



Bob Foreman

Senior Software Engineer
LexisNexis
Risk Solutions



Hugo Watanuki

Senior Technical Support Engineer
LexisNexis
Risk Solutions

Remote HPCC Systems/ECL Training

Data Visualization, NLP, and
ML made easy with
HPCC Systems ECL

ODSC@ WEST

DATA SCIENCE
VIRTUAL TRAINING CONFERENCE

OCT 27-30

Workshop Agenda:

- ✓ History of HPCC (High Performance Computing Cluster)
- ✓ Architecture
- ✓ THOR and ROXIE (Two data clusters – factory and delivery)
- ✓ The language behind both: Enterprise Control Language, or ECL
- ✓ ETL with ECL
- ✓ Built-in Data Visualization
- ✓ ECL and NLP
- ✓ Parsing and Text Vectors
- ✓ ML with ECL
- ✓ Learning Trees Tutorial
- ✓ Summary and Resources



HPCC and ECL Fundamentals

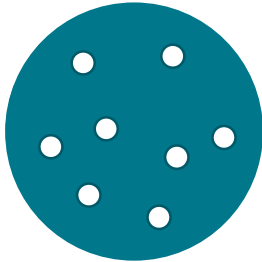
A Brief History of HPCC!

Why does HPCC Systems exist?

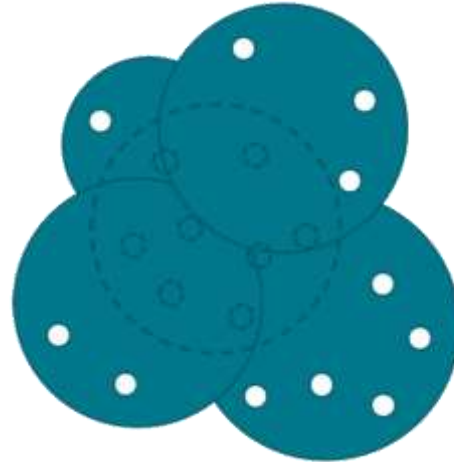
- ✓ It was NOT developed with the idea of selling the technology to anybody else!
- ✓ It was all created only to solve some of the data-handling problems that we encountered as we were developing our products.
- ✓ HPCC defined is a *distributed data parallel processing* platform.

The Data Centric Approach

A single source of data is insufficient to overcome inaccuracies



Our platform is built on the premise of absorbing data from many data sources and transforming them to actionable smart data



Scale from Small to Big

The stack can run on a single laptop or desktop.



Oracle's Virtual Box

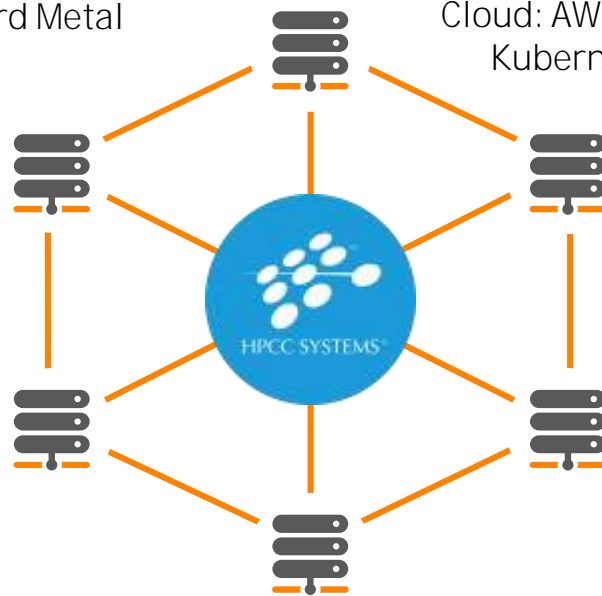


HPCC Virtual Machine

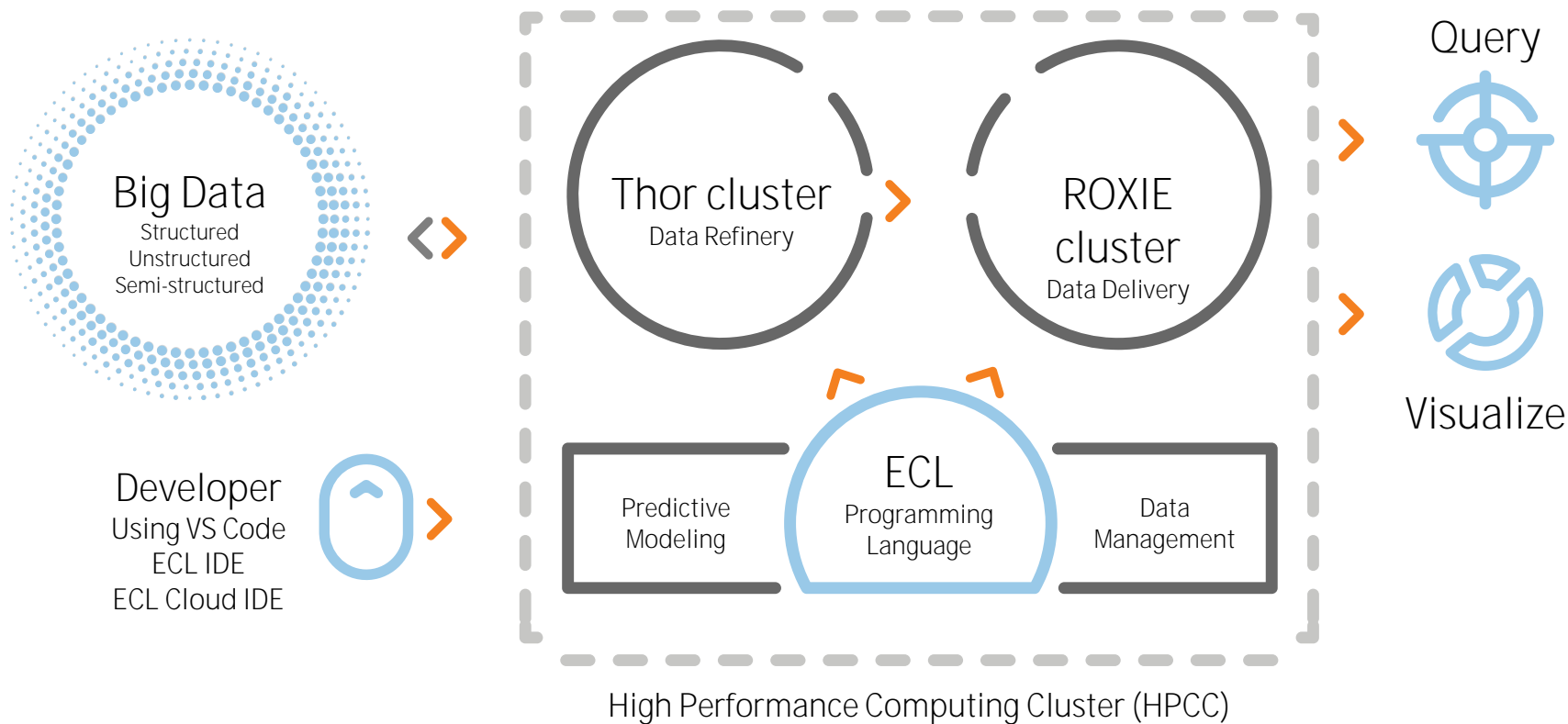
In more sophisticated cases, HPCC Systems run *clusters*, hundreds of servers working as a single processing entity, to transform and deliver big data.

Hard Metal

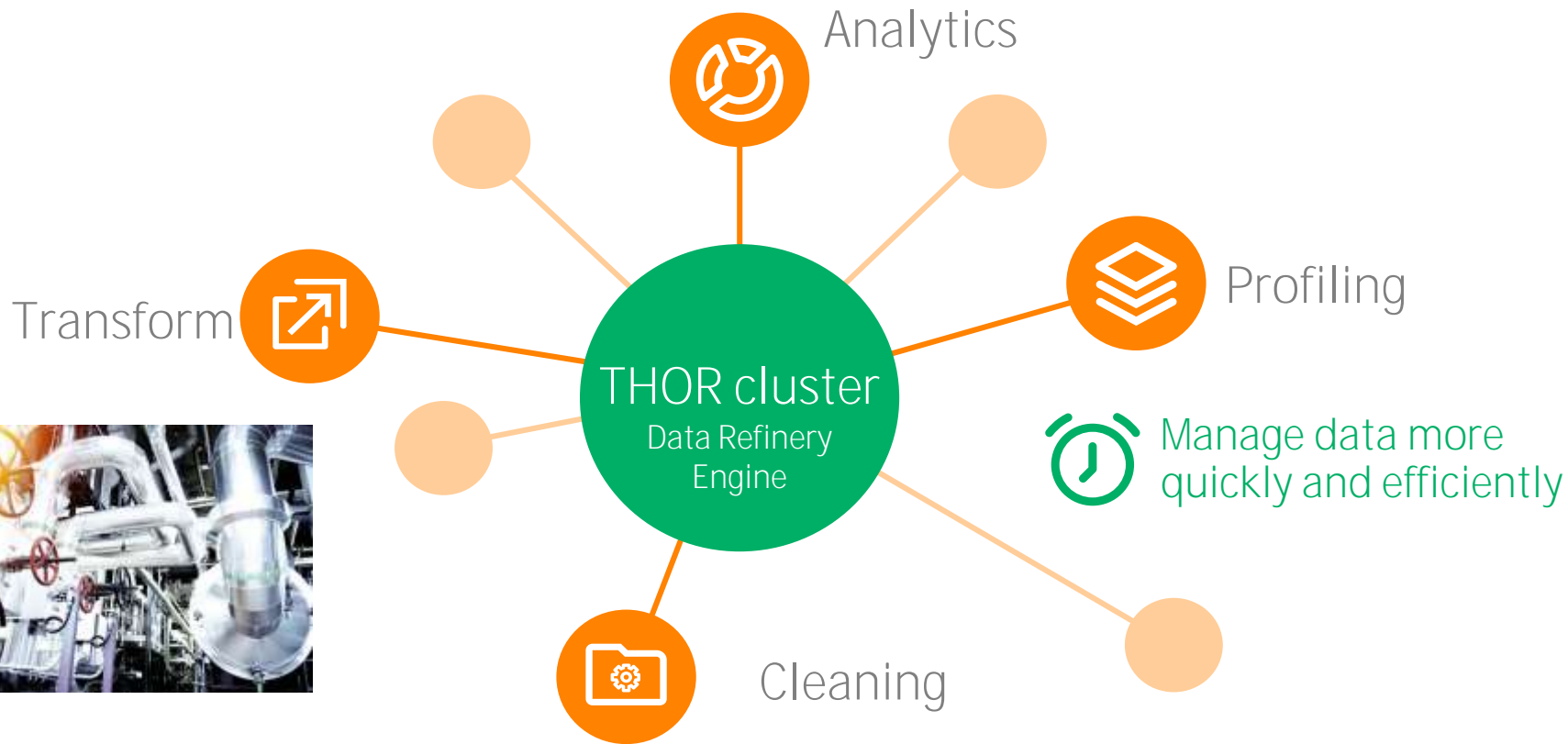
Cloud: AWS/Azure
Kubernetes



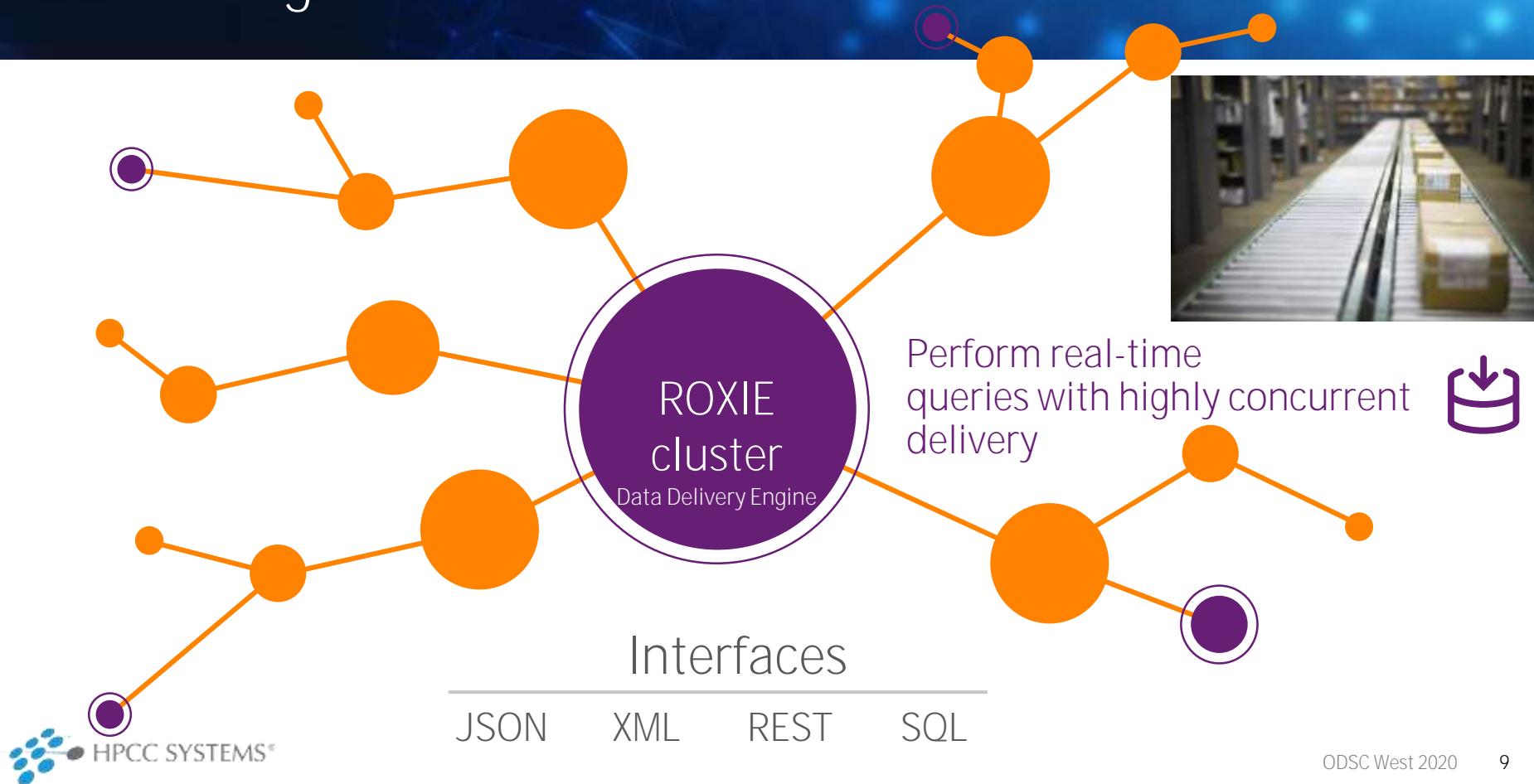
The HPCC Systems Components



THOR at a glance:



ROXIE at a glance:



Technology Summary — Layer View



Data Consumers

Real-time Services

Batch Services

Visualization

R

Spark

ODBC/JDBC



Analytics Tools

Normalization

Batch Services

Profiling

Cleaning

Data Management

ECL ML

Machine Learning



Common Programming Language

ECL



Data Engines

Thor cluster

ROXIE cluster

An Introduction to ECL

ECL

Enterprise Control
Language



```
IMPORT $, STD, ML;
EXPORT Func(UNSIGNED C, UNSIGNED2 Dist, UNSIGNED size, STRING Fld, REAL Parm1=0, REAL Parm2=0, REAL Parm3=0) := MODULE
  SHARED Node := STD.system.ThorLib.Node();
  SHARED PersistPrefix := $.Params.PersistPrefix;
  SHARED TotalRecs := $.Params.RecCnt*CLUSTER_SIZE;
  SHARED UIDval := IF(C=1, node, node + ((C-1)*CLUSTER_SIZE));
  SHARED BOOLEAN IsRandFile := $.Params.Randomness = $.ut.RandomSrc.file;
  SHARED Normal := FUNCTION
    Thisdist := IF(Parm3=0,
      ML.Distribution.Normal(Parm1, Parm2),
      ML.Distribution.Normal(Parm1, Parm2, Parm3));
    RetVals := ML.Distribution.GenData(TotalRecs, Thisdist, 1); PERSIST(PersistPrefix + 'NormalDistInt' + Fld, EXPIRE(1));
    RETURN RetVals;
  END;
  SHARED Normal2 := FUNCTION
    Thisdist := IF(Parm3=0,
      ML.Distribution.Normal2(Parm1, Parm2),
      ML.Distribution.Normal2(Parm1, Parm2, Parm3));
    RetVals := ML.Distribution.GenData(TotalRecs, Thisdist, 1); PERSIST(PersistPrefix + 'Normal2DistInt' + Fld, EXPIRE(1));
    RETURN RetVals;
  END;
  SHARED Uniform := FUNCTION
    Thisdist := IF(Parm3=0,
      ML.Distribution.Uniform(Parm1, Parm2),
      ML.Distribution.Uniform(Parm1, Parm2, Parm3));
    RetVals := ML.Distribution.GenData(TotalRecs, Thisdist, 1); PERSIST(PersistPrefix + 'UniformDistInt' + Fld, EXPIRE(1));
    RETURN RetVals;
  END;
  SHARED StudentT := FUNCTION
    Thisdist := ML.Distribution.StudentT(Parm1, Parm2);
    RetVals := ML.Distribution.GenData(TotalRecs, Thisdist, 1); PERSIST(PersistPrefix + 'StudentTDistInt' + Fld, EXPIRE(1));
    RETURN RetVals;
  END;
END
```



Powerful
language built
for big data

How to do it



vs.

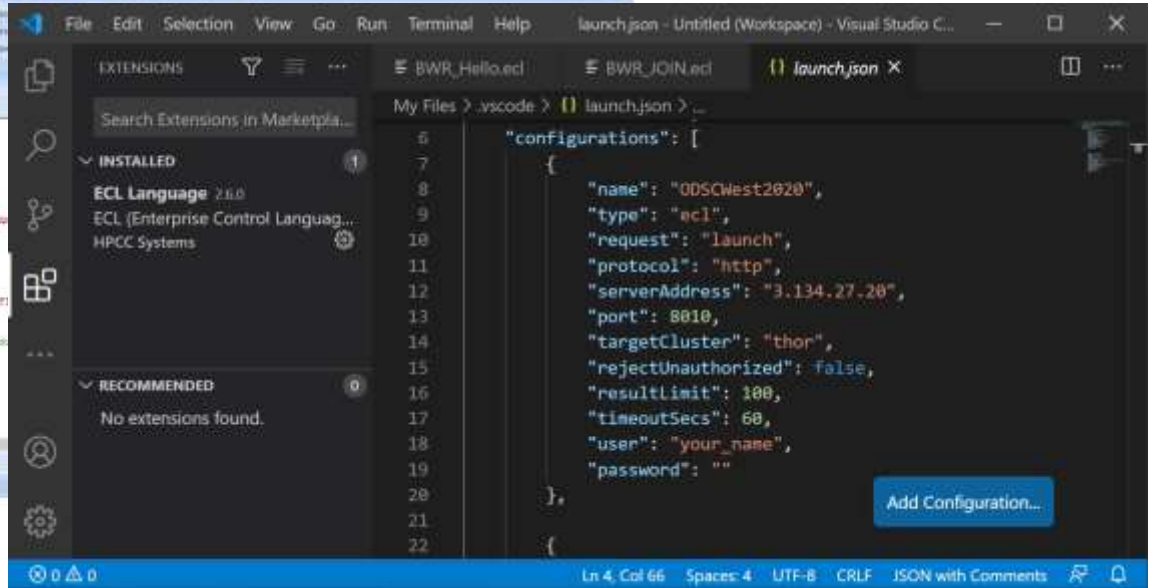
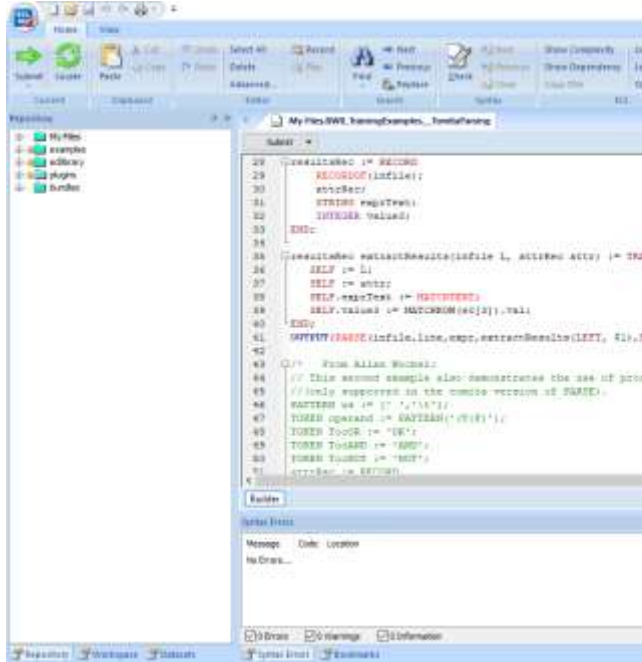


What to do

Integrated Development Environments

✓ECL IDE (Win)

✓VSCode (Ux/MacOS)



✓CLI Too! ECL.EXE

ECL IDE Features:

A full-featured GUI for ECL development providing access to the ECL repository and many of the ECL Watch capabilities.

Uses various ESP services via SOAP.



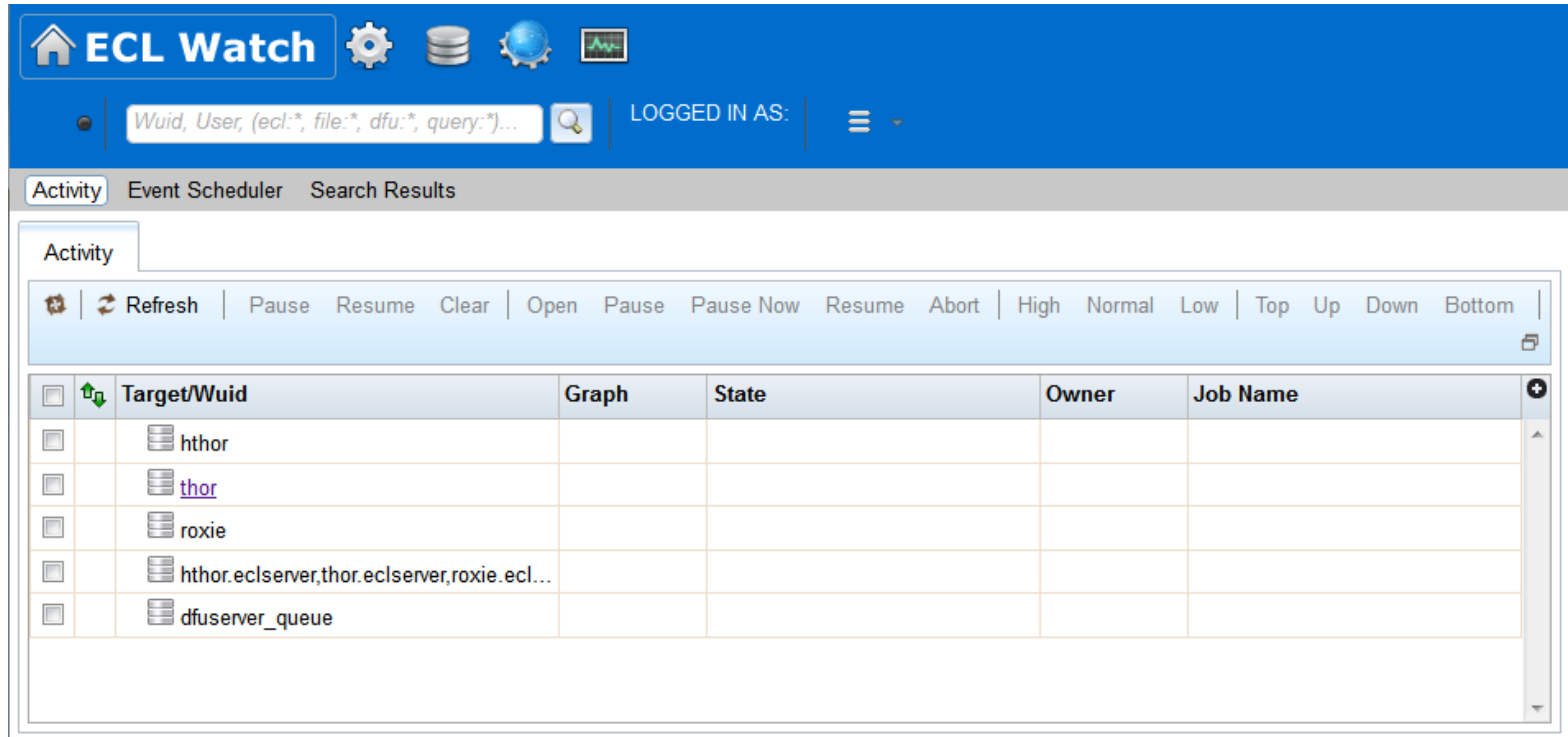
Provides the easiest way to create:

Queries into your data.

ECL Definitions to build your queries which:

- Are created by coding an expression that defines how some calculation or record set derivation is to be done.
- Once defined, can be used in succeeding ECL definitions.

The ECL Watch



The screenshot displays the ECL Watch web application interface. At the top, there is a blue header bar with the "ECL Watch" logo, several icons (gear, database, globe, line graph), and a search bar containing the text "Wuid, User, (ecl:*, file:*, dfu:*, query:*)...". To the right of the search bar, it says "LOGGED IN AS:" followed by a menu icon. Below the header, there is a navigation bar with tabs for "Activity", "Event Scheduler", and "Search Results". The "Activity" tab is currently selected. Below the navigation bar, there is a sub-header for "Activity" with a toolbar containing buttons: "Refresh", "Pause", "Resume", "Clear", "Open", "Pause", "Pause Now", "Resume", "Abort", and a set of priority filters: "High", "Normal", "Low", and a set of sort order filters: "Top", "Up", "Down", "Bottom". Below the toolbar is a table with the following columns: "Target/Wuid", "Graph", "State", "Owner", and "Job Name". The table contains five rows of data:

	Target/Wuid	Graph	State	Owner	Job Name
<input type="checkbox"/>	hthor				
<input type="checkbox"/>	thor				
<input type="checkbox"/>	roxie				
<input type="checkbox"/>	hthor.eclserver,thor.eclserver,roxie.ecl...				
<input type="checkbox"/>	dfusever_queue				

ECL Watch: What is it?

Browser-based interface that provides admin control over a cluster

- Monitor cluster status
- Job information
- Schedule and reschedule periodic tasks
- Manage data
- Maintain Roxie queries
- Security management

Default URL

- <http://hostname:8010> or [18010](http://hostname:18010) (SSL)

Useful Documentation

http://cdn.hpccsystems.com/releases/CE-Candidate-7.12.0/docs/The_ECL_Watch_Manual-7.12.0-1.pdf

ECL: What is it?

