}
```

TokeniserQGram3.ToString

override the ToString method to give accurate information on current settings

```
public override string ToString();
```

Returns

details of current tokeniser

Body Source

```
public override string ToString() {
    if (String.IsNullOrEmpty(SuppliedWord)) {
        return string.Format("{0} : not word passed for tokenising yet.", ShortDescriptionString);
    }
    else {
        return
```

```
                string.Format("{0} - currently holding : {1}.{2}The method is using a QGram length of {3}.",
                              ShortDescriptionString, SuppliedWord, Environment.NewLine, Convert.ToInt32(QGramLength));
        }
    }
```

1.3.11 TokeniserQGram3Extended

implementation of a Bigram tokeniser using extended logic

Class Hierarchy

```
AbstractTokeniserQGramN
    SimMetricsUtilities.TokeniserQGram3
        TokeniserQGram3Extended
            SimMetricsUtilities.TokeniserSGram3Extended
```

```
public class TokeniserQGram3Extended : TokeniserQGram3;
```

Methods

Method	Description
Tokenize (see page 83)	Return tokenized version of a string.
ToString (see page 83)	override the ToString method to give accurate information on current settings

Properties

Property	Description
ShortDescriptionString (see page 83)	displays the tokenisation method.

TokeniserQGram3Extended.ShortDescriptionString

displays the tokenisation method.

```
public override string ShortDescriptionString;
```

TokeniserQGram3Extended.Tokenize

Return tokenized version of a string.

```
public override Collection<string> Tokenize(string word);
```

Returns

tokenized version of a string

Body Source

```
public override Collection<string> Tokenize(string word) {
    return Tokenize(word, true, QGramLength, CharacterCombinationIndex);
}
```

TokeniserQGram3Extended.ToString

override the ToString method to give accurate information on current settings

```
public override string ToString();
```

Returns

details of current tokeniser

Body Source

```
public override string ToString() {
    if (String.IsNullOrEmpty(SuppliedWord)) {
        return string.Format("{0} : not word passed for tokenising yet.", ShortDescriptionString);
    }
    else {
        return
            string.Format("{0} - currently holding : {1}.{2}The method is using a QGram length of {3}.",
                          ShortDescriptionString, SuppliedWord, Environment.NewLine, Convert.ToInt32(QGramLength));
    }
}
```

1.3.12 TokeniserSGram2

implementaton of the Sgram tokeniser

Class Hierarchy

```
AbstractTokeniserQGramN
    SimMetricsUtilities.TokeniserQGram2
        TokeniserSGram2
```

```
public class TokeniserSGram2 : TokeniserQGram2;
```

Methods

Method	Description
TokeniserSGram2 (see page 84)	constructor
ToString (see page 84)	override the ToString method to give accurate information on current settings

Properties

Property	Description
ShortDescriptionString (see page 84)	displays the tokenisation method.

TokeniserSGram2.ShortDescriptionString

displays the tokenisation method.

```
public override string ShortDescriptionString;
```

TokeniserSGram2.TokeniserSGram2

constructor

```
public TokeniserSGram2();
```

Body Source

```
public TokeniserSGram2() : base() {
    CharacterCombinationIndex = 1;
}
```

TokeniserSGram2.ToString

override the ToString method to give accurate information on current settings

```
public override string ToString();
```

Returns

details of current tokeniser

Body Source

```
public override string ToString() {
    if (String.IsNullOrEmpty(SuppliedWord)) {
        return string.Format("{0} : not word passed for tokenising yet.", ShortDescriptionString);
    }
    else {
        if (CharacterCombinationIndex == 0) {
            return
                string.Format("{0} - currently holding : {1}.{2}The method is using a QGram length of {3}.",
                              ShortDescriptionString, SuppliedWord, Environment.NewLine,
Convert.ToInt32(QGramLength));
        }
        else {
            return
                string.Format(
                    "{0} - currently holding : {1}.{2}The method is using a character combination index of {3} and
{4}a QGram length of {5}.",
                    ShortDescriptionString, SuppliedWord, Environment.NewLine,
                    Convert.ToInt32(CharacterCombinationIndex), Environment.NewLine, Convert.ToInt32(QGramLength));
        }
    }
}
```

1.3.13 TokeniserSGram2Extended

implementation of a SGram tokeniser using extended logic

Class Hierarchy

```
AbstractTokeniserQGramN
    SimMetricsUtilities.TokeniserQGram2
        SimMetricsUtilities.TokeniserQGram2Extended
            TokeniserSGram2Extended
```

```
public class TokeniserSGram2Extended : TokeniserQGram2Extended;
```

Methods

Method	Description
TokeniserSGram2Extended (see page 85)	constructor
ToString (see page 85)	override the ToString method to give accurate information on current settings

Properties

Property	Description
ShortDescriptionString (see page 85)	displays the tokenisation method.

TokeniserSGram2Extended.ShortDescriptionString

displays the tokenisation method.

