DMX-h ETL Use Case Accelerator

Word Count

Table of Contents

1 Introduction

Word Count is a common application used to explain how Hadoop works. The application identifies all uniquely occurring words in a set of source data, and for each word, computes the total number of occurrences of that word. This use case accelerator implements Word Count in DMX-h ETL by efficiently tokenizing lines into words and aggregating on the words to obtain a total count for each.

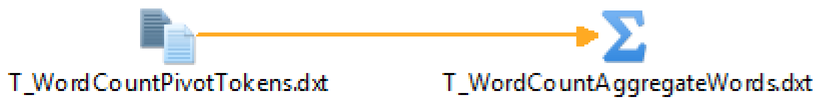
Two solutions are given:

- DMX-h Intelligent Execution (IX) – this solution, presented in the next section, is the simplest and most efficient way to develop this example. Taking advantage of IX, it allows the job to be run on the edge node, on a single cluster node, or in a Spark or MapReduce cluster as specified.
- User-defined MapReduce – this solution, provided as a reference in Appendix A. is more complex, and is used when you want to specifically control how the job is run. It can also be specified to run either standalone or in the cluster.

For guidance on setting up and running this and other use case accelerators, see the Guide to DMX-h ETL Use Case Accelerators.

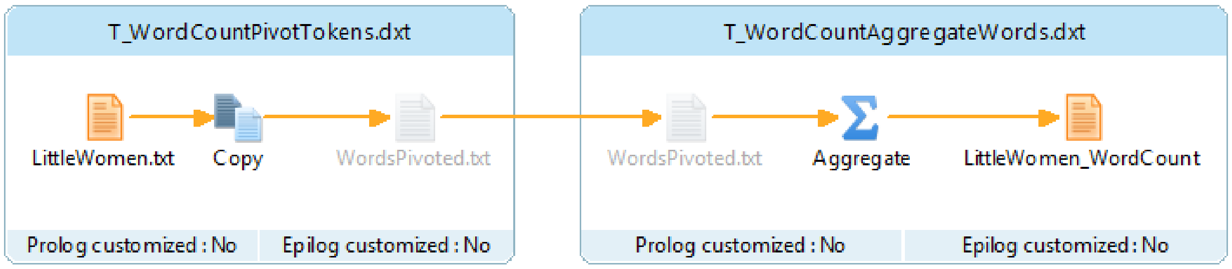
2 Word Count with DMX-h IX

The Word Count solution in DMX-h ETL consists of a job, J_WordCount.dxj, which contains a copy task followed by an aggregate task.



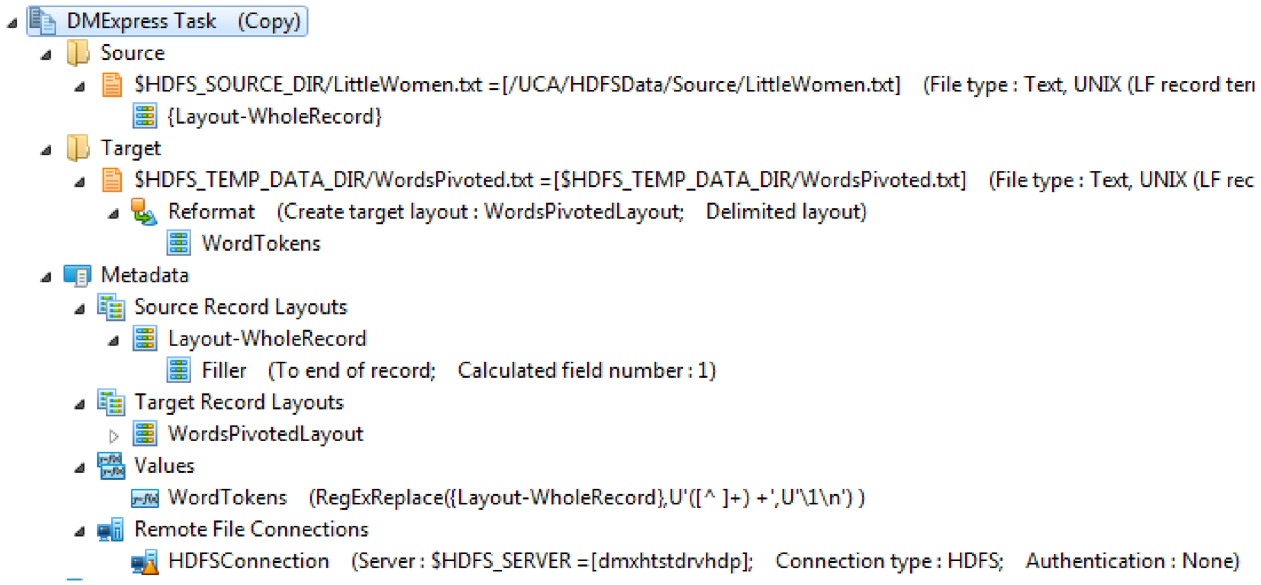
2.1 J_WordCount job

This job consists of a copy task responsible for tokenizing the input file, and an aggregate task to count the number of occurrences of a word.



