Double Range

```
//http://isotopescreencapture.codeplex.com/
//The MIT License (MIT)
using System.Collections.Generic;

namespace Isotope.Collections.Ranges
{
public struct DoubleRange
{
public readonly double _Lower;
public readonly double _Upper;

public double Lower
{
get { return this._Lower; }
}

public double Upper
{
get { return this._Upper; }
}

private DoubleRange(double lower, double upper)
{
this._Lower = lower;
this._Upper = upper;
}

///

/// Creates a range from the lower and upper values
///
///
/// (0,0) -> [0]
/// (0,1) -> [0,1]
/// (0,5) -> [0,1,2,3,4,5]
/// (2,5) -> [2,3,4,5]
```

```
///
///  ///  ///
public static DoubleRange FromEndpoints(double lower, double
upper)
{
return new DoubleRange(lower, upper);
}

public double Length
{
get { return this.Upper – this.Lower; }
}

public override string ToString()
{
var                invariant_culture                =
System.Globalization.CultureInfo.InvariantCulture;
return  string.Format(invariant_culture,  “Range({0},{1})”,
this.Lower, this.Upper);
}

private static bool _intersects_exclusive(DoubleRange range1,
DoubleRange range2)
{
bool val = (range1.Lower < range2.Lower) && (range1.Upper >
range2.Lower);
return val;
}

private static bool _intersects_inclusive(DoubleRange range1,
DoubleRange range2)
{
bool val = (range1.Lower <= range2.Lower) && (range1.Upper >=
range2.Upper);
return val;
}

///
```

```
/// Tests if this range interests with another
///
/// the other range /// true if they intersect
public bool IntersectsExclusive(DoubleRange range)
{
bool val = _intersects_exclusive(this, range) ||
_intersects_exclusive(range, this);
return val;
}

public bool IntersectsInclusive(DoubleRange range)
{
bool val = _intersects_inclusive(this, range) ||
_intersects_inclusive(range, this);
return val;
}

private static bool _touches(DoubleRange range1, DoubleRange
range2)
{
var val = (range1.Upper == range2.Lower);
return val;
}

///

/// Tests if this range touches another. For example (1-2)
touches (3,5) but (1,2) does not touch (4,5)
///
/// the other range /// true if they touch
public bool Touches(DoubleRange range)
{
var val = _touches(this, range) || _touches(range, this);
return val;
}

///

/// Tests if this range contains a specific double
```

```
///
/// the double /// true if the number is contained
public bool Contains(double n)
{
return (this.Lower <= n) && (n <= this.Upper); } ///
/// Join this range with another and return a single range
that contains them both. The ranges must either touch or
interest.
/// for example (0,2) amd (3,7) will yield (0,7)
///

/// the other range /// the merged range
public DoubleRange JoinWith(DoubleRange range)
{
if (this.IntersectsExclusive(range) || this.Touches(range))
{
double new_Upper = System.Math.Max(this.Upper, range.Upper);
double new_Lower = System.Math.Min(this.Lower, range.Lower);
return DoubleRange.FromEndpoints(new_Lower, new_Upper);
}
else
{
throw new System.ArgumentException("Ranges cannot be joined
because they do not touch or overlap");
}
}

public IEnumerable Values
{
get
{
double step = 1.0;
return this.GetValues(step);
}
}

public IEnumerable GetValues(double step)
{
```

```
if (step <= 0.0) { throw new System.ArgumentOutOfRangeException("step", "must be positive"); } double i = this.Lower; while (i <= this.Upper) { yield return i; i += step; } } } } [/csharp]
```