Package 'rHpcc'

August 22, 2013

Type Package

Title Interface between HPCC and R

Version 1.0

Date 2013-08-22

Author Dinesh Shetye

Maintainer Dinesh Shetye <dinesh.shetye@lexisnexis.com>

Description

rHpcc is an R package providing an Interface between R and HPCC.Familiarity with ECL (Enterprise Control Language) is a must to use this package.HPCC is a massive parallel-processing computing platform that solves Big Data problems.ECL is the Enterprise Control Language designed specifically for huge data projects using the HPCC platform.Its extreme scalability comes from a design that allows you to leverage every query you create for reuse in subsequent queries as needed. To do this, ECL takes a dictionary approach to building queries wherein each ECL definition defines an Attribute. Each previously defined Attribute can then be used in succeeding ECL Attribute definitions as the language extends itself as you use it.

License GPL-2

Depends R (>= 3.0.1), methods, RCurl, XML

URL http://hpccsystems.com

R topics documented:

`rHpcc-package` *R integration with HPCC(High Performance Computing Cluster)*

Description

This package helps in integration of R and HPCC (High Performance Computing Cluster). HPCC is a massive parallel-processing computing platform that solves Big Data problems. The users of this package must be familiar with the ECL(Enterprise Control Language) language used for writting queries on the HPCC platform. ECL is the Enterprise Control Language designed specifically for huge data projects using the HPCC platform. Its extreme scalability comes from a design that allows you to leverage every query you create for re-use in subsequent queries as needed. To do this, ECL takes a dictionary approach to building queries wherein each ECL definition defines an Attribute. Each previously defined Attribute can then be used in succeeding ECL Attribute definitions as the language extends itself as you use it. Refer to ECL Language reference for more details(http://hpccsystems.com/community/docs/ecl-language-reference)

Details

Package:	rHpcc
Type:	Package
Version:	1.0
Date:	2013-08-22
License:	GPL-2
Depends:	R (>= 3.0.1), methods, RCurl, XML

Author(s)

Dinesh Shetye <dinesh.shetye@lexisnexis.com>

References

HPCC Systems (http://hpccsystems.com) ECL Language Reference (http://hpccsystems.com/download/docs/ecl-language-reference)

`ECL` *The base class that generates the ECL(Enterprise Control Language) code.*

Description

The base class that generates the ECL(Enterprise Control Language) code and executes it on the HPCC cluster. This class is used to add ECL code/import definitions.

References

HPCC Systems (http://hpccsystems.com) ECL Language Reference (http://hpccsystems.com/download/docs/ecl-language-reference)

Examples

```
## Not run:
  ecl1 <- ECL$new(hostName="127.0.0.1")
  recPerson <- ECLRecord$new(name="Person")
  recPerson$addField("STRING", "code")
  recPerson$addField("STRING", "firstName")
  recPerson$addField("STRING", "lastName")
  recPerson$addField("STRING", "address")
  recPerson$addField("STRING", "stateCode")
  recPerson$addField("STRING", "city")
  recPerson$addField("STRING", "zip")
  ecl1$add(recPerson)

## End(Not run)
```

`ECL-class` *Class* "ECL"

Description

The base class that generates the ECL(Enterprise Control Language) code and executes it on the HPCC cluster.

Fields

`hostName`: Object of class `character` HPCC server hostname

`port`: Object of class `character` HPCC server port

`eclCode`: Object of class `character` ECL code you want to execute

`clusterName`: Object of class `character` Cluster name on which the ECL code will execute

Methods

`execute()`: This method internally calls the `eclDirectCall` method which executes the ECL code.

`print()`: Prints the ECL code.

`addImport(value)`: Used to add Import definitions.

`add(obj)`: Used to add definitions to the existing code.

References

HPCC Systems (http://hpccsystems.com) ECL Language Reference (http://hpccsystems.com/download/docs/ecl-language-reference)

Examples

```
## Not run:
  ecl1 <- ECL$new(hostName="127.0.0.1")
  recPerson <- ECLRecord$new(name="Person")
  recPerson$addField("STRING", "code")
  recPerson$addField("STRING", "firstName")
  recPerson$addField("STRING", "lastName")
  recPerson$addField("STRING", "address")
  recPerson$addField("STRING", "stateCode")
  recPerson$addField("STRING", "city")
  recPerson$addField("STRING", "zip")
  ecl1$add(recPerson)

## End(Not run)
```

ECLDataset — *Creates a DATASET definition. The DATASET declaration defines a file of records, on disk or in memory.*

Description

Creates a DATASET definition. The DATASET declaration defines a file of records, on disk or in memory.

References

HPCC Systems (http://hpccsystems.com) ECL Language Reference (http://hpccsystems.com/download/docs/ecl-language-reference)

Examples

```
## Not run:
  ecl1 <- ECL$new(hostName="127.0.0.1")
  recPerson <- ECLRecord$new(name="Person")
  recPerson$addField("STRING", "code")
  recPerson$addField("STRING", "firstName")
  recPerson$addField("STRING", "lastName")
  recPerson$addField("STRING", "address")
  recPerson$addField("STRING", "stateCode")
  recPerson$addField("STRING", "city")
  recPerson$addField("STRING", "zip")
  ecl1$add(recPerson)
  dsPerson <- ECLDataset$new(name="ds_person", datasetType = recPerson, logicalFileName ="~ds::person", f
  ecl1$add(dsPerson)

## End(Not run)
```

ECLDataset-class *Class* "ECLDataset"

Description

Creates a DATASET definition. The DATASET declaration defines a file of records, on disk or in memory.

