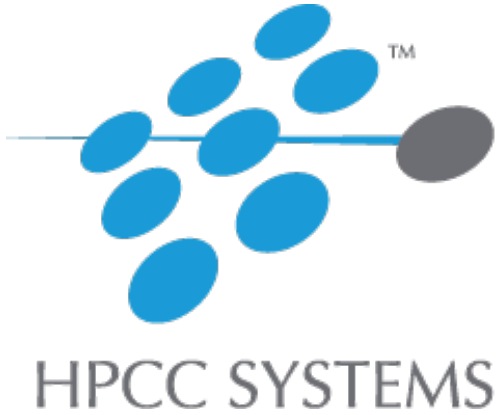


Advanced ECL (Part 2) – Superfiles, Working with XML and JSON, NLP and XML Parsing

Lesson 1

Course Introduction – Working with Superfiles – Spraying the Course Data

Introduction:

- ✓ Working with SuperFiles
- ✓ Working with XML
- ✓ Parsing Text and XML

Goals:

- ✓ How to define and work with SuperFiles and SuperKeys.
- ✓ How to spray and define XML data files
- ✓ How to work with “hybrid” XML data
- ✓ How to extract data from free form text

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 multiple logical *Files*.

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*).

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.

```
IMPORT STD;
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

```
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, maintenance changes to the superfile are encased in a transaction, which must execute sequentially:

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

            //AddSuperFile(), RemoveSuperfile(), ClearSuperFile()
            //SwapSuperFile(), and ReplaceSuperFile() are valid here

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

StartSuperFileTransaction

StartSuperFileTransaction()

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

            //stuff happens here

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

FinishSuperFiletransaction

FinishSuperFileTransaction()

```
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.

```
IMPORT STD;

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.

```
IMPORT STD;
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.

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

                STD.File.ClearSuperFile ('SuperFilename') ,

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

Lesson 1 Exercise:

1. Download and extract training data located on main course page (Superfiles.ZIP)
2. Upload extracted files to your target landing zone
3. Spray five (5) files to THOR – Spray Fixed:

Source File Name	Record Length	Destination Label
Online namephonesupd1	93	ecltraining::in::namephonesupd1
Online namephonesupd2	93	ecltraining::in::namephonesupd2
Online namephonesupd3	93	ecltraining::in::namephonesupd3
Online namephonesupd4	93	ecltraining::in::namephonesupd4
Online namephonesupd5	93	ecltraining::in::namephonesupd5

We sprayed in the Intro to ECL class – review if needed.

Recommended Spray Options:

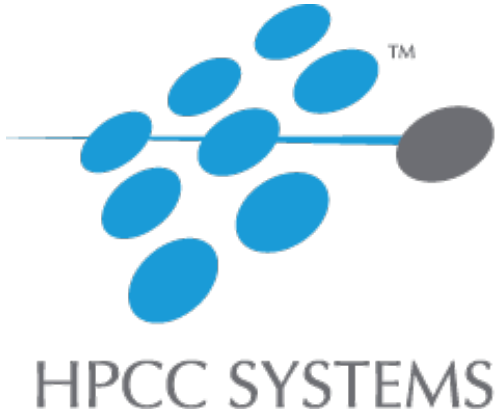
Overwrite and Replicate ON (checked)

No Split and Compress OFF (unchecked)

Lesson Completed!

Proceed to Lesson 2: Superfile Lab Exercises

HPCC SYSTEMS®



Advanced ECL (Part 2) – Superfiles, Working with XML and JSON, NLP and XML Parsing

Lesson 2

Creating Superfiles

LexisNexis® RISK SOLUTIONS

RELX Group

Lesson 2 Exercise:

Before:

Logical Name	Owner	Descr	Cluster	Records	Size
ecltraining::in::namephone...			mythor	150,731	14,017,983
ecltraining::in::namephone...			mythor	115,803	10,769,679
ecltraining::in::namephone...			mythor	98,247	9,136,971
ecltraining::in::namephone...			mythor	87,933	8,177,769
ecltraining::in::namephone...			mythor	80,829	7,517,097

After:

Logical Name	Owner	Descr	Cluster	Records	Size	Parts
online::bmf::sf::alldata						
online::bmf::sf::weekly						
online::bmf::sf::daily						

Lesson 2 Lab Exercise:

Exercise Spec:
Create three (3) superfiles. Use the *CreateSuperFile* function that was introduced in the last lesson. Download the *Standard Library Reference* PDF on the HPCC Systems web site for more detailed syntax information regarding all Superfile function support.

Requirements:

1. The superfile names to create for this exercise must start with *~ONLINE::* followed by *your initials*, followed by *::SF::filename* as in this example:

   ~ONLINE::XXX::SF::AllData

2. The superfile names to create for this exercise are:

   ~ONLINE::XXX::SF::AllData
   ~ONLINE::XXX::SF::Weekly
   ~ONLINE::XXX::SF::Daily

3. Save your code in a file named: **BWR_Create_SF**

4. In addition to the above requirements, create a *MODULE* structure in a new ECL file named **File_AllData** to define filename and other constants. This allows you to define them once and use the defined values in the multiple places that working with superfiles will require.

Result Comparison
Execute (Submit) the job and check that the result looks good in the ECL Watch.

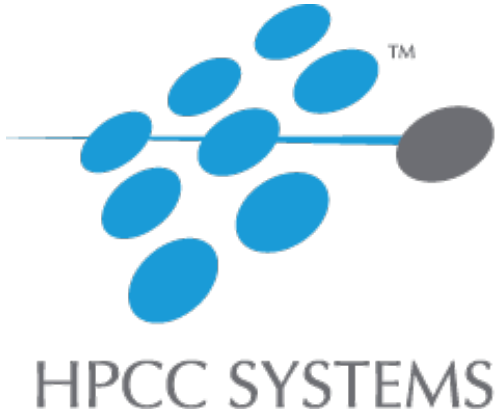
Lab Exercise Solution

```
//File_AllData

EXPORT File_AllData := MODULE
 EXPORT AllDataSF := '~ONLINE::XXX::SF::Alldata';
 EXPORT WeeklySF  := '~ONLINE::XXX::SF::Weekly';
 EXPORT DailySF   := '~ONLINE::XXX::SF::Daily';
END;


//BWR_Create_SF

IMPORT $,STD;
STD.File.CreateSuperFile($.File_AllData.AllDataSF);
STD.File.CreateSuperFile($.File_AllData.WeeklySF);
STD.File.CreateSuperFile($.File_AllData.DailySF);
```

Lesson Completed!

Proceed to Lesson 3: Superfile Lab Exercise

HPCC SYSTEMS®



Advanced ECL (Part 2) – Superfiles, Working with XML and JSON, NLP and XML Parsing

Lesson 3

Adding Subfiles to Superfiles

LexisNexis® RISK SOLUTIONS

RELX Group

Lesson 3 Exercise:

Before:

Logical Name	Owner	Super Owner	Description	Cluster	Records	Size	Parts
ecltraining::in::namephonesupd1				mythor	150,731	14,017,983	4
ecltraining::in::namephonesupd4				mythor	87,933	8,177,769	4
ecltraining::in::namephonesupd5				mythor	80,829	7,517,097	4
ecltraining::in::namephonesupd2				mythor	115,803	10,769,679	4
ecltraining::in::namephonesupd3				mythor	98,247	9,136,971	4
online::bmf::sf::alldata						0	
online::bmf::sf::weekly						0	
online::bmf::sf::daily						0	

After:

Logical Name ▲	Owner	Super Owner	Description	Cluster	Records	Size
online::bmf::sf::alldata					150,731	14,017,983
online::bmf::sf::daily		online::bmf::sf::alldata				0
online::bmf::sf::weekly		online::bmf::sf::alldata				0

Lab Exercise Lesson 4:

Exercise Spec:
Add three (3) sub-files to the **AllDataSF** superfile. Use the **AddSuperFile** function and enclose it in a transaction frame as discussed in Lesson 1.

Requirements:
1. The sub-files to add are:
    $.File_AllData.WeeklySF (aka *~ONLINE::BMF::SF::Weekly*)
    $.File_AllData.DailySF (aka *~ONLINE::BMF::SF::Daily*)
    $.File_AllData.Base1 (aka *~ecltraining::in::namephonesupd1*)
2. Write your ECL code into a new file named: **BWR_Add_SF1**

3. Add the named Base1 file (~ecltraining::in::namephonesupd1) to your previously defined MODULE structure (**File_AllData**) defining the filename constants. This gives you one place to update your code if/when the name of the base dataset changes.

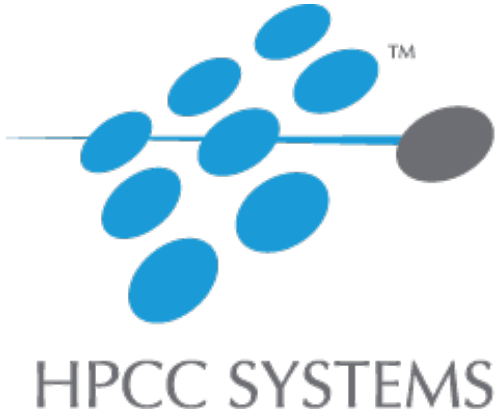
Result Comparison
Execute (submit) the job and check that the result looks good in ECL Watch.

Lab Exercise Solution