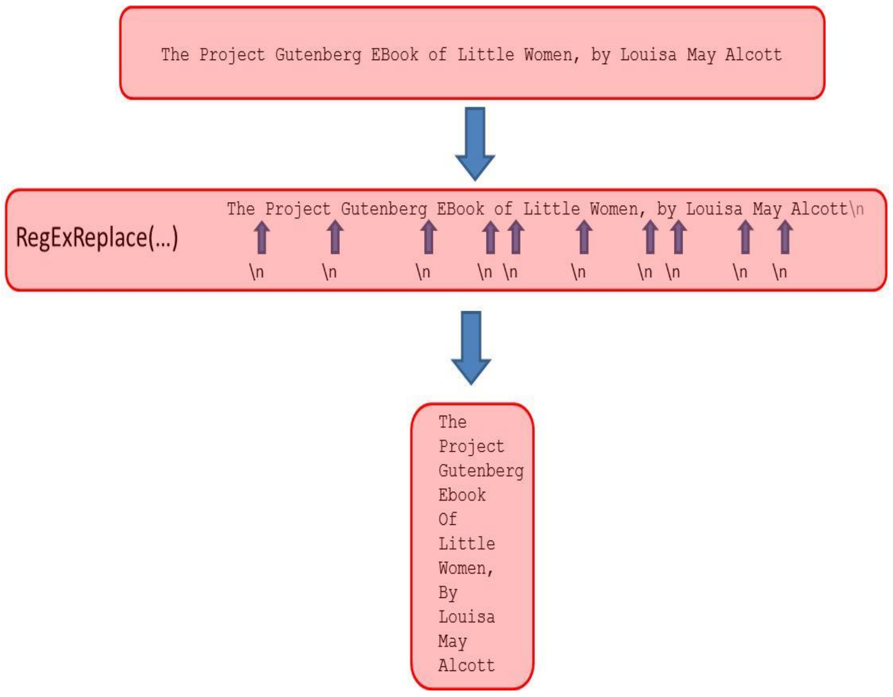
2.1.1 T_WordCountPivotTokens Task

This task uses a copy to read the file and replaces spaces between words with record terminators.



```
DMExpress Task   (Copy)
  Source
    $HDFS_SOURCE_DIR/LittleWomen.txt =[/UCA/HDFSData/Source/LittleWomen.txt]   (File type : Text, UNIX (LF record ter
      {Layout-WholeRecord}
  Target
    $HDFS_TEMP_DATA_DIR/WordsPivoted.txt =[$HDFS_TEMP_DATA_DIR/WordsPivoted.txt]   (File type : Text, UNIX (LF rec
      Reformat   (Create target layout : WordsPivotedLayout;   Delimited layout)
        WordTokens
  Metadata
    Source Record Layouts
      Layout-WholeRecord
        Filler   (To end of record;   Calculated field number : 1)
    Target Record Layouts
      WordsPivotedLayout
    Values
      WordTokens   (RegExReplace({Layout-WholeRecord},U'([^ ]+) +',U'\1\n') )
    Remote File Connections
      HDFSConnection   (Server : $HDFS_SERVER =[dmxhtstdrvhdp];   Connection type : HDFS;   Authentication : None)
```

The source in this example is an unstructured text file consisting of lines of text broken up by UNIX linefeed terminators. Although we specify a pipe (|) as a delimiter, no predetermined fields exist in the free-form line or record, so we describe a layout that consists of one field that extends to the end of the record. A copy task is sufficient to read the source file, as no ordering requirements exist for the first step.


The Project Gutenberg EBook of Little Women, by Louisa May Alcott
RegExReplace(...)
The Project Gutenberg EBook of Little Women, by Louisa May Alcott\n
\n \n \n \n \n \n \n \n \n \n
The
Project
Gutenberg
Ebook
Of
Little
Women,
By
Louisa
May
Alcott

We accomplish the pivoting by inserting record terminators between the words. Using the regular expression `'([^ ]+) +'`, we search for a group of one or more non-space characters (indicated with parentheses) followed by one or more spaces.