Fields

`name`: Object of class `character` Class name.

`datasetType`: Object of class `ECLRecord` Input record name.

`logicalFileName`: Object of class `character` A string constant containing the logical file name.

`fileType`: Object of class `character` One of the following keywords, optionally followed by relevant options for that specific type of file: THOR/FLAT, CSV, XML, PIPE.

`def`: Object of class `character` ECl definition/code.

Methods

`print()`: Prints the ECL code.

`addExpression(fieldName)`: Used to add ECL definitions.

`getDatasetType()`: Returns input dataset name.

`getName()`: Returns class name.

References

HPCC Systems (http://hpccsystems.com) ECL Language Reference (http://hpccsystems.com/download/docs/ecl-language-reference)

Examples

```
## Not run:
  ecl1 <- ECL$new(hostName="127.0.0.1")
  recPerson <- ECLRecord$new(name="Person")
  recPerson$addField("STRING", "code")
  recPerson$addField("STRING", "firstName")
  recPerson$addField("STRING", "lastName")
  recPerson$addField("STRING", "address")
  recPerson$addField("STRING", "stateCode")
  recPerson$addField("STRING", "city")
  recPerson$addField("STRING", "zip")
  ecl1$add(recPerson)
  dsPerson <- ECLDataset$new(name="ds_person", datasetType = recPerson, logicalFileName ="~ds::person", f
  ecl1$add(dsPerson)

## End(Not run)
```

ECLDedUp *Creates an ECL "DEDUP" definition.*

Description

The DEDUP function evaluates the recordset for duplicate records, as defined by the condition parameter, and returns a unique return set. This is similar to the DISTINCT statement in SQL. The recordset should be sorted, unless ALL is specified

References

HPCC Systems (http://hpccsystems.com) ECL Language Reference (http://hpccsystems.com/download/docs/ecl-language-reference)

Examples

```
## Not run:
  ecl1 <- ECL$new(hostName="127.0.0.1", port="8008")
  recPerson <- ECLRecord$new(name="rec_person")
  recPerson$addField("STRING", "code")
  recPerson$addField("STRING", "firstName")
  recPerson$addField("STRING", "lastName")
  recPerson$addField("STRING", "address")
  recPerson$addField("STRING", "stateCode")
  recPerson$addField("STRING", "city")
  recPerson$addField("STRING", "zip")
  ecl1$add(recPerson)

  dsPerson <- ECLDataset$new(name="ds_person", datasetType = recPerson, logicalFileName ="~ds::person", f
  ecl1$add(dsPerson)

  recPersonTable <- ECLRecord$new(name="personNewTableFormat")
  recPersonTable$addField(dsPerson$getName(), "code", seperator=".")
  recPersonTable$addField(dsPerson$getName(), "firstName", seperator=".")
  recPersonTable$addField(dsPerson$getName(), "lastName", seperator=".")

  ecl1$add(recPersonTable)
```

```
tblPerson <- ECLTable$new(name="PersonNewTable", inDataset = dsPerson, format= recPersonTable)
ecl1$add(tblPerson)

PersonNewTableSorted <- ECLSort$new(name="PersonNewTableSorted", inDataset = tblPerson)
PersonNewTableSorted$addField("lastName")
ecl1$add(PersonNewTableSorted)

mySets <- ECLDedUp$new(name="mySets", inDataset = PersonNewTableSorted)
mySets$addField("lastName")
ecl1$add(mySets)
ecl1$print()

## End(Not run)
```

ECLDedUp-class *Class* "ECLDedUp"

Description

Creates an ECL "DEDUP" definition. The DEDUP function evaluates the recordset for duplicate records, as defined by the condition parameter, and returns a unique return set. This is similar to the DISTINCT statement in SQL. The recordset should be sorted, unless ALL is specified

Fields

`name`: Object of class `character` Class name.

`inDataset`: Object of class `ECLDataset` Input record name.

`def`: Object of class `character` ECl definition/code

Methods

`print()`: Prints the ECL code.

`addField(value)`: Used to add ECL definitions.

`getName()`: Returns class name.

