Bob Foreman/Hugo Watanuki
Senior Software Engineers
LexisNexis RISK Solutions

# ECL Alive! - a Deep-Dive into Definitive HPCC Systems

Hour 1 – The ETL Way with ECL: Data Ingest, Profiling, Hygiene, Standardization, and Exporting

## Introduction

This workshop is based on the new book by Richard Taylor:

**Definitive HPCC Systems**

**Volume II: Data Transformation and Delivery**

**Hour 1: Chapters 2 and 3**

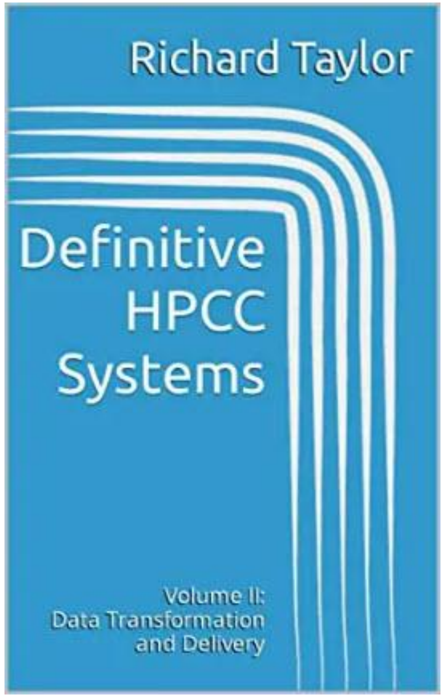
Hour 2: Chapter 4

Hour 3: Chapter 5 (ECL Cookbook)

Volume I and II is currently available on Amazon!

https://www.amazon.com/Definitive-HPCC-Systems-Overview-Platform-ebook/dp/B087Y1FMDH

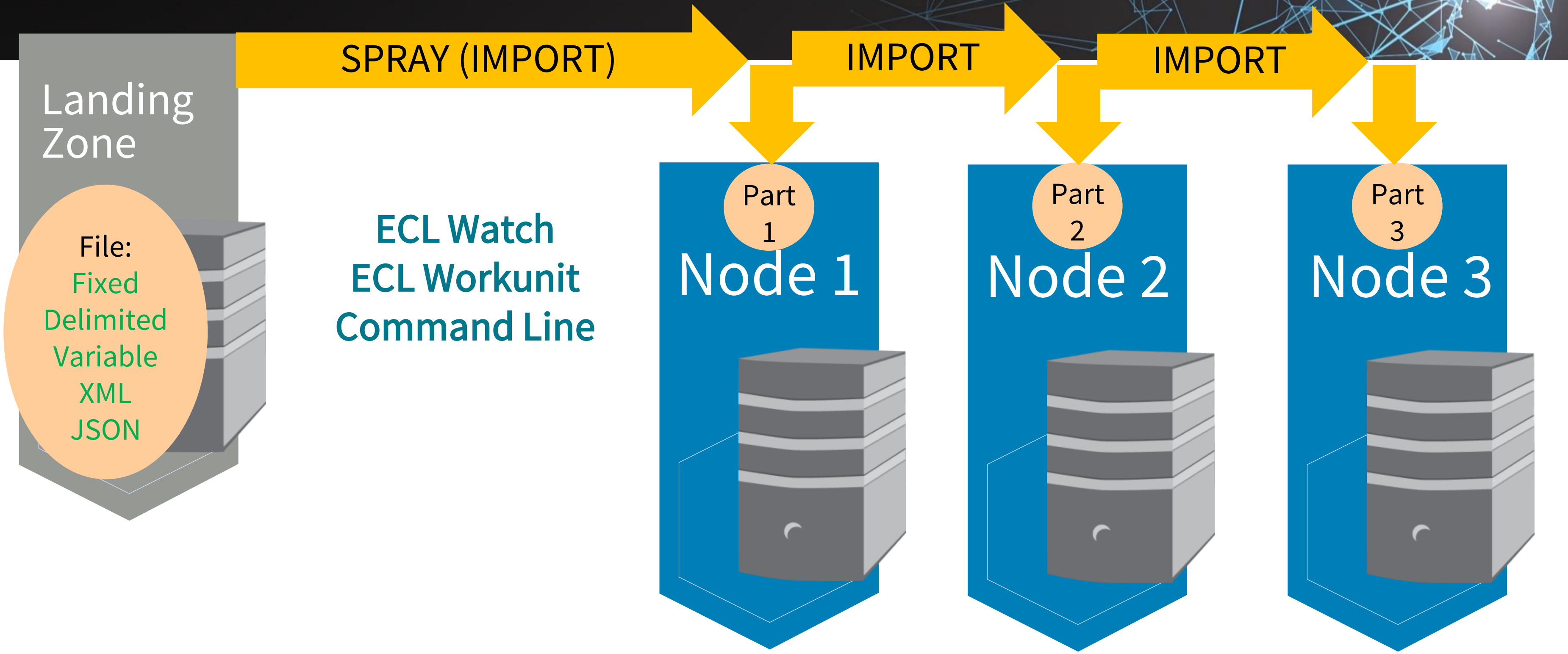
https://www.amazon.com/Definitive-HPCC-Systems-Transformation-Delivery-ebook/dp/B0BCMZCXDD