When we find this, we replace it with the preserved non-space group of characters followed by a newline (`'\n'`) instead of a space. The RegExReplace function finds and replaces all occurrences of the pattern. This introduces line breaks or record terminators between all previously space-separated tokens and words as well as between punctuation.

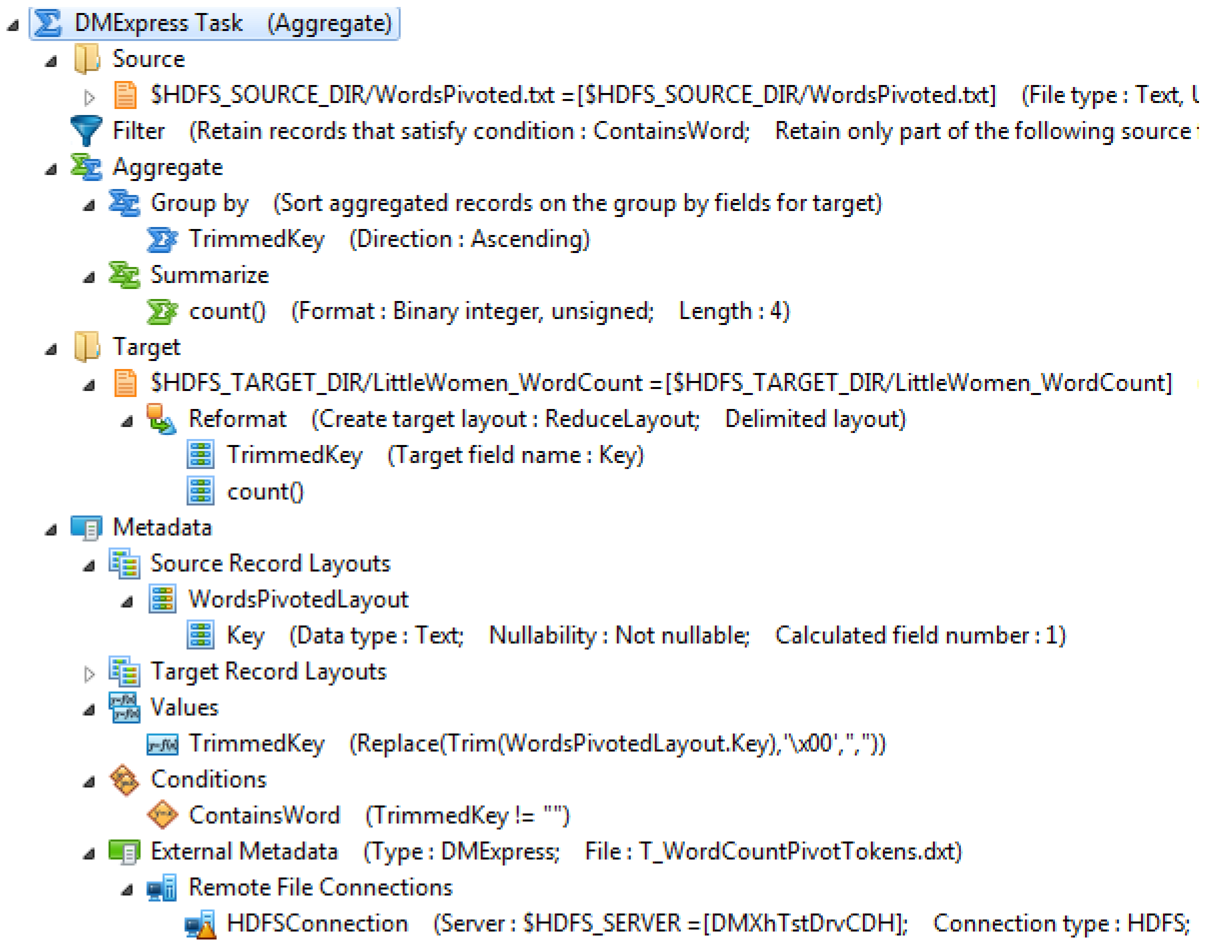
The RegExReplace expression is defined in the WordTokens value, which is included in the target via a reformat.

Whether the overall job is run in or outside of Hadoop, this task will run on the edge node.

