Definitive HPCC Systems

Day 4 Workshop:
ECL Cookbook and Working with XML and JSON

Richard Taylor/Bob Foreman
Senior Software Engineers

LexisNexis RISK Solutions

Workshop Agenda

This workshop are based on the new book by Richard Taylor, and our core ECL Introduction to ECL and Advanced ECL training courses:

Definitive HPCC Systems

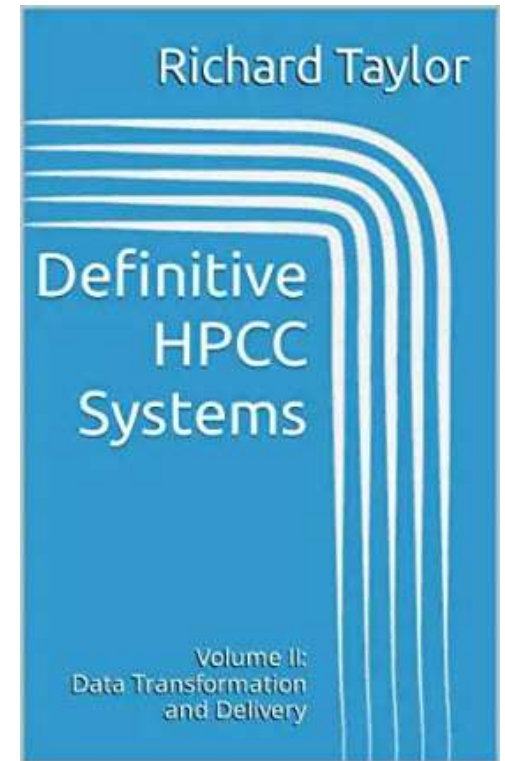**Volume II: Data Transformation and Delivery**

Days 1 and 2: Chapters 1-3

Day 3: Chapter 4

Day 4: Chapter 5 (Selections from 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

Introduction: The ECL Cookbook

- In our first two hours, we have examined the complete product life cycle, from data ingest, through data transformation, and product delivery -- the standard process that every data product must go through no matter what platform is used -- and been introduced to how that process is done on the HPCC Systems platform.

- This workshop examines many more ECL tools and demonstrates aspects of ECL that have not yet been covered.

- You may use these tools, or modify them, or extrapolate from their specific use case to something similar but not quite the same in your projects.

File Tips

- The HPCC Systems platform is all about working with Big Data, and the choice was made early on to do that using standard file system datasets. This section is about doing things with files that are not simply "straight out of the box" tasks.

- XML and JSON file formats have been industry standards for many years. Advantages are their built-in metadata information which makes them easy to read and interpret.

- Disadvantages are their file sizes, since field information is stored and duplicated in every record.

- Let's look at the ECL way to generate these formats:

File Tips – Generating:

Creating XML and JSON Files

XMLandJSONFiles.ecl

- Simple XML and JSON
- XML Attributes
- XML row and file tags
- JSON row and file tags

BWR_CustomXMLandJSON.ecl

- Generating custom XML and JSON rows

File Tips – Automatic File Cleanup

- As you work with your HPCC Systems platform you will invariably create a lot of files.

- And it's typical that, as you migrate from project to project, some of those files will no longer be relevant to your production work.

- That means you will periodically need to eliminate files that are no longer needed, and one major determinant in whether or not a file is still needed is found by asking the question, "How long ago was this file last accessed?"

- If the answer to that question is "too long" then it's a candidate for removal. And the code in this section covers how you can do that automatically.

- The *AutoFileCleanup* FUNCTION is designed to clean up logical files by either setting their EXPIRE option or simply deleting them.

AutoFileCleanup.ecl

File Tips – Automatic File Cleanup: Usage

The *AutoFileCleanup* function takes three parameters. The first is the *WhichMethod* parameter that specifies which of the four supported methods the caller is choosing to use:

> 1 – Set EXPIRE for all matching files matching *FilePattern* parameter
> 2 – Same as 1, but only delete files that haven't been accessed (*Expiry* parameter)
> 3 – Immediate delete of all files matching *FilePattern* parameter
> 4 - Delete of all files matching *FilePattern* parameter that haven't been accessed (*Expiry* parameter)

BWR_DeleteXMLJSONSamples.ecl

Files and SuperFiles:

✓ A File is a single logical entity comprised of multiple physical parts

Each logical file in the DFU has component physical files on each of the disks in the cluster to which it was written

✓ A SuperFile is a single logical entity comprised of one to many logical files. A logical file in a superfile is referred to as a *subfile*.

