# Distance Generator 1.0 Component Specification

## 1. Design

The Distance Generator is a custom component for TopCoder's web site. The component will calculate the distance between a member and a set of related members. The component will define at minimum three distances to calculate - rating, geographical, and match overlap. The component is responsible for calculating these distances, as well as weighting them to calculate an overall distance between the two members. After calculation, the component must deliver the results in XML, to provide a data feed for a user interface to display.

When calculation is performed, the component will retrieve a requested member, and a list of related members. For the purposes of any single calculation request, these are the only members that exist. The output of this component will be used in a graphical display, and the intention is to make each individual display as visually appealing and informative as possible. For this reason, this component does not calculate distance in the strict mathematical sense - the component does not define distance functions as symmetrical, for instance.

### 1.1 Design Patterns

The strategy pattern is used to make IMemberDataAccess implementations pluggable to the DefaultDistanceGenerator class, IDistanceCalculator implementations pluggable to DefaultDistanceGenerator class and IDistanceGenerator implementations pluggable to client code.

### 1.2 Industry Standards

XML

### 1.3 Required Algorithms

Distance between two members is defined as a floating point value from zero to one, inclusive. Larger values indicate further distance. Distances must always be calculated in a relative manner, to maximize the distinction between related members. The component supports the following calculations. Note that minimum and maximum functions mentioned below are local and only cover the member and its related members.

#### *1.3.1 Rating Distance*

The component supports calculation of the distance between two members based on their ratings. Members may have more than one rating (for instance, one member may be rated in design, development, and algorithm competitions). The types of ratings available are: algorithm, development, design, marathon, and high-school. All rating data will provided by the data access interface.

When comparing a member to another member, these rules should be followed:

| From (A) | To (B) | Value | Reason |
|---|---|---|---|
| Rated Member | Unrated Member | 1.0 | A rated member is equidistant to all unrated members of the type, with a non-zero distance. To maximize the effect of this distance, it will be considered the maximum distance (1.0). |

| | | | |
|---|---|---|---|
| Unrated member | Rated member | B.Rating/max(rating) | An unrated member is assumed to have a zero rating for determining the distance between themselves and rated members. |
| Unrated member | Unrated member | 0.0 | An unrated member is in the same state as another unrated member and may be assumed to be equidistant. The distance between them is assumed to be zero. |
| Rated member | Rated member | abs(A.Rating – B.Rating) / max(rating) | This function produces the relative distance between two members. |

Please note here that max(rating) here refers to the max rating of all members involved in the calculation for a given type. In the data provided to the component a rating less than or equal to zero should be considered as "unrated".

To compute the overall rating distance between one member and another member, the average (mean) distance must be computed, based on the individual distances per rating. If all provided members are unrated in a competition type, the distance should be dropped from the computation to prevent skewing the mean. Note this applies to the other two distance calculations too. Take a look at this example:

Suppose we have three members related with member X with three ratings:
X { -1, -1, -1 }
A { 1, 2, 3 }
B { 2, 3, 4 }
C { 3, 4, 5 }

max(rating1) = 3, max(rating2) = 4, max(rating3) = 5

distances() =
A { (1/3 + 2/4 + 3/5) / 3 }
B { (2/3 + 3/4 + 4/5) / 3 }
C { (3/3 + 4/4 + 5/5) / 3 }

### *1.3.2 Geographical Distance*

The component supports calculation of the distance between two members geographically. The geographic distance in uniform units will be provided by the data access interface. A geographical distance of zero indicates no geographical distance between two members, and a distance less than zero indicates an unknown geographical distance between the members.

| From (A) | To (B) | Value | Reason |
|---|---|---|---|
| Known Member | Unknown Member | Undefined | The distance between a known and unknown member is undefined and the distance should not be |

| | | | |
|---|---|---|---|
| | | | included. |
| Unknown member | Known member | Undefined | The distance between a known and unknown member is undefined and the distance should not be included. |
| Unknown member | Unknown member | Undefined | The distance between unknown members is undefined and the distance should not be included. |
| Known member | Known member | B.GeoDistance / max(GeoDistance) | This function produces the relative distance between two members. If max(GeoDistance) is zero, the function should return zero, and not undefined. |

max(GeoDistance) is also calculated locally for all involved members, similar to max(Rating) as described in section 1.3.1.

*1.3.3 Match Overlap*

The component supports calculation of the distance between two members based on the number of shared matches. Note that this function is inverse (higher overlap indicates lower distance). Additionally, there is no "unknown" value for this data - a member will share zero or more matches with another member.

The recommended function for this calculation is 1-((B.MatchOverlap-min(overlap))/ max(overlap)). If max(overlap) is zero the function should return 1 and not undefined.