References

HPCC Systems (http://hpccsystems.com) ECL Language Reference (http://hpccsystems.com/download/docs/ecl-language-reference)

Examples

```
## Not run:
  ecl1 <- ECL$new(hostName="127.0.0.1", port="8010")
  recPerson <- ECLRecord$new(name="rec_person")
  recPerson$addField("STRING", "code")
  recPerson$addField("STRING", "firstName")
  recPerson$addField("STRING", "lastName")
  recPerson$addField("STRING", "address")
  recPerson$addField("STRING", "stateCode")
  recPerson$addField("STRING", "city")
```

```
recPerson$addField("STRING", "zip")
ecl1$add(recPerson)

dsPerson <- ECLDataset$new(name="ds_person", datasetType = recPerson, logicalFileName ="~ds::person", t
ecl1$add(dsPerson)

recPersonTable <- ECLRecord$new(name="personNewTableFormat")
recPersonTable$addField(dsPerson$getName(), "code", seperator=".")
recPersonTable$addField(dsPerson$getName(), "firstName", seperator=".")
recPersonTable$addField(dsPerson$getName(), "lastName", seperator=".")

ecl1$add(recPersonTable)

tblPerson <- ECLTable$new(name="PersonNewTable", inDataset = dsPerson, format= recPersonTable)
ecl1$add(tblPerson)

PersonNewTableSorted <- ECLSort$new(name="PersonNewTableSorted", inDataset = tblPerson)
PersonNewTableSorted$addField("lastName")
ecl1$add(PersonNewTableSorted)

mySets <- ECLDedUp$new(name="mySets", inDataset = PersonNewTableSorted)
mySets$addField("lastName")
ecl1$add(mySets)
ecl1$print()

## End(Not run)
```

`eclDirectCall` *Executes the ECL code on the cluster specified and returns the XML response*

Description

Executes the ECL code on the cluster specified and returns the XML response.

Usage

```
eclDirectCall(hostName, port, eclCode, clusterName)
```

Arguments

`hostName`	HPCC server hostname.
`port`	HPCC server port.
`eclCode`	ECL code you want to execute
`clusterName`	Cluster name on which the ECL code will execute.

References

HPCC Systems (http://hpccsystems.com) ECL Language Reference (http://hpccsystems.com/download/docs/ecl-language-reference)

Examples

```
## Not run:
  eclCode <- "recCountyFipsCode := RECORD
      String CountyNames;
      UNSIGNED3 CountyFipsCode;
      UNSIGNED2 StateFipsCode;
    END;

    dsCountyFips := DATASET('~seer::countyfipscode', recCountyFipsCode, CSV);
    OUTPUT(COUNT(dsCountyFips));"

    eclDirectCall(hostName="127.0.0.1", eclCode=eclCode)

## End(Not run)
```

ECLDistribute *Creates an ECL "DISTRIBUTE" definition.*

Description

Creates an ECL "DISTRIBUTE" definition. The DISTRIBUTE function re-distributes records from the recordset across all the nodes of the cluster

References

HPCC Systems (http://hpccsystems.com) ECL Language Reference (http://hpccsystems.com/download/docs/ecl-language-reference)

Examples

```
## Not run:
    ecl1 <- ECL$new(hostName="127.0.0.1", port="8010")
    recPerson <- ECLRecord$new(name="rec_person")
    recPerson$addField("STRING", "code")
    recPerson$addField("STRING", "firstName")
    recPerson$addField("STRING", "lastName")
    recPerson$addField("STRING", "address")
    recPerson$addField("STRING", "stateCode")
    recPerson$addField("STRING", "city")
    recPerson$addField("STRING", "zip")
    ecl1$add(recPerson)

    condition <- "SKEW(0.1)"
    distribute <- ECLDistribute$new(inECLRecord=recPerson, condition=condition)
    ecl1$add(distribute)
    ecl1$print()

## End(Not run)
```

ECLDistribute-class *Class* "ECLDistribute"

Description

Creates an ECL "DISTRIBUTE" definition. The DISTRIBUTE function re-distributes records from the recordset across all the nodes of the cluster

Fields

`name`: Object of class `character` Class name.

`inECLRecord`: Object of class `ECLRecord` Input record name.

`condition`: Object of class `character` ECl definition/code

Methods

`print()`: Prints the ECL code.

`getName()`: Returns class name.

References

HPCC Systems (http://hpccsystems.com) ECL Language Reference (http://hpccsystems.com/download/docs/ecl-language-reference)

Examples

```
## Not run:
   ecl1 <- ECL$new(hostName="127.0.0.1", port="8010")
   recPerson <- ECLRecord$new(name="rec_person")
   recPerson$addField("STRING", "code")
   recPerson$addField("STRING", "firstName")
   recPerson$addField("STRING", "lastName")
   recPerson$addField("STRING", "address")
   recPerson$addField("STRING", "stateCode")
   recPerson$addField("STRING", "city")
   recPerson$addField("STRING", "zip")
   ecl1$add(recPerson)

   condition <- "SKEW(0.1)"
   distribute <- ECLDistribute$new(inECLRecord=recPerson, condition=condition)
   ecl1$add(distribute)
   ecl1$print()

## End(Not run)
```

ECLIterate *Creates an ECL "ITERATE" definition.*

Description

Creates an ECL "ITERATE" definition. The ITERATE function processes through all records in the recordset one pair of records at a time, performing the transform function on each pair in turn.