Creating a SuperFile:

✓A SuperFile must first be explicitly created

STD.File.CreateSuperFile()

✓NOTE: ALL SuperFile functions are contained in the FileServices library (exported as *File*).

IMPORT STD;

CreateSuperFile

✓ **CreateSuperFile(***superfile* [, *sequentialflag*]**)**

    ✓ *Superfile* - A null-terminated string containing the logical name of the superfile.

    ✓ *Sequentialflag* - A boolean value indicating whether the sub-files must be sequentially numbered. If omitted, the default is FALSE.

        The **CreateSuperFile** function creates an empty *superfile*.

STD.File.CreateSuperFile('~CLASS::XX::IN::SF1');

NOTE: Does NOT use or require a transaction frame.

SEQUENTIAL Action

✓ [*name* :=] **SEQUENTIAL(** *actionlist* **)**

   ✓ *actionlist* – A comma-delimited list of the actions to execute in order. These may be ECL actions or external actions

   The **SEQUENTIAL** action executes the items in the *actionlist* in the order in which they appear in the *actionlist*. This is useful when a subsequent action requires the output of a precedent action.

R := {fname, lname,UNSIGNED8 fpos {virtual(fileposition)}},;

A := OUTPUT(A_People, R, '//hold01/fred.out');

D := DATASET('//hold01/fred.out', R, THOR);

B := BUILDINDEX(D,{fname, lname, UNSIGNED8 fpos});

SEQUENTIAL(A,B);   //do A first and then B, not both at once

SuperFile Transactions

✓ Once created, changes to the SuperFile are encased in a transaction, which must execute sequentially

```
SEQUENTIAL(
        STD.File.StartSuperFileTransaction()

        //stuff happens here

        STD.File.FinishSuperFileTransaction()
                );
```

StartSuperFileTransaction

✓ **StartSuperFileTransaction()**

   The **StartSuperFileTransaction** function begins a transaction frame.

```
SEQUENTIAL(
        STD.File.StartSuperFileTransaction()

                //stuff happens here

        STD.File.FinishSuperFileTransaction()

                );
```

FinishSuperFiletransaction

✓ **FinishSuperFileTransaction()**

The **FinshSuperFileTransaction** function ends a transaction frame.

SEQUENTIAL(
        STD.File.StartSuperFileTransaction()

        //stuff happens here

        STD.File.FinishSuperFileTransaction()

                );

AddSuperFile

✓**AddSuperFile (**_superfile, subfile_ **)**

   ✓_superfile_ - A null-terminated string containing the logical name of the superfile.

   ✓_subfile_ - A null-terminated string containing the logical name of the sub-file.

   The **AddSuperFile** function adds the _subfile_ to the list of files comprising the _superfile_. All _subfiles_ in the _superfile_ must have exactly the same structure type and format.

```
SEQUENTIAL(
      STD.File.StartSuperFileTransaction() ,
      STD.File.AddSuperFile ('SuperFilename', 'SubFilename') ,
      STD.File.FinishSuperFileTransaction() );
```

RemoveSuperFile

✓ **RemoveSuperFile (***superfile***,** *subfile* **)**

    ✓ *superfile* - A null-terminated string containing the logical name of the superfile.

    ✓ *subfile* - A null-terminated string containing the logical name of the sub-file.

        The **RemoveSuperFile** function removes the *subfile* from the list of files comprising the *superfile*.

```
SEQUENTIAL(STD.File.StartSuperFileTransaction() ,
           STD.File.RemoveSuperFile ('SuperFilename', SubFilename') ,
           STD.File.FinishSuperFileTransaction()

           );
```

ClearSuperFile

✓ **ClearSuperFile (***superfile* **)**

   ✓ *superfile* - A null-terminated string containing the logical name of the superfile.