2.2 T_WordCountAggregateWords Task

This task reads the output of the previous task and performs two processing steps:

1. Empty records are filtered out.
2. Records are trimmed to remove whitespace, and aggregated to obtain a final count.



```
◢ DMExpress Task   (Aggregate)
    ◢ Source
        ▷ $HDFS_SOURCE_DIR/WordsPivoted.txt =[$HDFS_SOURCE_DIR/WordsPivoted.txt]   (File type : Text, U
      Filter   (Retain records that satisfy condition : ContainsWord;   Retain only part of the following source
    ◢ Aggregate
        ◢ Group by   (Sort aggregated records on the group by fields for target)
              TrimmedKey   (Direction : Ascending)
        ◢ Summarize
              count()   (Format : Binary integer, unsigned;   Length : 4)
    ◢ Target
        ◢ $HDFS_TARGET_DIR/LittleWomen_WordCount =[$HDFS_TARGET_DIR/LittleWomen_WordCount]
            ◢ Reformat   (Create target layout : ReduceLayout;   Delimited layout)
                  TrimmedKey   (Target field name : Key)
                  count()
    ◢ Metadata
        ◢ Source Record Layouts
            ◢ WordsPivotedLayout
                  Key   (Data type : Text;   Nullability : Not nullable;   Calculated field number : 1)
        ▷ Target Record Layouts
        ◢ Values
              TrimmedKey   (Replace(Trim(WordsPivotedLayout.Key),'\x00',"",""))
        ◢ Conditions
              ContainsWord   (TrimmedKey != "")
        ◢ External Metadata   (Type : DMExpress;   File : T_WordCountPivotTokens.dxt)
            ◢ Remote File Connections
                  HDFSConnection   (Server : $HDFS_SERVER =[DMXhTstDrvCDH];   Connection type : HDFS;
```

The source, a pipe-delimited UNIX text file, is the target of the previous task. The source record contains only one field, WordTokens, whose field values consist of the single token or word produced by the previous task.

Before any other processing, we use a bulk filter to eliminate empty lines since they contain no words. We also want to eliminate lines that consist of only spaces. To do this, first we create a value, TrimmedKey, that uses the Trim() function to eliminate leading and trailing spaces, leaving just the word itself, if any. We then use a condition, ContainsWord, to retain only records that contain words (TrimmedKey != “”).

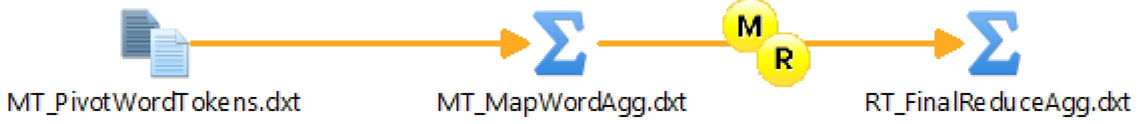
A sorted aggregation is performed on the remaining records, grouping by the field TrimmedKey, which is the word being counted. The count() function is used to obtain the final occurrence count for each key.

A record is produced as output for each unique word, followed by the tab character and the total count of occurrences of that word in the original source file.

Appendix A Word Count with DMX-h User-defined MapReduce

This user-defined MapReduce solution is provided as a reference in the event that particular knowledge of your application's data would benefit from manual control of the MapReduce process.

The Word Count job in DMX-h contains a map step and a reduce step, separated by a MapReduce data flow connector between the final map task and the initial reduce task, as follows:

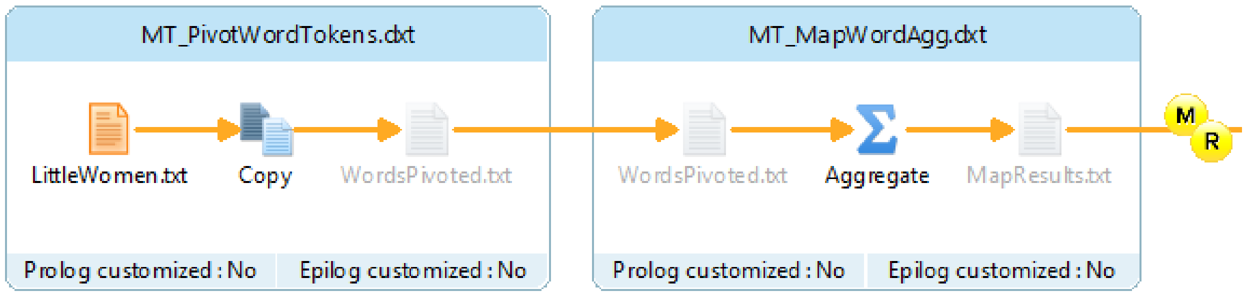


The map step first reads an input file containing the full text of the novel *Little Women* by Louisa May Alcott and pivots the data to tokenize individual words. It then assigns partition IDs to the data such that all identical words will go to the same reducer. Each mapper also aggregates the words known to the mapper, producing a partial count of those words. The aggregation is partial since each mapper is passed only a subset of the input data. The partial map-side aggregation is useful, however, considering the frequency of words such as “a”, “the”, and “and”. This partial aggregation subsequently reduces the amount of data that must be processed by the reducers.

