

HPCC Systems™

Six Degrees of Kevin Bacon

Boca Raton Documentation Team

Six Degrees of Kevin Bacon: ECL Programming Example

Boca Raton Documentation Team

Copyright © 2012 HPCC Systems. All rights reserved

We welcome your comments and feedback about this document via email to <docfeedback@hpccsystems.com> Please include **Documentation Feedback** in the subject line and reference the document name, page numbers, and current Version Number in the text of the message.

LexisNexis and the Knowledge Burst logo are registered trademarks of Reed Elsevier Properties Inc., used under license. Other products, logos, and services may be trademarks or registered trademarks of their respective companies. All names and example data used in this manual are fictitious. Any similarity to actual persons, living or dead, is purely coincidental.

February 2012 Version 3.6.2.1

Working with Data	4
Introduction	4
Processing the Data	5
Getting Useful Information from Data	17
Next Steps	21

Working with Data

Introduction

This exercise shows the methodology to extract useful information from data. Finding interesting links and relationships from large or massive datasets is a typical use of the HPCCSystems High Performance Computing Cluster (HPCC) platform.

In this example, we will download the data files from the Internet Movie Database (IMDB) and see one technique to extract links and find relationships.

Since the concept of actors and movies is conceptually simple; everyone should understand the data and relationships intuitively. However, the data is comprehensive enough to provide a solid example and inspiration for new users to gain skills to attack their own real-world problems with an HPCC.

In this example, we will:

- Download raw data files and supporting documentation about the data
- Analyze the data file to understand its format and contents
- Spray the file to a Data Refinery (THOR) cluster
- Examine the data and determine the pre-processing needed
- Pre-process the data to produce a new data file



While this example will run on a single-node HPCC, you will see a dramatic difference in performance on a multi-node system. The true power of an HPCC is its ability to work on different portions of the data file in parallel. This is known as Massively Parallel Processing (MPP).

Processing the Data

We get a data file

The Internet Movie Database (IMDB) database is a freely downloadable set of data files about Movies.

It can be downloaded in many formats, including text file format. The set includes approximately 48 files about Actors, Actresses, Directors, Producers, and other aspects of motions pictures.

It is manageable in size (~400MB) and is sufficient in size to exercise an HPCC platform but not too big to download.

The plain text data files are available from the following ftp sites:

- <ftp://ftp.fu-berlin.de/pub/misc/movies/database/> (Germany)
- <ftp://ftp.funet.fi/pub/mirrors/ftp.imdb.com/pub/> (Finland)
- <ftp://ftp.sunet.se/pub/tv+movies/imdb/> (Sweden)

The files are compressed using GNUzip to save space and bandwidth.

We will focus initially on two of the larger data sets in the IMDB database

- The Actors Dataset (Approximately 4 million Records)
- The Actresses Dataset (Approximately 2 million Records)
- Download the plain text data files (*actors.list.gz* and *actresses.list.gz*) to your local drive using any ftp interface you choose.
- Extract the two data files (*actors.list* and *actresses.list*) using any GNUzip interface.

Analyze the data file to understand its format and its contents

Here is the sample of the data in the Actors.list file from IMDB

```
Koolout' Starks, Johnny Nothing Like the Holidays (2008) [Alexis' Thug] <35>
Subtle Seduction (2008) [Officer Ward]
The Godfather of Green Bay (2005) (as Johnny Starks) [Marcus] <18>

La Chispa', Tony Caceria de judiciales (1997) <11>
Violencia en la sierra (1995) [Victoriano] <4>
```

Notice the actors text file is structured as follows

```
Blankline
Actorname_i Moviename (year) [role] <listing position>
      Moviename (year) [role] <listing position>
      Moviename (year) [role] <listing position>
      :
Blankline
Actorname_j \t Moviename (year) [role] <listing position>
      :
Blankline
```

Load the Incoming Data File to your Landing Zone

In this step, you will copy the data files to a location from which it can be sprayed to your HPCC cluster. A Landing Zone is a storage location attached to your HPCC. It has a utility running to facilitate file spraying to a cluster.

For smaller data files, maximum of 2GB, you can use the upload/download file utility in ECL Watch. The sample data files are ~400 mb.

Next you will distribute (or Spray) the dataset to all the nodes in the HPCC cluster. The power of the HPCC comes from its ability to assign multiple processors to work on different portions of the data file in parallel.

1. Download the sample data files from the ftp sites as described in the previous section, if you have not done so already.
2. Extract them to a folder on your local machine.
3. In your browser, go to the **ECL Watch** URL For example, <http://nnn.nnn.nnn.nnn:8010>, where nnn.nnn.nnn.nnn is your ESP Server's IP address.



Your IP address could be different from the ones provided in the example images. Please use the IP address provided by **your** installation.

- From ECL Watch page, click on the **Upload/download File** link in the menu on the left side.

Figure 1. Upload/download



Once you click on the Upload/download file link, it will take you to the Dropzones and Files page, where you can choose to **Browse** your machine for a file to upload:

Figure 2. Dropzones and Files



- Press the **Browse** button to browse the files on your local machine, select the file to upload and then press the **Open** button.

The file you selected should appear in the **Select a file to upload:** field. The data files are named: *actors.list* and *actresses.list*

6. Press on **Upload Now** to complete the file upload.
7. Repeat the process for the other data file.

Spray the Data File to your *Data Refinery (THOR) Cluster*

To use the data file in our HPC system, we must "spray" it to all the nodes. A *spray* or *import* is the relocation of a data file from one location (such as a Landing Zone) to multiple file parts on nodes in a cluster.