References

HPCC Systems (http://hpccsystems.com) ECL Language Reference (http://hpccsystems.com/download/docs/ecl-language-reference)

Examples

```
## Not run:
  ecl1 <- ECL$new(hostName="192.168.217.128", port="8010")
  resType <- ECLRecord$new(name="rec_resType")
  resType$addField("INTEGER1", "Val")
  resType$addField("INTEGER1", "Rtot")
  ecl1$add(resType)

  dsRecords <- ECLDataset$new(name="ds_records", datasetType = resType, logicalFileName ="~ds::iterate",
  ecl1$add(dsRecords)

  iterate <- ECLIterate$new(name="ECLIterate", inDataset=dsRecords, outECLRecord=resType);
  iterate$addField("SELF.Rtot", "LEFT.Rtot+RIGHT.Val");
  iterate$addField("SELF", "RIGHT");
  ecl1$add(iterate)

  outputIterate <- ECLOutput$new(name="outputIterate", def = iterate$getName())
  ecl1$add(outputIterate)
  ecl1$print()

  xmlContent <- ecl1$execute()
  parseResults(xmlContent)

## End(Not run)
```

ECLIterate-class *Class "ECLIterate"*

Description

Creates an ECL "ITERATE" definition. The ITERATE function processes through all records in the recordset one pair of records at a time, performing the transform function on each pair in turn.

Fields

`name`: Object of class `character` Class name.

`inDataset`: Object of class `ECLDataset` Input record name.

`outECLRecord`: Object of class `ECLRecord` Output record name.

`def`: Object of class `character` ECl definition/code

Methods

`print()`: Prints the ECL code.

`addField(id, value)`: Used to add ECL definitions.

`getName()`: Returns class name.

References

HPCC Systems (http://hpccsystems.com) ECL Language Reference (http://hpccsystems.com/download/docs/ecl-language-reference)

Examples

```
## Not run:
  ecl1 <- ECL$new(hostName="192.168.217.128", port="8010")
  resType <- ECLRecord$new(name="rec_resType")
  resType$addField("INTEGER1", "Val")
  resType$addField("INTEGER1", "Rtot")
  ecl1$add(resType)

  dsRecords <- ECLDataset$new(name="ds_records", datasetType = resType, logicalFileName ="~ds::iterate",
  ecl1$add(dsRecords)

  iterate <- ECLIterate$new(name="ECLIterate", inDataset=dsRecords, outECLRecord=resType);
  iterate$addField("SELF.Rtot", "LEFT.Rtot+RIGHT.Val");
  iterate$addField("SELF", "RIGHT");
  ecl1$add(iterate)

  outputIterate <- ECLOutput$new(name="outputIterate", def = iterate$getName())
  ecl1$add(outputIterate)
  ecl1$print()

  xmlContent <- ecl1$execute()
  parseResults(xmlContent)

## End(Not run)
```

ECLJoin *Creates an ECL "JOIN" definition.*

Description

Creates an ECL "JOIN" definition. A inner join if omitted, else one of the listed types in the JOIN Types JOIN Types: INNER,LEFT OUTER,RIGHT OUTER,FULL OUTER,LEFT ONLY,RIGHT ONLY,FULL ONLY

References

HPCC Systems (http://hpccsystems.com) ECL Language Reference (http://hpccsystems.com/download/docs/ecl-language-reference)

Examples

```
## Not run:
  transfrm <- ECLTransform$new(name="transfrm", outECLRecord=rec_revenueDef);
  transfrm$addField("SELF.orderNumber", "RIGHT.orderNumber");
  transfrm$addField("SELF.prodCode", "LEFT.productCode");
  transfrm$addField("SELF.prodName", "LEFT.productName");
  transfrm$addField("SELF.revenue", "RIGHT.priceEach * RIGHT.quantityOrdered");

  joinCondition <- "LEFT.productCode=RIGHT.productCode"
  ds_revenue <- ECLJoin$new(name="ds_revenue", leftRecordSet= ds_products, rightRecordSet=ds_orderDetails
  ecl1$add(ds_revenue)
  output <- ECLOutput$new(name="output", def = ds_revenue$getName())
  ecl$add(output)
  ecl$print()
  xmlContent <- ecl$execute()
  data <- parseResults(xmlContent)

## End(Not run)
```

ECLJoin-class *Class* "ECLJoin"

Description

Creates an ECL "JOIN" definition. A inner join if omitted, else one of the listed types in the JOIN Types JOIN Types: INNER,LEFT OUTER,RIGHT OUTER,FULL OUTER,LEFT ONLY,RIGHT ONLY,FULL ONLY

Extends

Class "ECLDataset", directly.

Fields

`name`: Object of class `character` Class name.

`datasetType`: Object of class `ECLRecord` Input record name.

`logicalFileName`: Object of class `character` A string constant containing the logical file name.

`fileType`: Object of class `character` One of the following keywords, optionally followed by relevant options for that specific type of file: THOR/FLAT, CSV, XML, PIPE.

`def`: Object of class `character` ECl definition/code.

`leftRecordSet`: Object of class `ECLDataset` The left set of records to process

`rightRecordSet`: Object of class `ECLDataset` The right set of records to process

`joinCondition`: Object of class `character` An expression specifying how to match records in the leftrecset and rightrecset

`joinType`: Object of class `character` Optional. An inner join if omitted, else one of the listed types in the JOIN Types section below

Methods

`setName(value)`: Define class name.