```
public override string ShortDescriptionString;
```

TokeniserSGram2Extended.TokeniserSGram2Extended

constructor

```
public TokeniserSGram2Extended();
```

Body Source

```
public TokeniserSGram2Extended() : base() {
    CharacterCombinationIndex = 1;
}
```

TokeniserSGram2Extended.ToString

override the ToString method to give accurate information on current settings

```
public override string ToString();
```

Returns

details of current tokeniser

Body Source

```
public override string ToString() {
    if (String.IsNullOrEmpty(SuppliedWord)) {
        return string.Format("{0} : no word passed for tokenising yet.", ShortDescriptionString);
    }
    else {
        if (CharacterCombinationIndex == 0) {
            return
                string.Format("{0} - currently holding : {1}.{2}The method is using a QGram length of {3}.",
                              ShortDescriptionString, SuppliedWord, Environment.NewLine,
Convert.ToInt32(QGramLength));
        }
        else {
            return
                string.Format(
                    "{0} - currently holding : {1}.{2}The method is using a character combination index of {3} and
{4}a QGram length of {5}.",
                    ShortDescriptionString, SuppliedWord, Environment.NewLine,
                    Convert.ToInt32(CharacterCombinationIndex), Environment.NewLine, Convert.ToInt32(QGramLength));
        }
    }
}
```

1.3.14 TokeniserSGram3

implementaton of the Sgram tokeniser

Class Hierarchy

```
AbstractTokeniserQGramN
    SimMetricsUtilities.TokeniserQGram3
        TokeniserSGram3
```

```
public class TokeniserSGram3 : TokeniserQGram3;
```

Methods

Method	Description
TokeniserSGram3 (see page 86)	constructor
ToString (see page 86)	override the ToString method to give accurate information on current settings

Properties

Property	Description
ShortDescriptionString (see page 86)	displays the tokenisation method.

TokeniserSGram3.ShortDescriptionString

displays the tokenisation method.

```
public override string ShortDescriptionString;
```

TokeniserSGram3.TokeniserSGram3

constructor

```
public TokeniserSGram3();
```

Body Source

```
public TokeniserSGram3() : base() {
    CharacterCombinationIndex = 1;
}
```

TokeniserSGram3.ToString

override the ToString method to give accurate information on current settings

```
public override string ToString();
```

Returns

details of current tokeniser

Body Source

```
public override string ToString() {
    if (String.IsNullOrEmpty(SuppliedWord)) {
        return string.Format("{0} : not word passed for tokenising yet.", ShortDescriptionString);
    }
    else {
        if (CharacterCombinationIndex == 0) {
            return
                string.Format("{0} - currently holding : {1}.{2}The method is using a QGram length of {3}.",
                              ShortDescriptionString, SuppliedWord, Environment.NewLine,
Convert.ToInt32(QGramLength));
        }
        else {
            return
                string.Format(
                    "{0} - currently holding : {1}.{2}The method is using a character combination index of {3} and
{4}a QGram length of {5}.",
                    ShortDescriptionString, SuppliedWord, Environment.NewLine,
                    Convert.ToInt32(CharacterCombinationIndex), Environment.NewLine, Convert.ToInt32(QGramLength));
        }
    }
}
```

1.3.15 TokeniserSGram3Extended

implementation of a SGram tokeniser using extended logic

Class Hierarchy

```
AbstractTokeniserQGramN
    SimMetricsUtilities.TokeniserQGram3
        SimMetricsUtilities.TokeniserQGram3Extended
            TokeniserSGram3Extended
```

```
public class TokeniserSGram3Extended : TokeniserQGram3Extended;
```

Methods

Method	Description
TokeniserSGram3Extended (see page 87)	constructor
ToString (see page 87)	override the ToString method to give accurate information on current settings

Properties

Property	Description
ShortDescriptionString (see page 87)	displays the tokenisation method.

TokeniserSGram3Extended.ShortDescriptionString

displays the tokenisation method.