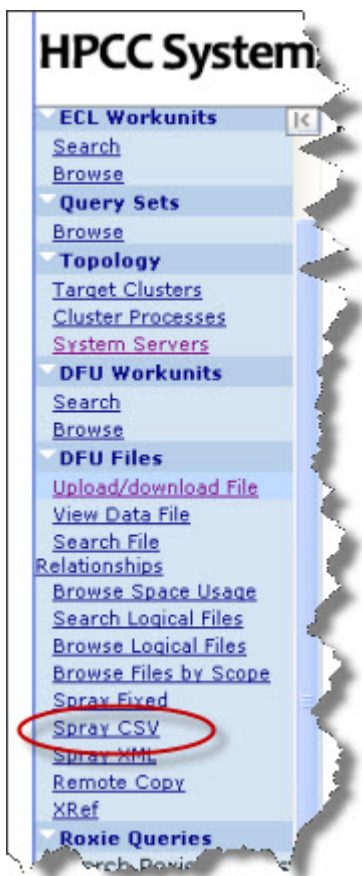
The distributed or sprayed file is given a *logical-file-name* as follows: **~thor::in::IMDB::actors.list** The system maintains a list of logical files and the corresponding physical file locations of the file parts.

- Open ECL Watch using the following URL:

http://nnn.nnn.nnn.nnn:pppp(where **nnn.nnn.nnn.nnn** is your ESP Server's IP Address and **pppp** is the port. The default port is 8010)

- CLICK on the Spray CSV hyperlink under the DFU Files menu on the left.

Figure 3. Spray CSV



The DFU Spray CSV page displays.

- Select mydropzone in the Source **Machine/dropzone** drop-list.

The IP Address is automatically filled and the Local Path is partially filled with the default folder on your landing zone. Note: The VM and Community Edition typically only has one landing zone defined.

- Complete the Local Path to include the complete file name or use the **Choose File** button to select the file from a list of files in the folder. The file is **actors.list**
- Fill in the rest of the parameters (if they are not filled in already).
 - Max Record Length 8192
 - Separator \,
 - Line Terminator \n,\r\n
 - Quote: '
- Fill in the Label using the Logical File name desired: : ~thor::in::IMDB::actors.list
- Make sure the **Overwrite** and **Replicate** boxes are checked.

Figure 4. Spray the File

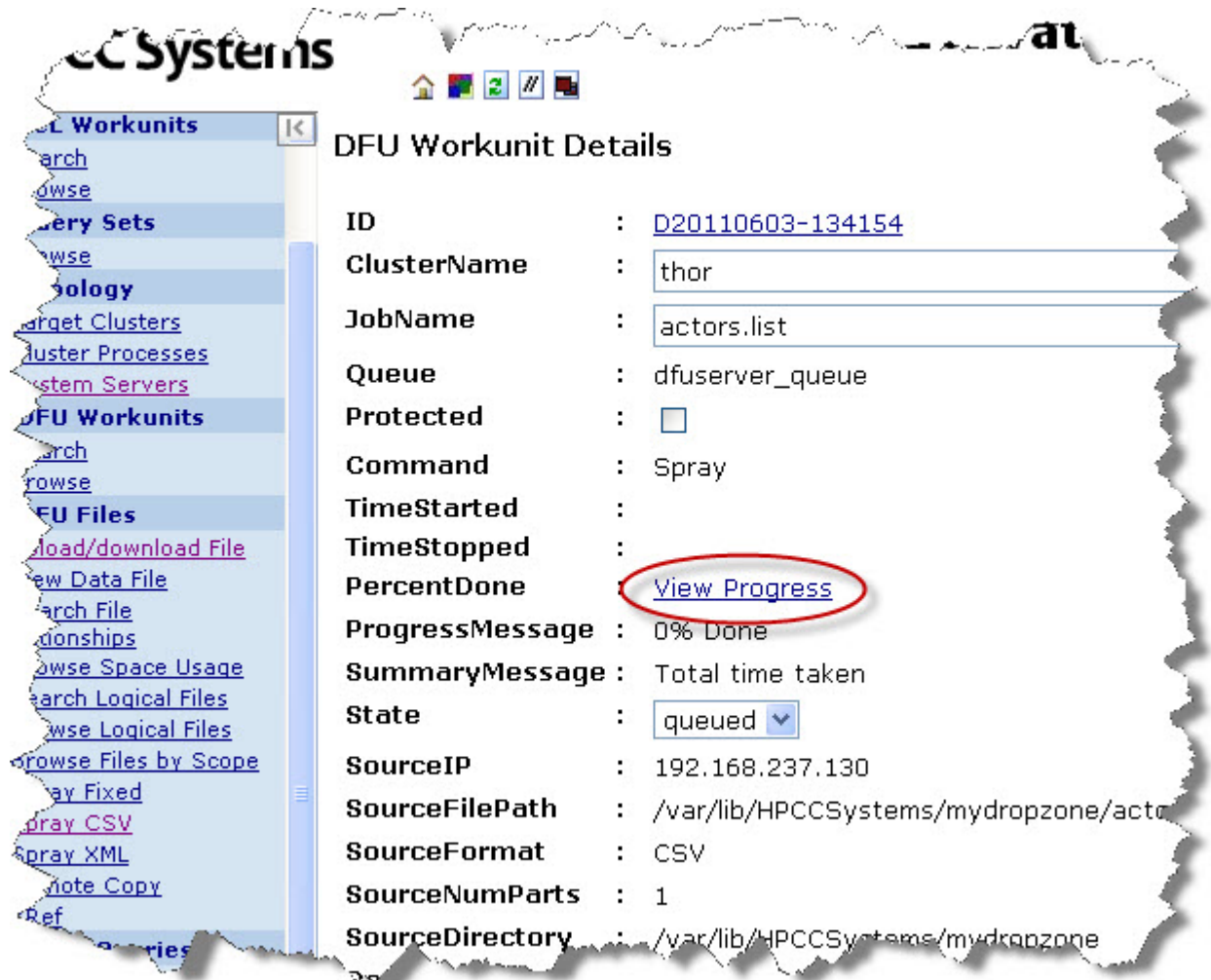
The screenshot shows the EclWatch web interface for configuring a 'Spray CSV' operation. The interface is titled 'HPCC Systems' and 'EclWatch'. A sidebar on the left contains navigation links such as 'ECL Workunits', 'Search', 'Browse', 'Query Sets', 'Topology', 'Target Clusters', 'Cluster Processes', 'System Servers', 'DFU Workunits', 'DFU Files', 'Roxie Queries', and 'Resources'. The main content area is titled 'Spray CSV' and contains the following configuration fields:

- Source:** Machine/dropzone: localhost/mydropzone (dropdown), IP Address: 192.168.237.130, Local Path: /var/lib/HPCCSystems/mydropzone/actors.list (with a 'Choose File' button), Network Path: //192.168.237.130/var/lib/HPCCSystems/mydropzone, Format: ASCII (dropdown), Max Record Length: 8192, Separator: \, Line Terminator: \n,\r\n, Quote: '.
- Destination:** Group: mythor (dropdown), Label: ~thor::in::IMDB::actors.list, Mask: ~thor::in::IMDB::actors.list,_\${P\$}_of_\${N\$}, Prefix: (empty).
- Options:** Overwrite: , Replicate: , No Split: , Compress: .

A red circle highlights the 'Submit' button at the bottom of the form.

- Press the Submit button

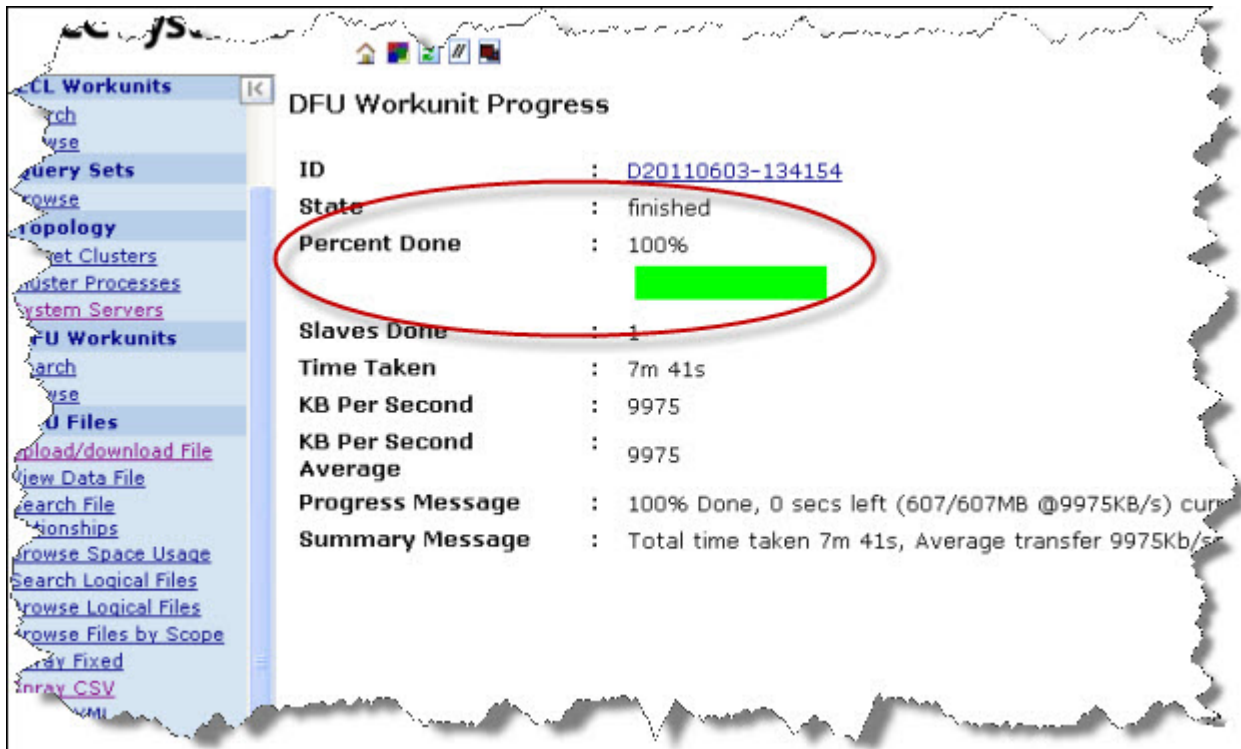
Figure 5. View Progress



- Click the View Progress link

- The Workunit progress page displays.

Figure 6. Workunit Progress

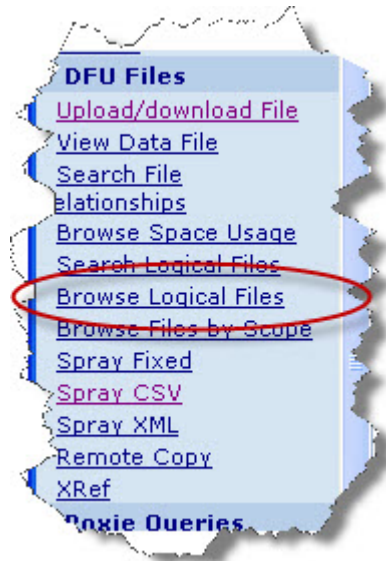


- Repeat the spray steps for the actresses file. (Just replace *actor.list* with the *actresses.list* in both the Local Path, Logical File name, and Label field.)