`getName()`: Returns class name.

`print()`: Prints the ECL code.

The following methods are inherited (from the corresponding class): print ("ECLDataset"), getName ("ECLDataset"), getDatasetType ("ECLDataset"), addExpression ("ECLDataset")

References

HPCC Systems (http://hpccsystems.com) ECL Language Reference (http://hpccsystems.com/download/docs/ecl-language-reference)

Examples

```
## Not run:
  transfrm <- ECLTransform$new(name="transfrm", outECLRecord=rec_revenueDef);
  transfrm$addField("SELF.orderNumber", "RIGHT.orderNumber");
  transfrm$addField("SELF.prodCode", "LEFT.productCode");
  transfrm$addField("SELF.prodName", "LEFT.productName");
  transfrm$addField("SELF.revenue", "RIGHT.priceEach * RIGHT.quantityOrdered");

  joinCondition <- "LEFT.productCode=RIGHT.productCode"
  ds_revenue <- ECLJoin$new(name="ds_revenue", leftRecordSet= ds_products, rightRecordSet=ds_orderDetails
  ecl1$add(ds_revenue)
  output <- ECLOutput$new(name="output", def = ds_revenue$getName())
  ecl$add(output)
  ecl$print()
  xmlContent <- ecl$execute()
  data <- parseResults(xmlContent)

## End(Not run)
```

ECLOutput *Creates a ECL action "Output".*

Description

Creates a ECL action "Output". The OUTPUT action produces a recordset result from the supercomputer, based on which form and options you choose. If no file to write to is specified, the result is stored in the workunit and returned to the calling program as a data stream.

References

HPCC Systems (http://hpccsystems.com) ECL Language Reference (http://hpccsystems.com/download/docs/ecl-language-reference)

Examples

```
## Not run:
 ecl1 <- ECL$new(hostName="127.0.0.1")
 recPerson <- ECLRecord$new(name="Person")
 recPerson$addField("STRING", "code")
 recPerson$addField("STRING", "firstName")
 recPerson$addField("STRING", "lastName")
 recPerson$addField("STRING", "address")
 recPerson$addField("STRING", "stateCode")
 recPerson$addField("STRING", "city")
 recPerson$addField("STRING", "zip")
 ecl1$add(recPerson)
 dsPerson <- ECLDataset$new(name="ds_person", datasetType = recPerson, logicalFileName ="~ds::person", f
 ecl1$add(dsPerson)
 outputPerson <- ECLOutput$new(name="outputPerson", def = dsPerson$getName())
 ecl1$add(outputPerson)
 xmlContent <- ecl1$execute()
 parseResults(xmlContent)

## End(Not run)
```

ECLOutput-class *Class* "ECLOutput"

Description

Creates a ECL action "Output". The OUTPUT action produces a recordset result from the supercomputer, based on which form and options you choose. If no file to write to is specified, the result is stored in the workunit and returned to the calling program as a data stream.

Fields

`name`: Object of class `character` Returns class name.

`def`: Object of class `character` ECl definition/code.

Methods

`print()`: Prints the ECL code.

`getName()`: Returns class name.

References

HPCC Systems (http://hpccsystems.com) ECL Language Reference (http://hpccsystems.com/download/docs/ecl-language-reference)

Examples

```
## Not run:
  ecl1 <- ECL$new(hostName="127.0.0.1")
  recPerson <- ECLRecord$new(name="Person")
  recPerson$addField("STRING", "code")
  recPerson$addField("STRING", "firstName")
```

```
    recPerson$addField("STRING", "lastName")
    recPerson$addField("STRING", "address")
    recPerson$addField("STRING", "stateCode")
    recPerson$addField("STRING", "city")
    recPerson$addField("STRING", "zip")
    ecl1$add(recPerson)
    dsPerson <- ECLDataset$new(name="ds_person", datasetType = recPerson, logicalFileName ="~ds::person", f
    ecl1$add(dsPerson)
    outputPerson <- ECLOutput$new(name="outputPerson", def = dsPerson$getName())
    ecl1$add(outputPerson)
    xmlContent <- ecl1$execute()
    parseResults(xmlContent)

## End(Not run)
```

ECLProject *Creates an ECL "PROJECT" definition.*

Description

Creates an ECL "PROJECT" definition. The PROJECT function processes through all records in the recordset performing the transform function on each record in turn.