## Data Ingest

- The term "Spray" (or "Import" in ECL Watch 9 and greater) is used to describe the process of copying data from external files into an HPCC Systems cluster. This term is appropriate because the HPCC Systems environment always works with distributed data files.
- So, to get a single physical data file into the cluster, the spray/import process divides the data into *n* chunks (where *n* is the number of nodes in the cluster) and puts one file part (approximately evenly sized) on each node.
- Before any file can be imported, it must first be in a location that is accessible to the cluster. That location is commonly referred to as a Landing Zone or Drop Zone – another middleware component described in the first volume of this book series.

# SPRAY(Import) Operation

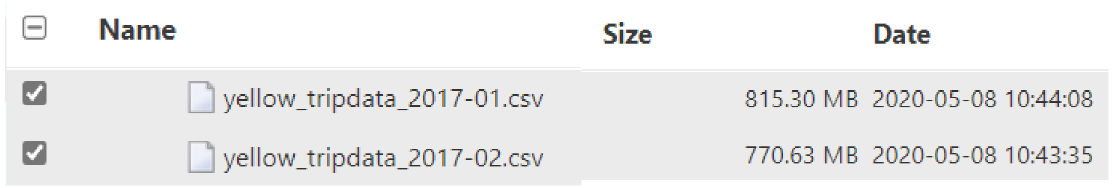


ECL Watch
ECL Workunit
Command Line

Referenced in ECL as a single logical file…

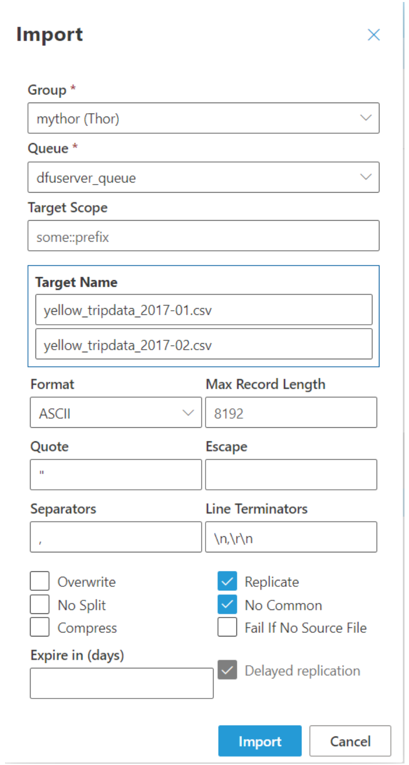
## Workshop Data

- We'll begin with getting some publicly available data to work with. The **New York City Taxi & Limousine Commission** makes its trip data freely available to everyone here: https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page
- Before any file can be sprayed, it must first be in a location that is accessible to the cluster. That location is commonly referred to as a Landing Zone or Drop Zone – another middleware component described in the first volume of this book series.
- After locating the files that we need to spray on our workshop cluster Landing Zone, we will use a Delimited Spray to move these files to the cluster.



| | Name | Size | Date |
|---|---|---|---|
| ✓ | yellow_tripdata_2017-01.csv | 815.30 MB | 2020-05-08 10:44:08 |
| ✓ | yellow_tripdata_2017-02.csv | 770.63 MB | 2020-05-08 10:43:35 |