Declarative Programming Language

- “... a programming paradigm ... that expresses the logic of a computation without describing its control flow.” – Wikipedia

Designed For Big Data Scenarios

Any Cluster Size

Source-to-source compiler

- ECL code translated to C++ that is compiled to shared libraries and executed within a custom framework

Useful Documentation (on your laptops in PDF format)

- Language Reference
 - <https://hpccsystems.com/training/documentation/ecl-language-reference/html>
- Standard Library
 - <https://hpccsystems.com/training/documentation/standard-library-reference/html>

ECL: Basic (but important) Stuff

Two Statement Types

- Definition
 - Define an expression to an attribute (aka definition name)
 - Definition Operator (:=)
 - One naming rule – must start with a letter
- Action
 - Actually do something that affects the outside world
 - **Let's you see the definition result**
 - Can be implicit or explicit

Plot Twist

- You can define an attribute (definition) as an action to use later

ECL: Common Data Types

Character

- STRING[n]
- UTF8
- UNICODE[_locale][n]

Numeric

- INTEGER[n]
- UNSIGNED[n]
- REAL[n]
- DECIMAL<n>[_y]
- UDECIMAL<n>[_y]

Other

- BOOLEAN
- SET OF <type>
- RECORD
- DATASET

ECL: Important Minutia

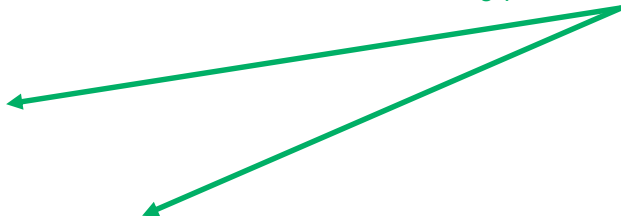
- ✓ Case-insensitive
- ✓ Whitespaces ignored
- ✓ String literals are quoted with apostrophes
- ✓ Semicolon terminator
- ✓ C++/Java style commenting
- ✓ Definition operator is `:=`
- ✓ Equality test operator is `=`
- ✓ **Inequality has two operators `<>` or `!=`**
- ✓ Attributes can be defined only once
- ✓ Single-pass code parser – (Only previously-defined attributes can be referenced)
- ✓ Only those definitions that contribute to a result are actually compiled and used
- ✓ There are no loops

ECL: RECORD and DATASET

Defining a record structure

- *attr* := **RECORD**
 data_type field1;
 ...
 data_type field100;
 END;
- *attr* := {*data_type* field1, ..., *data_type* field100};

List of data types and their attribute names



Defining a reference to saved data

- *attr* := **DATASET**(*path*, *record_structure*, *file_type*);
- *file_type*
 - FLAT: Native file type for Thor; also used for fixed-length raw records
 - CSV: Any kind of delimited data, including CSV-encoded data
 - JSON: Data stored as a series of JSON objects
 - XML: Data stored as a series of XML documents
 - PIPE: Data obtained dynamically via process calls

ECL: Filtering Data

Filters can be specified using a simple syntax:

- *attr := dataset(boolean_expression);*
- Basically, put your filter within parenthesis after the dataset/recordset representing the data
- All records within *dataset* will be evaluated
- If *boolean_expression* evaluates to TRUE for a particular record, it will be included in the result
- Logical operators:
 - AND
 - OR
 - NOT or ~
- Comparison operators:
 - =
 - <> or !=
 - <
 - >
 - <=
 - >=
 - <=> Equivalency comparison, returns -1, 0 or 1
 - Can use parenthesis to group subconditions

ECL Examples

Randomly-Chosen ECL Examples

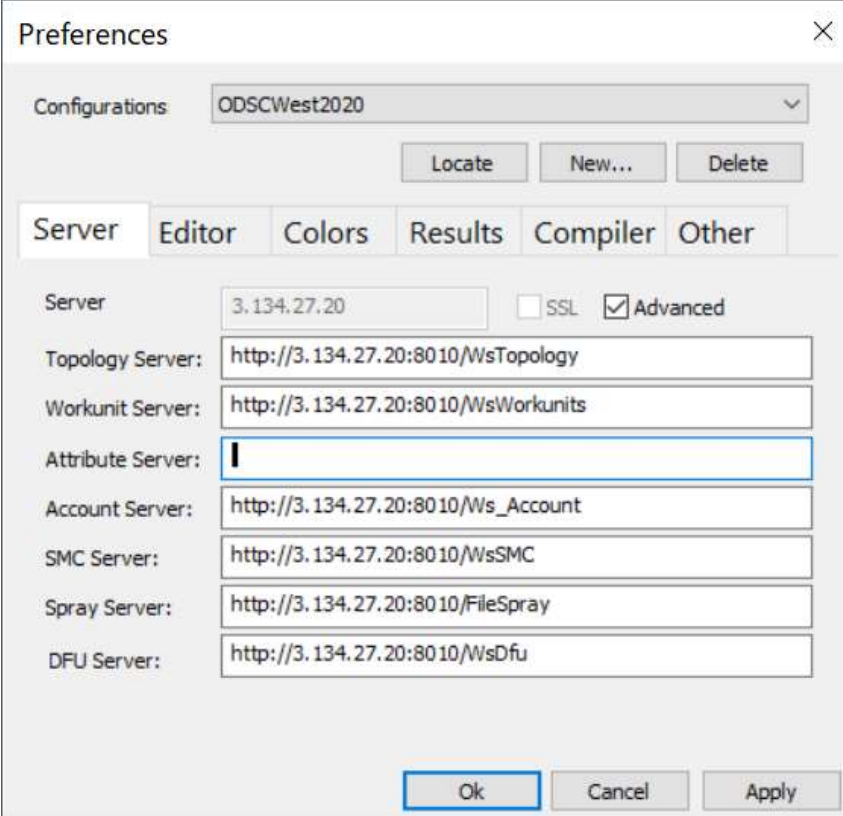
(<https://github.com/hpccsystems-solutions-lab/hpcc-systems-BR.git>)

Let's fire up the ECL IDE!

Over to Hugo!

ECL IDE Setup

Our Cluster:
8 node THOR
3.134.27.20

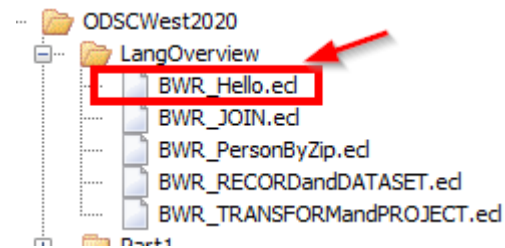


The screenshot shows the 'Preferences' dialog box for the ECL IDE, with the 'Server' tab selected. The 'Configurations' dropdown is set to 'ODSCWest2020'. Below the tabs, the 'Server' field is set to '3.134.27.20', with 'SSL' unchecked and 'Advanced' checked. The following servers are configured with their respective URLs:

Field	Value
Server	3.134.27.20
Topology Server:	http://3.134.27.20:8010/WsTopology
Workunit Server:	http://3.134.27.20:8010/WsWorkunits
Attribute Server:	
Account Server:	http://3.134.27.20:8010/Ws_Account
SMC Server:	http://3.134.27.20:8010/WsSMC
Spray Server:	http://3.134.27.20:8010/FileSpray
DFU Server:	http://3.134.27.20:8010/WsDfu

At the bottom, there are 'Ok', 'Cancel', and 'Apply' buttons.

ECL: Hello World



```
*My Files.CodeCamp.BWR_MyFirstCodeProject
Submit ▼ Target: thor ▼
1  // EXPORT BWR_MyFirstCodeProject := 'todo';
2
3  // Define an attribute with a string literal
4  MY_STRING := 'Hello World!';
5
6  // Action that displays the string in the workunit
7  OUTPUT(MY_STRING);
8
```


ECL: Overly-Complicated Hello World

```
9  // Define an attribute with a string literal
10 SUFFIX := 'World!';
11
12 // Define a function
13 □ STRING AddSuffix(STRING prefix) := FUNCTION
14     RETURN prefix + ' ' + SUFFIX;
15 END;
16
17 // Define an attribute as the result of a function all
18 myValue := AddSuffix('Hello');
19
20 // Action that displays the result ("Hello World!") in the workunit
21 OUTPUT(myValue);
22
```

ECL: Output Existing File – RECORD and DATASET



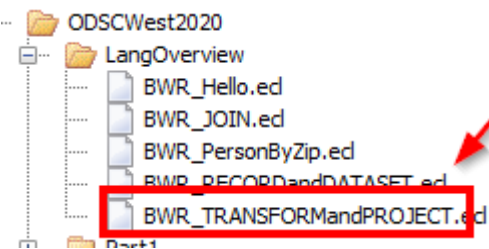
```
Submit | Target: thor More

1 // Record definition of the file
2 PersonRec := RECORD
3     UNSIGNED4    id;
4     STRING40     name;
5     STRING5      zip_code;
6 END;
7
8 // Reference to the data within the file
9 personData := DATASET
10 (
11     'wperson_file', // Path to file
12     PersonRec,      // Record structure of file
13     FLAT             // File type
14 );
15
16 // Output first 100 records
17 OUTPUT(personData, NAMED('PersonSample'));
18
```

ECL: Simple Filter Data Example

```
Submit Target: thor More
1 // Record definition of the file
2 PersonRec := RECORD
3     UNSIGNED4 id;
4     STRING40 name;
5     STRING5 zip_code;
6 END;
7
8 // Reference to the data within the file
9 personData := DATASET
10 (
11     'person_file', // Path to file
12     PersonRec,     // Record structure of file
13     THOR           // File type
14 );
15
16 // Get only those records with a certain zip code
17 oneZip := personData(zip_code = '60505');
18
19 // Output first 100 records
20 OUTPUT(oneZip, NAMED('Zip60505'));
21
```

ECL: Process Every Record – TRANSFORM and PROJECT



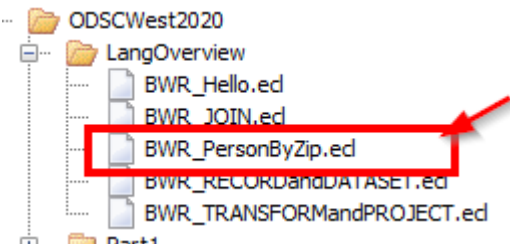
```
Submit Target: thor More

2 // Record definition of the file
3 PersonRec := RECORD
4   UNSIGNED4 id;
5   STRING40  name;
6   STRING5   zip_code;
7 END;
8 // Reference to the data within the file
9 personData := DATASET
10  (
11    '~person_file', // Path to file
12    PersonRec,      // Record structure of file
13    FLAT            // File type
14  );
15 // Output first 100 records from original file
16 OUTPUT(personData, NAMED('PersonSample'));
17
18 // Define a transform that titlecases the name and simply
19 // copies all other attributes
20 PersonRec TitlecaseName(PersonRec rec) := TRANSFORM
21   SELF.name := Std.Str.ToTitleCase(rec.name);
22   SELF := rec;
23 END;
24
25 // Define an attribute to hold the transformed data
26 titleCasedPersonData := PROJECT(personData, TitlecaseName(LEFT));
27
28 // Output first 100 records of the uppercased data
29 OUTPUT(titleCasedPersonData, NAMED('TitlecasePersonSample'));
30
```

ECL: Process Every Record – Inline TRANSFORM

```
20
21 // Define an attribute to hold the transformed data
22 titleCasedPersonData := PROJECT
23   (
24     personData,
25     TRANSFORM
26       (
27         PersonRec,
28         SELF.name := Std.Str.ToTitleCase(LEFT.name),
29         SELF := LEFT
30       )
31   );
32
33 // Output first 100 records of the uppercased data
34 OUTPUT(titleCasedPersonData, NAMED('TitlecasePersonSample'));
35
```

ECL: Simple TABLE example

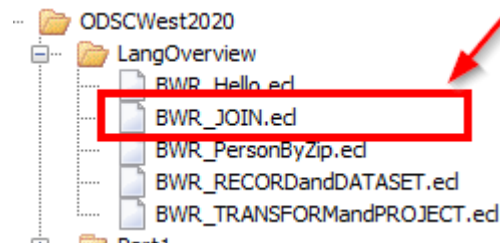


```
1 PersonRec := RECORD
2   UNSIGNED4 id;
3   STRING40 name;
4   STRING5 zip_code;
5 END;
6
7 personData := DATASET('person_file', PersonRec, FLAT);
8
9 // Count the number of people for each zip code
10 zipPeopleCount := TABLE
11 (
12   personData,
13   {
14     zip_code,
15     UNSIGNED4 num_people := COUNT(GROUP) // INLINE RECORD structure // Field used in grouping condition // Computed field
16   },
17   zip_code,
18   MERGE // Grouping condition // Flag
19 );
20
21 OUTPUT(zipPeopleCount, NAMED('NumPeopleInEachZipCode'));
22
```

ECL: JOIN Example

```
1 PersonRec := RECORD
2   UNSIGNED4 id;
3   STRING40 name;
4   STRING5 zip_code;
5 END;
6
7 personData := DATASET('wperson_file',
8   PersonRec, FLAT);
9
10 ZipLookupRec := RECORD
11   STRING5 zip5;
12   STRING20 city;
13   STRING2 state;
14 END;
15
16 zipData := DATASET('wzip_lookup',
17   ZipLookupRec, FLAT);
18
19 MergedRec := RECORD
20   UNSIGNED4 id;
21   STRING40 name;
22   STRING20 city;
23   STRING2 state;
24   STRING5 zip_code;
25 END;
```