Once the **two** sprays are complete, we can query the logical files on the HPCC to see the files we sprayed.

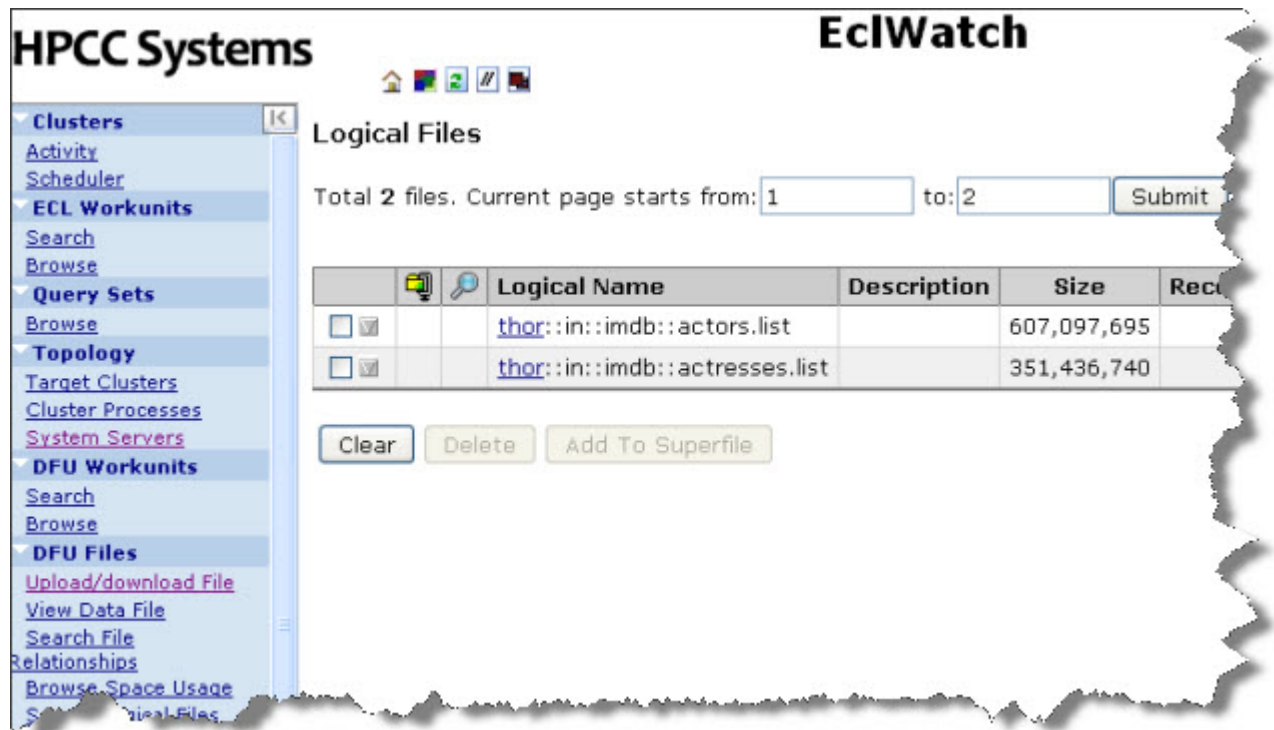
- Click on the **Browse Logical Files** hyperlink in ECL Watch

Figure 7. Browse Logical Files



The files display in the Logical Files list:

Figure 8. Display Logical Files



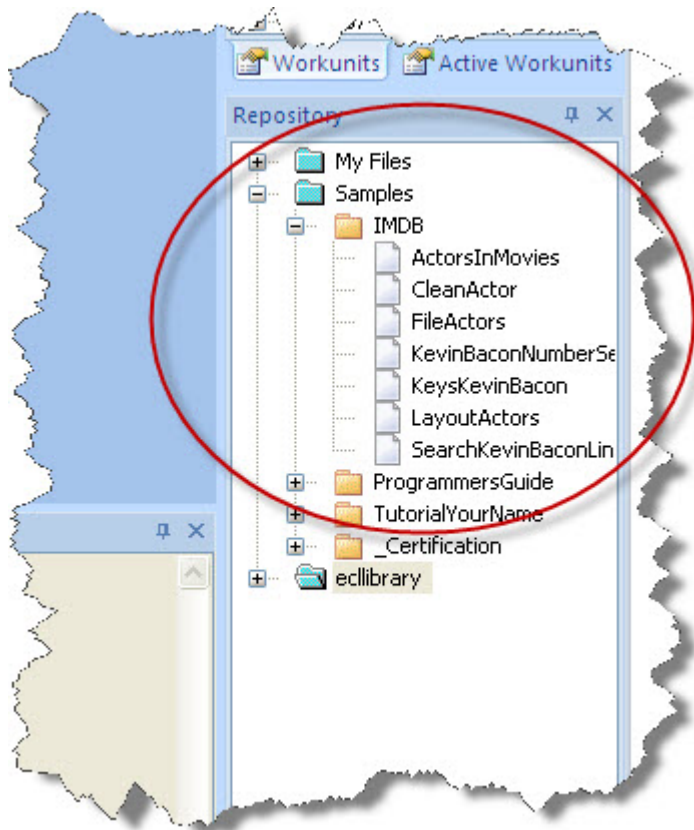
Working With the Data

In this portion of the example, we will write ECL code to make sure we can read the sprayed data file .We will define and execute simple queries on it so we can evaluate it and determine any necessary pre-processing.