```
//BWR_Add_SF1

IMPORT $,STD;
SEQUENTIAL
 (STD.File.StartSuperFileTransaction(),
  STD.File.AddSuperFile($.File_AllData.AllDataSF,$.File_AllData.WeeklySF),
  STD.File.AddSuperFile($.File_AllData.AllDataSF,$.File_AllData.DailySF),
  STD.File.AddSuperFile($.File_AllData.AllDataSF,$.File_AllData.Base1),
  STD.File.FinishSuperFileTransaction()
 );



//File_AllData update
EXPORT File_AllData := MODULE
 EXPORT AllDataSF := '~ONLINE::BMF::SF::Alldata';
 EXPORT WeeklySF  := '~ONLINE::BMF::SF::Weekly';
 EXPORT DailySF   := '~ONLINE::BMF::SF::Daily';

 EXPORT Base1     := '~ecltraining::in::namephonesupd1';
END;
```

Lesson Completed!

Proceed to Lesson 4: Superfile Lab Exercise



Advanced ECL (Part 2) – Superfiles, Working with XML and JSON, NLP and XML Parsing

Lesson 4

Defining Superfiles

LexisNexis® RISK SOLUTIONS

RELX Group

Lesson 4 Lab Exercise:

Exercise Spec:
Create the necessary DATASET definitions so that the three (3) superfiles created in *Lesson 2* may be queried.
Requirements:
1. The RECORD fields to define are:

4-byte unsigned integer	record identifier
4-byte unsigned integer	foreign key
10-character string	home phone
10-character string	cell phone
20-character string	first name
20-character string	middle name
20-character string	last name
5-character string	name suffix

2. Add the RECORD and DATASETs to the existing **File_AllData** MODULE structure to define the DATASETs. Make sure to share your RECORD definition. Name each DATASET **AllDataDS**, **WeeklyDS**, and **DailyDS** respectively.

Result Comparison
Query the dataset, using this query in a Builder or BWR window:

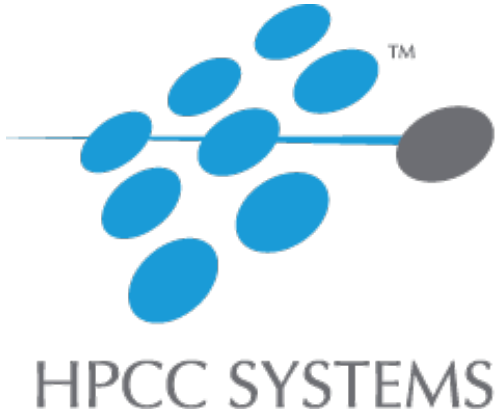
```
IMPORT TrainingAdvECL;
COUNT(TrainingAdvECL.File_AllData.AllDataDS); //should be 150731
```

Lesson 4 Lab Exercise Solution

```
EXPORT File_AllData := MODULE
 EXPORT AllDataSF := '~ONLINE::XXX::SF::Alldata';
 EXPORT WeeklySF  := '~ONLINE::XXX::SF::Weekly';
 EXPORT DailySF   := '~ONLINE::XXX::SF::Daily';
 EXPORT Base1     := '~ecltraining::in::namephonesupd1';
// Lesson 4
 SHARED Rec := RECORD
  UNSIGNED4 recid;
  UNSIGNED4 foreignkey;
  STRING10  homephone;
  STRING10  cellphone;
  STRING20  fname;
  STRING20  mname;
  STRING20  lname;
  STRING5   name_suffix;
 END;
 EXPORT AllDataDS := DATASET(AllDataSF,Rec,THOR);
 EXPORT WeeklyDS  := DATASET(WeeklySF,Rec,THOR);
 EXPORT DailyDS   := DATASET(DailySF,Rec,THOR);

END;
```

Lesson Completed!

Proceed to Lesson 5:
More Superfile Lab Exercises



Advanced ECL (Part 2) – Superfiles, Working with XML and JSON, NLP and XML Parsing

Lesson 5

Adding More Subfiles to Existing Superfiles

LexisNexis® RISK SOLUTIONS

RELX Group

Lesson 5 – Adding to “Daily”:

Before:

Logical Name	Owner	Super Owner	Description	Cluster	Records	Size
ecltraining::in::namephonesupd1		online::bmf::sf::alldata		mythor	150,731	14,017,983
ecltraining::in::namephonesupd2				mythor	115,803	10,769,679
ecltraining::in::namephonesupd3				mythor	98,247	9,136,971
ecltraining::in::namephonesupd4				mythor	87,933	8,177,769
ecltraining::in::namephonesupd5				mythor	80,829	7,517,097
online::bmf::sf::alldata					150,731	14,017,983
online::bmf::sf::daily		online::bmf::sf::alldata				0
online::bmf::sf::weekly		online::bmf::sf::alldata				0

After:

Logical Name	Owner	Super Owner	Description	Cluster	Records	Size
ecltraining::in::namephonesupd1		online::bmf::sf::alldata		mythor	150,731	14,017,983
ecltraining::in::namephonesupd2		online::bmf::sf::daily		mythor	115,803	10,769,679
ecltraining::in::namephonesupd3		online::bmf::sf::daily		mythor	98,247	9,136,971
ecltraining::in::namephonesupd4				mythor	87,933	8,177,769
ecltraining::in::namephonesupd5				mythor	80,829	7,517,097
online::bmf::sf::alldata					364,781	33,924,633
online::bmf::sf::daily		online::bmf::sf::alldata			214,050	19,906,650
online::bmf::sf::weekly		online::bmf::sf::alldata				0

Lesson 5 Lab Exercise:

Exercise Spec:
Add two (2) sub-files to the “Daily” superfile. Use the same coding technique that was introduced in *Lesson 3*.

Requirements:
1. The sub-files to add are:
    ~ecltraining::IN::namephonesupd2
    ~ecltraining::IN::namephonesupd3
2. Save your code in a file named **BWR_Add_SF2**.

Best Practices Hint
Since these files are daily update files, their names are one-time use and do not need to be added to your previously defined **File_AllData** MODULE structure defining the filename constants.

Result Comparison
Query the dataset, using this query:

```
IMPORT TrainingAdvECL;
COUNT(TrainingAdvECL.File_AllData.AllDataDS);
```

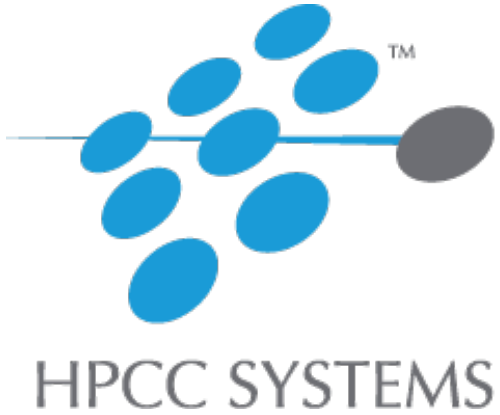
You'll find that the number returned is now larger than the previous query, since the superfile now contains three additional sub-files.

Lesson 5 Lab Exercise Solution:

```
//BWR_Add_SF2

IMPORT $,STD;
Daily := $.File_AllData.DailySF;
SEQUENTIAL(
 STD.File.StartSuperFileTransaction(),
 STD.File.AddSuperFile(Daily,'~ecltraining::in::namephonesupd2'),
 STD.File.AddSuperFile(Daily,'~ecltraining::in::namephonesupd3'),
 STD.File.FinishSuperFileTransaction()
          );
```

Lesson Completed!

Proceed to Lesson 6: Superfile Consolidation



Advanced ECL (Part 2) – Superfiles, Working with XML and JSON, NLP and XML Parsing

Lesson 6

Superfile Consolidation

LexisNexis® RISK SOLUTIONS

RELX Group

Lesson 6:

Before:

Logical Name	Owner	Super Owner	Description	Cluster	Records	Size
online::bmf::sf::alldata					364,781	33,924,633
online::bmf::sf::daily		online::bmf::sf::alldata			214,050	19,906,650
online::bmf::sf::weekly		online::bmf::sf::alldata				0

After:

Logical Name	Owner	Super Owner	Description	Cluster	Records	Size
online::bmf::out::weeklyrollup1	BobF			mythor	214,050	19,906,650
online::bmf::sf::alldata					364,781	33,924,633
online::bmf::sf::daily		online::bmf::sf::alldata				0
online::bmf::sf::weekly	BobF	online::bmf::sf::alldata			214,050	19,906,650

Lesson 6 Lab Exercise:

Exercise Spec:
Consolidate all “Daily” subfiles into a new single “Weekly” file, and then replace the “Daily” subfiles with the new “Weekly” file. You will use a standard OUTPUT, and AddSuperFile and ClearSuperFile within a transaction frame.

Requirements:
1. The sub-files to consolidate that are already a part of the “Daily” superfile are:
    ~ecltraining::IN::namephonesupd2
    ~ecltraining::IN::namephonesupd3
2. OUTPUT the daily file to a backup file to store on the THOR cluster. The file name to create for this exercise must start with **~ONLINE::** followed by *your initials*, followed by **::OUT::filename** as in this example:
    ~ONLINE::XXX::OUT::WeeklyRollup1
3. Save your code in a file named **BWR_WeeklyRollup_SF1**.

Best Practices Hint
Since this file is a weekly update file, the name is one-time use and does not need to be added to your previously defined **File_AllData** MODULE structure defining the filename constants.

Result Comparison
Use a Builder window to query the dataset, using this query (or simply verify the results in your ECL Watch):

```
IMPORT TrainingAdvECL;
COUNT(TrainingAdvECL.File_AllData.AllDataDS);
```

You'll find that the number returned is the same as the previous query, since the superfile now contains the same amount of data.

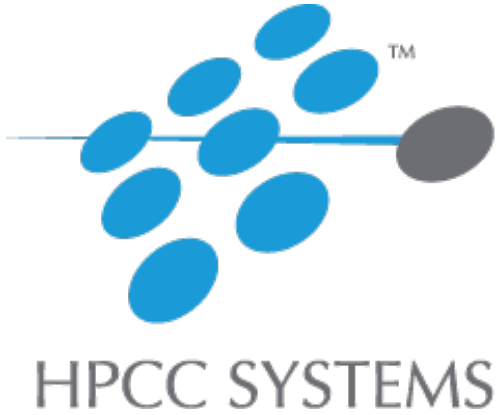
Lesson 6 Lab Exercise Solution:

```
//BWR_WeeklyRollup_SF1

IMPORT $,STD;

SEQUENTIAL(
  OUTPUT($.File_AllData.DailyDS,,'~ONLINE::BMF::out::WeeklyRollup1',overwrite),
  STD.File.StartSuperFileTransaction(),
   STD.File.AddSuperFile($.File_AllData.WeeklySF,'~ONLINE::BMF::out::WeeklyRollup1'),
   STD.File.ClearSuperFile($.File_AllData.DailySF),
  STD.File.FinishSuperFileTransaction());
```

Lesson Completed!

Proceed to Lesson 7:
More Superfile Lab Exercises

HPCC SYSTEMS®



Advanced ECL (Part 2) – Superfiles, Working with XML and JSON, NLP and XML Parsing

Lesson 7

Adding More Subfiles to a Consolidated Superfile

Lesson 7 – Add More Sub Files to “Daily”:

Before:

Logical Name	Owner	Super Owner	Description	Cluster	Records	Size
online::bmf::out::weeklyrollup1	BobF			mythor	214,050	19,906,650
online::bmf::sf::alldata					364,781	33,924,633
online::bmf::sf::daily		online::bmf::sf::alldata				0
online::bmf::sf::weekly	BobF	online::bmf::sf::alldata			214,050	19,906,650

After:

Logical Name	Owner	Super Owner	Description	Cluster	Records	Size
ecltraining::in::namephonesupd4		online::bmf::sf::daily		mythor	87,933	8,177,7...
ecltraining::in::namephonesupd5		online::bmf::sf::daily		mythor	80,829	7,517,0...
online::bmf::out::weeklyrollup1	BobF	online::bmf::sf::weekly		mythor	214,050	19,906,...
online::bmf::sf::alldata					533,543	49,619,...
online::bmf::sf::daily		online::bmf::sf::alldata			168,762	15,694,...
online::bmf::sf::weekly	BobF	online::bmf::sf::alldata			214,050	19,906,...

Lesson 7 Lab Exercise:

Exercise Spec:

Add two (2) sub-files to the "Daily" superfile. The coding technique that you will use here is the same as in *Lesson 3* and *Lesson 5*. Review those prior lessons if needed.

Requirements:

1. The sub-files to add are:
   ~ecltraining::IN::namephonesupd4
   ~ecltraining::IN::namephonesupd5
2. Save your code in a file named **BWR_Add_SF3**.

Best Practices Hint

Since these files are daily update files, their names are one-time use and do not need to be added to your previously defined **File_AllData** MODULE structure that defines the filename constants.

Result Comparison

Query the dataset, using this query:

```
IMPORT TrainingAdvECL;
COUNT(TrainingAdvECL.File_AllData.AllDataDS);
```

You'll find that the number returned is now larger than the previous query, since the superfile now contains two additional sub-files.

Lesson 7 Lab Exercise Solution:

```
//BWR_Add_SF3

IMPORT $,STD;
SEQUENTIAL(STD.File.StartSuperFileTransaction(),
        STD.File.AddSuperFile($.File_AllData.DailySF,'~ecltraining::in::namephonesupd4'),
        STD.File.AddSuperFile($.File_AllData.DailySF,'~ecltraining::in::namephonesupd5'),
           STD.File.FinishSuperFileTransaction()
           );
```

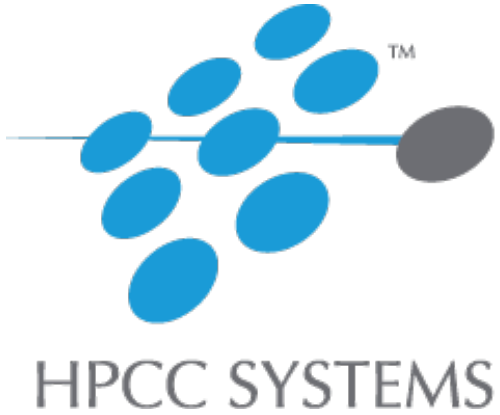
Syntax Errors

Message	Code
Information: StartSuperFileTransaction	0
Information: AddSuperFile ('ONLINE::BMF::SF::Daily', 'edtraining::in::namephonesupd4') trans	0
Information: AddSuperFile ('ONLINE::BMF::SF::Daily', 'edtraining::in::namephonesupd5') trans	0
Information: FinishSuperFileTransaction commit	0

HPCC SYSTEMS®

Lesson Completed!

Proceed to Lesson 8: Final Consolidation



Advanced ECL (Part 2) – Superfiles, Working with XML and JSON, NLP and XML Parsing

Lesson 8

Daily/Weekly Superfile Consolidation

LexisNexis® RISK SOLUTIONS

RELX Group

Lesson 8 – Daily/Weekly Consolidation:

Before:

Logical Name	Owner	Super Owner	Description	Cluster	Records	Size
ecltraining::in::namephonesupd4		online::bmf::sf::daily		mythor	87,933	8,177,7...
ecltraining::in::namephonesupd5		online::bmf::sf::daily		mythor	80,829	7,517,0...
online::bmf::out::weeklyrollup1	BobF	online::bmf::sf::weekly		mythor	214,050	19,906,...
online::bmf::sf::alldata					533,543	49,619,...
online::bmf::sf::daily		online::bmf::sf::alldata			168,762	15,694,...
online::bmf::sf::weekly	BobF	online::bmf::sf::alldata			214,050	19,906,...

After:

Logical Name	Owner	Super Owner	Description	Cluster	Records	Size
ecltraining::in::namephonesupd5				mythor	80,829	7,517,0...
online::bmf::out::weeklyrollup1	BobF			mythor	214,050	19,906,...
online::bmf::sf::alldata				mythor	533,543	49,619,...
online::bmf::sf::daily						0
online::bmf::sf::newbaserollup1	BobF	online::bmf::sf::alldata		mythor	533,543	49,619,...
online::bmf::sf::weekly						0

Lesson 8 Lab Exercise:

Exercise Spec:
Consolidate all “Daily” and “Weekly” subfiles into a *new* Base file, and then replace all subfiles with that new Base file. Use the same consolidation coding technique that was introduced in *Lesson 6*.

Requirements:
1. The sub-files to consolidate are - all of them!
2. Create a new ECL definition file in the **File_AllData** module named **Base2**. The file name to create for this exercise that you will OUTPUT to your target cluster must start with **~ONLINE::** followed by *your initials*, followed by **::OUT::filename** as in this example:
   ~ONLINE::XXX::OUT::NewBaseRollup1
3. Save your code in a file named **BWR_NewBaseRollup_SF1**.

Best Practices Hint
Since this file is a new Base file, the name you previously defined in your **File_AllData** MODULE structure for the Base file should be updated before you restructure the subfiles in your superfile (See Requirement (2.) above).

Result Comparison
Query the dataset, using this query (or use ECL Watch to verify your results):

```
IMPORT TrainingAdvECL;
COUNT(TrainingAdvECL.File_AllData.AllDataDS);
```

You'll find that the number returned is the same as the previous query, since the superfile still contains the same amount of data.

Lesson 8 Lab Exercise Solution:

```
//File_AllData.ECL
EXPORT File_AllData := MODULE
 EXPORT AllDataSF := '~ONLINE::BMF::SF::Alldata';
 EXPORT WeeklySF  := '~ONLINE::BMF::SF::Weekly';
 EXPORT DailySF   := '~ONLINE::BMF::SF::Daily';
 EXPORT Base1     :=
'~ecltraining::in::namephonesupd1';
 EXPORT Base2     :=
'~ONLINE::BMF::SF::NewBaseRollup1';
 SHARED Rec := RECORD
  UNSIGNED4 recid;
  UNSIGNED4 foreignkey;
  STRING10  homephone;
  STRING10  cellphone;
  STRING20  fname;
  STRING20  mname;
  STRING20  lname;
  STRING5   name_suffix;
 END;
 EXPORT AllDataDS := DATASET(AllDataSF,Rec,THOR);
 EXPORT WeeklyDS  := DATASET(WeeklySF,Rec,THOR);
 EXPORT DailyDS   := DATASET(DailySF,Rec,THOR);
END;
```

```
//BWR_NewBaseRollup_SF1.ECL
IMPORT $,STD;
SEQUENTIAL(
  OUTPUT($.File_AllData.AllDataDS,,$.File_AllData.Base2,overwrite),
  STD.File.StartSuperFileTransaction(),
   STD.File.ClearSuperFile($.File_AllData.AllDataSF),
   STD.File.ClearSuperFile($.File_AllData.WeeklySF),
   STD.File.ClearSuperFile($.File_AllData.DailySF),
   STD.File.AddSuperFile($.File_AllData.AllDataSF,$.File_AllData.WeeklySF),
   STD.File.AddSuperFile($.File_AllData.AllDataSF,$.File_AllData.DailySF),
   STD.File.AddSuperFile($.File_AllData.AllDataSF,$.File_AllData.Base2),
  STD.File.FinishSuperFileTransaction()
         );
```

Superfiles Final Review:

- ✓ We have examined many key Superfile functions that are used in typical day-to-day operations.

- ✓ In the Standard Library Reference PDF, there are many more Superfile functions that will help you understand everything you need to know about Superfiles and Superkeys

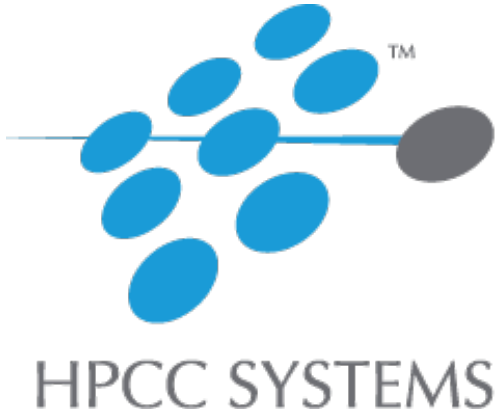
Superfile Functions – The Rest:

We have examined many key Superfile functions that are used in typical day-to-day operations. Here are all of them again:

CreateSuperFile	SuperFileExists
DeleteSuperFile	GetSuperFileSubCount
GetSuperFileSubName	LogicalFileSuperOwners
LogicalFileSuperSubList	SuperFileContents
FindSuperFileSubName	StartSuperFileTransaction
AddSuperFile	RemoveSuperFile
ClearSuperFile	SwapSuperFile
ReplaceSuperFile	PromoteSuperFileList
FinishSuperFileTransaction	RemoveOwnedSubFiles

Lesson Completed!

Proceed to Lesson 9: Working with XML/JSON



Advanced ECL (Part 2) – Superfiles, Working with XML and JSON, NLP and XML Parsing

Lesson 9

Working with XML

LexisNexis® RISK SOLUTIONS

RELX Group

Working with XML:

- The HPCC platform works with
  - FLAT
  - Variable Length
  - JSON
  - …and XML!

- We will be working with:
  - Simple XML/JSON
  - Complex XML (using attributes)
  - Nested Child XML (Relational Data)

RECORD for XML:

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*') }

{ XMLDEFAULT('value‘)}

{ 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: <, <=, >, >=, =, !=

Examples of supported xpath:

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

To emulate AND conditions:

```
/a/b[@x="1"][@y="2"]
```

Non-standard XPATH:

```
STRING text{xpath('a/b<>')};
```

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,* **XML(** *path*[**, NOROOT]) [,ENCRYPT(***key***)]);**
name **:= DATASET(** *file, recorddef,* **JSON(** *path*[**, NOROOT]) [,ENCRYPT(***key***)]);**

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.
- ✓*path* – The XPATH to the XML row tag.
- ✓**NOROOT -** Specifies the *file* is an XML file with no file tags, only row tags.
- ✓**ENCRYPT -** Optional. Specifies the *file* was created by OUTPUT with the ENCRYPT option.
- ✓*key* - A string constant containing the encryption key used to create the file

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('~ONLINE::XXX::IN::timezones',r,XML('Dataset/area'));
```

JSON DATASET Example:

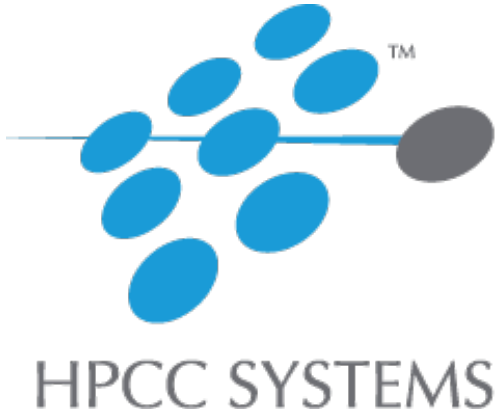
```
/*
{"Row": [{"id": "15520054709326826887", "firstname": "Otilia ", "lastname": "Tuscano ", "middlename": " ",    "namesuffix": " ", "filedate":
"19950119", "bureaucode": 575, "maritalstatus": " ",    "gender": "F", "dependentcount": 0, "birthdate": "19750624",    "streetaddress": "30 DWELLY
RD ", "city": "MIAMI ", "state": "FL", "zipcode": "33155"}]}
*/
Layout_PeopleFile := RECORD
  UNSIGNED8 ID;
  STRING15   FirstName;
  STRING25   LastName;
  STRING15   MiddleName;
  STRING2    NameSuffix;
  STRING8    FileDate;
  STRING1    Gender;
  STRING8    BirthDate;
  STRING42   StreetAddress;
  STRING20   City;
  STRING2    State;
  STRING5    ZipCode;
END;

ds    := DATASET('~ONLINE::bmf::in::peoplefljson',Layout_PeopleFile,JSON('Row'));
```

Lesson Completed!

