```
/// <summary>
/// This method takes an input sequence of objects, formats the object, and pushes it to the output.
/// </summary>
/// <typeparam name="TSource">The type of object which the input sequence is composed of.</typeparam>
/// <param name="Input">The input sequence of objects which are to be processed.<br/>
/// The type of these objects is exposed in the <typeparamref name="TSource">"/> type parameter.
/// </param>
/// <param name="WherePredicate">
/// [Optional] A where predicate used to select the objects to be output.<br/>
/// Predicate signature <code>Func&lt;TSource InputObject, int PositionInSquence, bool ReturnValue&gt;</code>
/// Predicate Arguments:
/// <list type="number">
///     <item>
/// [Input Parameter] <br/>
/// The object from the input sequence. <br/>
/// The type of the object is the same as the declaration of the sequence.
/// This type of the objects comes from the type parameter <typeparamref name="TSource">"/> in the declaration of this method.<br/>
/// For compound sequences like Dictionary the object is a KeyValuePair.
///     </item>
///     <item>
/// [Input Parameter]<br/>
/// The position in the input sequence which object occupies.<br/>
/// This is a base zero number.
///     </item>
///     <item>>
/// [Return Parameter]<br/>
/// Indicates if the object should be selected (true case) or excluded (false case).
///     </item>
/// </list>
/// </param>
///
///
/// <param name="FormatFunction">
/// [Optional] A function which is used to format the conversion of the objects in the sequence.<br/>
/// Function signature:
/// <code>Func FormatFunction &lt;TSource InputObject, int PositionInSequence, string ReturnValue&gt;</code>
/// <list type="number">
/// <item>
///     [Input Parameter] <br/>
///     The object from the input sequence. <br/>
///     The type of the object is the same as the declaration of the sequence.
///     This type of the objects comes from the type parameter <typeparamref name="TSource">"/> in the declaration of this method.<br/>
///     For compound sequences like Dictionary the object is a KeyValuePair.
/// </item>
```

```
/// <item>
///     [Input Parameter]<br/>
///     The position the object occupies in the input sequence.
/// </item>
/// <item>
///     [Return Value]<br/>
///     A string representation of the object.
/// </item>
/// <item>[Default Value]<br/>
/// The default supplied if the argument is null is:
/// <code>(InputObject, Position) => string.Format("[{1}] {0}, ", Position, InputObject)</code>
/// </item>
/// </list>
/// </param>
///
/// <param name="OutputFunction">[Optional]
/// A function which is used to emit the formatted string into the output stream.<br/>
/// The function signature is:
/// <code>Func&lt;string ValueToEmit, bool OutputSucceeded&gt;</code>
/// Function Arguments:
/// <list type="number">
///     <item>
/// [Input Parameter]<br/>
/// This is the string value created by the format function,
/// and this sting is expected to be the value which is pushed into an output.
///     </item>
///     <item>
/// [Return Value]<br/>
/// A bool value indicating if the output was successful (true case).
///     </item>
///     <item>
/// [Default Value]<br/>
///     The default value supplied if the argument is null is as follows:
///     <code>
/// (StringRepesentation) =>
/// {
///     System.Diagnostics.Debug.Write(StringRepesentation);
///     return true;
/// };
///     </code>
///     </item>
/// </list>
/// </param>
///
```

```
/// <param name="SummaryCount">
/// [Optional] A bool value which causes the return value to be a count of the objects processed and written.</param>
/// <returns>The number of objects processed and written.</returns>
/// <remarks>
/// I will think of something.
/// </remarks>
/// <example>
/// The following provides a number of examples of invoking the ToOutput extension method.
/// <code>
/// private void TestTheToOutputMethod()
///{
///    // Declaring a simple array
///    int[] SimpleArray = new int[] { 1, 2, 3, 4 };
///    // Declaring a list of objects
///    List&lt;Tuple&lt;int, string&gt;&gt; SimpleObjects = new List&lt;Tuple&lt;int, string&gt;&gt;()
///    {
///Tuple.Create(1, "Test"),Tuple.Create(200, "String Test"),
///Tuple.Create(-21, "Testing"),   Tuple.Create(0, "the quick brown")
///    };
///    // A more complex object collection to test against
///    Dictionary&lt;long, int?&gt; DictTest = new Dictionary&lt;long, int?&gt;()
///    {
///{ 234L, null }, {-44345L, 65742 },
///{ -5644, null },{6799032L, 8765464 }
///    };
///
///    // Simplest invocation of the extension method
///    long outputCount = "test".ToOutput();
///    Debug.WriteLine(
///string.Format("Char Objects written {0}", outputCount));
///
///    // Test Against an array with no arguments.