- Start the ECL IDE (Start >> All Programs >> HPCC Systems >> ECL IDE)
- Log in to your environment.
- Expand the **Samples** ECL folder in the Repository toolbox.
- Expand the **IMDB** folder inside.

All the ECL files needed to complete this tutorial are located in the IMDB folder.

Figure 9. IMDB ECL Files



- Open the CleanActor ECL file and examine the code.

This code reads and processes the raw text file. The comments below describe the processing:

```
IMPORT Std;

EXPORT STRING CleanActor(STRING infld) := FUNCTION
  //this can be refined later
  s1 := Std.Str.FindReplace(infld, '\'', ''); // replace apostrophe
  s2 := Std.Str.FindReplace(s1, '\t', ''); //replace tabs
  s3 := Std.Str.FindReplace(s2, '----', ''); // replace multiple ----
```

```
return TRIM(s3, LEFT, RIGHT);  
END;
```

Examine the Data

In this section, we will look at the data and determine if there is any pre-processing we want to perform. This is the step in the development process where we convert the raw data into a form we can actually use.

- Open a new Builder window (CTRL+N) and write the following code:

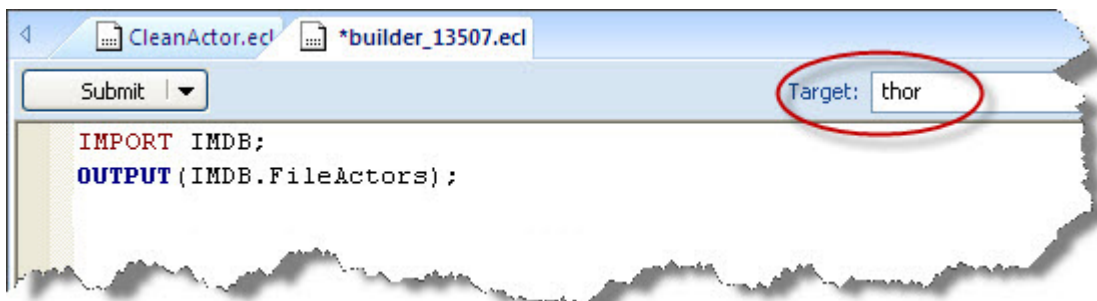
```
IMPORT IMDB;  
OUTPUT(IMDB.FileActors);
```

- Press the syntax check button on the main toolbar (or press F7).

It is always a good idea to check syntax before submitting.

- Make sure the selected cluster is your *thor* cluster, then press the **Submit** button.

Figure 10. Submit to Thor

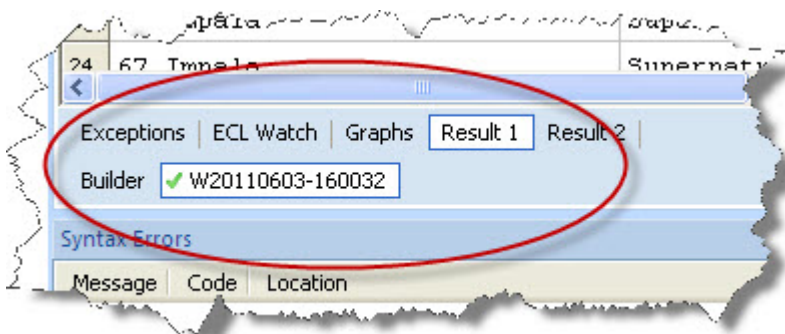


- When the Workunit completes it displays a green checkmark. ✓

Note: Depending on the size of your cluster and the speed of your server(s), this process could take several minutes. If you are running this on a virtual machine, it could take as long as 45 minutes to complete.

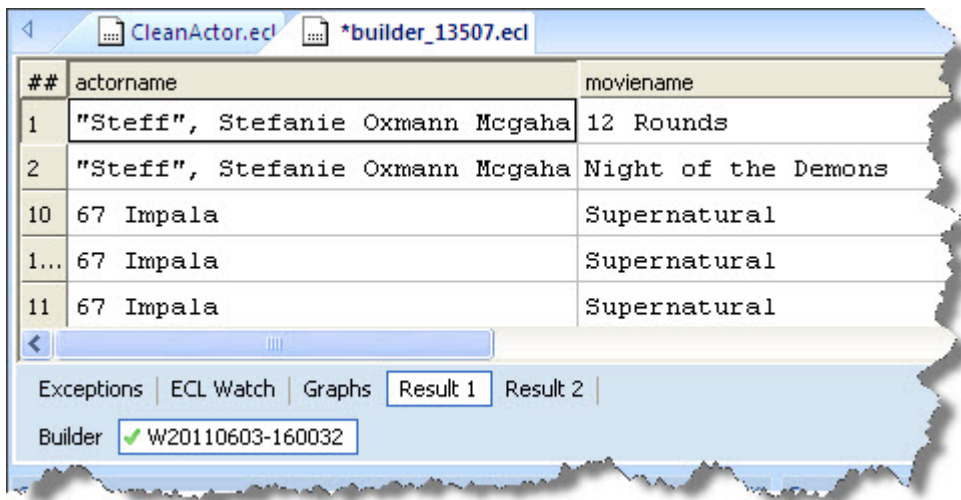
- Select the Workunit tab (the one with the number and the checkmark next to it) and select the **Result 1** tab.