References

HPCC Systems (http://hpccsystems.com) ECL Language Reference (http://hpccsystems.com/download/docs/ecl-language-reference)

Examples

```
## Not run:
  ecl <- ECL$new(hostName="127.0.0.1")
  ecl$addImport("IMPORT STD;")
  person <- ECLRecord$new(name="Person")
  person$addField("STRING", "code")
  person$addField("STRING", "firstName")
  person$addField("STRING", "lastName")
  ecl$add(person)

  personOut <- ECLRecord$new(name="PersonOut")
  personOut$addField("STRING", "code")
  personOut$addField("STRING", "firstName")
  personOut$addField("STRING", "lastName")
  ecl$add(personOut)

  personDS <- ECLDataset$new(name="personDS", datasetType = person, logicalFileName ="~ds::person", file
  ecl$add(personDS)

  personProject <- ECLProject$new(name="PersonProject", inDataset=personDS, outECLRecord=personOut);
  personProject$addField("SELF.firstName", "Std.Str.ToUpperCase(LEFT.firstName)");
  personProject$addField("SELF", "LEFT");
  ecl$add(personProject)
  outputProject <- ECLOutput$new(name="outputProject", def = personProject$getName())
  ecl$add(outputProject)
```

```
    ecl$print()
    xmlContent <- ecl$execute()
    data <- parseResults(xmlContent)

## End(Not run)
```

ECLProject-class *Class* "ECLProject"

Description

Creates an ECL "PROJECT" definition. The PROJECT function processes through all records in the recordset performing the transform function on each record in turn.

Fields

`name`: Object of class `character` Class name.

`inDataset`: Object of class `ECLDataset` Input record name.

`outECLRecord`: Object of class `ECLRecord` Output record name.

`def`: Object of class `character` ECl definition/code.

Methods

`print()`: Prints the ECL code.

`addField(id, value)`: Used to add ECL definitions.

`getName()`: Returns class name.

References

HPCC Systems (http://hpccsystems.com) ECL Language Reference (http://hpccsystems.com/download/docs/ecl-language-reference)

Examples

```
## Not run:
  ecl <- ECL$new(hostName="127.0.0.1")
  ecl$addImport("IMPORT STD;")
  person <- ECLRecord$new(name="Person")
  person$addField("STRING", "code")
  person$addField("STRING", "firstName")
  person$addField("STRING", "lastName")
  ecl$add(person)

  personOut <- ECLRecord$new(name="PersonOut")
  personOut$addField("STRING", "code")
  personOut$addField("STRING", "firstName")
  personOut$addField("STRING", "lastName")
  ecl$add(personOut)

  personDS <- ECLDataset$new(name="personDS", datasetType = person, logicalFileName ="~ds::person", file
  ecl$add(personDS)
```

```
personProject <- ECLProject$new(name="PersonProject", inDataset=personDS, outECLRecord=personOut);
personProject$addField("SELF.firstName", "Std.Str.ToUpperCase(LEFT.firstName)");
personProject$addField("SELF", "LEFT");
ecl$add(personProject)
outputProject <- ECLOutput$new(name="outputProject", def = personProject$getName())
ecl$add(outputProject)
ecl$print()
xmlContent <- ecl$execute()
data <- parseResults(xmlContent)

## End(Not run)
```

ECLRecord *Creates an ECL "Record Set" definition.*

Description

Record layouts are Attribute definitions whose expression is a RECORD structure terminated by the END keyword. The attr name creates a user-defined value type that can be used in built-in functions and TRANSFORM function definitions. The delimiter between field definitions in a RECORD structure can be either the semi-colon (;) or a comma (,).

References

HPCC Systems (http://hpccsystems.com) ECL Language Reference (http://hpccsystems.com/download/docs/ecl-language-reference)

Examples

```
## Not run:
  ecl1 <- ECL$new(hostName="127.0.0.1")
  recPerson <- ECLRecord$new(name="Person")
  recPerson$addField("STRING", "code")
  recPerson$addField("STRING", "firstName")
  recPerson$addField("STRING", "lastName")
  recPerson$addField("STRING", "address")
  recPerson$addField("STRING", "stateCode")
  recPerson$addField("STRING", "city")
  recPerson$addField("STRING", "zip")
  ecl1$add(recPerson)

## End(Not run)
```

ECLRecord-class *Class "ECLRecord"*

Description

Creates an ECL "Record Set" definition.

Fields

`name`: Object of class `character` Class name.

`def`: Object of class `character` ECl definition/code.

Methods

`print()`: Prints the ECL code.

`addField(fieldName, fieldValue, seperator)`: Used to add ECL definitions.

`getName()`: Returns class name.

References

HPCC Systems (http://hpccsystems.com) ECL Language Reference (http://hpccsystems.com/download/docs/ecl-language-reference)

Examples

```
## Not run:
  ecl1 <- ECL$new(hostName="127.0.0.1")
  recPerson <- ECLRecord$new(name="Person")
  recPerson$addField("STRING", "code")
  recPerson$addField("STRING", "firstName")
  recPerson$addField("STRING", "lastName")
  recPerson$addField("STRING", "address")
  recPerson$addField("STRING", "stateCode")
  recPerson$addField("STRING", "city")
  recPerson$addField("STRING", "zip")
  ecl1$add(recPerson)

## End(Not run)
```

ECLRollUp *Creates an ECL "ROLLUP" definition.*

Description

Creates an ECL "ROLLUP" definition. The ROLLUP function is similar to the DEDUP function with the addition of the call to the transform function to process each duplicate record pair. This allows you to retrieve valuable information from the "duplicate" record before it's thrown away.