Lesson 10 – Spray/Define Simple XML/JSON files

Lesson 11 – Spray/Define Complex XML file

Lesson 12 – Spray/Define Nested Child XML file



Advanced ECL (Part 2) – Superfiles, Working with XML and JSON, NLP and XML Parsing

Lesson 10

Simple XML and Simple JSON Processing

LexisNexis® RISK SOLUTIONS

RELX Group

Lesson 10 – Simple XML - Tag Based

```
<Dataset>
<area>
 <code>201</code>
 <description>PA Pennsylvania (Philadelphia area, overlays with 267 and
445)</description>
</area>
<area>
 <code>202</code>
 <description>OH Ohio (Cleveland area)</description>
</area>
<area>
 <code>203</code>
 <description>IL Illinois(Champaign, Urbana, Springfield and central
Illinois)</description>
</area>
</Dataset>
```

Lesson 10 Spray Exercise Part 1:

Exercise Spec:
Using the ECL Watch **Spray XML** option, spray this XML file on the landing zone to your target cluster:

timezones.xml

Requirements:
1. The **Row** tag is: *area*
2. The **Target Scope** must start with *~ONLINE::*, followed by *your initials* followed by *IN::Timezones* as in this example:

~ONLINE::XXX::IN

3. The **Target Name** is *Timezones.*

Best Practices Hint
Remember that the XML XPATH information is always case sensitive.

Result Comparison
The spray must complete with no errors to be considered a successful exercise.

Lesson 10 Spray Exercise Part 2:

Exercise Spec:
Using the ECL Watch **Spray JSON** option, spray the following JSON file on the landing zone to your target cluster:

peoplefljson

Requirements:
1. The **Target Scope** must start with *~ONLINE::*, followed by *<your initials>,* followed by *IN* as in this example:
   ~ONLINE::XX::IN
2. The **Target Name** is *PeopleJSON*
3. The **Row Path** is: *Row*
4. Accept all of the other defaults.

Best Practices Hint
Remember that JSON is similar to XML in processing.

Result Comparison
The spray must complete with no errors to be considered a successful exercise.

Lesson 10 Define Exercise Part 1:

Exercise Spec:

Create Builder Window Runnable code that defines the RECORD structure and DATASET definition for the Timezones file sprayed in the previous exercise. The data in this file looks like this:

```
<Dataset>
<area>
 <code>201</code>
 <description>PA Pennsylvania</description>
 <zone>Eastern Time Zone</zone>
</area>
<area>
 <code>202</code>
 <description>OH Ohio (Cleveland area)</description>
 <zone>Eastern Time Zone</zone>
</area>
</Dataset>
```

Requirements:

1. The ECL file name to create for this exercise is **BWR_SimpleXML**
2. The layout of the fields is as follows:

```
Code - unsigned 2-byte integer
Description - 110-character string
Zone - 42-character string
```

Best Practices Hint

The key to this exercise is the XML option on the DATASET declaration and how the RECORD structure is constructed.

Result Comparison

Do a simple OUTPUT of the dataset to check that the result looks good (non-garbage data).

Lesson 10 Solution Part 1:

```
//BWR_SimpleXML
/* <Dataset>
    <area>
     <code>201</code>
     <description>PA Pennsylvania</description>
     <zone>Eastern Time Zone</zone>
    </area>
    <area>
     <code>202</code>
     <description>OH Ohio (Cleveland area)</description>
     <zone>Eastern Time Zone</zone>
    </area>
   </Dataset>*/

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

d := DATASET('~ONLINE::xxx::IN::timezones',r,XML('Dataset/area'));

OUTPUT(d);
```

Lesson 10 Define Exercise Part 2:

Exercise Spec:
Create Builder Window Runnable code that defines the RECORD structure and DATASET definition for the PeopleFLJSON file sprayed in the previous exercise.

The data in this file is formatted like this:

```
{"Row": [
{"id": "15520054709326826887", "firstname": "Otilia ", "lastname": "Tuscano ", "middlename":" ","namesuffix": " ",
"filedate": "19950119", "bureaucode": 575, "maritalstatus": " ","gender": "F","dependentcount": 0, "birthdate":
"19750624", "streetaddress":"30 DWELLY RD ", "city": "MIAMI ", "state": "FL", "zipcode": "33155"}
]}
```

Requirements:

1. The file name to create for this exercise is: BWR_SimpleJSON
2. The layout of the fields is as follows:

```
id                unsigned 8 byte integer
firstname         15 character string
lastname          25 character string
middlename        15 character string
namesuffix        2 character string
filedate          8 character string
maritalstatus     1 character string
gender            1 character string
dependentcount    unsigned 1 byte integer
birthdate         8 character string
streetaddress     42 character string
city              20 character string
state             2 character string
zipcode           5 character string
```

Best Practices Hint
The key to this exercise is using the JSON option on the DATASET declaration with the proper RECORD structure referenced.

Result Comparison
Do a simple OUTPUT of the dataset to check that the result looks good and includes child records. View the result through the ECL Watch page.

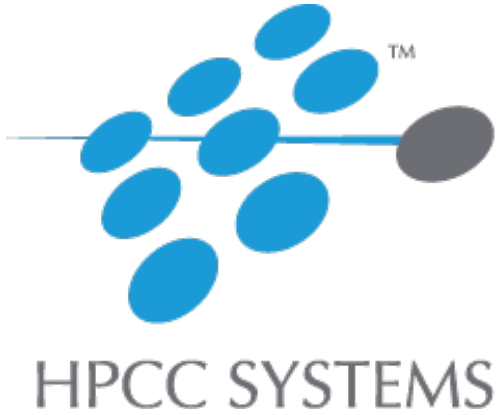
Lesson 10 Solution Part 2:

```
Layout_PeopleFile := RECORD
  UNSIGNED8 ID;
  STRING15  FirstName;
  STRING25  LastName;
  STRING15  MiddleName;
  STRING2   NameSuffix;
  STRING8   FileDate;
  UNSIGNED2 BureauCode;
  STRING1   MaritalStatus;
  STRING1   Gender;
  UNSIGNED1 DependentCount;
  STRING8   BirthDate;
  STRING42  StreetAddress;
  STRING20  City;
  STRING2   State;
  STRING5   ZipCode;
END;

ds := DATASET('~ONLINE::XXX::in::peoplefljson',Layout_PeopleFile,JSON('Row'));
OUTPUT(ds);
```

Lesson Completed!