Figure 11. Select Workunit



- Scroll down to see more records.

Figure 12. See more records



- Close the Builder Window.

Processing the Data : Extract, Transform, and Load

In this section, we will write code to transform the original actor data as follows:

- From the raw actors data, we will do an ETL operation (Extract, Transform, Load) to build an **actor_movie** relation set.
- We will also construct a Kevin Bacon degrees of separation lookup set. This is the structure we will query to answer the question:

How many degrees of separation exist between Actor X and Kevin Bacon?

For example: Using Jon Lovitz as the actor, we want information as follows:

Jon Lovitz ((was in) Movie X ((with) Actor2 ((who was in) Movie Y ((with) Kevin Bacon

We will then write this new file to our Thor cluster so it can be used in parameterized queries.

- In the ECL IDE , go to the Repository panel and expand the IMDB folder.
- Open the ECL File ActorsInMovies.

The code in this ECL file looks like this:

```
/* *****
## Copyright 2011 HPCC Systems. All rights reserved.
***** */
/**
```

Six Degrees of Kevin Bacon: ECL Programming Example

Working with Data

```
* Produce a slimmed down version of the IMDB actor AND actress files to
* permit more efficient join operations.
* Filter out the movie records we do not want in building our KBacon Number sets.
*
*/

IMPORT $ AS IMDB;
IMPORT Std;

// Filter out TV movies, Videos AND some documentary type collections
ds_IMDB := IMDB.FileActors(actorname!='' AND moviename != '' AND
                          Std.Str.Find(moviename,'Boffo',1) = 0 AND
                          Std.Str.Find(moviename,'Slasher Film',1) = 0 AND
                          movie_type != 'Video' AND isTVseries = 'N' AND
                          movie_type != 'For TV');

//Slim the records down to bare essentials for searching AND joining
slim_IMDB_rec := RECORD
  STRING50 actor;
  STRING150 movie;
END;

slim_IMDB_rec slim_it(ds_IMDB L):= TRANSFORM
  SELF.actor := Std.Str.FindReplace(L.actorname,'(I)','');
  SELF.movie := L.moviename;;
END;

IMDB_names := PROJECT(ds_IMDB, slim_it(LEFT));

export ActorsInMovies := IMDB_Names : persist('~temp::IMDB::ActorsInMovies');
```

This defines a relational data set:-- actor:movie. We will use this definition later.

Getting Useful Information from Data

Links and Degrees of Separation

Now that we have our data in a useful format, have a relation defined, and the file is in place, we can write code to use the new data file.

We want to know how many actors are a distance N from Kevin Bacon. To accomplish this, we will construct sets of Kevin Bacon's costars that are KBacon number apart.

- Open the KevinBaconNumberSets ECL file.

This ECL code counts the number of actors with "bacon numbers" starting from 1 thru 7, that is up to 7 Levels of separation. We will use this later to do searches by building an index.

```
/* *****  
ATTRIBUTE PURPOSE:  
Produce a series of sets for Actors and Movies that are : distance-0  
away (KBacon's Direct movies ), distance-2 Away KBacon's Costars Movies ,  
distance-3 away - Movies of Costars of Costars etc all the way upto level 7  
  
The nested attributes below are shown here together for the benefit of the reader.  
  
Notes on variable naming convention used for costars and movies  
KBMovies          : Movies Kevin Bacon Worked in (distance 0)  
KBCoStars         : Stars who worked in KBMovies (distance 1)  
KBCoStarMovies    : Movies worked in by KBCoStars  
                   except KBMovies (distance 1)  
KBCo2Stars        : Stars(Actors) who worked in KBCoStarMovies (distance 2)  
KBCo2StarMovies   : Movies worked in by KBCo2Stars  
                   except KBCoStarMovies (distance 2)  
KBCo3Stars        : Stars(Actors) who worked in KBCo2StarMovies (distance 3)  
KBCo3StarMovies   : Movies worked in by KBCo3Stars  
                   except KBCo2StarMovies (distance 3)  
  
etc..  
***** */  
  
IMPORT Std;  
IMPORT IMDB;  
  
EXPORT KevinBaconNumberSets := MODULE  
  // Constructing a proper name match function is an art within itself  
  // For simplicity we will define a name as matching if both first and last name  
  //are found within the string  
  
  NameMatch(string full_name, string fname,string lname) :=  
    Std.Str.Find(full_name,fname,1) > 0 AND  
    Std.Str.Find(full_name,lname,1) > 0;  
  
  //----- Get KBacon Movies  
  AllKBEntries := IMDB.actorsInMovies(NameMatch(actor,'Kevin','Bacon'));  
  EXPORT KBMovies := DEDUP(AllKBEntries, movie, ALL); // Each movie should ONLY occur once  
  
  //----- Get KBacon CoStars  
  CoStars := IMDB.actorsInMovies(Movie IN SET(KBMovies,Movie));  
  EXPORT KBCoStars := DEDUP( CoStars(actor<>'Kevin Bacon'), actor, ALL);  
  
  //----- Get KBacon Costars' Movies  
  // CSM = First find all of the movies that a KBCoStar has been in  
  
  CSM := DEDUP(JOIN(IMDB.actorsInMovies,KBCoStars, LEFT.actor=RIGHT.actor,
```