   The **ClearSuperFile** function removes all *subfiles* from the list of files comprising the *superfile*.

```
SEQUENTIAL(
        STD.File.StartSuperFileTransaction() ,

        STD.File.ClearSuperFile ('SuperFilename') ,

        STD.File.FinishSuperFileTransaction()

                );
```

Lab Exercises:

18 – Create superfiles

19 – Add sub files to superfile

20 – Define superfiles

21 – Add more sub files to superfile
(Daily Update)

22 – Weekly consolidation

23 – Add more sub files to superfile
(Daily Update)

24 – Consolidate weekly and daily to
new base file
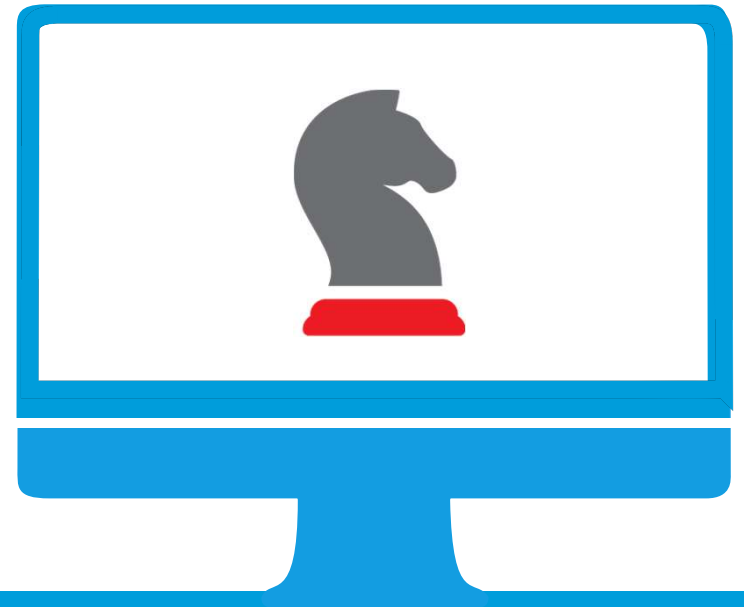
Exercise 1a:

Before:

				Logical Name	Owner	Description	Cluster	Records	Size
☐		ⓘ		ecltraining::in::namephonesupd				5,335,364	496,188,852
☐				ecltraining::in::namephonesupd1				1,507,292	140,178,156
☐				ecltraining::in::namephonesupd2				1,158,027	107,696,511
☐				ecltraining::in::namephonesupd3				982,446	91,367,478
☐				ecltraining::in::namephonesupd4				879,319	81,776,667
☐				ecltraining::in::namephonesupd5				808,280	75,170,040

After:

ⓘ		Logical Name	Owner	Description	Cluster	Records	Size
		class::bmf::sf::alldata					
		class::bmf::sf::daily					
		class::bmf::sf::weekly					
		ecltraining::in::namephonesupd				5,335,364	496,188,852
		ecltraining::in::namephonesupd1				1,507,292	140,178,156
		ecltraining::in::namephonesupd2				1,158,027	107,696,511
		ecltraining::in::namephonesupd3				982,446	91,367,478
		ecltraining::in::namephonesupd4				879,319	81,776,667
		ecltraining::in::namephonesupd5				808,280	75,170,040

Exercise 1h:

Before:

ⓘ 🖥	Logical Name	Owner	Description	Cluster	Records	Size
🖥	class::bmf::sf::alldata					
🖥	class::bmf::sf::daily					
🖥	class::bmf::sf::weekly					
	ecltraining::in::namephonesupd				5,335,364	496,188,852
	ecltraining::in::namephonesupd1				1,507,292	140,178,156
	ecltraining::in::namephonesupd2				1,158,027	107,696,511
	ecltraining::in::namephonesupd3				982,446	91,367,478
	ecltraining::in::namephonesupd4				879,319	81,776,667
	ecltraining::in::namephonesupd5				808,280	75,170,040

After:

ⓘ 🖥	Logical Name	Owner	Description	Cluster	Records	Size
🖥	class::bmf::sf::alldata				1,507,292	140,178,156
🖥	class::bmf::sf::daily					
🖥	class::bmf::sf::weekly					
	ecltraining::in::namephonesupd				5,335,364	496,188,852
	ecltraining::in::namephonesupd1				1,507,292	140,178,156
	ecltraining::in::namephonesupd2				1,158,027	107,696,511
	ecltraining::in::namephonesupd3				982,446	91,367,478
	ecltraining::in::namephonesupd4				879,319	81,776,667
	ecltraining::in::namephonesupd5				808,280	75,170,040