# Delimited Spray Options



**Import** ×

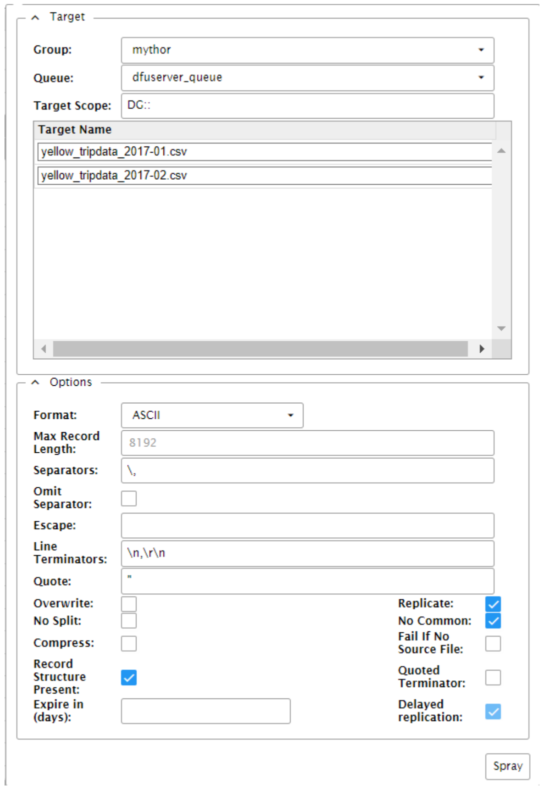
Group *: mythor (Thor)

Queue *: dfuserver_queue

Target Scope: some::prefix

**Target Name**
- yellow_tripdata_2017-01.csv
- yellow_tripdata_2017-02.csv

| | |
|---|---|
| Format: ASCII | Max Record Length: 8192 |
| Quote: " | Escape: |
| Separators: , | Line Terminators: \n,\r\n |
| ☐ Overwrite | ☑ Replicate |
| ☐ No Split | ☑ No Common |
| ☐ Compress | ☐ Fail If No Source File |
| Expire in (days): | ☑ Delayed replication |



**Target**

Group: mythor

Queue: dfuserver_queue

Target Scope: DC::

**Target Name**
- yellow_tripdata_2017-01.csv
- yellow_tripdata_2017-02.csv

**Options**

Format: ASCII

Max Record Length: 8192

Separators: \,

Omit Separator: ☐

Escape:

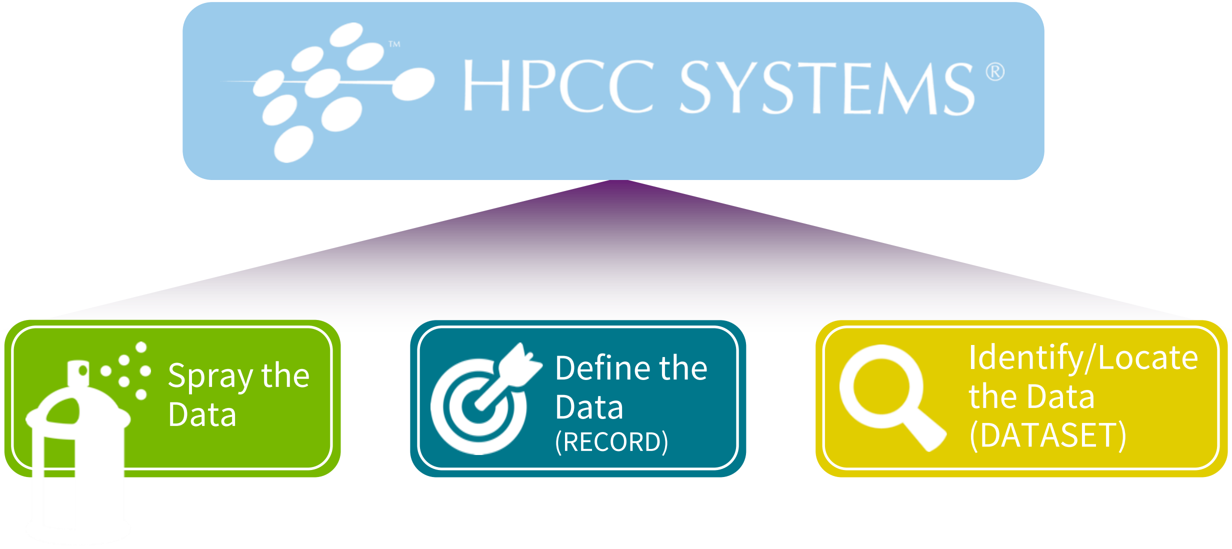
Line Terminators: \n,\r\n

Quote: "