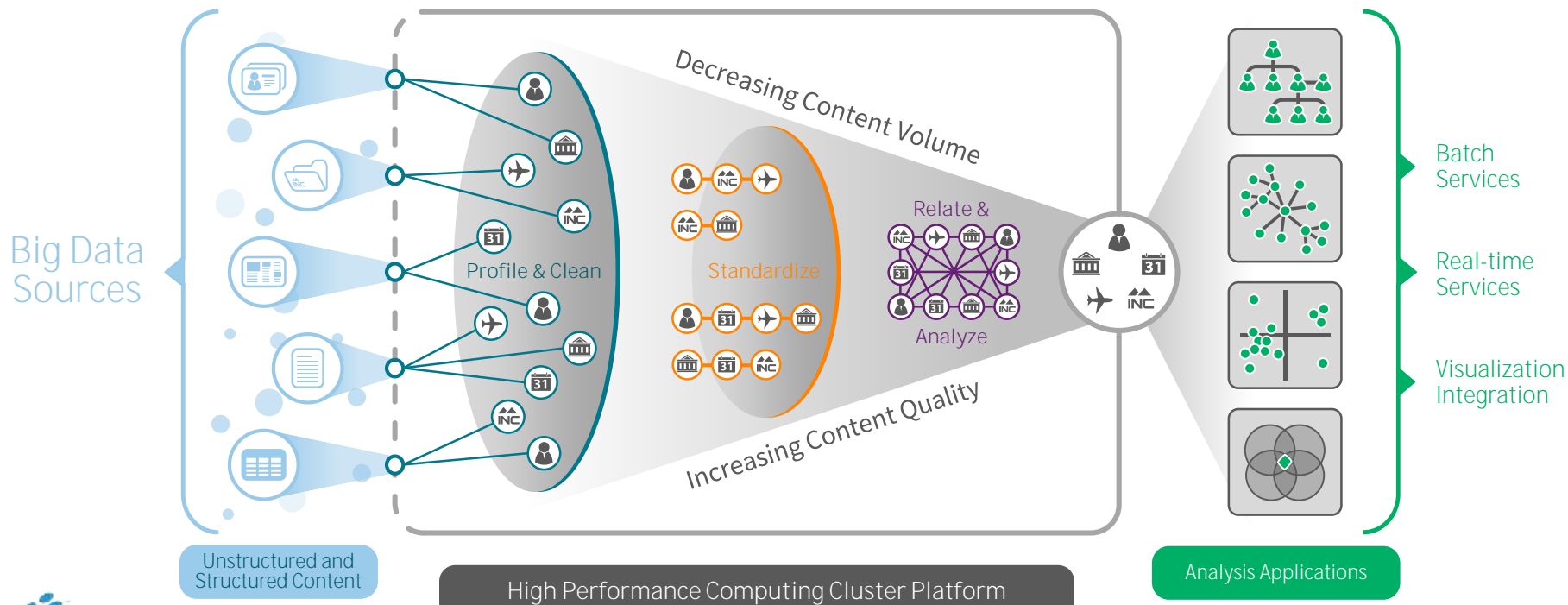
```
27 MergedRec MergePersonAndZip(PersonRec personInfo,
28   ZipLookupRec zipInfo) := TRANSFORM
29   SELF := personInfo; // Copy all values from personInfo
30   SELF := zipInfo; // Copy all values from zipInfo
31 END;
32
33 personAndZipInfo := JOIN
34   (
35     personData, // LEFT
36     ZipData, // RIGHT
37     LEFT.zip_code = RIGHT.zip5, // Join condition
38     MergePersonAndZip(LEFT, RIGHT), // Transform
39     LEFT OUTER // Join flag
40   );
41
42 OUTPUT(personAndZipInfo, NAMED('MergedPersonInfo'));
```





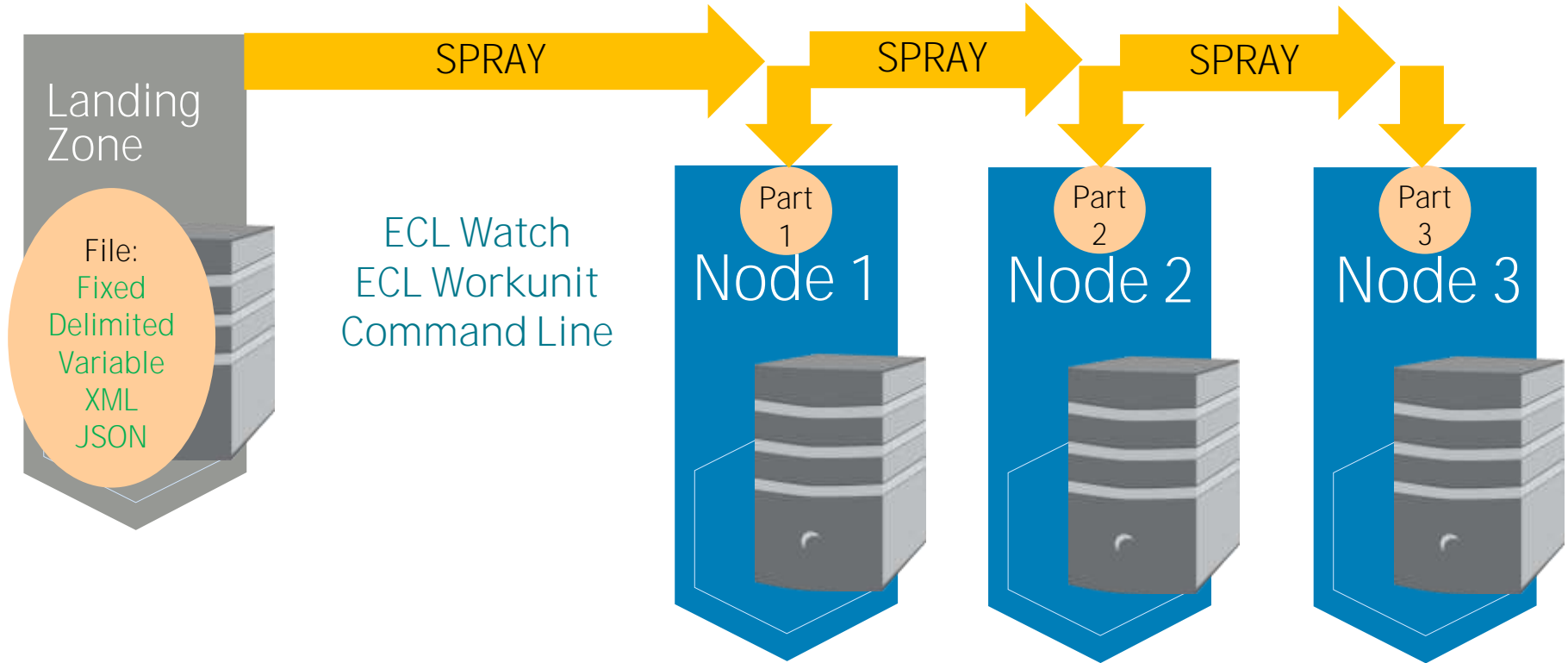
ECL, ETL and Visualization

HPCC Systems (Small to Big Data) ETL



SPRAY Operation

HPCC Cluster



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:

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

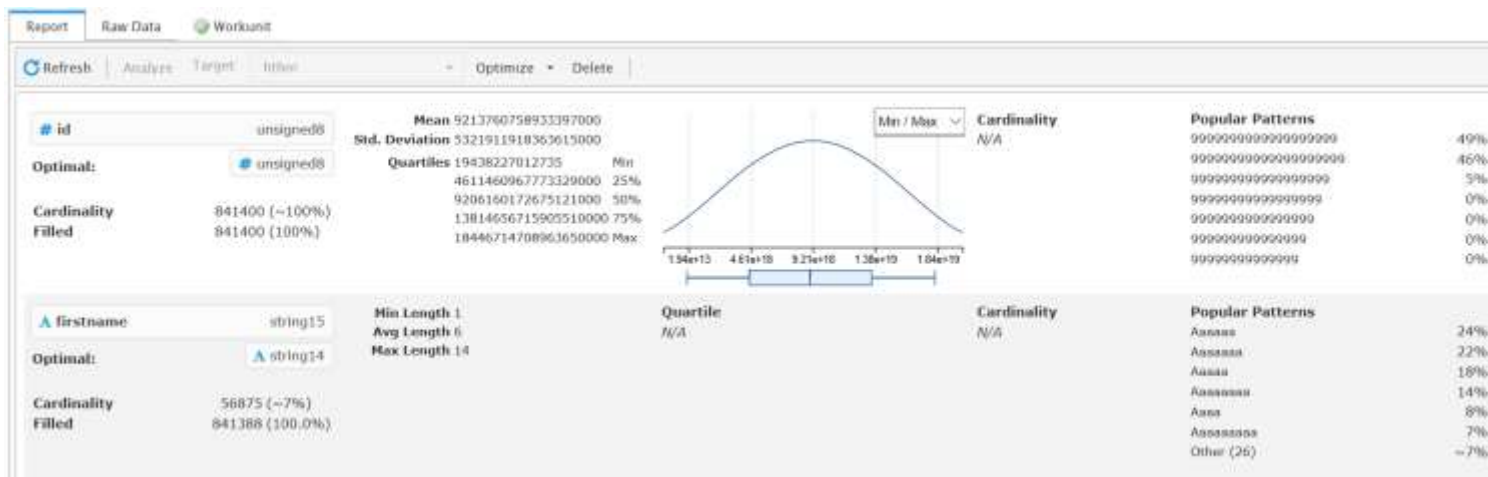
Built-In Data Profiling: Data Patterns

```
IMPORT STD.DataPatterns;
```

```
filePath := '~aaa::bmf::reptest::accounts';  
ds := DATASET(filePath, RECORDOF(filePath, LOOKUP), csv);  
profileResults := DataPatterns.Profile(ds);  
OUTPUT(profileResults, ALL, NAMED('profileResults'));
```

Requirements (for ECL Watch Analysis);

1. File must have an ECL RECORD
2. Size of the file should not be too large (Use SAMPLE for larger files).



Built-In Visualization Tools

HPCC Systems provides built-in Visualization of your output data in a variety of charts and graphs. You can visualize your data in three ways:

1. Using the Chart Tool in the ECL Playground.
2. Accessing the **Visualize** tab in all ECL workunits
3. Using the **Resources** tab in conjunction with the ECL Visualizer bundle.

Installing:

```
ecl bundle install https://github.com/hpcc-systems/Visualizer.git
```

Visualization Function Summary

Global or Helpers (**Any** MODULE):

Meta

Creates a special output record set that contains the meta information for use in the target visualization. Outputs visualization meta information.

Grid

Used with the Meta function to render data into the appropriate data grid or table. Mappings can be used to limit and/or rename the columns.

Two Dimensional Ordinal (**TwoD** MODULE):

Bubble - a series of circles whose size is proportional to the field's value

Pie - a single circle divided into proportional slices

Summary - values are displayed in designated intervals

RadialBar - basically a bar chart plotted on polar coordinates instead of a Cartesian plane

WordCloud - a visual representation of text data, whose size is proportional to its value.

Visualization Function Summary

Two Dimensional Linear (**TwoDLinear** MODULE):

Scatter - uses Cartesian (X/Y) coordinates to display dot or point values for two numeric fields in a dataset

HexBin - useful to represent 2 numerical fields when you have a lot of data points. Instead of overlapping, the plotting window is split into several hexbins, and the number of points per hexbin is counted. The color denotes this number of points.

Contour - A graphical technique that uses contour lines. A contour line of a function of two variables is a curve along which the function has a constant value, so that the curve joins points of equal value.

Multi-Dimensional (**MultiD** MODULE):

Area - Label is plotted on X-axis, and values for each label are plotted on the Y-Axis. Common fields are joined by line and area below is shaded.

Bar - Label is plotted on Y-axis, and values for each charted by bar on the X-Axis.

Column - Similar to Bar, but Label is plotted on the X-Axis, and associated values charted by bar on the Y-Axis.

Line - Similar to Area, but no shaded area is colored (lines only)

Radar - Similar to the Area plotting, but uses a center point in a circle for the X and Y axis zero coordinate. Think of a radar scope for this graph type.

Scatter - Points are marked only on this graph type

Step - Uses vertical and horizontal lines to connect data points, resembling a step ladder type of display.

Visualization Function Summary

Relational (**Relational** MODULE):

Network - assembles the vertices and edges defined into an entity relation chart

Geospatial (**Choropleth** MODULE):

USStates - A US States Map that maps a two letter State code with its associated value.

USCounties - A US County Map that maps a numeric FIPS (Federal Information Processing Standard) county code with its associated value.

Euro - a base map graph that allows you to select any country in the European continent. A two letter international code is required for the region.

EuroIE - Example function in the bundle that uses the Euro function to show the country map of Ireland.

EuroGB - Example function in the bundle that uses the Euro function to show the country map of Great Britain.

Each of these categories have an associated **__Test** function that can be used to view all graphs/charts/maps in that category:

```
IMPORT Visualizer;  
Visualizer.Choropleth.__test;
```


Visualization Quick Start

Using the Visualization Bundle is essentially a three step process.

