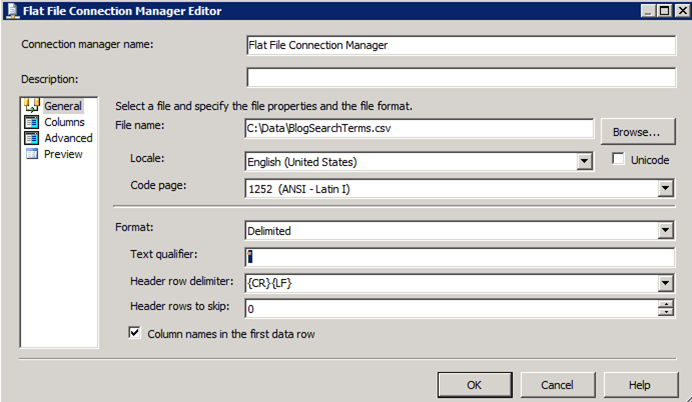
The Term Extraction transformation extracts terms from text in a transformation input column, and then writes the terms to a transformation output column**. The transformation works only with English text** and it uses **its own English dictionary and linguistic information about English**.

You can use the Term Extraction transformation to discover the content of a data set. For example, text that contains e-mail messages may provide useful feedback about products, so that you could use the Term Extraction transformation to extract the topics of discussion in the messages, as a way of analyzing the feedback.

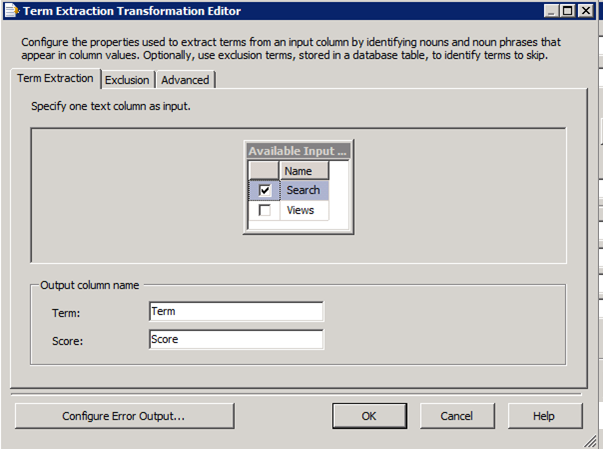
I created a new SSIS package, added a flat file to a CSV file. I specified the **Text qualifier**(Excel uses ” (double quote) by default), and marked that the first row contains headers.



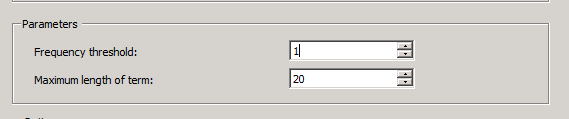
On Advanced tab, I set the data types (DT\_WSTR(500) and DT\_I4).

Note, even though the text values in the CSV are DT\_STR (ansi strings – no special characters), I used DT\_WSTR because that is what the Term Extraction transform supports. Alternatively, we could have used a Data Convert or Derived Column transform to perform the conversion, but I’d rather do it directly in the Source component.

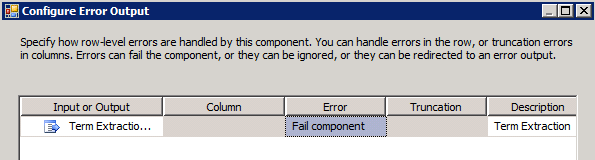
I then added a Data Flow Task, Flat File Source which used my new connection manager, and a Term Extraction transform. There’s not much configuration involved with the term extraction – you select the column you want to analyze, and (optionally) provide names for the output columns.



On the Advanced tab, I changed the **Frequency threshold** to 1 and **Maximum length of term** to 20. Changing these values can affect performance of the component, but since we’re processing a limited number of rows (about 500 total), I figured it was safe to do.



The Term Extraction transform redirects error rows by default, so you’ll get a warning if you don’t configure the output path. Instead, I configured the error output to **Fail Component** on error since we’re not expecting to have any errors in our list of simple terms.



I then closed off the data flow to see the results with a data viewer.

Running the package gave me a break down of the common terms.

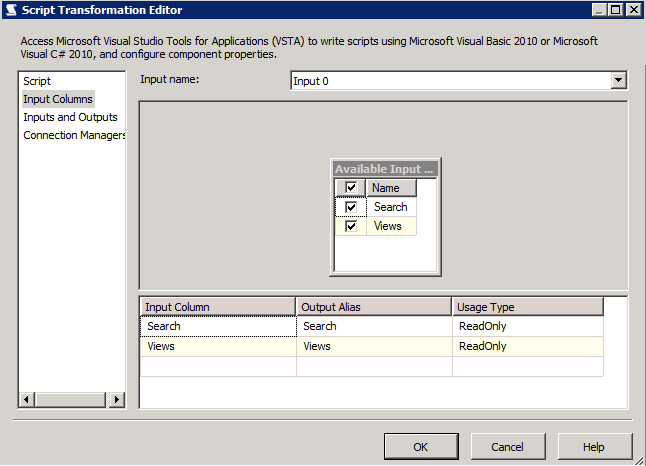
|  |  |
| --- | --- |
|  | clip_image008 |

The more the term appears in the input, the higher the score will be. This is really useful, but unfortunately it’s not completely representative of the actual search terms since the values had already been aggregates (the “Views” value from the original data set). To get a more accurate number, I had to expand the results before processing them.

Expanding the Search with a Script Component

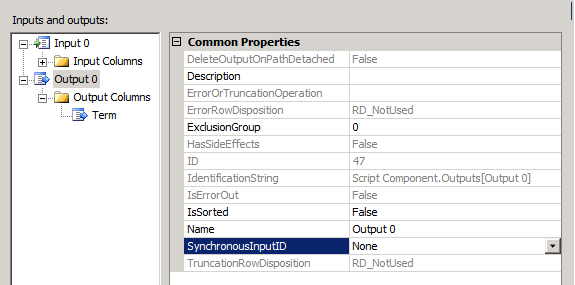
By default Script Components are synchronous – among other things, this means they will have the same number of rows going in as they have coming out. To expand the values counts correctly, we’ll need to add more rows to the data flow. This means I’ll need to [make the Script Component Asynchronous](http://msdn.microsoft.com/en-us/library/ms136133.aspx).

I selected both input columns.



I went to the **Inputs and Outputs** tab, selected the single output, and set the Synchronous ID value to None.

I added a column for the Term (DT\_WSTR(500)).



Open the script editor to start entering the code.

The code is pretty simple. Using an Asynchronous Script Component isn’t that different from a Synchronous one – you just need to remember that you’re adding rows in a new buffer, rather than changing values in the current one. The only method I needed to worry about is the one that processes the incoming rows – **Input0\_ProcessInputRow**.

For each incoming row, I want to look at the term (Search) and the number of times it appears (Views). I then programmatically add a row which contains the term to the output buffer for each time it was viewed.

|  |  |
| --- | --- |
|  | public override void Input0\_ProcessInputRow(Input0Buffer Row)      {          // retrieve the values once          string term = Row.Search;          int count = Row.Views;            // we output a row for each view          for (int i = 0; i < count; i++)          {              Output0Buffer.AddRow();              Output0Buffer.Term = term;          }      } |

Running the package after this change, I see that I have a lot more rows going into the Term Extraction transform then I did before (1043 vs. 498), and more accurate scores as a result.

