**The following standard query operators belong to Element Operators category**  
First / FirstOrDefault  
Last / LastOrDefault  
ElementAt / ElementAtOrDefault  
Single / SingleOrDefault  
DefaultIfEmpty   
  
   
  
**Element Operators**retrieve a single element from a sequence using the element index or based on a condition. All of these methods have a corresponding overloaded version that accepts a predicate.  
  
**First :** There are 2 overloaded versions of this method. The first overloaded version that does not have any parameters simply returns the first element of a sequence.  
  
**Example 1:**Returns the first element from the sequence

int[] numbers = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };

int result = numbers.First();

Console.WriteLine("Result = " + result);

**Output:**  
Result = 1  
  
If the sequence does not contain any elements, then First() method throws an InvalidOperationException.  
  
**Example 2:**Throws InvalidOperationException.

int[] numbers = { };

int result = numbers.First();

Console.WriteLine("Result = " + result);

**Output:**  
Unhandled Exception: System.InvalidOperationException: Sequence contains no elements  
  
**The second overloaded version is used to find the first element in a sequence based on a condition.** If the sequence does not contain any elements or if no element in the sequence satisfies the condition then an InvalidOperationException is thrown.  
  
**Example 3:**Returns the first even number from the sequence

int[] numbers = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };

int result = numbers.First(x => x % 2 == 0);

Console.WriteLine("Result = " + result);

**Output:**  
Result = 2  
  
**Example 4:**Throws InvalidOperationException, as no element in the sequence satisfies the condition specified by the predicate.

int[] numbers = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };

int result = numbers.First(x => x % 2 == 100);

Console.WriteLine("Result = " + result);

**Output:**  
Unhandled Exception: System.InvalidOperationException: Sequence contains no matching element  
  
**FirstOrDefault :** This is very similar to First, except that this method does not throw an exception when there are no elements in the sequence or when no element satisfies the condition specified by the predicate. Instead, a default value of the type that is expected is returned. For reference types the default is NULL and for value types the default depends on the actual type expected.  
  
**Example 5:**Returns ZERO. No element in the sequence satisfies the condition, so the default value (ZERO) for int is returned.

int[] numbers = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };

int result = numbers.FirstOrDefault(x => x % 2 == 100);

Console.WriteLine("Result = " + result);

**Last :** Very similar to First, except it returns the last element of the sequence.  
  
**LastOrDefault :**Very similar to FirstOrDefault, except it returns the last element of the sequence.  
  
**ElementAt :**Returns an element at a specified index. If the sequence is empty or if the provided index value is out of range, then an ArgumentOutOfRangeException is thrown.  
  
**Example 6:**Returns element from the sequence that is at index position 1.

int[] numbers = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };

int result = numbers.ElementAt(1);

Console.WriteLine("Result = " + result);

**Output:**  
Result = 2  
  
**Example 7:**Throws ArgumentOutOfRangeException

int[] numbers = { };

int result = numbers.ElementAt(0);

Console.WriteLine("Result = " + result);

**Output:**  
Unhandled Exception: System.ArgumentOutOfRangeException: Index was out of range. Must be non-negative and less than the size of the collection.  
  
**ElementAtOrDefault :** Similar to ElementAt except that this method does not throw an exception, if the sequence is empty or if the provided index value is out of range. Instead, a default value of the type that is expected is returned.  
  
**Single :** There are 2 overloaded versions of this method. The first overloaded version that does not have any parameters returns the only element of the sequence.  
  
**Example 8:**Returns the only element (1) of the sequence.

int[] numbers = { 1 };

int result = numbers.Single();

Console.WriteLine("Result = " + result);

**Output:**  
Result = 1  
  
**Single()**method throws an exception if the sequence is empty or has more than one element.  
  
**Example 9:**Throws InvalidOperationException as the sequence contains more than ONE element.

int[] numbers = { 1, 2 };

int result = numbers.Single();

Console.WriteLine("Result = " + result);

**Output:**  
Unhandled Exception: System.InvalidOperationException: Sequence contains more than one element  
  
**The second overloaded version of the Single() method is used to find the only element in a sequence that satisfies a given condition.** An exception will be thrown if any of the following is true  
**a)** If the sequence does not contain any elements OR  
**b)** If no element in the sequence satisfies the condition OR  
**c)** If more than one element in the sequence satisfies the condition  
  
**Example 10:**Throws InvalidOperationException as more than one element in the sequence satisfies the condition

int[] numbers = { 1, 2, 4 };

int result = numbers.Single(x => x % 2 == 0);

Console.WriteLine("Result = " + result);

**Output:**  
Unhandled Exception: System.InvalidOperationException: Sequence contains more than one matching element  
  
**SingleOrDefault :**Very similar to Single(), except this method does not throw an exception when the sequence is empty or when no element in the sequence satisfies the given condition. Just like Single(), this method will still throw an exception, if more than one element in the sequence satisfies the given condition.  
  
**Example 11:**Throws InvalidOperationException as more than one element in the sequence satisfies the given condition

int[] numbers = { 1, 2, 4 };

int result = numbers.SingleOrDefault(x => x % 2 == 0);

Console.WriteLine("Result = " + result);

**Output:**  
Unhandled Exception: System.InvalidOperationException: Sequence contains more than one matching element  
  
**DefaultIfEmpty :** If the sequence on which this method is called is not empty, then the values of the original sequence are returned.  
  
**Example 12 :**Returns a copy of the original sequence

int[] numbers = { 1, 2, 3 };

IEnumerable<int> result = numbers.DefaultIfEmpty();

foreach (int i in result)

{

    Console.WriteLine(i);

}

**Output:**  
1  
2  
3  
  
If the sequence is empty, then **DefaultIfEmpty**() returns a sequence with the default value of the expected type.  
  
**Example 13 :**Since the sequence is empty, a sequence containing the default value (ZERO) of int is returned.

int[] numbers = { };

IEnumerable<int> result = numbers.DefaultIfEmpty();

foreach (int i in result)

{

    Console.WriteLine(i);

}

Output:  
0  
  
The other overloaded version with a parameter allows us to specify a default value. If this method is called on a sequence that is not empty, then the values of the original sequence are returned. If the sequence is empty, then this method returns a sequence with the specified defualt value.  
  
**Example 14 :** Since the sequence is empty, a sequence containing the specified default value (10) is returned.

int[] numbers = { };

IEnumerable<int> result = numbers.DefaultIfEmpty(10);

foreach (int i in result)

{

    Console.WriteLine(i);

}

**Output:**  
10