White Paper

HPCC Systems: ECL For Hadoopers

LexisNexis®

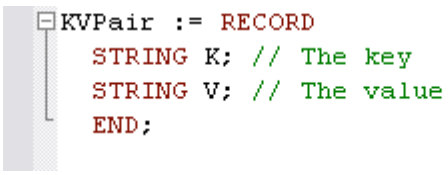
Table of Contents

ECL for Hadoopers

ECL has an excellent reference manual and many good 'starting from scratch' training resources that can teach ECL; this document does not aim to replace any of those. Rather this paper is designed for those already experienced in Hadoop to answer the question: "how do I do X in ECL" – where X is a common Hadoop function. The code presented here is designed to work and to 'translate' from Hadoop as simply as possible. It does **not** guarantee that the result is necessarily the **ideal** solution that would have been reached if the developer were thinking in ECL natively. That said there will be some 'advanced' and 'very advanced' tips given designed to tease the Hadoop developer into a more ECL style mind-set.

The Key-Value Pair

Hadoop is centered round a data-model of the key-value pair[1]. The key value pair is not explicitly built in to ECL but one can be declared easily using the RECORD structure.



```
KVPair := RECORD
  STRING K; // The key
  STRING V; // The value
  END;
```

Advanced: The type of the Key does not have to be a variable length string; making it a fixed length string or integer will improve performance – sometimes considerably.

Advanced: ECL does not restrict the user to one value. You can have multiple values each with their own type. This avoids the need to 'parse and unparse' structured values from the value portion of the KV pair.

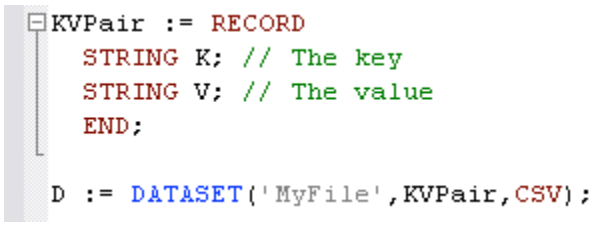
Very Advanced: ECL doesn't restrict you to one key either; you can have multiple keys of multiple types stored in the same record and used as appropriate.

Don't Read This: In fact in ECL, any value, combination of values, or expressions upon values can be used as a key at any part of the process

1. This is not always ideal; see "Data Models for Big Data"

Reading Data

In Hadoop the data is generally read in using one of the default or adapted file reading classes. In ECL the reading of data is handled using the DATASET function. The layout of the data is specified using the RECORD structure we have already seen. Therefore if 'myfile' was a file that was a saved KVPair structure using comma separated variable format one would say:



```
KVPair := RECORD
  STRING K; // The key
  STRING V; // The value
  END;

D := DATASET('MyFile',KVPair,CSV);
```

Basic Note: The CSV format has many options (and there are other options such as XML and Fixed File) which are explained in the LRM.

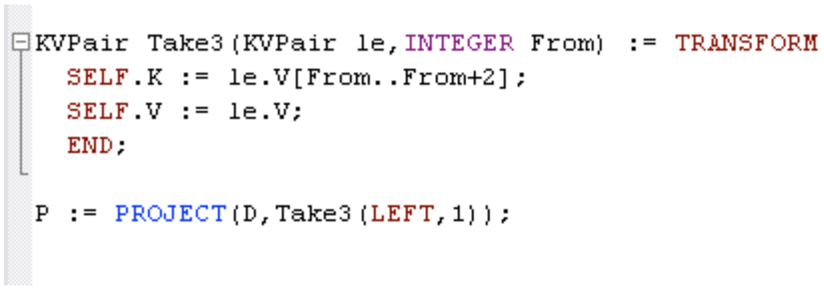
Basic Note: For information regarding how to get data in and out of an HPCC cluster see our Data Tutorial

Advanced Note: For files that are NOT saved in a 'Key Value' pair structure one needs to adapt the record structure to reflect the layout of the data on disk. Then if a KVPair structure is required a Mapper (see next section) is used.