1. Get your data ready. Too many points can make a graph unreadable, too little points diminishes the analysis benefit.
2. Import the Visualizer folder. If you use the "git" method, your folder will be easily located.
3. OUTPUT your data using the NAMED attribute.
4. Call your Visualizer chart using the NAMED attribute as your chart source

Example:

```
IMPORT $, Visualizer;  
GenderDS := DATASET([{'Female', 404988},  
                     {'Male', 384182},  
                     {'Neutral', 20508},  
                     {'Unknown', 70722}],  
                     {STRING Label, UNSIGNED4 Value});  
OUTPUT(GenderDS, NAMED('VizPie'));  
Visualizer.TwoD.Pie('Pie',, 'VizPie');
```

Let's practice
(ECL, ETL and Visualization)

Let's spray the data

<http://3.134.27.20:8010/> (ECL Watch)

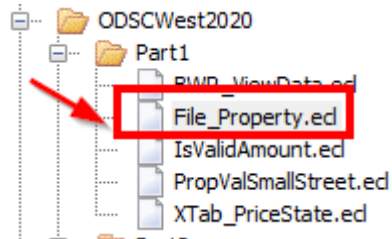
Your initials here!!!

The screenshot shows the ECL Watch interface for configuring a data spray. The 'Landing Zones' tab is active, displaying a file tree on the left. The file 'wkshpropertys154' is selected. The right-hand configuration panel shows the following settings:

- Group:** mythor
- Queue:** rfinserver.queue
- Target Scope:** WKSHCP_ODSCWest2020.XXX
- Target Name:** property
- Record Length:** 154
- Options:**
 - Overwrite:** ☒
 - No Split:** ☐
 - Compress:** ☐
 - Expire in (days):**
- Replicate:**
 - No Common:** ☐
 - Fail If No Source File:** ☐
 - Delayed replication:** ☒

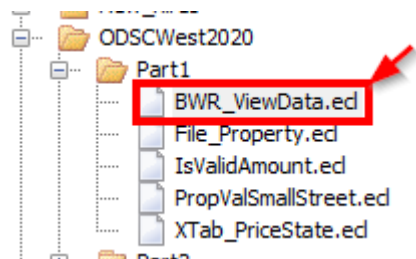
The 'Spray' button is located at the bottom right of the configuration panel.

The data structure



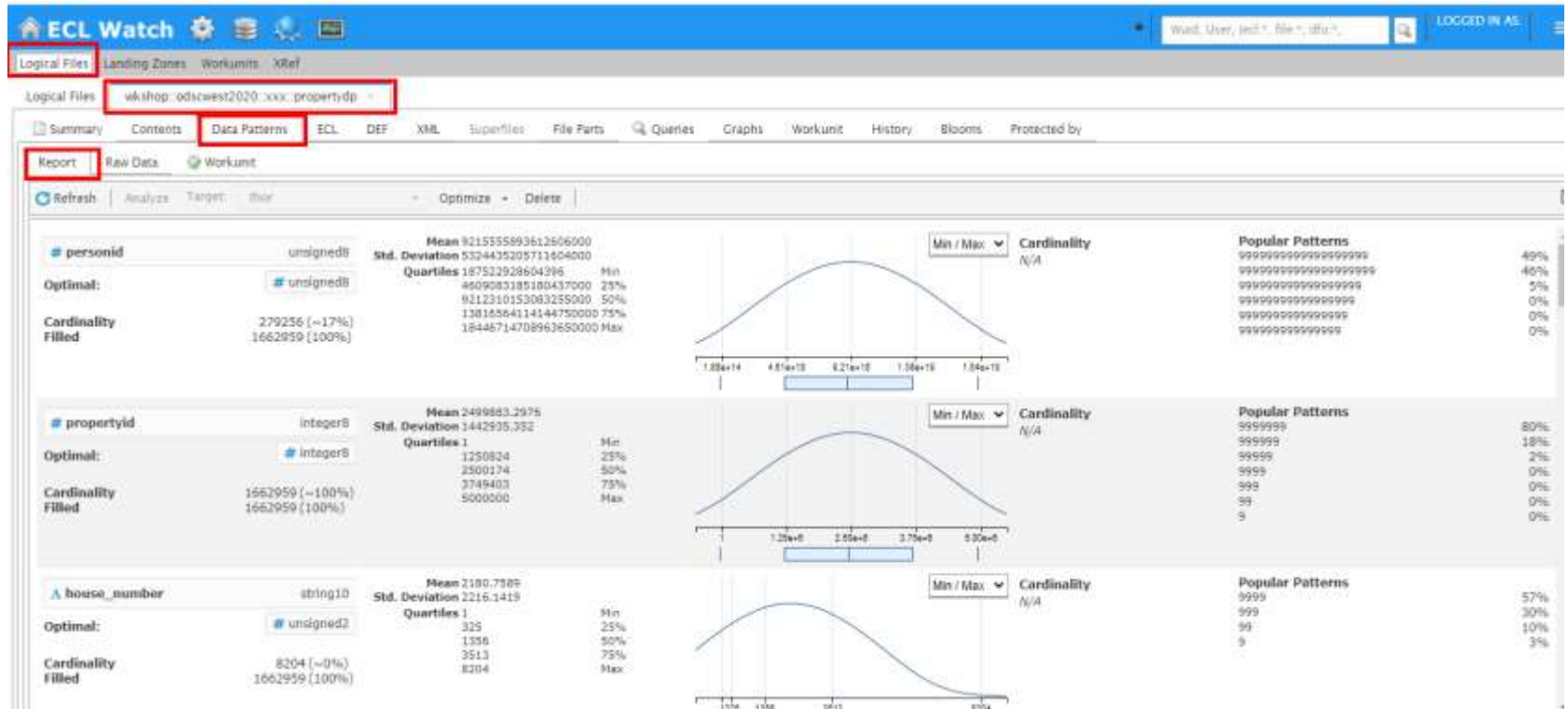
```
EXPORT File_Property := MODULE
  EXPORT Layout := RECORD
    UNSIGNED8 personid;
    INTEGER8 propertyid;
    STRING10 house_number;
    STRING10 house_number_suffix;
    STRING2 predir;
    STRING30 street;
    STRING5 streettype;
    STRING2 postdir;
    STRING6 apt;
    STRING40 city;
    STRING2 state;
    STRING5 zip;
    UNSIGNED4 total_value;
    UNSIGNED4 assessed_value;
    UNSIGNED2 year_acquired;
    UNSIGNED4 land_square_footage;
    UNSIGNED4 living_square_feet;
    UNSIGNED2 bedrooms;
    UNSIGNED2 full_baths;
    UNSIGNED2 half_baths;
    UNSIGNED2 year_built;
  END;
  EXPORT File := DATASET('~\WKSHOP::ODSCWest2020::XXX::property', Layout, THOR);
END;
```

Viewing the data (raw view)



##	personid	propertyid	house_number	house_number_suffix	predir	street	streettype	postdir	apt	city	state	zip
1	187522928604396	828195	144			MCKIERNAN	DR			WALNUT CREEK	CA	9459
2	187522928604396	1144455	281			CENTER	ST			BALTIMORE	MD	2113
3	187522928604396	1494347	483			NEWTON	RD			FLAGSTAFF	AZ	8601
4	187522928604396	1910847	802			HATCHERY	CT			WOODLAND	WA	9867
5	187522928604396	4267562	5007		E	ROY ROGERS	RD			TROY	MI	4808
6	187522928604396	4888602	7607			PEBBLESTONE	DR		000009	KERNVILLE	CA	9323
7	214582956185891	54135	4			WAINWRIGHT	DR			NORTH FORT MYERS	FL	3391
8	214582956185891	762012	125			SHIPYARD	DR		000150	MELBOURNE VILLAGE	FL	3290
9	214582956185891	2331721	1190			LITTLEOAK	DR			HOUSTON	TX	7701
10	214582956185891	3276109	2506			MEADOW	DR			LA QUINTA	CA	9225

Viewing the data (profile view)



Viewing the data (marketing view)

Need:

Calculate the aggregate Property values for all properties on “small” streets (CT, LN, WAY, CIR, PL, or TRL).

Requirements:

1. Use the total value and assessed value fields from the Property dataset
2. If both are valid, use the larger of the two valid values
3. If there are no properties return -9
4. If there are no “small” street properties return -9
5. If there are no “small” street properties w/valid values return -9

Viewing the data (marketing view)

```
IMPORT $;

Property := $.File_Property.File;
Total    := Property.Total_value;
Assessed := Property.Assessed_value;

IsSmallStreet := Property.StreetType IN ['CT','LN','WAY','CIR','PL','TRL'];

HighValue := IF($.IsValidAmount(Total) AND
                $.IsValidAmount(Assessed),
                IF(Total > Assessed,Total,Assessed),
                IF($.IsValidAmount(Total),Total,Assessed));

SmallProperties := Property(IsSmallStreet AND $.IsValidAmount(HighValue));
EXPORT PropValSmallStreet := IF(NOT EXISTS(SmallProperties),
                                -9,
                                SUM(SmallProperties,HighValue));
```

Result Comparison

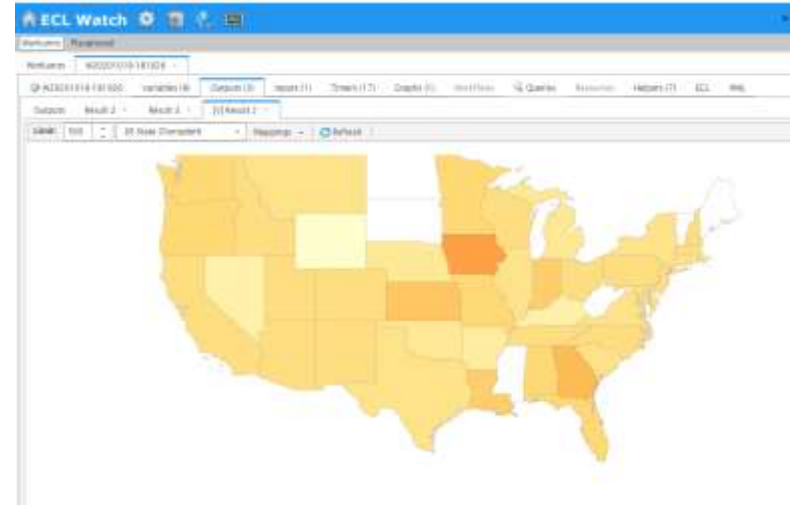
Open a new Builder window and run a query on the Property file like this:

```
IMPORT $;
$.PropValSmallStreet;
```

The expected result from this query is
4,539,672,569.

Viewing the data (graphical view)

```
IMPORT $;  
  
Property := $.File_Property.File;  
  
OutRec := RECORD  
    Property.state;  
    UNSIGNED4 avg_value := AVE(GROUP,Property.total_value);  
    UNSIGNED4 cnt:=COUNT(GROUP);  
END;  
  
EXPORT XTAB_PriceState := TABLE(Property,OutRec,state);
```



More graphical options

The image shows a Windows Command Prompt window and a file explorer window. The Command Prompt window is titled "Administrator: Command Prompt" and shows the following commands and output:

```
Microsoft Windows [Version 10.0.17134.1792]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>cd "C:\Program Files (x86)\HPCCSystems\7.6.64\clienttools\bin"

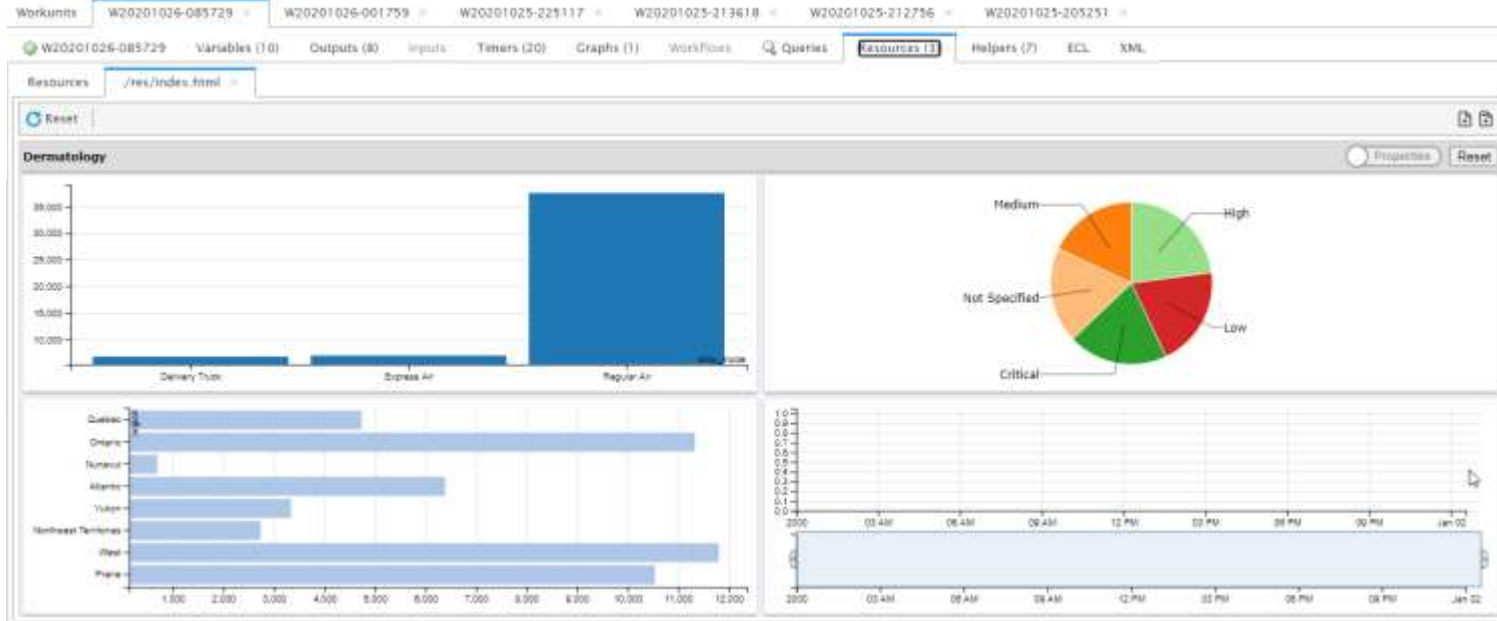
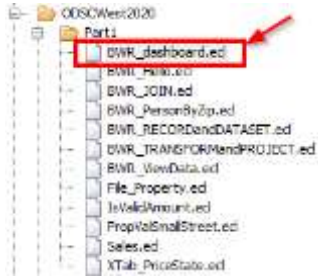
C:\Program Files (x86)\HPCCSystems\7.6.64\clienttools\bin>ecl bundle install https://github.com/hpcc-systems/Visualizer.git --update --force
Installing bundle Visualizer version 2.0.0
Existing active version 2.0.0 is newer or same version
--force specified - updating anyway
Visualizer 2.0.0 ECL Visualization Bundle
Installation complete
ecl 'bundle' command error 0
```

The file explorer window shows the "Repository" folder, which contains the following files and folders:

- My Files
- examples
- eclibrary
- plugins
- bundles
- versions
- DataPatterns.ed
- LearningTrees.ed
- ML_Core.ed
- TextVectors.ed
- Visualizer.ed

Red boxes highlight the "bundles" folder in the file explorer and the "Visualizer.ed" file. A red arrow points to the "Visualizer.ed" file.