Six Degrees of Kevin Bacon: ECL Programming Example

Working with Data

```
        TRANSFORM(LEFT), LOOKUP),
        movie,ALL);

// Now we need to remove all of those that KB was in himself
// We can use a set; KB has not been in (quite!) that many movies

EXPORT KBCoStarMovies := CSM(movie NOT IN SET(KBMovies,movie));

//----- Bacon # 2 Actors
// To be a Co2Star of Kevin Bacon you must have appeared in a movie with a
//CoStar of Kevin Bacon
// This corresponds to having a Bacon number of 2
// We are now getting towards the expensive part of the process
KBCo2S := DEDUP(JOIN(IMDB.actorsInMovies, KBCoStarMovies, LEFT.movie=RIGHT.movie,
        TRANSFORM(LEFT), LOOKUP),
        actor, ALL);

// KCCo2S = ALL Actors appearing in Movies of KBacon's CoActors
// The above is all the people in the movies; but some will have been co-stars of KB
//directly - these must be removed
// The LEFT ONLY join removes items in one list from another

EXPORT KBCo2Stars := JOIN(KBCo2S, KBCoStars, LEFT.actor=RIGHT.actor,
        TRANSFORM(LEFT), LEFT ONLY);

//----- bacon # 2 Movies
// Co2SM = what movies have all the Co2Stars been in?
Co2SM := DEDUP(JOIN(IMDB.actorsInMovies, KBCo2Stars, LEFT.actor=RIGHT.actor,
        TRANSFORM(LEFT), LOOKUP),
        movie, ALL);
// Co2SM = ALL Movies KBCo2Stars have been in
// Of course some of these movies will have CoStars in too and thus will already have
//been listed. Note this list will not contain any Kevin Bacon movies OR the movie would
//have been reached earlier!

Export KBCo2StarMovies := JOIN(Co2SM, KBCoStarMovies, LEFT.movie=RIGHT.movie,
        TRANSFORM(LEFT),LEFT ONLY);

//----- bacon #3 Actors
// Find people with a Bacon number of 3
// This code is very similar to KBCo2Stars; one might be tempted to common up into a
// function or macro. However it is worth looking at the attribute counts first; we may be
// down to a small enough set that we can start using in-memory functions (e.g.,SET) again.

KBCo3S := DEDUP(JOIN(IMDB.actorsInMovies, KBCo2StarMovies, LEFT.movie=RIGHT.movie,
        TRANSFORM(LEFT), LOOKUP),
        actor, ALL);

// KBCo3S = ALL CoStars in KBCo2Star Movies
// The above is all the people in the movies; but some will have been co2stars of KB
// directly - these must be removed. The LEFT ONLY join removes items in one list from
// another. There should not be any direct CoStars in this list (or the movie would have
// been a CoStarMovie not a CoCoStarMovie)

EXPORT KBCo3Stars := JOIN(KBCo3S, KBCo2Stars, LEFT.actor=RIGHT.actor,
        TRANSFORM(LEFT),LEFT ONLY);

//----- bacon #3 Movies
// So what movies have all the KBCo3Stars been in?

Co3SM := DEDUP(JOIN(IMDB.actorsInMovies, KBCo3Stars, LEFT.actor=RIGHT.actor,
        TRANSFORM(LEFT), LOOKUP),
        movie, ALL);

// Co3SM = ALL Movies KBCo3Stars have been in
```

Six Degrees of Kevin Bacon: ECL Programming Example

Working with Data

```
// Of course some of these movies will have KBCo2Stars in too and thus will already have
// been listed. Note We ONLY have to remove one level back from the list; previous levels
// cannot be reached by definition

EXPORT KBCo3StarMovies := JOIN(Co3SM, KBCo2StarMovies, LEFT.movie=RIGHT.movie,
                              TRANSFORM(LEFT),LEFT ONLY);

//-----bacon #4 Actors
KBCo4S := DEDUP(JOIN(IMDB.actorsInMovies, KBCo3StarMovies, LEFT.movie=RIGHT.movie,
                   TRANSFORM(LEFT), LOOKUP),
              actor, ALL);

EXPORT KBCo4Stars := JOIN(KBCo4S, KBCo3Stars, LEFT.actor=RIGHT.actor,
                          TRANSFORM(LEFT),LEFT ONLY);

//----- bacon #4 Movies
// So what movies have all the Co4Stars been in?

Co4SM := DEDUP(JOIN(IMDB.actorsInMovies, KBCo4Stars, LEFT.actor=RIGHT.actor,
                   TRANSFORM(LEFT), LOOKUP),
              movie, ALL);

// Co4SM = ALL Movies KBCo4Stars have been in
// Of course some of these movies will have Co3Stars in too and thus will already have
// been listed. Note We ONLY have to remove one level back from the list; previous levels
// cannot be reached by definition

EXPORT KBCo4StarMovies := JOIN(Co4SM, KBCo3StarMovies, LEFT.movie=RIGHT.movie,
                              TRANSFORM(LEFT),LEFT ONLY);

//----- bacon #5 Stars
KBCo5S := DEDUP(JOIN(IMDB.actorsInMovies, KBCo4StarMovies, LEFT.movie=RIGHT.movie,
                   TRANSFORM(LEFT), LOOKUP),
              actor, ALL);

EXPORT KBCo5Stars := JOIN(KBCo5S, KBCo4Stars, LEFT.actor=RIGHT.actor,
                          TRANSFORM(LEFT),LEFT ONLY);

//----- bacon #5 Movies
Co5SM := DEDUP(JOIN(IMDB.actorsInMovies, KBCo5Stars, LEFT.actor=RIGHT.actor,
                   TRANSFORM(LEFT), LOOKUP),
              movie,ALL);

EXPORT KBCo5StarMovies := JOIN(Co5SM, KBCo4StarMovies, LEFT.movie=RIGHT.movie,
                              TRANSFORM(LEFT),LEFT ONLY);

//----- bacon #6 Stars
// Find people with a Bacon number of 6
// KBCo5 is getting small again - can move back down to the SET?

KBCo6S := DEDUP(IMDB.actorsInMovies(movie IN SET(KBCo5StarMovies, movie)),
              actor, ALL);

EXPORT KBCo6Stars := JOIN(KBCo6S, KBCo5Stars, LEFT.actor=RIGHT.actor,
                          TRANSFORM(LEFT),LEFT ONLY);

//----- bacon #6 Movies
Co6SM := DEDUP(IMDB.actorsInMovies(actor IN SET(KBCo6Stars, actor)), movie, ALL);

EXPORT KBCo6StarMovies := Co6SM(movie NOT IN SET(KBCo5StarMovies, movie));

//----- bacon #7 Movies
// Find people with a Bacon number of 7
KBCo7S := DEDUP(IMDB.actorsInMovies(movie IN SET(KBCo6StarMovies,movie)), actor, ALL);
EXPORT KBCo7Stars := KBCo7S(actor NOT IN SET(KBCo6Stars, actor));
```