Very Advanced Note: It is possible for ECL to use a language written in another program to read disk (or even a socket) into ECL as if it were a file – in a fashion similar to Streaming Hadoop. This is done using the PIPE option on the dataset command.

The Mapper

ECL does have something called MAP but it is not directly related to the Hadoop Mapper. The equivalent of the Hadoop Mapper is the PROJECT built-in function; or rather more accurately the transform passed to the PROJECT function. Thus suppose we wished to take the file D declared previously and give ourselves a key of the first three characters of the value – we would use:



```
KVPair Take3(KVPair le,INTEGER From) := TRANSFORM
  SELF.K := le.V[From..From+2];
  SELF.V := le.V;
  END;

P := PROJECT(D,Take3(LEFT,1));
```

The TRANSFORM is really performing the mapping operation. It is declared as returning a KVPair and as taking a KVPair and an integer parameter (which is used to pick where the 3 characters come from). The PROJECT statement is used to say 'apply this transform once to each record in the file'.

Basic Note: Transforms can be used by any number of projects

Advanced Note: There is a special form of project that allows the records to receive a **sequential** number as the key (or value!) – this is the PROJECT with a COUNTER

Very Advanced Note: It is quite possible to produce more than one record of output for every record of input; in this case one uses NORMALIZE rather than PROJECT to drive the transform. For example to produce all the trigrams from the value – substitute this line for the P := PROJECT

```
N := NORMALIZE(D,LENGTH(LEFT.V)-2,Take3(LEFT,COUNTER));
```

The Shuffle

The Hadoop 'reduce' is split into three different phases within Hadoop and each of them has an equivalent in ECL. The first part of the Hadoop shuffle is used to distribute the data across a number of reducers – the equivalent of this in ECL is the DISTRIBUTE function:

```
Di := DISTRIBUTE(N,HASH(K));
```

Basic Note: Within ECL every node in a cluster is numbered; the numbers range from 0 to the number of nodes-1. The HASH value is used modulus the cluster size to assign every record to a node.

Advanced Note: Generally ECL works best when all of the nodes available are used; however by placing your own modulus operator into the equation it is possible to reduce the number of nodes used.

The second part of the shuffle sorts the data at the reducer so that those elements with the same value for the key are together. This is achieved in ECL using SORT,LOCAL:

```
S := SORT(Di,K,LOCAL);
```

Advanced Note: It is possible to not just sort by K but also to sort by some or all of the values in the key-value pair too.

Advanced Note: Here the default HASH was used; HASH32 would probably be faster in this context

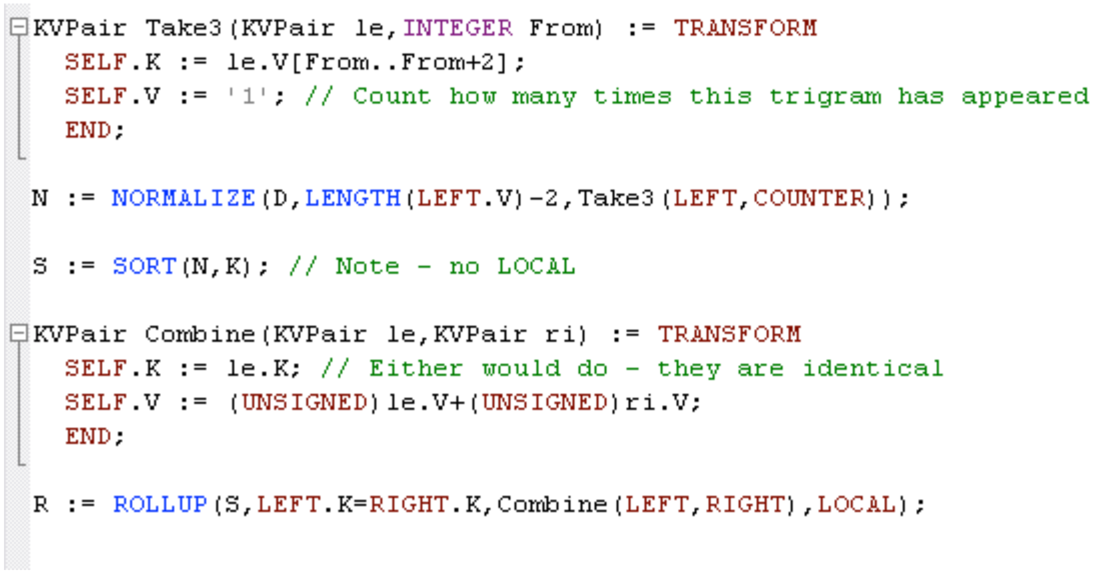
Very Advanced: The above works well (and like Hadoop) in the case where the data is evenly distributed by K. If your key cardinalities have a skewed distribution then one or more of the reducing servers will be overworked – slowing the whole process. It is therefore possible to replace the entire shuffle with:

```
S := SORT(N,K); // Note - no LOCAL
```