The reduce step receives partial word counts from the mappers. Since for any given word, all partial counts for that word go to the same reducer, the reducers total all incoming counts of each word into a final count for that word. Finally, the reduce step writes all words and corresponding word counts to the target.

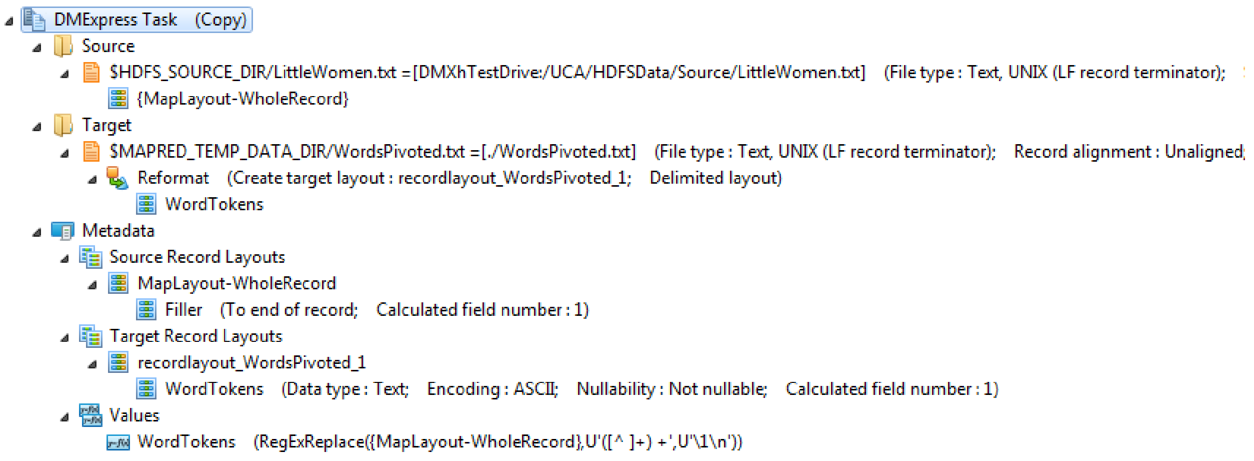
A.1 Map Step

The Word Count map step in DMX-h consists of a task that pivots the data to create one record per word, and another task that assigns a partition ID to each record based on the key and performs a partial count of each word before sending the partitioned data to the reduce step.



A.1.1 MT_PivotWordTokens.dxt

This task uses a copy to read the file and replaces spaces between words with record terminators.



```
DMExpress Task    (Copy)
  Source
    $HDFS_SOURCE_DIR/LittleWomen.txt =[DMXhTestDrive:/UCA/HDFSData/Source/LittleWomen.txt]    (File type : Text, UNIX (LF record terminator);
      {MapLayout-WholeRecord}
  Target
    $MAPRED_TEMP_DATA_DIR/WordsPivoted.txt =[./WordsPivoted.txt]    (File type : Text, UNIX (LF record terminator);    Record alignment : Unaligned;
      Reformat    (Create target layout : recordlayout_WordsPivoted_1;    Delimited layout)
        WordTokens
  Metadata
    Source Record Layouts
      MapLayout-WholeRecord
        Filler    (To end of record;    Calculated field number : 1)
    Target Record Layouts
      recordlayout_WordsPivoted_1
        WordTokens    (Data type : Text;    Encoding : ASCII;    Nullability : Not nullable;    Calculated field number : 1)
    Values
      WordTokens    (RegExReplace({MapLayout-WholeRecord},U'([^ ]+) +',U'\1\n'))
```

The source in this example is an unstructured text file. The source consists of lines of text broken up by UNIX line terminators (linefeed). Although we specify a pipe (|) as a delimiter, no predetermined fields exist in the free form line or record, so we describe a layout that consists of one field that extends to the end of the record. A copy task is used to read the source file as no ordering requirements exist for the first step.