More graphical options (cont.)





Break and Questions

Natural Language Parsing

- ✓ Pattern Type Definitions
- ✓ Parse Pattern Definitions
- ✓ NLP RECORD Structure Functions
- ✓ XML RECORD Structure Functions
- ✓ PARSE Function

Overview of Natural Language Parsing:

Natural Language Parsing is accomplished in ECL by combining pattern definitions with an output RECORD structure specifically designed to receive the parsed values, then using the PARSE function to perform the operation.

Pattern definitions are used to detect "interesting" text within the data. Just as with all other ECL definitions, these patterns typically define specific parsing elements and may be combined to form more complex patterns, tokens, and rules.

The output RECORD structure (or TRANSFORM function) defines the format of the resulting recordset. **It typically contains specific pattern matching functions that return the "interesting" text, its length or position.**

The PARSE function implements the parsing operation. It returns a recordset that may then be post-processed as needed using standard ECL syntax, or simply output.

PATTERN and TOKEN

PATTERN *patternid* := *parsepattern*;

- ✓ *patternid* – The definition name of the pattern.
- ✓ *parsepattern* – The pattern, very similar to regular expressions. This may contain other previously defined PATTERN definitions.

The PATTERN value type defines a parsing expression very similar to regular expression patterns.

TOKEN *tokenid* := *parsepattern*;

- ✓ *tokenid* – The definition name of the token.
- ✓ *parsepattern* – The token pattern, very similar to regular expressions. This may contain PATTERN definitions but no TOKEN or RULE definitions.

The TOKEN value type defines a parsing expression very similar to a PATTERN, but once matched, the parser doesn't backtrack to find alternative matches as it would with PATTERN.

RULE

RULE *ruleid* := *parsepattern*;

- ✓ *ruleid* – The definition name of the rule.
- ✓ *parsepattern* – The rule pattern, very similar to regular expressions. This may contain previously defined PATTERN, TOKEN, and RULE definitions.

The RULE value type defines a parsing expression containing combinations of TOKENs. If a RULE definition contains a PATTERN it is implicitly converted to a TOKEN. Like PATTERN, once matched the parser does backtrack to find alternative RULE matches.

Parse Pattern Definitions:

pattern-name	The name of any previously defined PATTERN attribute.
(pattern)	Parentheses may be used for grouping.
pattern1 pattern2	Pattern1 followed by pattern2.
'string'	A fixed text string, which may contain escaped octal string control characters (for example, CtrlZ is '\032').
FIRST	Matches the start of the string to search. This is similar to the regular expression ^ token, which is not supported.
LAST	Matches the end of the string to search. This is similar to the regular expression \$ token, which is not supported.
ANY	Matches any character.
REPEAT(pattern)	Repeat the pattern any number of times. The regular expression syntax pattern* is supported as a shorthand for REPEAT(pattern).
REPEAT(pattern, expression)	Repeat the pattern expression times.
REPEAT(pattern, low, ANY [,MIN])	Repeat the pattern low or more times (with the MIN option making it a minimal match).
REPEAT(pattern, low, high)	Repeat the pattern from low to high times.
OPT(pattern)	An optional pattern. The regular expression syntax pattern? is supported as a shorthand for OPT(pattern).
pattern1 OR pattern2	Either pattern1 or pattern2. The regular expression syntax pattern1 pattern2 is supported as a shorthand for OR.
[list-of-patterns]	A comma-delimited list of alternative patterns, useful for string sets. This is the same as OR.

Parse Pattern Definitions (cont.):

pattern1 [NOT] IN pattern2	Does the text matched with pattern1 also match pattern2?
pattern1 [NOT] BEFORE pattern2	Check if the given pattern2 does [not] follow pattern1.
pattern1 [NOT] AFTER pattern2	Check if the given pattern2 does [not] precede pattern1.
pattern LENGTH(range)	Check whether the length of a pattern is in the range.
VALIDATE(pattern, isValidExpression)	Evaluate isValidExpression to check if the pattern is valid or not.
VALIDATE(pattern, isValidAsciiExpression, isValidUnicodeExpression)	A two parameter variant. Use the first isValidAsciiExpression if the string being searched is ASCII; use the second if it is Unicode.
NOCASE(pattern)	Matches the pattern case insensitively, overriding the CASE option on the PARSE function.
CASE(pattern)	Matches the pattern case sensitively, overriding the NOCASE option on the PARSE function. This may be nested within a NOCASE pattern.
pattern PENALTY(cost)	Associate a penalty cost with this match of the pattern.
TOKEN(pattern)	Treat the pattern as a token.
PATTERN('regular expression')	

Parse Example:

```
ds := DATASET([{'the fox; and the hen'}], {STRING100 line});
```

```
PATTERN ws           := PATTERN('[\t\r\n]');  
PATTERN Alpha        := PATTERN('[A-Za-z]');  
PATTERN Word         := Alpha+;  
PATTERN Article      := ['the', 'A'];  
TOKEN JustAWord      := Word PENALTY(1);  
PATTERN notHen       := VALIDATE(Word, MATCHTEXT != 'hen');  
TOKEN NoHenWord      := notHen PENALTY(1);  
RULE NounPhraseComp1 := JustAWord | Article ws Word;  
RULE NounPhraseComp2 := NoHenWord | Article ws Word;
```

```
ps1 := {out1 := MATCHTEXT(NounPhraseComp1) };  
ps2 := {out2 := MATCHTEXT(NounPhraseComp2) };
```

```
p1 := PARSE(ds, line, NounPhraseComp1, ps1, BEST,MANY,NOCASE);  
p2 := PARSE(ds, line, NounPhraseComp2, ps2, BEST,MANY,NOCASE);
```

NLP RECORD Structure Functions:

- ✓ MATCHED([*patternreference*])

returns true or false as to whether the *patternreference* found a match. If the *patternreference* is omitted, it indicates whether the entire pattern matched or not (for use with the NOT MATCHED option).

- ✓ MATCHTEXT([*patternreference*])

returns the matching ASCII text the *patternreference* found, or blank if not found. If the *patternreference* is omitted, MATCHTEXT returns all matching text.

- ✓ MATCHUNICODE([*patternreference*])

returns the matching Unicode text the *patternreference* found, or blank if not found.

NLP RECORD Structure Functions (cont.):

- ✓ MATCHLENGTH([*patternreference*])

returns the number of characters in the matching text the *patternreference* found, or 0 if not found.

- ✓ MATCHPOSITION([*patternreference*])

returns the position within the text of the first character in the matching text the *patternreference* found, or 0 if not found.

- ✓ MATCHROW([*patternreference*])

returns the entire row of the matching text the *patternreference* found for a RULE (valid only when the PARSE option is used on the PARSE function). This may be used to fully qualify a field in the RECORD structure of the row.

Pattern References

✓ Pattern References

The *patternreference* parameter to these functions is a slash-delimited (/) list of previously defined PATTERN, TOKEN, or RULE definitions with or without an instance number appended in square brackets. If an instance number is supplied, it matches a particular occurrence, otherwise it matches any. The *patternreference* provides a path through the regular expression grammar to a particular result. The path to a particular definition can either be fully or partially specified.

Pattern References Example

```
PATTERN arb          := PATTERN('[!.,\t a-zA-Z0-9]')+;
PATTERN number       := PATTERN('[0-9]')+;
PATTERN age          := '(' number OPT('/I') ')';
PATTERN role         := '[' arb ']';
PATTERN m_rank       := '<' number '>';
PATTERN actor        := arb OPT(ws '(I)' ws);
PATTERN line         := actor '\t' arb ws OPT(age) ws OPT(role) ws OPT(m_rank) ws;
```

```
NLP_layout_actor_movie := RECORD
  STRING30 actor_name := MATCHTEXT(actor);
  STRING50 movie_name := MATCHTEXT(arb[2]);           //2nd instance of arb
  UNSIGNED2 movie_year := (UNSIGNED)MATCHTEXT(age/number); //number within age
  STRING20 movie_role := MATCHTEXT(role/arb);        //arb within role
  UNSIGNED1 cast_rank := (UNSIGNED)MATCHTEXT(m_rank/number);
END;
```

Instructor Note: Complete example can be found in the LRM.

PARSE (NLP Form)

PARSE(*dataset,data,pattern,result,flags*)

- ✓ *dataset* - The set of records to process.
- ✓ *data* - An expression specifying the text to parse, typically the name of a field in the *dataset*.
- ✓ *pattern* - The pattern to parse with.
- ✓ *result* - The name of the RECORD structure definition that specifies the format of the output record set.
- ✓ *flags* - One or more parsing options, as defined below.

This form of PARSE operates on the *dataset*, using the *pattern* and creating a result set in the *result* structure format. This is similar to the TABLE function, but PARSE may also operate on variable length text.

Flags can have the following values:

FIRST

Only return a row for the first match starting at a particular position

ALL

Return a row for every possible match of the string at a particular position.

WHOLE

Only match the whole string.

NOSCAN

If a position matches, don't continue searching for other matches.

SCAN

If a position matches, continue searching from end of the match, otherwise continue from the next position.

SCAN ALL

Return matches for every possible start position. (TRIM recommended)

PARSE Flags (cont.)

Flags (continued):

NOCASE

Perform a case insensitive comparison.

CASE

Perform a case sensitive comparison (this is the default).

SKIP(*separator-pattern*)

Pattern to insert between tokens

KEEP(*max*)

Only keep the first *max matches*.

ATMOST(*max*)

Don't produce any matches if there are more than *max matches*.

MAX

Return a row for the result that matches the longest sequence of the input.

MIN

Return a row for the result that matches the shortest sequence of the input.

PARSE Flags (cont.)

Flags (continued):

MATCHED([*rule-reference*])

Used when *rule-reference* is used in a user-matching function.

MATCHED(ALL)

Retain all rule-names – if they are used by user match functions.

NOT MATCHED

Generate a row if there were no matches on the input row.

NOT MATCHED ONLY

Only generate a row if no matches were found.

BEST

Pick the match with the highest score (lowest penalty).

MANY

Return multiple matches for BEST, MAX, or MIN options.

PARSE Example:

```
datafile := DATASET([{'And when Shechem the son of Hamor the Hivite, prince of
Reuel'},{'the son of Bashemath the wife of Esau.'}], {STRING10000 line});
PATTERN ws1                := [' ', '\t', ',', '.'];
PATTERN ws                  := ws1 ws1?;
PATTERN article             := ['A', 'The', 'Thou', 'a', 'the', 'thou'];
TOKEN Name                  := PATTERN('[A-Z][a-zA-Z]+');
RULE Namet                  := name OPT(ws ['the', 'king of', 'prince of'] ws name);
PATTERN produced            := OPT(article ws) ['begat', 'father of', 'mother of'];
PATTERN produced_by         := OPT(article ws) ['son of', 'daughter of'];
PATTERN produces_with       := OPT(article ws) ['wife of'];
RULE relationtype           := ( produced | produced_by | produces_with );
RULE progeny                := namet ws relationtype ws namet;
results := {STRING60 Le      := MATCHTEXT(Namet[1]),
            STRING60 Ri      := MATCHTEXT(Namet[2]),
            STRING30 Rel      := MATCHTEXT(relationtype) };
outfile1 := PARSE(datafile, line, progeny, results, SCAN ALL);
```

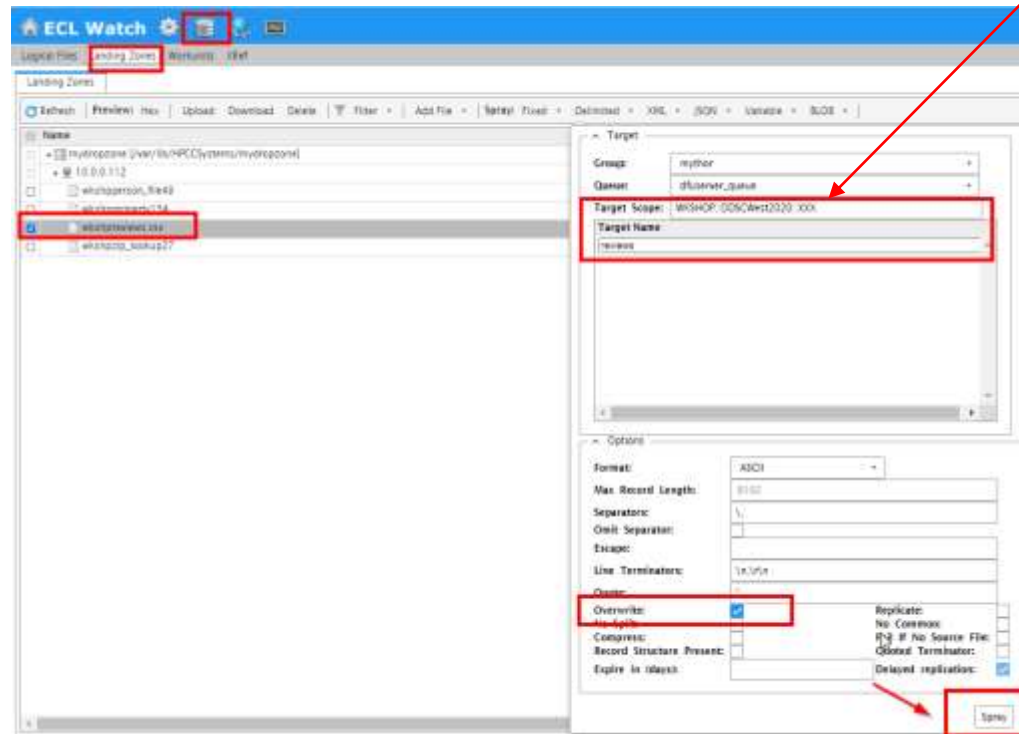
Let´s practice

(ECL and Natural Language Parsing)

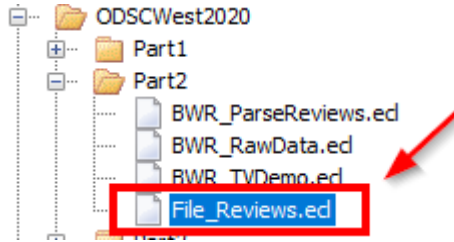
Let's spray the data

<http://3.134.27.20:8010/> (ECL Watch)

Your initials here!!!