This will then use a patented algorithm to distribute the data according to the frequencies of the keys.

The Reduce

Once the data has been distributed and sorted appropriately then it is possible to reduce it using the ROLLUP function. For completeness and by way of example here is the full 'trigram' counting code:



```
KVPair Take3(KVPair le,INTEGER From) := TRANSFORM
  SELF.K := le.V[From..From+2];
  SELF.V := '1'; // Count how many times this trigram has appeared
  END;

N := NORMALIZE(D,LENGTH(LEFT.V)-2,Take3(LEFT,COUNTER));

S := SORT(N,K); // Note - no LOCAL

KVPair Combine(KVPair le,KVPair ri) := TRANSFORM
  SELF.K := le.K; // Either would do - they are identical
  SELF.V := (UNSIGNED)le.V+(UNSIGNED)ri.V;
  END;

R := ROLLUP(S,LEFT.K=RIGHT.K,Combine(LEFT,RIGHT),LOCAL);
```

It can be seen that the reduce really has two pieces; the transform that does the reducing and the rollup which specifies the records upon which the transform occurs.

Basic Note: Having our value as a string here made life more difficult; if V had been an integer the cast to UNSIGNED could have been avoided.

Advanced Note: The default rollup combines all the records one at a time – starting with the first in line. This is the most flexible solution. However it is also possible to form a rollup which takes ALL of the values for a given key and processes them all in one function. This is documented in the LRM.

Very Advanced Note: As you may have guessed; the rollup condition does not have to be simply 'Key=Key'. You could perform the operation on a subset of the key, a superset of the key and even some expression not involving the key at all[2].

And Finally

In closing it should be stressed; the preceding is designed to be a 'quick start index' into ECL for those familiar with Hadoop. Hopefully it has served to show how you can do what you already do. However I hope it has also whetted your appetite to see what **else** you can do and to encourage deeper exploration of a very rich and flexible language.

2. Although you would then need to adapt your SORT to ensure that the records to be compared were still next to each other

For more information:
Website: http://hpccsystems.com/
Email: info@hpccsystems.com
US inquiries: 1.877.316.9669
International inquiries: 1.678.694.2200

About HPCC Systems

HPCC Systems from LexisNexis® Risk Solutions offers a proven, data-intensive supercomputing platform designed for the enterprise to solve big data problems. As an alternative to Hadoop, HPCC Systems offers a consistent data-centric programming language, two processing platforms and a single architecture for efficient processing. Customers, such as financial institutions, insurance carriers, insurance companies, law enforcement agencies, federal government and other enterprise-class organizations leverage the HPCC Systems technology through LexisNexis® products and services. For more information, visit http://hpccsystems.com.

About LexisNexis Risk Solutions

LexisNexis® Risk Solutions (http://lexisnexis.com/risk/) is a leader in providing essential information that helps customers across all industries and government predict, assess and manage risk. Combining cutting-edge technology, unique data and advanced scoring analytics, Risk Solutions provides products and services that address evolving client needs in the risk sector while upholding the highest standards of security and privacy. LexisNexis Risk Solutions is headquartered in Alpharetta, Georgia, United States.

LexisNexis®