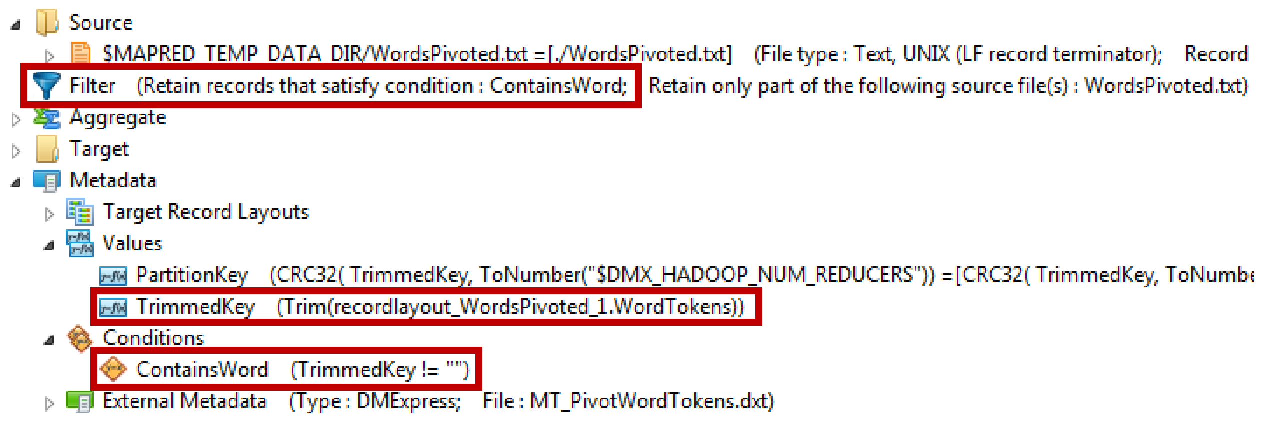
We now accomplish the pivoting by inserting record terminators between the words. Using the expression `'([^ ]+) +'`, we search for a group of one or more non-space characters (indicated with parentheses) followed by one or more spaces.

When we find this, we replace it with the preserved non-space group of characters followed by a newline (`'\n'`) instead of a space. The RegExReplace function finds and replaces all occurrences of the pattern. This introduces line breaks or record terminators between all previously space-separated tokens and words as well as between punctuation.

The RegExReplace expression is defined in the WordTokens value, which is included in the target via a reformat.

A.1.2 MT_MapWordAgg.dxt

The second task performs an aggregation on the words and partitions the data in preparation for the reduce step.



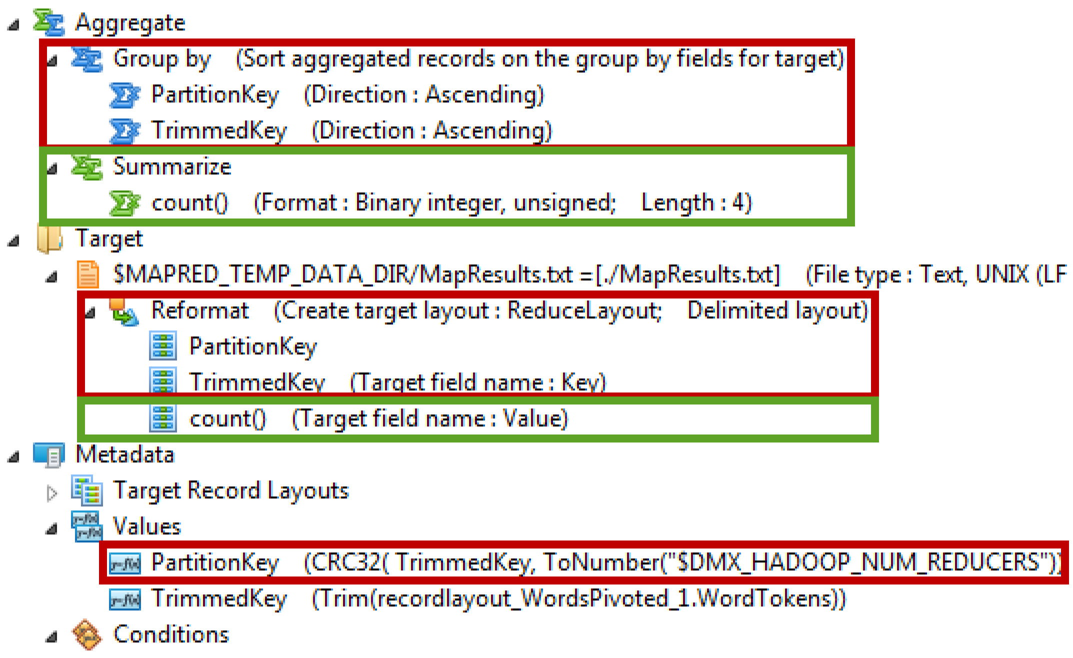
```
Source
  $MAPRED_TEMP_DATA_DIR/WordsPivoted.txt =[./WordsPivoted.txt]   (File type : Text, UNIX (LF record terminator);   Record
Filter   (Retain records that satisfy condition : ContainsWord;  Retain only part of the following source file(s) : WordsPivoted.txt)
Aggregate
Target
Metadata
  Target Record Layouts
  Values
    PartitionKey   (CRC32( TrimmedKey, ToNumber("$DMX_HADOOP_NUM_REDUCERS")) =[CRC32( TrimmedKey, ToNumbe
    TrimmedKey   (Trim(recordlayout_WordsPivoted_1.WordTokens))
  Conditions
    ContainsWord   (TrimmedKey != "")
  External Metadata   (Type : DMExpress;   File : MT_PivotWordTokens.dxt)
```