```
public override string ShortDescriptionString;
```

TokeniserSGram3Extended.TokeniserSGram3Extended

constructor

```
public TokeniserSGram3Extended();
```

Body Source

```
public TokeniserSGram3Extended() : base() {
    CharacterCombinationIndex = 1;
}
```

TokeniserSGram3Extended.ToString

override the ToString method to give accurate information on current settings

```
public override string ToString();
```

Returns

details of current tokeniser

Body Source

```
public override string ToString() {
    if (String.IsNullOrEmpty(SuppliedWord)) {
        return string.Format("{0} : no word passed for tokenising yet.", ShortDescriptionString);
    }
    else {
        if (CharacterCombinationIndex == 0) {
            return
                string.Format("{0} - currently holding : {1}.{2}The method is using a QGram length of {3}.",
                              ShortDescriptionString, SuppliedWord, Environment.NewLine,
Convert.ToInt32(QGramLength));
        }
        else {
            return
                string.Format(
                    "{0} - currently holding : {1}.{2}The method is using a character combination index of {3} and
{4}a QGram length of {5}.",
                    ShortDescriptionString, SuppliedWord, Environment.NewLine,
                    Convert.ToInt32(CharacterCombinationIndex), Environment.NewLine, Convert.ToInt32(QGramLength));
        }
    }
}
```

1.3.16 TokeniserUtilities

type for token collection

Summary

class containing utility functions for the tokenisers to use. these are in two main version collections or sets a collection can contain the same value multiple times ad set can only have the value once.

Class Hierarchy

`TokeniserUtilities`

```
[Serializable]
public class TokeniserUtilities;
```

1.3.17 TokeniserWhitespace

implements a simple whitespace tokeniser.

Class Hierarchy

```
ITokeniser
    TokeniserWhitespace
```

```
[Serializable]
public sealed class TokeniserWhitespace : ITokeniser;
```

Methods

Method	Description
TokeniserWhitespace (see page 89)	default constructor
Tokenize (see page 89)	Return tokenized version of a string.
TokenizeToSet (see page 90)	Return tokenized set of a string.

Properties

Property	Description
Delimiters (see page 89)	displays the delimiters used.
ShortDescriptionString (see page 89)	displays the tokenisation method.
StopWordHandler (see page 89)	gets the stop word handler used.

TokeniserWhitespace.Delimiters

displays the delimiters used.

```
public string Delimiters;
```

TokeniserWhitespace.ShortDescriptionString

displays the tokenisation method.

```
public string ShortDescriptionString;
```

TokeniserWhitespace.StopWordHandler

gets the stop word handler used.

```
public ITermHandler StopWordHandler;
```

TokeniserWhitespace.TokeniserWhitespace

default constructor

```
public TokeniserWhitespace();
```

Body Source

```
public TokeniserWhitespace() {
    stopWordHandler = new DummyStopTermHandler();
    tokenUtilities = new TokeniserUtilities<string>();
}
```

TokeniserWhitespace.Tokenize

Return tokenized version of a string.

```
public Collection<string> Tokenize(string word);
```

Returns

tokenized version of a string

Body Source

```
public Collection<string> Tokenize(string word) {
    Collection<string> returnVect = new Collection<string>();
    if (word != null) {
        int nextGapPos;
        for (int curPos = 0; curPos < word.Length; curPos = nextGapPos) {
            char ch = word[curPos];
            if (Char.IsWhiteSpace(ch)) {
                curPos++;
            }
            nextGapPos = word.Length;
```

```
                for (int i = 0; i < delimiters.Length; i++) {
                    int testPos = word.IndexOf(delimiters[i], curPos);
                    if (testPos < nextGapPos && testPos != -1) {
                        nextGapPos = testPos;
                    }
                }

                string term = word.Substring(curPos, (nextGapPos) - (curPos));
                if (!stopWordHandler.IsWord(term)) {
                    returnVect.Add(term);
                }
            }
        }
        return returnVect;
    }
```

TokeniserWhitespace.TokenizeToSet

Return tokenized set of a string.

```
public Collection<string> TokenizeToSet(string word);
```

Returns

tokenized set of a string

Body Source

```
public Collection<string> TokenizeToSet(string word) {
    if (word != null) {
        return tokenUtilities.CreateSet(Tokenize(word));
    }
    return null;
}
```

Index