*1.3.4 Distance Aggregation*

The consumer of the component is able to specify which of the above distances to calculate per request. When more than one type is specified, the distances must be averaged together. The component uses a weighted average to compute the final distance. The weighting should default to equal weight, but the consumer must be able to alter the weighting if desired. For example, if the following distances were calculated with equal weighting:

| Handle | Rating Distance | Geographical Distance | Final Distance |
|---|---|---|---|
| AdamSelence | 1.0 | 0.0 | 0.5 |
| dok | 0.5 | 0.0 | 0.25 |
| mess | 0.25 | Undefined | Undefined |

*If we were to give higher weighting to geographical distance, the final distances would decrease. Note that any aggregate distance which includes an undefined distance will be undefined.*

Unused weight must be re-distributed evenly. For example, if the default weightings are GEO=40, OVR=40, RAT=20, if we only calculate GEO and OVR, we redistribute the 20 points evenly between the other two weights - which is 40+10, or 50. Similarly, for OVR and RAT, we redistribute GEO to result in 60/40.

Since weights can be float numbers, developers should be careful of the precision issue. A small correction number should be used when comparing float numbers, for example:

abs(100 – sum_of_weights) < eps where eps is a very small float number.

*1.3.5 XML Output*

The component must return an XML string conforming to the provided XSD schema. The first coder element in the output must always be the requested member and any member with an undefined distance must not be included in the output.

The elements in the XSD map as follows:

| Element | Content |
|---|---|
| coder_id | Member.Id |
| Handle | Member.Handle |
| rating | Member.MaxRating |
| image | Member.Image |
| distance | The calculated distance or 0 for the first member. |
| overlap | Member.MatchOverlap |
| country | Member.Country |

## 1.4 Component Class Overview

**IDistanceGenerator interface**:
This interface defines the contract to calculate the various distances. This component provides 3 implementations to support 3 kinds of distance calculations: rating distance, geographical distance and match overlap distance. Implementations of this interface will be plugged to IDistanceGenerator implementations and will not be used directly. Implementations of this interface should be thread safe.

**IXmlGenerator interface**:
This interface defines the contract to generate xml string for the calculated distance data. Implementations of this interface should be thread safe.

**DistanceTypes enum**:
This enumeration contains the basic distances between users. This enum must be marked with the [Flags] attribute.
Enums are always thread safe.

**CompetitionTypes enum**:
This enumeration contains the various competition types available for ratings and competition overlap. This enum must be marked with the [Flags] attribute.
Enums are always thread safe.

**IDistanceCalculator interface**:
This interface defines the contract to calculate the various distances. Currently 3 kinds of distance calculation are supported: rating distance, geographical distance and match overlap distance. This interface is defined so calculating formulas could be easily replaced or plugged in.
Implementations of this interface should be thread safe.

**DefaultDistanceGenerator class:**
This class is the default implementation of the IDistanceGenerator interface. It uses pluggable member data access instance distance calculator and xml generator to accomplish the task. The calculation formula could be easily replaced or plugged in by providing new IDistanceCalculator implementations, which is a nice feature since this

way we don't need to modify existing code in case of such requirements.
This class is thread safe since it's immutable.

**RatingDistanceCalculator class:**
This class is the implementation of the IDistanceCalculator interface that calculates rating distance. It implements the default distance calculation formula/algorithm as stated in component specification algorithm section.
This class is thread safe since it's immutable.

**GeographicalDistanceCalculator class:**
This class is the implementation of the IDistanceCalculator interface that calculates geographical distance. It implements the default geographical calculation formula/algorithm as stated in component specification algorithm section.
This class is thread safe since it's immutable.

**OverlapDistanceCalculator class:**
This class is the implementation of the IDistanceCalculator interface that calculates match overlap distance. It implements the default match overlap calculation formula/algorithm as stated in component specification algorithm section.
This class is thread safe since it's immutable.

**DefaultXmlGenerator class:**
The default implementation of the IXmlGenerator interface. It generates xml string that conforms to the defined XSD.
This class is thread safe since it's immutable.

**IMemberDataAccess interface:**
The data access interface for retrieving member data. It defines methods to retrieve a specific member by id and retrieve a members related members by given id and competition type.
Implementations of this interface should be thread safe.

**Member class:**
This class is an immutable data transport object that holds all data necessary to calculate the required distances between two members. Members unrated in a particular competition type are considered to have rating zero.
This class is thread safe since it's immutable.

### 1.5 Component Exception Definitions

**DistanceGenerationException**:
This exception is used to represent errors that might occur during generation of distance. For example: failed to retrieve data. This exception is thrown by methods in IDistanceGenerator interface and its implementations.

**MemberDataAccessException:**
This exception is thrown by IMemberDataAccess implementations, when errors occur in the member retrieval methods of the implementations.

### 1.6 Thread Safety

This component is completely thread safe as all classes in this component are immutable , and all future implementations of the defined interfaces are required to be thread safe.

## 2. Environment Requirements

### 2.1 Environment

Development language: #

Compile Target: .NET Framework 2.0 and 3.0

### 2.2 TopCoder Software Components

**Configuration API 1.0** is used to pass in configuration for this component.

**Object Factory Configuration API Plugin 1.1** is used to create objects from Configuration API style configuration.

*NOTE: The default location for TopCoder Software component jars is../lib/tcs/COMPONENT_NAME/COMPONENT_VERSION relative to the component installation. Setting the tcs_libdir property in topcoder_global.properties will overwrite this default location.*

### 2.3 Third Party Components

None.