The source, a pipe-delimited UNIX text file, is the target of the previous task. Only one field, which is named WordTokens, exists in the source record. The field values consist of the single token or word produced by the previous task.

Before any other processing, we use a bulk filter to eliminate empty lines since they contain no words. Instead of empty lines, we could also have lines that only consist of spaces. To eliminate trailing and leading spaces, we create a new value, TrimmedKey, which is the word coming in with spaces removed using the Trim() function. We then filter using a named condition, ContainsWord, to filter all records that when trimmed contain no words (TrimmedKey != “”).

Since all records with the same primary key value (PartitionKey + TrimmedKey) must go to the same reducer in order for all occurrences of each word to be totaled, the partition ID is determined by creating a CRC32() hash value based on this key. In addition, the environment variable DMX_HADOOP_NUM_REDUCERS (automatically set to the configured number of reducers when running in Hadoop) is passed to the function to limit the range of partition ID values to the number of reducers invoked for your MapReduce job.

For the aggregation, we group by the PartitionKey as well as the new value, TrimmedKey. We count the occurrences of the same word by introducing the count() aggregation function. This is only a partial count for each word, since the input to each mapper is a subset of the overall source data. The count format is set to 4-byte unsigned binary integer to improve performance.

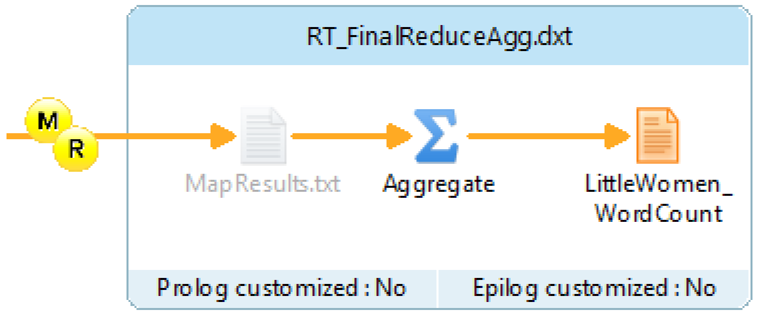


```
Aggregate
  Group by   (Sort aggregated records on the group by fields for target)
    PartitionKey   (Direction : Ascending)
    TrimmedKey   (Direction : Ascending)
  Summarize
    count()   (Format : Binary integer, unsigned;   Length : 4)
Target
  $MAPRED_TEMP_DATA_DIR/MapResults.txt =[./MapResults.txt]   (File type : Text, UNIX (LF
    Reformat   (Create target layout : ReduceLayout;   Delimited layout)
      PartitionKey
      TrimmedKey   (Target field name : Key)
      count()   (Target field name : Value)
Metadata
  Target Record Layouts
  Values
    PartitionKey   (CRC32( TrimmedKey, ToNumber("$DMX_HADOOP_NUM_REDUCERS"))
    TrimmedKey   (Trim(recordlayout_WordsPivoted_1.WordTokens))
  Conditions
```

The option to sort aggregated records by group by fields is selected so that the data is sorted by the partition ID, a requirement to ensure the data is routed to the correct reducer. Further, the partition ID is output as the first field in the Reformat, also a requirement for correct MapReduce operation.