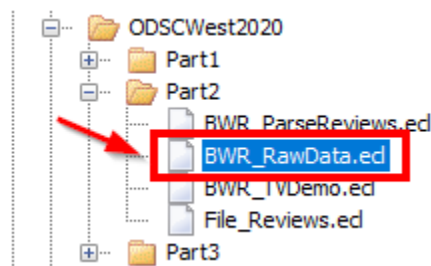


The data structure



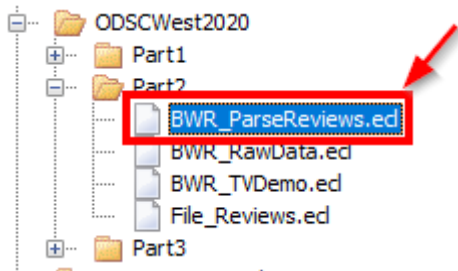
```
EXPORT File_Reviews := MODULE
  EXPORT Layout := RECORD
    UNSIGNED4 Property_id;
    UNSIGNED4 Review_id;
    UNSIGNED4 Review_date;
    UNSIGNED4 Reviewer_id;
    STRING Reviewer_name;
    STRING Review_text;
  END;
  EXPORT File := DATASET('~WKSHOP::ODSCWest2020::XXX::Reviews',
    Layout,CSV(SEPARATOR(','),HEADING(1)));
END;
```

Viewing the data (raw view)



##	property_id	review_id	review_date	reviewer_id	reviewer_name	review_text
1	7202016	38917982	2015	28943674	Bianca	Cute and cozy place. Perfect location to everything!
2	7202016	39087409	2015	32440555	Frank	Kelly has a great room in a very central location. Beautiful building , architecture an
3	7202016	39820030	2015	37722850	Ian	Very spacious apartment, and in a great neighborhood. This is the kind of apartment I
4	7202016	40813543	2015	33671805	George	Close to Seattle Center and all it has to offer - ballet, theater, museum, Space Needle
5	7202016	41986501	2015	34959538	Ming	Kelly was a great host and very accommodating in a great neighborhood. She has some gre
6	7202016	43979139	2015	1154501	Barent	Kelly was great, place was great, just what I was looking for-clean, simple, well kept
7	7202016	45265631	2015	37853266	Kevin	Kelly was great! Very nice and the neighborhood and place to stay was expected and comf
8	7202016	46749120	2015	24445447	Rick	hola all bnb erz - Just left Seattle where I had a simply fantastic time for the weeken
9	7202016	47783346	2015	249583	Todd	Kelly's place is conveniently located on a quiet street in Lower Queen Anne which is an
10	7202016	48388999	2015	38110731	Tatiana	The place was really nice, clean, and the most important aspect; it was close to everyt
11	7202016	49441269	2015	39852826	Tim	The place was really nice, clean and quiet at night.Clean Linen and Towels were provide
12	7202016	50490194	2015	384855	Tony	The listing was exactly as described! Kelly's place was wonderful and cleen. it was j
13	7202016	53862449	2015	21607838	Jason	Very welcoming and a nicer place to live in the Seattle area

Parsing the review comments



```
IMPORT $;

datafile      :=      $.File_Reviews.File;

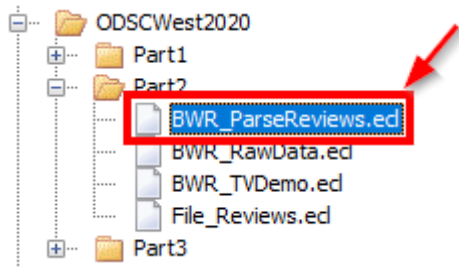
PATTERN ws      := ' ';
PATTERN alpha   := PATTERN('[a-zA-Z]');
PATTERN preposition := ['a','the','and','to','both','for','by','until','with'];
PATTERN adverb   := ['so','extremely','really','very','little','both','some','a lot of',
    'completely','more','close to','in a','kind of','right next to',
    'exactly as','just','while','when'];
PATTERN verb     := ['was','wasn\'t','was not','were','weren\'t','were not',
    'is','isn\'t','is not','are','aren\'t','are not',
    'had','hadn\'t','has','hasn\'t','has not','have','haven\'t','have not'];
    OPT(ws preposition) OPT(ws adverb);
TOKEN substantive := alpha alpha+;
PATTERN adjective := alpha alpha+ OPT(ws preposition) OPT(ws adverb) OPT(ws alpha+);
RULE compliment   := substantive ws verb ws adjective ;

results := {UNSIGNED4 prop_id      := datafile.property_id;
    STRING subst := MATCHTEXT(substantive);
    STRING verb_prep_adv:= MATCHTEXT(verb);
    STRING adjct  := MATCHTEXT(adjective));

outfile := PARSE(datafile,review_text,compliment,results);

outfile;
```

Parsing the review comments (cont.)



#	prop_id	subst	verb_prep_adv	adjct
13	7202016	Kelly	was	great
14	7202016	stay	was	expected and comfortable
15	7202016	all	was	good and mega
16	7202016	place	is	conveniently located
17	7202016	schedule	was completely	full and I
18	7202016	place	was really	nice
19	7202016	it	was close to	everything so we
20	7202016	place	was really	nice
21	7202016	Towels	were	provided and the
22	7202016	mattress	was	superb
23	7202016	Neighbourhood	is	practical with a lot of restaurants
24	7202016	Downtown	is	reachable by foot
25	7202016	Kelly	was a	fantastic host

Extract meaning with Text Vectors



Figure 1 -- 2D Word Vector Space showing select words

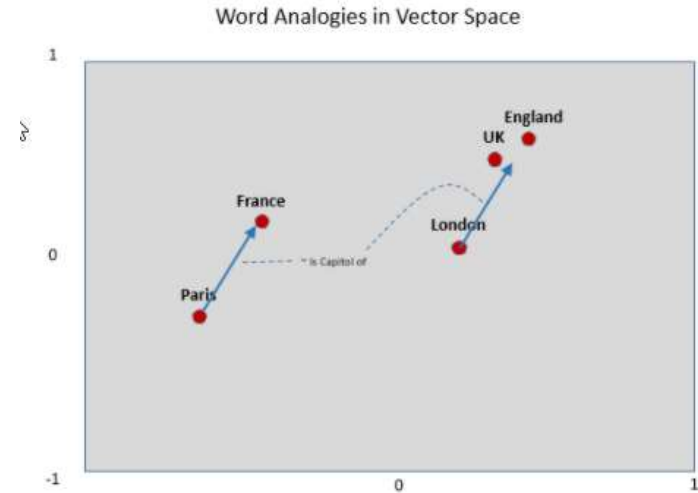


Figure 3 -- Word Analogies

Installing TextVectors bundle

```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.17134.1726]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>cd "C:\Program Files (x86)\HPCCSystems\7.6.64\clienttools\bin"

C:\Program Files (x86)\HPCCSystems\7.6.64\clienttools\bin>ec1 bundle install https://github.com/hpcc-systems/ML_Core.git --update --force
Installing bundle ML_Core version 3.2.2
Existing active version 3.2.2 is newer or same version
--force specified - updating anyway
ML_Core 3.2.2 Common definitions for Machine Learning
Installation complete
ec1 "bundle" command error 0

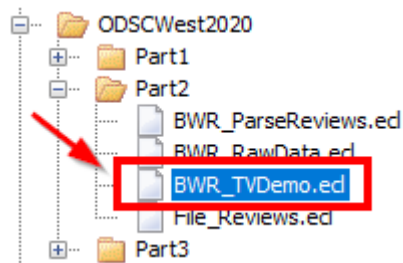
C:\Program Files (x86)\HPCCSystems\7.6.64\clienttools\bin>ec1 bundle install https://github.com/hpcc-systems/TextVectors.git --update --force
Installing bundle TextVectors version 1.0.0
Existing active version 1.0.0 is newer or same version
--force specified - updating anyway
TextVectors 1.0.0 Text Vectorization for words and sentences
Installation complete
ec1 "bundle" command error 0

C:\Program Files (x86)\HPCCSystems\7.6.64\clienttools\bin>
```

Repository

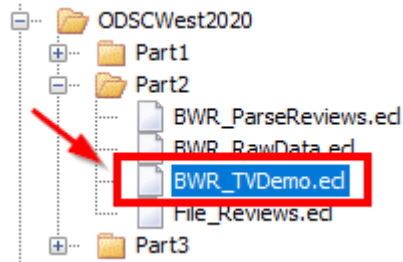
- ETL
- My Files
- reformatter-master
- eclibrary
- plugins
- bundles
 - _versions
 - DataPatterns.ed
 - LearningTrees.ed
 - ML_Core.ed
 - TextVectors.ed
 - Visualizer.ed

Training sentences



##	sentid	text
1	1	Cute and cozy place. Perfect location to everything!
2	2	Kelly has a great room in a very central location. Beautiful building , architecture and a style that we really like. I
3	3	Very spacious apartment, and in a great neighborhood. This is the kind of apartment I wish I had!Didn't really get to
4	4	Close to Seattle Center and all it has to offer - ballet, theater, museum, Space Needle, restaurants of all ilk just b.
5	5	Kelly was a great host and very accommodating in a great neighborhood. She has some great coffee and while I wasn't ar
6	6	Kelly was great, place was great, just what I was looking for-clean, simple, well kept place.5 min walk to the Seattle
7	7	Kelly was great! Very nice and the neighborhood and place to stay was expected and comfortable. Overall great and woul
8	8	hola all bnb erz - Just left Seattle where I had a simply fantastic time for the weekend , no small part because of th
9	9	Kelly's place is conveniently located on a quiet street in Lower Queen Anne which is an easy walk or bus/cab ride to B
10	10	The place was really nice, clean, and the most important aspect; it was close to everything so we moved across the cit
11	11	The place was really nice, clean and quiet at night.Clean Linen and Towels were provided and the air mattress was supe
12	12	The listing was exactly as described! Kelly's place was wonderful and cleen. it was just what we were looking for.We

TextVectors output



text	closest	similarity
	Item	Item
location is to quiet as place is to:	quiet,stay	1,0.999029278755188

text	closest	similarity
	Item	Item
neighbourhood	family,awesome,beautiful	0.9441138505935669,0.942000150680542,0.9396407008171082

text	closest	similarity
	Item	Item
the apartment was spacious	Exactly as described, easy to get in and spacious.,Wonderful place! Clean, quiet, spacious, and very comfortable! Would definitely stay again!	0.9995267987251282,0.999
the neighbourhood was great	Nice quiet neighbourhood. Room was comfortable and clean.,Everything was accurate about the listing. Great location and neighbourhood.	0.9992449283590854,0.998



Break and Questions



ECL and Machine Learning

What is Machine Learning?

The broad definition of Machine Learning (from Wikipedia) "the scientific study of algorithms and statistical models that computer systems use to perform a specific task without using explicit instructions, relying on patterns and inference instead. It is seen as a subset of artificial intelligence. "

The "learning" part of Machine Learning (or ML) has several categories, here is what we currently showcase in ECL:

Supervised - The most common type of ML. This method involves the training of the system where the recordsets along with the target output pattern is provided to the system for performing a task.

Unsupervised - This method does not involve the target output which means no training is provided to the system. The system has to learn by its own through determining and adapting according to the structural characteristics in the input patterns.

Deep - Moves into the area of the ML Neural Networks methods. Deep Learning implies multiple layers greater than two (2). Deep also implies techniques used with complex data, like video or audio analysis.

Machine Learning Basics

Supervised learning overview:

Given a set of data samples:

Record1: Field1, Field2, Field3, ..., FieldM
Record2: Field1, Field2, Field3, ..., FieldM
...
RecordN: Field1, Field2, Field3, ..., FieldM

Independent Variables

Note: The fields in the independent data are also known as “features” of the data

And a set of target values,

Record1: TargetValue
Record2: TargetValue
...
RecordN: TargetValue

Dependent Variables

Learn how to predict target values for new samples.