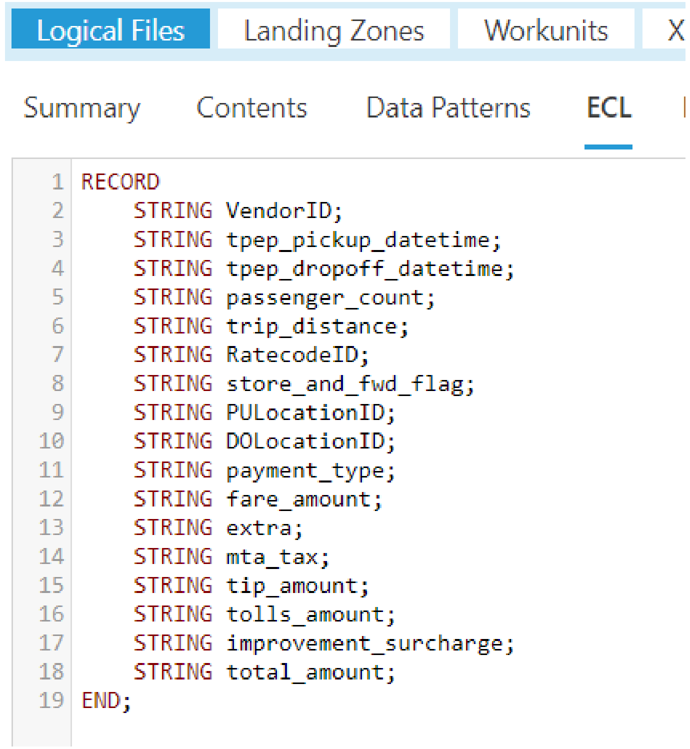
| | |
|---|---|
| Overwrite: ☐ | Replicate: ☑ |
| No Split: ☐ | No Common: ☑ |
| Compress: ☐ | Fail If No Source File: ☐ |
| Record Structure Present: ☑ | Quoted Terminator: ☐ |
| Expire in (days): | Delayed replication: ☑ |

# Three ECL Data Rules

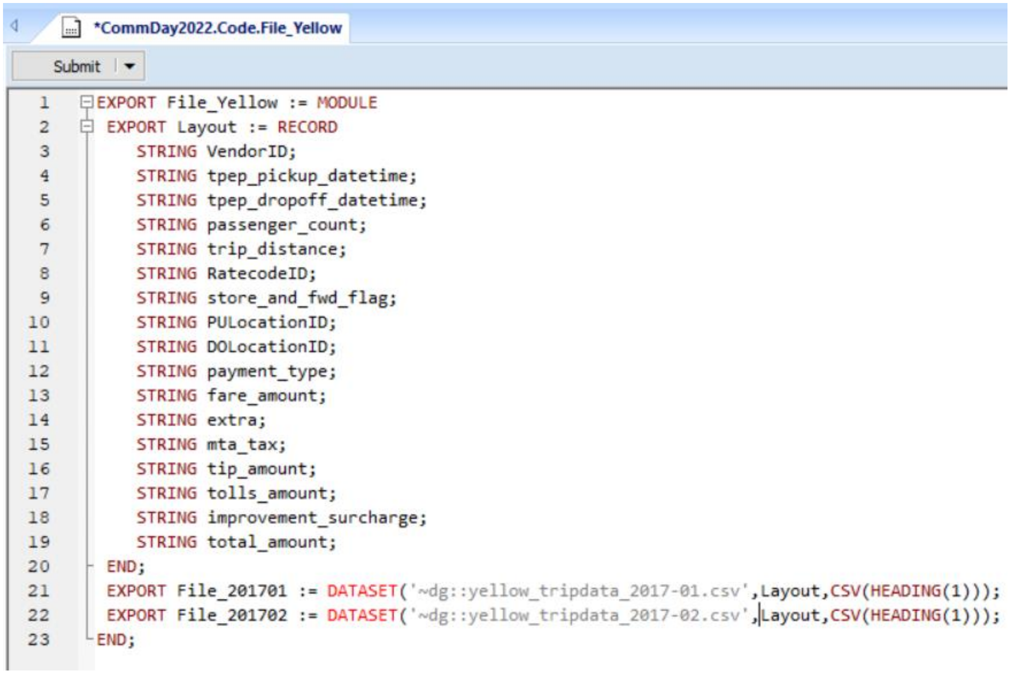
Before you begin to work on any data in the HPCC cluster, you must always do three things:



# Get the RECORD, create the DATASET:



```
RECORD
    STRING VendorID;
    STRING tpep_pickup_datetime;
    STRING tpep_dropoff_datetime;
    STRING passenger_count;
    STRING trip_distance;
    STRING RatecodeID;
    STRING store_and_fwd_flag;
    STRING PULocationID;
    STRING DOLocationID;
    STRING payment_type;
    STRING fare_amount;
    STRING extra;
    STRING mta_tax;
    STRING tip_amount;
    STRING tolls_amount;
    STRING improvement_surcharge;
    STRING total_amount;
END;
```

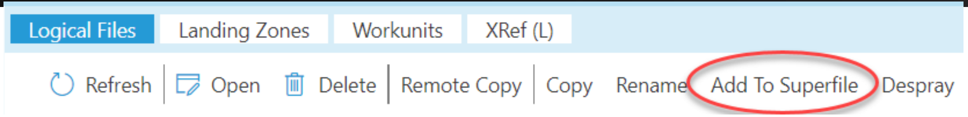


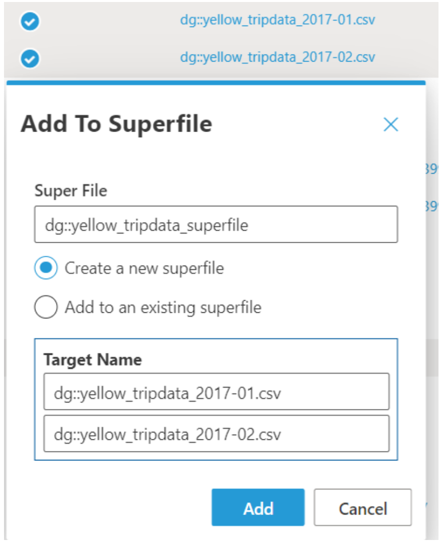
```
EXPORT File_Yellow := MODULE
 EXPORT Layout := RECORD
    STRING VendorID;
    STRING tpep_pickup_datetime;
    STRING tpep_dropoff_datetime;
    STRING passenger_count;
    STRING trip_distance;
    STRING RatecodeID;
    STRING store_and_fwd_flag;
    STRING PULocationID;
    STRING DOLocationID;
    STRING payment_type;
    STRING fare_amount;
    STRING extra;
    STRING mta_tax;
    STRING tip_amount;
    STRING tolls_amount;
    STRING improvement_surcharge;
    STRING total_amount;
 END;
 EXPORT File_201701 := DATASET('~dg::yellow_tripdata_2017-01.csv',Layout,CSV(HEADING(1)));
 EXPORT File_201702 := DATASET('~dg::yellow_tripdata_2017-02.csv',Layout,CSV(HEADING(1)));
END;
```

## Combining Common Data

- Given that we have two separate logical files that both have the same structure, and that we're working with a Big Data platform, then it would be advantageous to be able to work with both as if they were a single logical file instead of two. You could simply use the ECL record set append operators (+ and &) to combine them, but the better way is to define them as sub-files in a single SuperFile.
- The SuperFiles section in the *Standard Library Reference* documents all the functions that are available for SuperFile maintenance. Also, the **Working With SuperFiles** section of the *Programmer's Guide* contains several articles that describe how to use those functions for standard Superfile maintenance processes.

# Using the ECL Watch to Create a Superfile:



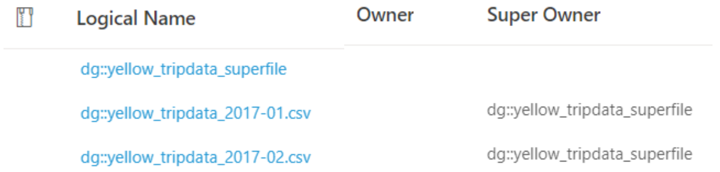

dg::yellow_tripdata_2017-01.csv
dg::yellow_tripdata_2017-02.csv

**Add To Superfile**

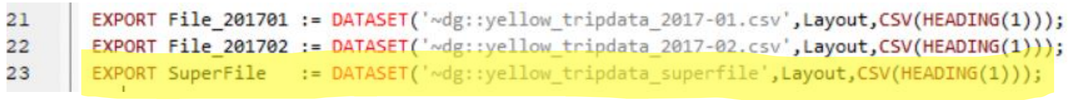
Super File: dg::yellow_tripdata_superfile