A.2 Reduce Step

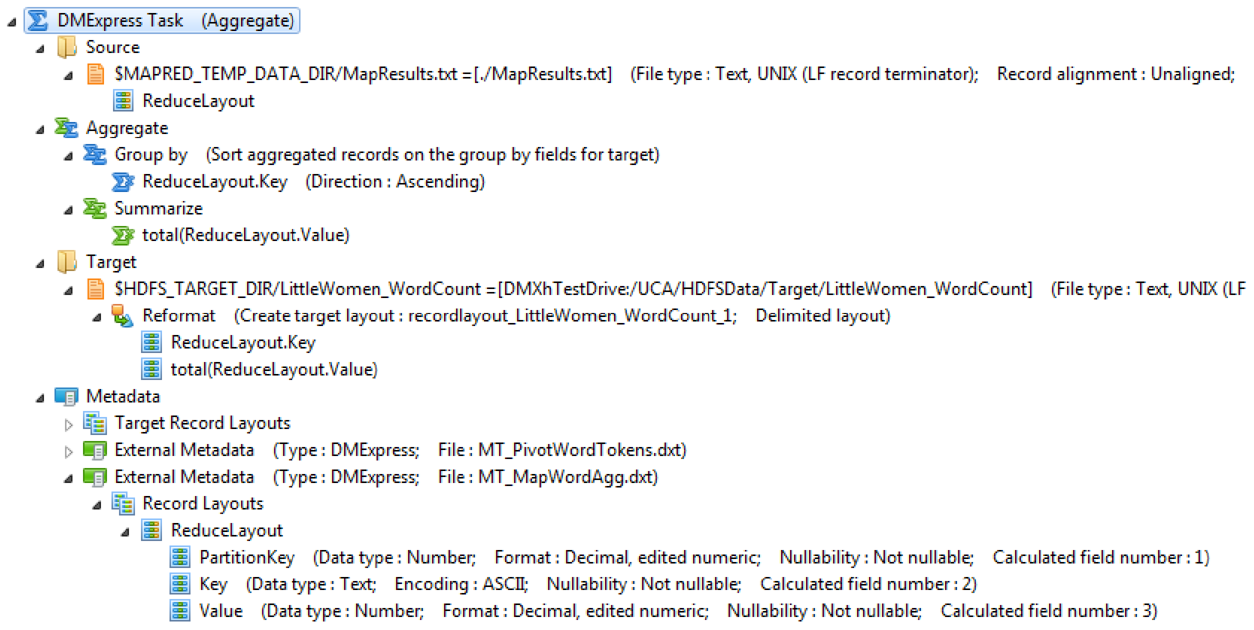
The Word Count reduce step in DMX-h ETL consists of one task that aggregates the partial word counts calculated during the map step into a final count for each word.



As the partition IDs were assigned by the map job based on individual words, partial counts from different mappers for each word are routed to the same reducer, ensuring that all occurrences of each word are accounted for in the reduce step.

A.2.1 RT_FinalReduceAgg.dxt

The input to this task is made up of pairs of words and partial word counts from different mappers. These partial word counts will be summed into a total word count for each word.



```
DMExpress Task   (Aggregate)
  Source
    $MAPRED_TEMP_DATA_DIR/MapResults.txt = [./MapResults.txt]   (File type : Text, UNIX (LF record terminator);   Record alignment : Unaligned;
      ReduceLayout
  Aggregate
    Group by   (Sort aggregated records on the group by fields for target)
      ReduceLayout.Key   (Direction : Ascending)
    Summarize
      total(ReduceLayout.Value)
  Target
    $HDFS_TARGET_DIR/LittleWomen_WordCount = [DMXhTestDrive:/UCA/HDFSData/Target/LittleWomen_WordCount]   (File type : Text, UNIX (LF
      Reformat   (Create target layout : recordlayout_LittleWomen_WordCount_1;   Delimited layout)
        ReduceLayout.Key
        total(ReduceLayout.Value)
  Metadata
    Target Record Layouts
    External Metadata   (Type : DMExpress;   File : MT_PivotWordTokens.dxt)
    External Metadata   (Type : DMExpress;   File : MT_MapWordAgg.dxt)
      Record Layouts
        ReduceLayout
          PartitionKey   (Data type : Number;   Format : Decimal, edited numeric;   Nullability : Not nullable;   Calculated field number : 1)
          Key   (Data type : Text;   Encoding : ASCII;   Nullability : Not nullable;   Calculated field number : 2)
          Value   (Data type : Number;   Format : Decimal, edited numeric;   Nullability : Not nullable;   Calculated field number : 3)
```

A sorted aggregation is performed, grouping by the field “Key,” which is the word being counted. The field “Value,” containing partial counts of each word, is summed for records with matching word keys. A record is produced as output for each unique word, which is followed by the tab character and the total count of occurrences of that word in the original source file.

The Hadoop framework will create a directory with the target name specified here, and each reducer’s output will be written to its own “part-reducer_ID” file within that directory.

About Syncsort

Syncsort Inc. ***50 Tice Boulevard, Suite 250, Woodcliff Lake, NJ 07677*** ***201.930.8200***