References

HPCC Systems (http://hpccsystems.com) ECL Language Reference (http://hpccsystems.com/download/docs/ecl-language-reference)

Examples

```
## Not run:
  ecl1 <- ECL$new(hostName="127.0.0.1", port="8008")
  recPerson <- ECLRecord$new(name="rec_person")
  recPerson$addField("STRING", "code")
  recPerson$addField("STRING", "firstName")
  recPerson$addField("STRING", "lastName")
  recPerson$addField("STRING", "address")
  recPerson$addField("STRING", "stateCode")
  recPerson$addField("STRING", "city")
  recPerson$addField("STRING", "zip")
  ecl1$add(recPerson)

  dsPerson <- ECLDataset$new(name="ds_person", datasetType = recPerson, logicalFileName ="~ds::person", t
  ecl1$add(dsPerson)

  recPersonContact <- ECLRecord$new(name="rec_myRec")
  recPersonContact$addField(dsPerson$getName(), "firstName", seperator=".")
  recPersonContact$addField(dsPerson$getName(), "lastName", seperator=".")

  ecl1$add(recPersonContact)

  tblPerson <- ECLTable$new(name="LnameTable ", inDataset = dsPerson, format= recPersonContact)
  ecl1$add(tblPerson)

  sort <- ECLSort$new(name="sortedTable", inDataset = tblPerson)
  sort$addField("firstName")
  ecl1$add(sort)

  condition <- "LEFT.firstName = RIGHT.firstName"
  rollUp <- ECLRollUp$new(name="TransformPersons ", inDataset=sort, outECLRecord=recPersonContact, condit
  rollUp$addField("SELF", "LEFT");
  ecl1$add(rollUp)
  ecl1$print()

## End(Not run)
```

`ECLRollUp-class` *Class* "ECLRollUp"

Description

Creates an ECL "ROLLUP" definition. The ROLLUP function is similar to the DEDUP function with the addition of the call to the transform function to process each duplicate record pair. This allows you to retrieve valuable information from the "duplicate" record before it's thrown away.

Fields

`name`: Object of class `character` Class name.

`inDataset`: Object of class `ECLDataset` Input record name.

`outECLRecord`: Object of class `ECLRecord` Output record name.

`condition`: Object of class `character` An expression that defines "duplicate" records. The keywords LEFT and RIGHT may be used as dataset qualifiers for fields in the recordset

`def`: Object of class `character` ECl definition/code.

Methods

`print()`: Prints the ECL code.

`addField(id, value)`: Used to add ECL definitions.

`getName()`: Returns class name.

References

HPCC Systems (http://hpccsystems.com) ECL Language Reference (http://hpccsystems.com/download/docs/ecl-language-reference)

Examples

```
## Not run:
  ecl1 <- ECL$new(hostName="127.0.0.1", port="8010")
  recPerson <- ECLRecord$new(name="rec_person")
  recPerson$addField("STRING", "code")
  recPerson$addField("STRING", "firstName")
  recPerson$addField("STRING", "lastName")
  recPerson$addField("STRING", "address")
  recPerson$addField("STRING", "stateCode")
  recPerson$addField("STRING", "city")
  recPerson$addField("STRING", "zip")
  ecl1$add(recPerson)

  dsPerson <- ECLDataset$new(name="ds_person", datasetType = recPerson, logicalFileName ="~ds::person", f
  ecl1$add(dsPerson)

  recPersonContact <- ECLRecord$new(name="rec_myRec")
  recPersonContact$addField(dsPerson$getName(), "firstName", seperator=".")
  recPersonContact$addField(dsPerson$getName(), "lastName", seperator=".")

  ecl1$add(recPersonContact)

  tblPerson <- ECLTable$new(name="LnameTable ", inDataset = dsPerson, format= recPersonContact)
  ecl1$add(tblPerson)

  sort <- ECLSort$new(name="sortedTable", inDataset = tblPerson)
  sort$addField("firstName")
  ecl1$add(sort)

  condition <- "LEFT.firstName = RIGHT.firstName"
  rollUp <- ECLRollUp$new(name="TransformPersons ", inDataset=sort, outECLRecord=recPersonContact, condit
  rollUp$addField("SELF", "LEFT");
  ecl1$add(rollUp)
  ecl1$print()

## End(Not run)
```

ECLSort *Creates an ECL "SORT" definition.*

Description

Creates an ECL "SORT" definition. The SORT function sorts the recordset according to the values specified.

References

HPCC Systems (http://hpccsystems.com) ECL Language Reference (http://hpccsystems.com/download/docs/ecl-language-reference)

Examples

```
## Not run:
  sort <- ECLSort$new(name="sortedTable", inDataset = tblCatalog)
  sort$addField("ProdLine")
  sort$addField("ProdName")
  ecl1$add(sort)

## End(Not run)
```

ECLSort-class *Class* "ECLSort"

Description

Creates an ECL "SORT" definition. The SORT function sorts the recordset according to the values specified.

Extends

Class "ECLDataset", directly.

Fields

`name`: Object of class `character` Class name.

`datasetType`: Object of class `ECLRecord` Input record name.

`logicalFileName`: Object of class `character` A string constant containing the logical file name.

`fileType`: Object of class `character` One of the following keywords, optionally followed by relevant options for that specific type of file: THOR/FLAT, CSV, XML, PIPE.