Exercise 1d:

Before:

ⓘ	☰	Logical Name	Owner	Description	Cluster	Records	Size
	☰	class::bmf::sf::alldata				1,507,292	140,178,156
	☰	class::bmf::sf::daily					
	☰	class::bmf::sf::weekly					
		ecltraining::in::namephonesupd				5,335,364	496,188,852
		ecltraining::in::namephonesupd1				1,507,292	140,178,156
		ecltraining::in::namephonesupd2				1,158,027	107,696,511
		ecltraining::in::namephonesupd3				982,446	91,367,478
		ecltraining::in::namephonesupd4				879,319	81,776,667
		ecltraining::in::namephonesupd5				808,280	75,170,040

After:

ⓘ	☰	Logical Name	Owner	Description	Cluster	Records	Size
	☰	class::bmf::sf::alldata				3,647,765	339,242,145
	☰	class::bmf::sf::daily				2,140,473	199,063,989
	☰	class::bmf::sf::weekly					
		ecltraining::in::namephonesupd				5,335,364	496,188,852
		ecltraining::in::namephonesupd1				1,507,292	140,178,156
		ecltraining::in::namephonesupd2				1,158,027	107,696,511
		ecltraining::in::namephonesupd3				982,446	91,367,478
		ecltraining::in::namephonesupd4				879,319	81,776,667
		ecltraining::in::namephonesupd5				808,280	75,170,040

Exercise 1e:

Before:

ⓘ		Logical Name	Owner	Description	Cluster	Records	Size
	🗐	class::bmf::sf::alldata				3,647,765	339,242,145
	🗐	class::bmf::sf::daily				2,140,473	199,063,989
	🗐	class::bmf::sf::weekly					
		ecltraining::in::namephonesupd				5,335,364	496,188,852
		ecltraining::in::namephonesupd1				1,507,292	140,178,156
		ecltraining::in::namephonesupd2				1,158,027	107,696,511
		ecltraining::in::namephonesupd3				982,446	91,367,478
		ecltraining::in::namephonesupd4				879,319	81,776,667
		ecltraining::in::namephonesupd5				808,280	75,170,040

After:

		Logical Name	Owner	Description	Cluster	Records	Size
		class::bmf::out::weeklyrollup1	Bob			2,140,473	199,063,989
	🗐	class::bmf::sf::alldata				3,647,765	339,242,145
	🗐	class::bmf::sf::daily					
	🗐	class::bmf::sf::weekly	Bob			2,140,473	199,063,989
		ecltraining::in::namephonesupd				5,335,364	496,188,852
		ecltraining::in::namephonesupd1				1,507,292	140,178,156
		ecltraining::in::namephonesupd2				1,158,027	107,696,511
		ecltraining::in::namephonesupd3				982,446	91,367,478
		ecltraining::in::namephonesupd4				879,319	81,776,667
		ecltraining::in::namephonesupd5				808,280	75,170,040

Exercise 1f:

Before:

	class::bmf::out::weeklyrollup1	Bob		2,140,473	199,063,989
📄	class::bmf::sf::alldata			3,647,765	339,242,145
📄	class::bmf::sf::daily				
📄	class::bmf::sf::weekly	Bob		2,140,473	199,063,989
	ecltraining::in::namephonesupd			5,335,364	496,188,852
	ecltraining::in::namephonesupd1			1,507,292	140,178,156
	ecltraining::in::namephonesupd2			1,158,027	107,696,511
	ecltraining::in::namephonesupd3			982,446	91,367,478
	ecltraining::in::namephonesupd4			879,319	81,776,667
	ecltraining::in::namephonesupd5			808,280	75,170,040