*NOTE: The default location for 3rd party packages is ../lib relative to this component installation. Setting the ext_libdir property in topcoder_global.properties will overwrite this default location.*

## 3. Installation and Configuration

### 3.1 Package Name

TopCoder.Web.Distance
TopCoder.Web.Distance.DistanceGenerators
TopCoder.Web.Distance.DistanceCalculators
TopCoder.Web.Distance.XmlGenerators
TopCoder.Web.Distance.Data

### 3.2 Configuration Parameters

Configuration for DefaultDistanceGenerator class:

| Parameter Name | Parameter Description | Parameter Value |
|---|---|---|
| dataAccessKey | The key used to create the IMemberDataAccess instance. Required. | Non-null and non-empty string. |
| xmlGeneratorKey | The key used to create the IXmlGenerator instance. Required | Non-null and non-empty string. |
| calculatorKeys | This should be a string array in which each element has format distance_type:calculator_key, while distance_type is a valid enum name as defined in DistanceTypes enum, and calculator_key is a valid key used to create the IDistanceCalculator instance. Required. | Should be an array containing elements with the defined format. |
| weights | This should be a string array in which each element has format distance_type:weight, while distance_type is a valid enum name as defined in DistanceTypes enum, and weight is a positive float number. Optional. If not present | If present must be an array containing elements with the defined format. And the sum of all weights should equal to 100. The elements cannot contain duplicate names of distance types. |

| | | |
|---|---|---|
| | we'll use the default equal weight. | |
| objectDefChild | The child IConfiguration object containing object definitions for the above configured keys. Required | Must be a valid IConfiguration child that contains definition to create IMemberDataAccess and IDistanceCalculator instances based on configured key. |

### 3.3 Dependencies Configuration

All dependent components should be properly configured before this component is used. Please consult the documents of the dependent components for configuration details.

## 4. Usage Notes

### 4.1 Required steps to test the component

Extract the component distribution.

Set up the component environment.

Execute 'ant test' within the directory that the distribution was extracted to.

### 4.2 Required steps to use the component

Nothing special. Just set up the configuration and use it.

### 4.3 Demo

The following section demonstrates how to set up the configuration and create a DefaultDistanceGenerator instance:

```
// set up the configuration object used by DefaultDistanceGenerator class
Iconfiguration config = new DefaultConfiguration("root");
config.SetSimpleAttribute("dataAccessKey", "fileBasedDataAccess");
config.SetSimpleAttribute("xmlGeneratorKey", "myXmlGenerator");
string[] calculatorKeys = new string[] {"Rating:rCal","Overlap:oCal"};
config.SetSimpleAttribute("calculatorKey", calculatorKeys);
// and the config should contain the object definitions, suppose objectDefChild is
// defined
config.AddChild(objectDefChild);

// create a DefaultDistanceGenerator instance
IDistanceGenerator generator = new DefaultDistanceGenerator(config);

// we may also create the generator without the usage of any configuration
generator = new DefaultDistanceGenerator(memberDataAccess, distanceCalculator);
```

Since we now have the IDistanceGenerator instance created, we can start using it.

```
// generate with default weights
// we suppose we use the data in section 1.3.1, X is the member with id 1, and the
// 3 ratings are for Algorithm, Design and Development respectively
string result = generator.GenerateDistanceXml(1, DistanceTypes.Rating,
(CompetitionTypes) 3 );

// we'll expect an XML string looks like the following(note, only care the handle
// and distance values, the other values are mocks
<coder>
    <coder_id>1</coder_id>
    <handle>X</handle>
    <rating>-1</rating>
    <image>image for X</image>
    <distance>0</distance>
```

```
    <overlap>1</overlap>
    <country>US</country>
</coder>
<coder>
    <coder_id>2</coder_id>
    <handle>A</handle>
    <rating>3</rating>
    <image>image for A</image>
    <distance>0.48</distance>
    <overlap>2</overlap>
    <country>UK</country>
</coder>
<coder>
    <coder_id>3</coder_id>
    <handle>B</handle>
    <rating>4</rating>
    <image>image for B</image>
    <distance>0.74</distance>
    <overlap>3</overlap>
    <country>CN</country>
</coder>
<coder>
    <coder_id>4</coder_id>
    <handle>C</handle>
    <rating>5</rating>
    <image>image for C</image>
    <distance>1</distance>
    <overlap>4</overlap>
    <country>JP</country>
</coder>

// usage of the other overload, suppose we are using the sample data in CS 1.3.4
// we can expect the final distance to be 0.4, 0.2 and undefined for AdamSelence, dok
// and mess respectively, the xml is not shown here since it's similar to the above
// one
IDictionary<DistanceTypes,float> weights = new Dictionary<DistanceTypes,float>();
weights.Add(DistanceTypes.Rating, 40);
weights.Add(DistanceTypes.Country, 60);
result = generator.GenerateDistanceXml(1, DistanceTypes.Rating, (CompetitionTypes) 3,
weights);
```

## 5. Future Enhancements

Future versions will include more distance types - for instance the difference in volatility, the average time to submit an algorithm problem, submission score similarity, and so forth.