- The set of Independent and Dependent data is known as the **Training Set**
- The encapsulated learning is known as the **Model**
- Each model represents a **Hypothesis** regarding the relationship of the Independent to Dependent variables.
- The hallmark of machine learning is the ability to **Generalize**

Machine Learning Example

Given the following data about trees in a forest:

Height	Diameter	Altitude	Rainfall	Age
50	8	5000	12	80
56	9	4400	10	75
72	12	6500	18	60
47	10	5200	14	53

Learn a Model that approximates Age (i.e. the Dependent Variable) from Height, Diameter, Altitude, and Rainfall (i.e. the Independent Variables).

It is hard to see from the data how to construct such a model

Machine Learning will automatically construct a model that (usually) minimizes “prediction error”.

In the process, depending on the algorithm, it may also provide insight into the relationships between the independent and dependent variables (i.e. inference).

Note: We normally want to map easy to measure (i.e. “observed”) features to hard-to-measure (“unobserved”) features.

Quantitative and Qualitative Models

Supervised Machine Learning supports two major types of models:

- **Quantitative** – e.g., Determine the numerical age of a tree
- **Qualitative** – e.g. Determine the species of the tree, Determine if the tree is healthy or not.

Learning a Quantitative model is called Regression (a term with archaic origins – **don't** try to make sense of it).

Learning a Qualitative model is called Classification.

Machine Learning Algorithms

There are many different ML algorithms that all try to do the same things (Classification or Regression)

Each ML algorithm has limitations on the types of model it can produce. The space of all possible models for an algorithm is called its **Hypothesis Set**.

- **Linear models** assume that the dependent variable is a function of the independent **variables multiplied by a coefficient** (e.g. $f(\text{field1} * \text{coef1} + \text{field2} * \text{coef2} + \dots + \text{fieldM} * \text{coefM} + \text{constant})$)
- **Tree models** assume that the dependent variable can be determined by asking a hierarchical set of questions about the independent data. (e.g. is height ≥ 50 feet? If so, is rainfall ≥ 11 inches?...then age is 65).
- Some other models (e.g. Neural Nets) are difficult to explain or visualize, and are therefore not considered **Explanatory**.

Bundles

HPCC Platform Machine Learning Approaches

Experimental Bundle – ecl-ml (<https://github.com/hpcc-systems/ecl-ml>)

- Rich set of machine learning methods at various levels of quality, performance, and documentation. Currently a private repo that requires login.
- Recently added Gradient Boosted Trees, DimensionReduction

Embedded Language support for Python and R allow industry ML platforms:

- Various R packages
- Scikit-learn (Python)
- Google TensorFlow (Python)

Production Bundles

- Fully supported, validation tested, HPCC optimized, and performance profiled.
- Bundles are generally independent of platform release, and easy to install
- The Bundle installer will alert if there is a platform version dependency for a particular bundle

ML Core and Supervised Bundles

ML Core – Machine Learning Core (https://github.com/hpcc-systems/ML_Core)

- Provides the base data types and common data analysis methods.
- Also includes methods to convert between your record oriented data and the matrix form used by the ML algorithms.
- This is a dependency for all of the ML bundles.

PBblas – Parallel Block Basic Linear Algebra Subsystem (<https://github.com/hpcc-systems/PBblas>)

- Provides parallelized large-scale matrix operations tailored to the HPCC Platform.
- This is a dependency for many of the ML bundles.

LinearRegression – Ordinary least squares linear regression (<https://github.com/hpcc-systems/LinearRegression>)

- Allows multiple dependent variables (Multi-variate)
- Provides a range of analytic methods for interpreting the results
- Can be useful for testing relationship hypotheses

LogisticRegression – Logistic Regression Bundle (<https://github.com/hpcc-systems/LogisticRegression>)

- **Despite “regression” in its name, provides a linear model based classification mechanism**
- Binomial (yes/no) classification
- Multinomial (n-way) classification using Softmax.
- Allows multiple dependent variables (Multi-variate)
- Includes analytic methods.

Supervised ML Bundles (cont'd)

[SupportVectorMachines](https://github.com/hpcc-systems/SupportVectorMachines) – SVM Bundle (<https://github.com/hpcc-systems/SupportVectorMachines>)

- Classification or Regression.
- Uses the popular LibSVM library under the hood
- Appropriate for moderate sized datasets or solving many moderate sized problems in parallel.
- Includes automatic parallelized grid-search to find the best regularization parameters.

[GLM](https://github.com/hpcc-systems/GLM) – General Linear Model (<https://github.com/hpcc-systems/GLM>)

- Provides various linear methods that can be used when the assumptions of Linear or Logistic Regression **don't apply to your data**.
- Includes: binomial, quasibinomial, Poisson, quasipoisson, Gaussian, inverse Gaussian and Gamma GLM families.
- Can be readily adapted to other error distribution assumptions.
- Accepts user-defined observation weights/frequencies/priors via a weights argument.
- Includes analytics.

Supervised ML Bundles (cont'd)

[LearningTrees](https://github.com/hpcc-systems/LearningTrees) – Random Forests (<https://github.com/hpcc-systems/LearningTrees>)

- Classification or Regression
- One of the best “out-of-box” ML methods as it makes few assumptions about the nature of the data, and is resistant to overfitting. Capable of dealing with large numbers of Independent Variables.
- Creates a “forest” of diverse Decision Trees and averages the responses of the different Trees.
- Provides “Feature Importance” metrics.
- Latest version now supports Gradient Boosting Trees and Boosted Forest models.

[Generalized Neural Network \(GNN\)](https://github.com/hpcc-systems/GNN) – (<https://github.com/hpcc-systems/GNN>)

- Combines the parallel processing power of HPCC Systems with the powerful Neural Network capabilities of Keras and Tensorflow
- Each node is attached to an independent Keras / Tensorflow environment, which can contain various hardware acceleration capabilities such as Graphical Processing Units (GPUs) or Tensor Processing Units (TPUs).
- Provides a distributed environment that can parallelize all phases of Keras / Tensorflow usage.
- Provides a Tensor module, allowing users to efficiently encode, decode, and manipulate Tensors within ECL. These Tensor data sets are used as the primary data interface to the GNN functions.

Unsupervised and Natural Language ML Bundles

K-Means – (<https://github.com/hpcc-systems/KMeans>)

- Used to automatically cluster unlabeled data
- It is an implementation of K-Means algorithm, an unsupervised machine learning algorithm to automatically cluster Big Data into different groups.
- Adopts a hybrid parallelism to enable K-Means calculations on massive datasets and also with a large count of hyper-parameters. The hybrid parallelism can be realized by utilizing the flexible data distribution control and Myriad Interface of HPCC Systems.

DBSCAN – Scalable Parallel DBSCAN Clustering (<https://github.com/hpcc-systems/DBSCAN>)

- DBSCAN is Density-Based Spatial Clustering of Applications with Noise
- An unsupervised machine learning algorithm to automatically cluster the data into subclasses or groups.
- Provides a range of analytic methods for interpreting the results

TextVectors – ML for Textual Data (<https://github.com/hpcc-systems/TextVectors>) – Natural Language

- Allows for the mathematical treatment of textual information.
- Words, phrases, sentences, and paragraphs can be organized as points.
- Closeness in space implies closeness in meaning
- Vectorization allows text to be analyzed numerically and used with other ML techniques.

Learning Trees Overview

Random Forests are based on Decision Trees.

In Decision Trees, you ask a hierarchical series of questions about the data (think a flowchart) until you have asked enough questions to split your data into small groups (leaves) that have a similar result value. Now you can ask the same set of questions of a new data point, and return the answer that is representative of the leaf into which it falls.

Random Forest builds a number of separate and diverse decision trees (a decision forest) and average the results across the forest. The use of many diverse decision trees reduces overfitting since each tree is fit to a different subset of the data, and it will therefore incorporate different "noise" into its model. The aggregation across multiple trees (for the most part) cancels out the noise.

Why Learning Trees?

Widely considered to be one of the easiest algorithms to use. Why?

- It makes very little assumption about the data's distribution or its relationships
- It can handle large numbers of records and large numbers of fields
- It can handle non-linear and discontinuous relations
- It almost always works well using the default parameters and without any tuning

It scales well on HPCC Systems clusters of almost any size

It's prediction accuracy is competitive with the best state-of-the-art algorithms

Tutorial – using Learning Trees

Tutorial Contents

This tutorial example demonstrates the following:

- Installing ML
- Preparing Your Data
- Training the Model
- Assessing the Model
- Making Predictions

Installing ML and LearningTrees Bundles

1. Be sure HPCC Systems Client Tools is installed on your system. Also, you will need Git for Windows if you do not already have it installed.
2. Install HPCC Systems ML_Core
From your *clienttools/bin* folder, run in the Command Window:

```
ecl bundle install https://github.com/hpcc-systems/ML_Core.git
```

3. Install the HPCC Systems LearningTrees bundle. Run:

```
ecl bundle install https://github.com/hpcc-systems/LearningTrees.git
```

NOTE: For PC users, the ecl bundle install must be run as an Administrator. Right click on the command icon and select "Run as administrator" when you start your command window.

Easy! Now lets get your data ready to use.

Preparing Your Data

The data should contain *all numeric values*, and the *first field* of your record is an *UNSIGNED RecordId*.

1. The first step is to segregate your data into a *training* set and a *testing* set, using a random sample. (Prep01)
2. Convert your data to the form used by the ML bundles. ML requires the data in a cell-oriented matrix layout known as the NumericField. (Convert02)

```
IMPORT ML_Core;  
ML_Core.ToField(myTrainData, myTrainDataNF);
```

At this point, *myTrainDataNF* will contain your converted training data. Note that the MACRO does not return the converted data but creates an *new* in-line definition that contains it.

Preparing Your Data (More)

3. Separate the Independent fields from the Dependent fields:

```
myIndTrainDataNF := myTrainDataNF(number < 10); // Number is the field number  
myDepTrainDataNF := PROJECT(myTrainDataNF(number = 10),  
                           TRANSFORM(RECORDOF(LEFT), SELF.number := 1, SELF := LEFT));
```

Note that using the PROJECT to set the number field to 1 is not strictly necessary, but is a good practice. This indicates that it is the first field of the dependent data. Since there is only one Dependent field, we number it accordingly.

4. Do the same thing for the test data:

```
myIndTestDataNF := myTestDataNF(number < 10); // Number is the field number  
myDepTestDataNF := PROJECT(myTestDataNF(number = 10),  
                           TRANSFORM(RECORDOF(LEFT), SELF.number := 1, SELF := LEFT));
```

We are now ready to train the model and assess our results!

Training the Model

1. Define our learner (Regression and Classification Forest)
2. Train and Retrieve the Model
 - Regression (MLTutorial.Train03)
 - ✓ RegressionForest
 - ✓ GetModel/Predict
 - ✓ Accuracy Assessment
 - Classification (MLTutorial.Train04)
 - ✓ Discretize (ByRounding)
 - ✓ ClassificationForest
 - ✓ GetModel/Classify
 - ✓ Accuracy Assessment

Assessing the Model

Assessing the Model is fairly easy:

1. Create a new set of Independent field samples
2. Use the Predict FUNCTION for Regression:

```
predictedValues := myLearnerR.Predict(myModelR, myNewIndData);
```

3. Use the Classify FUNCTION for Classification:

```
predictedClasses := myLearnerC.Classify(myModelC, myNewIndData);
```

Note: For best results, your train/test data should be representative of the population whose values you will try to predict.

Using Machine Learning

Machine Learning is fun and easy

- It is simple to invoke one of our ML algorithms, learn a model and make some predictions.
- If you have some data, give it a try.
- It will likely provide good insights, and may identify some significant possibilities for adding value.

Machine Learning is also dangerous (not *just* because of robot death rays)

- There are many ways to produce inaccurate and misleading predictions.
 - **Bad questions:** yesterday's temperature -> today's stock close
 - Bad data – Missing values, incorrect values -> bad predictions
 - Wrong assumptions vis-à-vis the algorithm chosen
 - Insufficient Training Data
 - Overfitting – the model is too good but fails with new data

Always consult a qualified Data Scientist before applying your models to a real production activity.



Conclusion

You now know everything you need to know to build and test ML models against your data and to use those models to predict qualitative or quantitative values (i.e. classification or regression).

If you want to use a different ML bundle, you'll find that all the bundles operate in a very similar fashion, with minor variation.

We've utilized only the most basic aspects of the ML bundles.

And finally:

ML's conceptual simplicity is somewhat misleading. Each algorithm has its own quirks, (assumptions and constraints) that need to be taken into account in order to maximize the predictive accuracy.

Acknowledgement and Resources

A big thank you to Roger Dev whose original presentation was debuted at our annual HPCC Systems Community Day. He is also the primary author and developer of the ML bundle algorithms.

More information on ML specific bundles and their implementation can be found here:

<https://hpccsystems.com/blog?uid=3842>

Thank you! Questions?

Useful Links

- Open Source HPCC Systems Platform: hpccsystems.com
- Machine Learning Bundles: hpccsystems.com/ml
- HPCC Systems Internship Program: hpccsystems.com/intern
- Student Wiki: hpccsystems.com/student-wiki
- Available HPCC Systems Projects: hpccsystems.com/ideas-list
- Getting Started: hpccsystems.com/getting-started
- Online Training: learn.lexisnexus.com/hpcc
- Download: hpccsystems.com/download
- GitHub portal: github.com/hpcc-systems
- Community Support Forums: hpccsystems.com/forums



Join our Community

Help us make HPCC Systems better. Register on our community portal.