After:

	class::bmf::out::weeklyrollup1	Bob		2,140,473	199,063,989
📄	class::bmf::sf::alldata			5,335,364	496,188,852
📄	class::bmf::sf::daily			1,687,599	156,946,707
📄	class::bmf::sf::weekly	Bob		2,140,473	199,063,989
	ecltraining::in::namephonesupd			5,335,364	496,188,852
	ecltraining::in::namephonesupd1			1,507,292	140,178,156
	ecltraining::in::namephonesupd2			1,158,027	107,696,511
	ecltraining::in::namephonesupd3			982,446	91,367,478
	ecltraining::in::namephonesupd4			879,319	81,776,667
	ecltraining::in::namephonesupd5			808,280	75,170,040

Exercise 1g:

Before:

	🔒	📄	*i*	Logical Name	Owner	Super Owner	Description	Cluster	Records	Size
☐				class::bmf::sf::alldata					5,335,364	496,188,852
☐				class::bmf::sf::daily	bforeman	class::bmf::sf::alldata			1,687,599	156,946,707
☐				class::bmf::out::weeklyrollup1	Bob	class::bmf::sf::weekly		mythor	2,140,473	199,063,989
☐				class::bmf::sf::weekly	Bob	class::bmf::sf::alldata			2,140,473	199,063,989

After:

	🔒	📄	*i*	Logical Name	Owner	Super Owner	Description	Cluster	Records	Size
☐				class::bmf::sf::alldata					5,335,364	496,188,852
☐				class::bmf::sf::weekly		class::bmf::sf::alldata				0
☐				class::bmf::sf::daily		class::bmf::sf::alldata				0
☐				class::bmf::sf::newbaserollup1	Bob	class::bmf::sf::alldata		mythor	5,335,364	496,188,852
☐				class::bmf::out::weeklyrollup1	Bob			mythor	2,140,473	199,063,989

Working with XML/JSON

Spraying and Defining

RECORD for XML/JSON:

name := **RECORD** [(*baserec*)] [, **MAXLENGTH(** *length*)] [, **LOCALE(**
locale)]

fields

END;

- ✓ *name* - The name of the RECORD structure.
- ✓ *baserec* - Optional. The name of a RECORD structure from which to inherit all fields. Any RECORD structure that inherits the *baserec* fields in this manner becomes compatible with any TRANSFORM function defined to take a parameter of *baserec* type (the extra *fields* will, of course, be lost).
- ✓ **MAXLENGTH -** Optional. Maximum characters in the RECORD structure or field. On the RECORD structure, it overrides any MAXLENGTH on a field definition, which overrides any MAXLENGTH specified in the TYPE structure if the *datatype* names an alien data type. This is typically used to define the maximum length of variable-length records. If omitted, the default is 4096 bytes.
- ✓ **LOCALE** - Optional. Specifies the Unicode *locale* for any UNICODE fields.
- ✓ *fields* – Field declarations.

Field Definitions

datatype identifier **[{** ***modifier*** **}] [:=** *defaultvalue***];**
identifier **:=** *defaultvalue* **;**
defaultvalue **;**
sourcefield **;**
recstruct **[** *identifier* **] ;**
sourcedataset **;**
childdataset identifier **[{** ***modifier*** **}];**

- ✓ *datatype* - The value type of the data field.
- ✓ *identifier* - The name of the field.
- ✓ *modifier* - Optional. One of the keywords listed in the **Field Modifiers**.
- ✓ *defaultvalue* - Optional. An expression defining the source of the data.
- ✓ *sourcefield* - The name of a previously defined data field, which implicitly provides the *datatype*, *identifier*, and *defaultvalue* for the new field—all inherited from the existing field.
- ✓ *recstruct* - The name of a previously defined RECORD structure.
- ✓ *sourcedataset* - The name of a previously defined DATASET or derived recordset definition. See the **Field Inheritance** section in the LRM.
- ✓ *childdataset* - A child DATASET declaration.

Field Modifiers

{ MAXLENGTH(*length*) }

{ MAXCOUNT(*records*) }

{ XPATH(*'tag'*) }

{ VIRTUAL(fileposition) }

{ VIRTUAL(localfileposition) }

XPATH Support

node[qualifier] / node[qualifier] …

node Can contain wildcards.

qualifier Can be a node or attribute, or a simple single expression of equality, inequality, or numeric or alphanumeric comparisons, or node index values. No functions or inline arithmetic, etc. are supported. String comparison is indicated when the right hand side of the expression is quoted.