Proceed to Lesson 11:
Working with Complex XML

HPCC SYSTEMS®



Advanced ECL (Part 2) – Superfiles, Working with XML and JSON, NLP and XML Parsing

Lesson 11

Spray/Define Complex XML

LexisNexis® RISK SOLUTIONS

RELX Group

Lesson 11 – XML – Attribute Based

<dataset><area code="201" zone="Eastern Time Zone"/><area code="202" zone="Eastern Time Zone"/><area code="203" zone="Eastern Time Zone"/><area code="204" zone="Central Time Zone"/><area code="205" zone="Central Time Zone"/><area code="206" zone="Pacific Time Zone"/><area code="207" zone="Eastern Time Zone"/><area code="208" zone="Mountain & Pacific Time Zones"/><area code="209" zone="Pacific Time Zone"/><area code="210" zone="Central Time Zone"/><area code="212" zone="Eastern Time Zone"/><area code="213" zone="Pacific Time Zone"/><area code="214" zone="Central Time Zone"/> <area code="260" description="IN Indiana (Fort Wayne, Decatur, Angola, Wabash and northeastern Indiana)" zone="Central & Eastern Time Zones"/><area code="262" description="WI Wisconsin (Menomonee Falls, Waukesha, Racine and southeastern Wisconsin excluding Milwaukee area)" zone="Central Time Zone"/> </dataset>

```
STRING code{xpath('@code')};
```

Lesson 11 Spray Exercise:

Exercise Spec:

Using the ECL Watch **Spray XML** option, spray this file on the landing zone:

complextimezones.xml

Requirements:

1. The Row tag is: area
2. The **Target Scope** must start with *ONLINE::*, followed by *your initials* followed by *IN::* as in this example:

   ONLINE::XXX::IN
3. The **Target Name** is *ComplexTimezones.*
4. Accept all other defaults.

Best Practices Hint

Remember that XML is always case sensitive.

Result Comparison

The spray must complete with no errors to be considered a successful exercise.

Lesson 11 Define Exercise:

Exercise Spec:
Create Builder Window Runnable code that defines the RECORD structure and DATASET definition for the Timezones file sprayed in the previous exercise.

The data in this file is the same, but formatted like this:

```
<dataset>
<area code="201" description="description" zone="Eastern Time Zone"/>
<area code="202" description="description" zone="Eastern Time Zone"/>
</dataset>
```

Requirements:
1. The file name to create for this exercise is **BWR_ComplexXML**
2. The layout of the fields is the same as the previous but the sizes should be variable length.

Best Practices Hint:
The key to this exercise is again the XML option on the DATASET declaration and how the RECORD structure is constructed.

Result Comparison:
Do a simple OUTPUT of the dataset to check that the result looks good (non-garbage data).

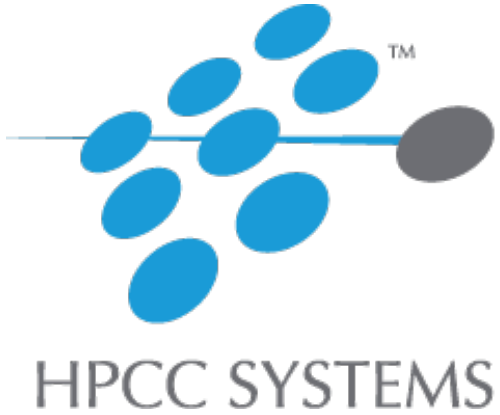
Lesson 11 Solution:

```
//BWR_ComplexXML
/* <dataset>
   <area code="201" description="description" zone="Eastern Time Zone"/>
   <area code="202" description="description" zone="Eastern Time Zone"/>
   </dataset>*/

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

d := DATASET('~ONLINE::XXX::IN::complextimezones',r,XML('dataset/area'));

OUTPUT(d);
```

Lesson Completed!

Proceed to Lesson 12:
Working with Relational XML

HPCC SYSTEMS®



Advanced ECL (Part 2) – Superfiles, Working with XML and JSON, NLP and XML Parsing

Lesson 12

Spray/Define Nested (Relational) XML

LexisNexis® RISK SOLUTIONS

RELX Group

Lesson 12 Spray Exercise:

Exercise Spec:

Using the ECL Watch Spray XML page, spray this file located on the landing zone to your target HPCC THOR cluster:

onlinenestedchildxml

Requirements:

1. The **Row** tag is: *Row* (don't forget that XML is case sensitive)
2. The **Target Scope** name must start with *ONLINE::*, followed by *your initials* followed by *IN::* as in this example:

   ONLINE::XXX::IN
3. The Target Name is *NestedChildXML*
4. Accept all other defaults.

Result Comparison

The spray must complete with no errors to be considered a successful exercise.

Lesson 12 Define Exercise:

<dataset>
<Row>
<id>187522928604396</id> <firstname>PETRONICA</firstname> <lastname>SPOCK</lastname> <middlename></middlename>
<namesuffix></namesuffix> <filedate>19900425</filedate>
<maritalstatus></maritalstatus> <gender>F</gender> <dependentcount>0</dependentcount> <birthdate>19240205</birthdate> <streetaddress>13
GLEN FORGE DR</streetaddress>
<city>LIVONIA</city> <state>MI</state> <zipcode>48150</zipcode>
<childaccts>
<Row>
<personid>187522928604396</personid> <reportdate>20001201</reportdate> <industrycode>DC</industrycode>
<opendate>19920801</opendate> <highcredit>146</highcredit> <balance>0</balance> <terms>0</terms>
<accountnumber>146399999999</accountnumber> <lastactivitydate>19990401</lastactivitydate>
</Row>
<Row>
<personid>187522928604396</personid> <reportdate>20001101</reportdate> <industrycode>OC</industrycode>
<opendate>19810301</opendate> <highcredit>142</highcredit> <balance>0</balance> <terms>0</terms>
<accountnumber>5400999999</accountnumber>
<lastactivitydate>20000701</lastactivitydate>
</Row>
</childaccts>
</Row>
</dataset>

```
**************************************************
DATASET(layout_accts) childaccts{XPATH('childaccts/Row'),MAXCOUNT(120)};
```


HPCC SYSTEMS®

Lesson 12 Define Exercise:

Exercise Spec:

Create Builder Window Runnable code that defines the RECORD structure and DATASET definition for the NestedChild file sprayed in the previous exercise. The data in this file is formatted as shown on the previous slide:

Requirements:

1. The file name to create for this exercise is: **BWR_NestedChildXML**
2. The layout of the fields is as follows:

```
Person Record:
id              unsigned 8 byte integer
firstname       15 character string
lastname        25 character string
middlename      15 character string
namesuffix      2 character string
filedate        8 character string
maritalstatus   1 character string
gender          1 character string
dependentcount  unsigned 1 byte integer
birthdate       8 character string
streetaddress   42 character string
city            20 character string
state           2 character string
zipcode         5 character string
```

```
Accounts Record:
personid            unsigned 8 byte integer
reportdate          8 character string
industrycode        2 character string
opendate            8 character string
highcredit          unsigned 4 byte integer
balance             unsigned 4 byte integer
terms               unsigned 2 byte integer
accountnumber       20 character string
lastactivitydate    8 character string
```

Best Practices Hint

The key to this exercise is again the XML option on the DATASET declaration and how the RECORD structure is constructed -- -especially the XPATH of the nested child dataset field.

Result Comparison

Do a simple OUTPUT of the dataset to check that the result looks good and includes child records. View the result through the ECL Watch page.

Lesson 12 Solution

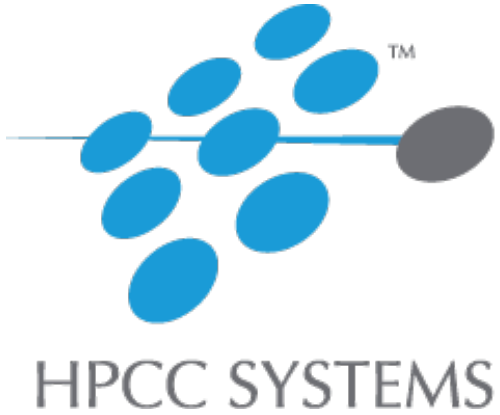
```
Layout_accts := RECORD
 UNSIGNED8 personid;
 STRING8   reportdate;
 STRING2   industrycode;
 STRING8   opendate;
 UNSIGNED4 highcredit;
 UNSIGNED4 balance;
 UNSIGNED2 terms;
 STRING20  accountnumber;
 STRING8   lastactivitydate;
END;
```

```
Layout_person := RECORD
 UNSIGNED8 id;
 STRING15  firstname;
 STRING25  lastname;
 STRING15  middlename;
 STRING2   namesuffix;
 STRING8   filedate;
 STRING1   maritalstatus;
 STRING1   gender;
 UNSIGNED1 dependentcount;
 STRING8   birthdate;
 STRING42  streetaddress;
 STRING20  city;
 STRING2   state;
 STRING5   zipcode;
 DATASET(layout_accts)childaccts{XPATH('childaccts/Row'),
           MAXCOUNT(120)};
END;

ds := DATASET('~ONLINE::XXX::IN::NestedChildXML',
              layout_person,XML('dataset/Row'));

OUTPUT(ds);
```

Lesson Completed!

