Int Range

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

namespace Isotope.Collections.Ranges
{
public struct IntRange
{
public readonly int _Lower;
public readonly int _Length;

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

public int Upper
{
get { return this.Lower + this.Length - 1; }
}

public int Length
{
get { return this._Length; }
}

private IntRange(int lower, int length)
{
this._Lower = lower;
this._Length = length;

if (this.Lower > this.Upper)
{
throw new System.OverflowException();
}
```

```
}

///

/// 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 IntRange FromEndpoints(int lower, int upper)
{
return new IntRange(lower, upper – lower + 1);
}

///

/// Creates a range of a single number
///
///
///  ///
public static IntRange FromInteger(int value)
{
return new IntRange(value, 1);
}

///

/// Creates a new range from a starting value and including
all the numbers following that
///
///
/// (2,0) -> []
/// (2,1) -> [2]
/// (2,2) -> [2,3]
/// (2,3) -> [2,3,4]
```

```
/// (2,4) -> [2,3,4,5]
///
///  ///  ///
public static IntRange FromLower(int lower, int length)
{
if (length < 0) { throw new System.ArgumentOutOfRangeException("length", "must be >= 0″);
}

return new IntRange(lower, length);
}

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(IntRange range1, IntRange range2)
{
return ((range1.Lower <= range2.Lower) && (range1.Upper >= range2.Lower));
}

///

/// Tests if this range interests with another
///
/// the other range /// true if they intersect
public bool Intersects(IntRange range)
{
return (_intersects(this, range) || _intersects(range, this));
}

private static int Delta = 1;
```

```
private static bool _touches(IntRange range1, IntRange range2)
{
var val = (range1.Upper + Delta) == 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(IntRange range)
{
var val = _touches(this, range) || _touches(range, this);
return val;
}

///

/// Tests if this range contains a specific integer
///
/// the integet /// true if the number is contained
public bool Contains(int 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 IntRange JoinWith(IntRange range)
{
if (this.Intersects(range) || this.Touches(range))
{
int new_Upper = System.Math.Max(this.Upper, range.Upper);
int new_Lower = System.Math.Min(this.Lower, range.Lower);
```

```
return IntRange.FromEndpoints(new_Lower, new_Upper);
}
else
{
throw new System.ArgumentException();
}
}

///

/// Get each int int he range from the lower value to the upper value
///
/// each int in the range
public IEnumerable Values
{
get
{
for (int i = this.Lower; i <= this.Upper; i++) { yield return i; } } } } } [/csharp]
```