These operators are valid for comparisons: <, <=, >, >=, =, !=

An example of a supported xpath:

/a/*/c*/*d/e[@attr]/f[child]/g[@attr="x"]/h[child>="5"]/i[@x!="2"]/j

XPATH Examples:

```
r := RECORD
 STRING code{xpath('@code')};
 STRING description{xpath('@description')};
 STRING zone{xpath('@zone')};
END;


layout_person := RECORD
 UNSIGNED8 id;
 STRING15 firstname;
 STRING25 lastname;
 DATASET(layout_accts) childaccts{xpath('childaccts/Row'),maxCount(120)};
 END;
```

DATASET for XML/JSON:

name := DATASET(*file, recorddef,* THOR *thoroptions*);
name := DATASET(*file, recorddef,* CSV [(*csvoptions*)]);
***name* := DATASET(*file, recorddef,* XML(*path*));**
***name* := DATASET(*file, recorddef,* JSON(*path*));**
name := DATASET(*file, recorddef,* PIPE(*command*));

✓ *name* – The definition name by which the file is subsequently referenced.
✓ *file* – A string constant containing the logical filename.
✓ *recorddef* – The RECORD structure of the dataset.
✓ *thoroptions* – Options specific to THOR/FLAT datasets.
✓ *csvoptions* – Options specific to CSV datasets.
✓ *path* – The XPATH to the XML or JSON row tag.
✓ *command* – The program call that will produce the dataset.

DATASET introduces a new data file into the system with the specified *recorddef* layout.

XML DATASET Example:

```
<Dataset>
<area>
<code>201</code>
<description>PA Pennsylvania</description>
<zone>Eastern Time Zone</zone>
</area>
</Dataset>


r := RECORD
  INTEGER2 code;
  STRING110 description;
  STRING42 zone;
END;


d := DATASET('~CLASS::XXX::IN::timezones',r,XML('Dataset/area'));
```
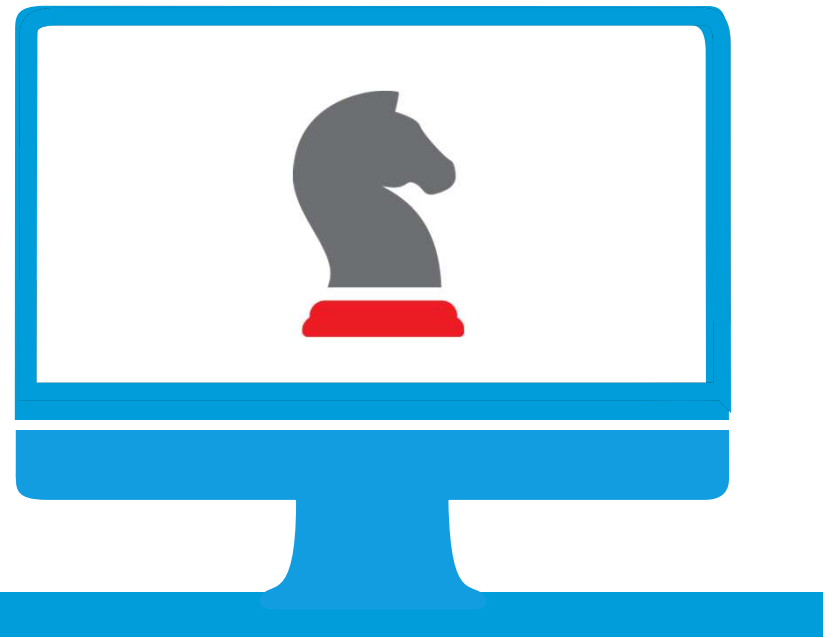
Lab Exercises:

25 – Spray Simple XML file
26 – Defining Simple XML file
27 – Spray Complex XML file
28 – Defining Complex XML file
29 – Spray Nested Child XML file
30 – Defining Nested Child XML file
31 – Spray Simple JSON file
32 – Defining Simple JSON file

End of Day 4 Workshop:

More to come!!

Thanks for attending!

✓ **Download it all at:**
https://github.com/hpcc-systems/NCF2023

✓ **Contact us:**   **robert.foreman@lexisnexisrisk.com**
                                 richard.taylor@lexisnexisrisk.com

HPCC SYSTEMS