Six Degrees of Kevin Bacon: ECL Programming Example

Working with Data

```
//----- We just have to count them all !! (How many holes in Albert Hall?)
EXPORT doCounts := PARALLEL(
  OUTPUT(COUNT(KBMovies), NAMED('KBMovies')),
  OUTPUT(COUNT(KBCoStars), NAMED('KBCoStars')),
  OUTPUT(COUNT(KBCoStarMovies), NAMED('KBCoStarMovies')),
  OUTPUT(COUNT(KBCo2Stars), NAMED('KBCo2Stars')),
  OUTPUT(COUNT(KBCo2StarMovies), NAMED('KBCo2StarMovies')),
  OUTPUT(COUNT(KBCo3Stars), NAMED('KBCo3Stars')),
  OUTPUT(COUNT(KBCo3StarMovies), NAMED('KBCo3StarMovies')),
  OUTPUT(COUNT(KBCo4Stars), NAMED('KBCo4Stars')),
  OUTPUT(COUNT(KBCo4StarMovies), NAMED('KBCo4StarMovies')),
  OUTPUT(COUNT(KBCo5Stars), NAMED('KBCo5Stars')),
  OUTPUT(COUNT(KBCo5StarMovies), NAMED('KBCo5StarMovies')),
  OUTPUT(COUNT(KBCo6Stars), NAMED('KBCo6Stars')),
  OUTPUT(COUNT(KBCo6StarMovies), NAMED('KBCo6StarMovies')),
  OUTPUT(COUNT(KBCo7Stars), NAMED('KBCo7Stars')),
  OUTPUT(KBCo7Stars)
);
END;
```

- Open a new Builder Window and type:

```
IMPORT IMDB;
IMDB.KevinBaconNumberSets.doCounts;
```

- Check the syntax then press the **Submit** button.

Note: Depending on the size of your cluster and the speed of your server(s), this process could take several minutes. If you are running this on a virtual machine, it could take as long as an hour to complete.

- When the process completes, each row shown below becomes it's own result tab. You will get a sample of the output as follows:

Note: The data files for this tutorial change frequently, your results may vary from those shown in this document.

KB Movies	71
KB Co Stars	3520
KB Co Star Movies	33504
KB Co 2 Stars	430145
KB Co 2 Star Movies	251867
KB Co 3 Stars	896009
KB Co 3 Star Movies	51650
KB Co 4 Stars	102729
KB Co 4 Star Movies	2634
KB Co 5 Stars	6080
KB Co 5 Star Movies	190
KB Co 6 Stars	450
KB Co 6 Star Movies	14
KB Co 7 Stars	22

Next Steps

Now that you have successfully processed the data and established links, what's next?

Two more ECL files are included in the IMDB folder that you can use in conjunction with the examples you have already worked through in this tutorial:

- `KeysKevinBacon` -- Builds an index of actors/actresses and the movies they have starred in.

You must build this index before you can run queries to find the degree of separation between Kevin Bacon and an actor of your choice.

To build the index, open a builder window and type the following code:

```
IMPORT IMDB;  
IMDB.KeysKevinBacon.BuildAll;
```

Press the **Submit** button to run the ECL code and build the index.

`SearchKevinBaconLinks` -- Searches the index you built to give you the degree of separation between an actor and Kevin Bacon.

For example, to find the degree of separation between Kevin Bacon and Andi Everingham, open a builder window and type the following code:

```
IMPORT IMDB;  
IMDB.SearchKevinBaconLinks('Everingham, Andi');
```

Make sure the selected cluster is your *hThor* cluster, then press the **Submit** button to run the query.

When it has completed, click on the Workunit ID tab.

Two results are shown.

Result1 shows the degree of separation between the actor and Kevin Bacon.

Interpret the results as follows:

Actor is at level 1 - The actor you chose and Kevin Bacon starred in a movie together.

Actor is at level 2 - The actor you chose starred in a movie with an actor who starred in a movie with Kevin Bacon.

The higher the level, the greater the degree of separation between the actor you chose and Kevin Bacon.

In this example, the actor is at level 6, indicating that there are 6 degrees of separation between Andi Everingham and Kevin Bacon.

Result2 shows the level (degree of separation), the name of the actor and the movie they starred in.

Each line shows an actor and the movie they starred in which links them to each other and eventually to Kevin Bacon.

Have fun finding the degrees of separation between any actor and Kevin Bacon.

Remember to build the index first.