[✓ Create a new superfile] Add to an existing superfile

Target Name:
dg::yellow_tripdata_2017-01.csv
dg::yellow_tripdata_2017-02.csv



| | Logical Name | Owner | Super Owner |
|---|---|---|---|
| | dg::yellow_tripdata_superfile | | |
| | dg::yellow_tripdata_2017-01.csv | | dg::yellow_tripdata_superfile |
| | dg::yellow_tripdata_2017-02.csv | | dg::yellow_tripdata_superfile |



```
EXPORT File_201701 := DATASET('~dg::yellow_tripdata_2017-01.csv',Layout,CSV(HEADING(1)));
EXPORT File_201702 := DATASET('~dg::yellow_tripdata_2017-02.csv',Layout,CSV(HEADING(1)));
EXPORT SuperFile   := DATASET('~dg::yellow_tripdata_superfile',Layout,CSV(HEADING(1)));
```

## Data Profiling

- Now that we have data in the HPCC Systems environment and have defined it for use, the next step is to explore the data to discover what's what. Whether you're doing standard ETL processing, or any other data work, you need to completely understand the data you're working with for two primary reasons:

1. So you can craft the best possible data structures for your end result (product) database.
2. To make your data transformation processes (from raw data to final product) as efficient as possible.

So, the first step in any data ingest process, once the files are available on your cluster, is to Profile the data. This section discusses several possible ways to accomplish that task.

## Data Profiling Questions

1. Are there any non-numeric characters in the field (IOW, is it text data or just numbers)?
2. If it is text, what is the maximum text length?
3. If it is numeric, are the values integers or floating point?
4. If it is numeric, what is the range of values?
5. How many unique values are present?
6. What do the data patterns look like?
7. How skewed are the values, and how sparsely populated?

**BWR_Profile0.ecl**

## Profiling Every Field (Issue 1)

- So, because we started our profiling code with the first two re-definitions, you could just change the *Fld* definition's expression to name a different field and re-run the code. That would work, but that means the information would be in a separate workunit for each field, and each question's answer would show up in a separate result tab for the workunit -- so the display wouldn't be terribly "user-friendly" (especially since you are the "user" of this information).
- Let's solve the first issue: the fact that all the answers show up on separate result tabs. We can do this by writing a FUNCTION structure to contain all the answer code and produce all the results on a single tab.

**Profile.ecl**

## Profiling *Every* Field (Issue 2)

- The second issue of automating our profiling to process every field in our dataset is accomplished using Template Language.
- Template language is a meta-language used to generate ECL code.
- Unlike ECL, the Template Language has variables (referred to as "symbols") that must be explicitly declared (with some few exceptions) and can be re-assigned values. It is also procedural, meaning it does have looping constructs and requires programming logic more similar to other procedural languages than to ECL.
- We can make use of that feature to automate ECL code generation to produce our Profile function results as a single dataset from every field in our CSV file.

**BWR_Profile2.ecl**

## Profile Automation

- So, you could just change the previous code to run it on a different dataset. Or you can modify that code and wrap it in either a MACRO or a FUNCTIONMACRO structure. That would give you a tool that you can just call, passing it an argument naming the dataset to profile.
- Like the Template Language, both the MACRO and FUNCTIONMACRO are code generation tools.

**fnMAC_Profile.ecl**

Testing:

```
OUTPUT($.fnMAC_Profile($.File_Yellow.SuperFile),,'~File_Yellow::Profile::SuperFile_' +
                       (STRING8)Std.Date.Today(),NAMED('ProfileInfo'), OVERWRITE);
```

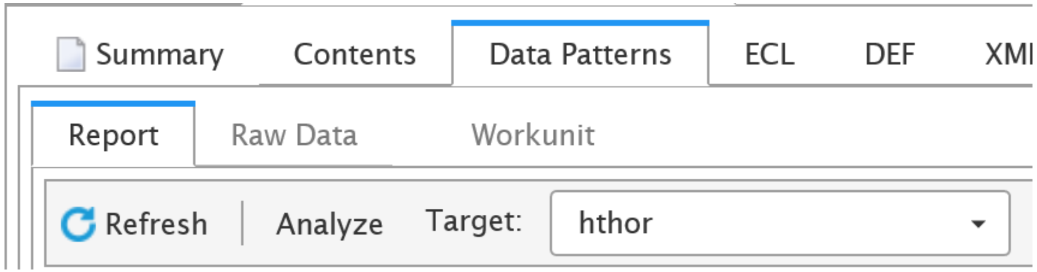
## Comparing Profiles

- If you periodically receive new files to add to the collection you already have, then it's a really good idea to re-run your profile code on the new file (alone) to see if there are any significant differences in the data that might require changes to your processing code or final product data format to handle. That's what we'll tackle now.
- Let’s run our Profiling macro on both input files, and compare:

**BWR_ProfileCompare.ecl**

# Built-In Data Profiling: Data Patterns

An extensive Data Profiling report is now built-in and available in the ECL Watch for all logical files. This report can be accessed via the Data Patterns tab:



There are three ways to use Data Patterns:

1. ECL Watch (via the Data Patterns tab)
2. Bundle (found on the HPCC Git Hub): https://github.com/hpcc-systems/DataPatterns.git
3. Standard Library Reference (STD.DataPatterns)

## Data Hygiene (Cleaning and Standardization)

- After analyzing and understanding your data, the Data Hygiene step (cleaning and standardization) is the next action you need to take in your data ingest process.
- The first part of that (at least, it is in LexisNexis Risk Solutions operations) is always to add your own globally unique identifier to each input record.
- Why? Because you always want to be able to get back to the original input record your final product data is derived from.
- The "global" context refers to the universe of your own input data, not the entire world. That makes the problem much more easily solvable -- you just need to ensure that your identifier values are unique across all of your input datasets for a single product file.

**BWR_BestRecord.ecl**

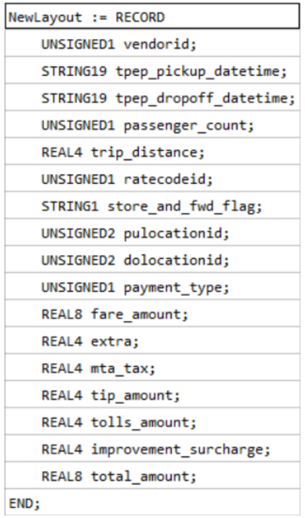
**fnMAC_GenUID.ecl**

## Data Standardization

- The CSV file format (the type of files that we sprayed) represents all data items as a string. That makes a human-readable file, but not a very efficient computer-readable data storage/operation mechanism. Therefore, the next thing we need to do is to decide what data types to use to contain each field's value so that our end product data is most efficient for both storage and usability.

We can start with the RECORD structure given to us by the DataPatterns.BestRecordStructure function:

**TaxiData.ecl**



```
NewLayout := RECORD
    UNSIGNED1 vendorid;
    STRING19 tpep_pickup_datetime;
    STRING19 tpep_dropoff_datetime;
    UNSIGNED1 passenger_count;
    REAL4 trip_distance;
    UNSIGNED1 ratecodeid;
    STRING1 store_and_fwd_flag;
    UNSIGNED2 pulocationid;
    UNSIGNED2 dolocationid;
    UNSIGNED1 payment_type;
    REAL8 fare_amount;
    REAL4 extra;
    REAL4 mta_tax;
    REAL4 tip_amount;
    REAL4 tolls_amount;
    REAL4 improvement_surcharge;
    REAL8 total_amount;
END;
```

## Data Standardization

- Now that we've defined exactly what storage format our data needs to be in going forward, we need to transform it from the input string data to the binary data types that we've decided upon.
- Data Hygiene is the step where you take the opportunity to clean up the data, getting rid of any “garbage” that you don’t want. It also gives you the opportunity to standardize your data (such as, if you get phone numbers that have been input with multiple formats you could standardize them all into a single format).

**BWR_CleanTaxiData.ecl**

HPCC SYSTEMS

## Data Exporting

- Once you have your data cleaned and standardized, you need to think about how to get that data to your end-users. There are several ways to do that:

1. Offload the data from your Thor to another platform (such as some kind of Business Intelligence software)
2. Use the WsSQL web service to make the data available to any SQL-based software
3. Create end-user queries and publish them to ROXIE

# End of Hour 1 Workshop:

**And Even <u>More</u> to Come!!**

**See you tomorrow for Hour 2!**

**Thanks for attending!**

✓Download it all at:
https://github.com/hpcc-systems/Community-Workshops

✓Contact us: robert.foreman@lexisnexisrisk.com
hugo.watanuki@lexisnexisrisk.com