Proceed to Lesson 13:
Free Form Text and XML Parsing

HPCC SYSTEMS®



Advanced ECL (Part 2) – Superfiles, Working with XML and JSON, NLP and XML Parsing

Lesson 13

Text and XML Parsing (Part 1 of 2)

LexisNexis® RISK SOLUTIONS

RELX Group

Natural Language Parsing – Part 1

- ✓Fundamentals of Natural Language Parsing (NLP)
- ✓Parse Pattern Value Type Definitions
  - ✓PATTERN
  - ✓TOKEN
  - ✓RULE
- ✓Parse Pattern Definitions

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.

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.

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.

Parse Pattern Definitions:

- *pattern-name*
- (*pattern*)
- *pattern1 pattern2*
- '*string*'
- **FIRST**
- **LAST**
- **ANY**
- **REPEAT**(*pattern*)
- **REPEAT**(*pattern*, *expression*)
- **REPEAT**(*pattern*, *low*, ANY [,MIN])
- **REPEAT**(*pattern*, *low*, *high*)
- **OPT**(*pattern*)
- *pattern1* **OR** *pattern2*
- [*list-of-patterns*]
- *pattern1* **[NOT] IN** *pattern2*
- *pattern1* **[NOT] BEFORE** *pattern2*
- *pattern1* **[NOT] AFTER** *pattern2*
- *pattern* **LENGTH**(*range*)
- **VALIDATE**(*pattern*, *isValidExpression*)
- **VALIDATE**(*pattern*, *isValidAsciiExpression*, *isValidUnicodeExpression*)
- **NOCASE**(*pattern)*
- **CASE**(*pattern*)
- *pattern* **PENALTY**(*cost*)
- **TOKEN**(*pattern*)
- **PATTERN**('*regular expression*')

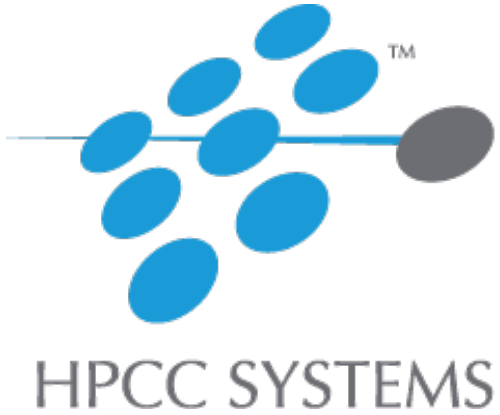
Parse Example:

```
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);
```

Lesson Completed!

Proceed to Lesson 14: Text and XML Parsing (Part 2 of 2)

HPCC SYSTEMS®



Advanced ECL (Part 2) – Superfiles, Working with XML and JSON, NLP and XML Parsing

Lesson 14

Text and XML Parsing (Part 2 of 2)

LexisNexis® RISK SOLUTIONS

RELX Group

Natural Language Parsing

- ✓NLP RECORD Structure Functions
  - ✓6 Options
- ✓NLP PARSE Function
- ✓NLP PARSE Flags
  - ✓19 options

NLP RECORD Structure Functions:

MATCHED([*patternreference*])
MATCHTEXT([*patternreference*])
MATCHUNICODE([*patternreference*])
MATCHLENGTH([*patternreference*])
MATCHPOSITION([*patternreference*])
MATCHROW([*patternreference*])

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.

NLP RECORD Structure Functions 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);

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;
```

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.

PARSE Flags

PARSE function *flags* can have the following values:

FIRST
ALL
WHOLE
NOSCAN
SCAN
SCAN ALL
NOCASE
CASE
SKIP(*separator-pattern*)

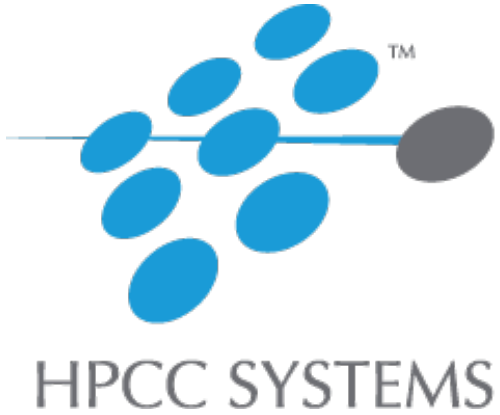
KEEP(*max*)
ATMOST(*max*)
MAX
MIN
MATCHED([*rule-reference*])
MATCHED(ALL)
NOT MATCHED
NOT MATCHED ONLY
BEST
MANY

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 RelationPhrase   :=  MATCHTEXT(relationtype) };
outfile1 := PARSE(datafile,line,progeny,results,SCAN ALL);
```

Lesson Completed!

Proceed to Lesson 15:
XML Parsing



Advanced ECL (Part 2) – Superfiles, Working with XML and JSON, NLP and XML Parsing

Lesson 15

XML Parsing

LexisNexis® RISK SOLUTIONS

RELX Group

Parsing XML Streams

- ✓XML RECORD Structure Functions
- ✓XML PARSE Function
- ✓XML Parsing Exercise – Spray and Parse

XML RECORD Structure Functions

XMLTEXT(*xmltag*) **XMLUNICODE(*xmltag*)** **XMLPROJECT(*xmltag, transform*)**

xmltag - An XPATH string constant to the tag containing the data.

```
d := DATASET([{'<library><book isbn="123456789X">' +
               '<author>Bayliss</author><title>A Way Too Far</title></book>' +
               '<book isbn="1234567801">' +
               '<author>Smith</author><title>A Way Too Short</title></book>' +
               '</library>'}],       {STRING line });
rform := RECORD
  STRING author := XMLTEXT('author');
  STRING title      := XMLTEXT('title');
END;
books := PARSE(d,line,rform,XML('library/book'));
output(books)
```

PARSE (XML Form)

PARSE(*dataset,data,xmlresult,***XML(***path***))**

- ✓*dataset* - The set of records to process.
- ✓*data* - An expression specifying the text to parse, typically the name of a field in the *dataset*.
- ✓*xmlresult* - The name of either the RECORD structure definition that specifies the format of the output record set, or the TRANSFORM function that produces the output record set.
- ✓*path* - An XPATH string constant naming the row tag in the *dataset*.

XML PARSE Example:

```
in1 := DATASET([{'<ENTITY eid="P101" type="PERSON" subtype="MILITARY">' +
'<ATTR name="fullname">JOHN SMITH</ATTR>' +
'<ATTRGRP descriptor="passport">' +
'<ATTR name="idNumber">W12468</ATTR><ATTR name="idType">pp</ATTR>' +
'<ATTR name="issuingAuthority">JAPAN PASSPORT AUTHORITY</ATTR>' +
'<ATTR name="country" value="L202"/></ATTRGRP></ENTITY>'}], {STRING line});
passportRec := { STRING id, STRING country};
outrec         := { STRING  id, UNICODE  fullname,  passportRec passport };
outrec t( in1 L) := TRANSFORM
  SELF.id           := XMLTEXT('@eid');
  SELF.fullname := XMLUNICODE('ATTR[@name="fullname"]');
  SELF.passport.id :=
                XMLTEXT('ATTRGRP[@descriptor="passport"]/ATTR[@name="idNumber"]');
  SELF.passport.country := XMLTEXT('ATTRGRP[@descriptor="passport"]‘ +
                                                        '/ATTR[@name="country"]/@value');
END;
textout := PARSE(in1, line, t(LEFT), XML('/ENTITY[@type="PERSON"]'));
```

Lesson 15 Spray Exercise:

Exercise Spec:
Using the ECL Watch **Spray Delimited** option, spray this file on the landing zone:
embeddedxmltimezones

Requirements:

1. Empty (clear) the **Quote** field, and check the **Omit Separator** checkbox option.
2. The **Line Terminator** is: **/>**
3. The **Target Scope** must start with *ONLINE::*, followed by *your initials* followed by *IN::* as shown in this example:
   ONLINE::XXX::IN::
4. The **Target Name** is *EmbeddedXMLtimezones*

Best Practices Hint
Although you're using the Spray Delimited option to accomplish the spray operation, the file itself is NOT a CSV file, but simply a *variable-length record flat file*. That's why you delete the contents of the Separator and Quote fields.