///    outputCount = SimpleArray.ToOutput();
///    Debug.WriteLine(
///string.Format("Simple 4 Element int array {0}", outputCount));
///    try{
///using(StreamWriter outFile = new StreamWriter(@"..\..\..\Testing_Output\Linq_Dump.txt"))
///{
///    // Invoking ToOutput with inline lambda expressions
///    outputCount = SimpleArray.ToOutput(
///(objValue, Position) =&gt;
///    {
///if(Position % 2 == 0) return true;
///else return false;
```

```
///    },
///(objValue, Position) =&gt;
///    {
///return string.Format("Position={0}, Value={1}", Position, objValue);
///    },
///(objValue) =&gt;
///    {
///outFile.WriteLine(objValue);
///return true;
///    },
///true);
///    // Declaring a function delegates
///    Func&lt;Tuple&lt;int, string&gt;, int, bool&gt; Where1 =
///(objValue, position) =&gt;
///{
///    if (objValue.Item1 &gt;= 0) return true;
///    else return false;
///};
///    Func&lt;Tuple&lt;int, string&gt;, int, string&gt; Format1 =
///(objValue, Position) =&gt;
///    {
///return string.Format("Position={0}, Int Value={1}, String Value={2}",
///    Position, objValue.Item1, objValue.Item2);
///    };
///    Func&lt;string, bool&gt; Output1 =
///(strValue) =&gt;
///{
///    outFile.WriteLine(strValue);
///    Debug.WriteLine(strValue);
///    return true;
///};
///    // Invoking the function delegate to put a string into the output
///    Output1("Working with SimpleObjects");
///    // Invoking ToOutput with externally declared function delegates
///    outputCount = SimpleObjects.ToOutput(Where1, Format1, Output1, true);
///    // Declaring a function delegates
///    Func&lt;KeyValuePair&lt;long, int?&gt;, int, bool&gt; Where2 =
///(objValue, Position) =&gt;
///{
///    return objValue.Value.HasValue;
///};
///    Func&lt;KeyValuePair&lt;long, int?&gt;, int, string&gt; Format2 =
///(objValue, Position) =&gt;
///{
```

```
///    if (objValue.Value.HasValue)
///return string.Format("Position={0}, Long Key Value={1}, Value Int={2}",
///    Position, objValue.Key, objValue.Value.Value);
///    else
///return string.Format("Position={0}, Long Key Value={1}, Value Int=null",
///    Position, objValue.Key);
///};
///    // Invoking the function delegate to put a string into the output
///    Output1("Working with Dictionary Test Object");
///    // Invoking ToOutput with externally declared function delegates
///    // and using the SummaryCount default value
///    outputCount = DictTest.ToOutput(Where2, Format2, Output1);
///    // Invoking ToOutput with externally declared function delegates
///    // and using named optional arguments
///    outputCount = DictTest.ToOutput(FormatFunction: Format2, OutputFunction: Output1);
///    outFile.Flush();
///    outFile.Close();
///}
///    }
///    catch(Exception ex)
///    {
///Debug.WriteLine(ex);
///    }
///    return;
///}
///
/// </code>
/// </example>
public static long ToOutput<TSource>(this IEnumerable<TSource> Input,
    Func<TSource, int, bool> WherePredicate = null,
    Func<TSource, int, string> FormatFunction = null,
    Func<string, bool> OutputFunction = null,
    bool SummaryCount = true)
{
    if (FormatFunction == null)
FormatFunction = (InputObject, Position) => string.Format("[{1}] {0}, ", Position, InputObject);
    if (OutputFunction == null)
OutputFunction = (StringRepesentation) =>
{
    Debug.Write(StringRepesentation);
    return true;
};
    long ElementsDone = 0L;
    if (WherePredicate == null)
```

```
    {
ElementsDone = Input
    .Select((InputObject, Position) => FormatFunction(InputObject, Position))
    .LongCount((StringRepesentation) => OutputFunction(StringRepesentation));
    }
    else
    {
ElementsDone = Input
    .Where((InputObject, Position) => WherePredicate(InputObject, Position))
    .Select((InputObject, Position) => FormatFunction(InputObject, Position))
    .LongCount((StringRepesentation) => OutputFunction(StringRepesentation));
    }

    if (SummaryCount)
    {
string OutputCount = string.Format("\nObjects Processed = {0:#,#}\n", ElementsDone);
bool done = OutputFunction(OutputCount);
    }
    return ElementsDone;
}
```