Result Comparison
The spray must complete with no errors to be considered a successful exercise.

Lesson 15 XML Parse Exercise:

Exercise Spec:
Create Builder Window Runnable code that defines the RECORD structure and DATASET definition for the **EmbeddedXMLtimezones** file sprayed in the last exercise, and then **use the XML form of PARSE** to retrieve the same timezones information from the embedded XML data.

This file contains variable-length records (hence the use of the Spray Delimited page) containing two fields—a 2-byte binary field and a variable-length string field whose content is XML text. The XML looks like this:

```
<area code="201" description="description" zone="Eastern Time Zone"/>
<area code="202" description="description" zone="Eastern Time Zone"/>
```

Requirements:
1. The file name to create for this exercise is **BWR_ParseXML**.
2. The layout of the fields is:
Sequence number - unsigned 2-byte integer
line - variable length string containing the XML data

Best Practices Hint
Remember, you used the **Spray Delimited** page to accomplish the spray operation, but the file itself is a variable-length record FLAT (THOR) file.

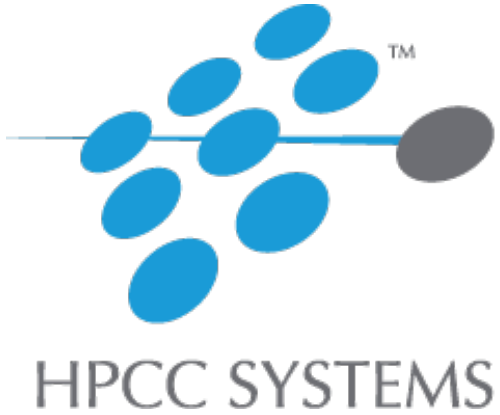
Result Comparison
Execute a simple OUTPUT of the dataset and check that the result looks good.

Lab Exercise Solution:

```
//BWR_ParseXML
rec := RECORD
 INTEGER2 seq;
 STRING   line{MAXLENGTH(200)};
END;
ds := DATASET('~ONLINE::XXX::IN::EmbeddedXMLtimezones',rec,FLAT);

outrec := RECORD
 STRING code{MAXLENGTH(3)};
 STRING description{MAXLENGTH(160)};
 STRING zone{MAXLENGTH(40)};
END;

outrec ParseIt(rec L) := TRANSFORM
 SELF.code        := XMLTEXT('@code');
 SELF.description := XMLTEXT('@description');
 SELF.zone        := XMLTEXT('@zone');
END;

x := PARSE(ds,line,ParseIt(LEFT),XML('area'));
OUTPUT(x);
```

Lesson Completed!

Proceed to Lesson 16: Free Form Text Parsing Lab Exercise



Advanced ECL (Part 2) – Superfiles, Working with XML and JSON, NLP and XML Parsing

Lesson 16

Free Form Text Parsing Lab Exercise

LexisNexis® RISK SOLUTIONS

RELX Group

Natural Language Parsing

- ✓ Free Form Text Parsing Exercise
  - ✓ Spray
  - ✓ Parse
- ✓ Using the Internet Movie Database public data

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 RelationPhrase   := MATCHTEXT(relationtype) };
outfile1 := PARSE(datafile,line,progeny,results,SCAN ALL);
```

NLP RECORD Structure Functions 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);

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;
```

Lesson 16 Spray Exercise:

Exercise Spec:

Using the ECL Watch Spray Delimited option, spray the *imdb_movies* file. The name of the file on the landing zone is:

imdb_movies

Requirements:

1. The **Target Scope** must start with *ONLINE::*, followed by *your initials* followed by *IN* as shown in this example:

   ONLINE::XXX::IN

2. The Target Name is *imdb_movies*
3. Empty (clear) the **Quote** field, and check the **Omit Separator** checkbox option.
4. The **Line Terminator** is using the default values.

Result Comparison

The spray must complete with no errors to be considered a successful exercise.

Lesson 16 NLP Parse Exercise:

Exercise Spec:
Create Builder Window Runnable code that defines the RECORD structure and DATASET definition for the imdb_movies file sprayed in the previous exercise. Then use PARSE to retrieve data from the free-form text. The free-form text in this file looks like this:

```
$40,000 (1996) 1996
$5,000 Reward (1918) 1918
$5,000,000 Counterfeiting Plot, The (1914) 1914
$5.15/Hr. (2004) (TV) 2004
$5.20 an Hour Dream, The (1980) (TV) 1980
$50,000 Challenge, The (1989) (TV) 1989 (unreleased)
$50,000 Climax Show, The (1975) 1975
$50,000 Jewel Theft, The (1915) 1915
```

Each line contains the Title, followed by the Year in parentheses, optionally followed by the Video type in parentheses, followed by the Release year, and optionally followed by a comment in parentheses. The Release year may contain multiple year values, separated by dashes or commas or both, and some years are listed just as ????. Note that the delimiters between the data values are spaces or tabs (ASCII 09). Tabs (one or more) are only used immediately before the Release Year, and any comment (if present). A new line character (\n) terminates each line of the text.

Requirements:
1. The file name to create for this exercise is **BWR_ParseText**
2. The layout of the OUTPUT fields is:

```
Title        variable length string
Titleyear    variable length string
Vidtype      variable length string
Releaseyear  variable length string
Comment      variable length string
```

Best Practices Hint
This file should be defined as a CSV file.
Result Comparison
Execute a simple OUTPUT of the dataset and check that the result looks good.

```
//BWR_ParseText
d := DATASET('~CLASS::XXX::IN::imdb_movies',{STRING line},
             CSV(SEPARATOR(''),QUOTE('')));
PATTERN arb       := ANY+;
PATTERN alpha     := PATTERN('[a-zA-Z]')+;
PATTERN Numbers   := PATTERN('[-0-9?,]')+;
PATTERN fs        := PATTERN('\t')+; //field separator
PATTERN Year      := '(' Numbers OPT('/' arb) ')';
PATTERN Vidtype   := ' (' alpha ')';
PATTERN Title     := arb Year OPT(Vidtype);
PATTERN movieyr   := Numbers;
PATTERN comment   := '(' arb ')';
PATTERN moviedata := Title fs movieyr OPT(fs comment);
```

Lesson 16 Lab Exercise Solution:

```
//BWR_ParseText
d := DATASET('~ONLINE::XXX::IN::imdb_movies',{STRING line},CSV(SEPARATOR(''),QUOTE('')));

PATTERN arb       := ANY+;
PATTERN alpha     := PATTERN('[a-zA-Z]')+;
PATTERN Numbers   := PATTERN('[-0-9?,]')+;
PATTERN fs        := PATTERN('\t')+; //field separator
PATTERN Year      := '(' Numbers OPT('/' arb) ')';
PATTERN Vidtype   := ' (' alpha ')';
PATTERN Title     := arb Year OPT(Vidtype);
PATTERN movieyr   := Numbers;
PATTERN comment   := '(' arb ')';
RULE    moviedata := Title fs movieyr OPT(fs comment);

Outrec := RECORD
 STRING MovieTitle{MAXLENGTH(250)}   := MATCHTEXT(Title/arb);
 STRING Titleyear{MAXLENGTH(20)}     := MATCHTEXT(Title/Year);
 STRING MovieType{MAXLENGTH(10)}     := MATCHTEXT(Title/Vidtype);
 STRING Releaseyear{MAXLENGTH(40)}   := MATCHTEXT(movieyr);
 STRING AddedComments{MAXLENGTH(50)} := MATCHTEXT(comment);
END;

x := PARSE(d,line,moviedata,Outrec,WHOLE,FIRST);
OUTPUT(x);
```

Lesson Completed!

This concludes the Advanced ECL (Part 2) Course

Thanks for Attending!