`def`: Object of class `character` ECl definition/code.

`inDataset`: Object of class `ECLDataset` Input record name.

Methods

`addField(value)`: Used to add ECL definitions.

`getName()`: Returns class name.

`print()`: Prints the ECL code.

The following methods are inherited (from the corresponding class): print ("ECLDataset"), getName ("ECLDataset"), getDatasetType ("ECLDataset"), addExpression ("ECLDataset")

References

HPCC Systems (http://hpccsystems.com) ECL Language Reference (http://hpccsystems.com/download/docs/ecl-language-reference)

Examples

```
## Not run:
  sort <- ECLSort$new(name="sortedTable", inDataset = tblCatalog)
  sort$addField("ProdLine")
  sort$addField("ProdName")
  ecl1$add(sort)

## End(Not run)
```

ECLTable *Creates an ECL "TABLE" definition.*

Description

Creates an ECL "TABLE" definition. The TABLE function is similar to OUTPUT, but instead of writing records to a file, it outputs those records in a new table (a new dataset in the supercomputer), in memory. The new table is temporary and exists only while the specific query that invoked it is running.

References

HPCC Systems (http://hpccsystems.com) ECL Language Reference (http://hpccsystems.com/download/docs/ecl-language-reference)

Examples

```
## Not run:
    ecl1 <- ECL$new(hostName="127.0.0.1")
    recPerson <- ECLRecord$new(name="rec_person")
    recPerson$addField("STRING", "code")
    recPerson$addField("STRING", "firstName")
    recPerson$addField("STRING", "lastName")
    recPerson$addField("STRING", "address")
    recPerson$addField("STRING", "stateCode")
    recPerson$addField("STRING", "city")
    recPerson$addField("STRING", "zip")
    ecl1$add(recPerson)

    dsPerson <- ECLDataset$new(name="ds_person", datasetType = recPerson, logicalFileName ="~ds::person"
    ecl1$add(dsPerson)

    recPersonTable <- ECLRecord$new(name="personNewTableFormat")
    recPersonTable$addField(dsPerson$getName(), "code", seperator=".")
    recPersonTable$addField(dsPerson$getName(), "firstName", seperator=".")
    recPersonTable$addField(dsPerson$getName(), "lastName", seperator=".")

    ecl1$add(recPersonTable)
```

```
tblPerson <- ECLTable$new(name="PersonNewTable", inDataset = dsPerson, format= recPersonTable)
ecl1$add(tblPerson)

## End(Not run)
```

ECLTable-class *Class* "ECLTable"

Description

Creates an ECL "TABLE" definition. The TABLE function is similar to OUTPUT, but instead of writing records to a file, it outputs those records in a new table (a new dataset in the supercomputer), in memory. The new table is temporary and exists only while the specific query that invoked it is running.

Extends

Class "ECLDataset", directly.

Fields

name: Object of class character Class name.

datasetType: Object of class ECLRecord Input record name.

logicalFileName: Object of class character A string constant containing the logical file name.

fileType: Object of class character One of the following keywords, optionally followed by relevant options for that specific type of file: THOR/FLAT, CSV, XML, PIPE.

def: Object of class character ECl definition/code.

inDataset: Object of class ECLDataset The set of records to process. This may be the name of a dataset or a record set derived from some filter condition, or any expression that results in a derived record set.

format: Object of class ECLRecord An output RECORD structure definition that defines the type, name, and source of the data for each field

Methods

getName(): Returns class name.

print(): Prints the ECL code.

The following methods are inherited (from the corresponding class): print ("ECLDataset"), getName ("ECLDataset"), getDatasetType ("ECLDataset"), addExpression ("ECLDataset")

References

HPCC Systems (http://hpccsystems.com) ECL Language Reference (http://hpccsystems.com/download/docs/ecl-language-reference)

Examples

```
## Not run:
   ecl1 <- ECL$new(hostName="127.0.0.1")
   recPerson <- ECLRecord$new(name="rec_person")
   recPerson$addField("STRING", "code")
   recPerson$addField("STRING", "firstName")
   recPerson$addField("STRING", "lastName")
   recPerson$addField("STRING", "address")
   recPerson$addField("STRING", "stateCode")
   recPerson$addField("STRING", "city")
   recPerson$addField("STRING", "zip")
   ecl1$add(recPerson)

   dsPerson <- ECLDataset$new(name="ds_person", datasetType = recPerson, logicalFileName ="~ds::person"
   ecl1$add(dsPerson)

   recPersonTable <- ECLRecord$new(name="personNewTableFormat")
   recPersonTable$addField(dsPerson$getName(), "code", seperator=".")
   recPersonTable$addField(dsPerson$getName(), "firstName", seperator=".")
   recPersonTable$addField(dsPerson$getName(), "lastName", seperator=".")

   ecl1$add(recPersonTable)

   tblPerson <- ECLTable$new(name="PersonNewTable", inDataset = dsPerson, format= recPersonTable)
   ecl1$add(tblPerson)

## End(Not run)
```

ECLTOPN *Returns the first count number of records in the sorts order from the recordset.*

Description

Returns the first count number of records in the sorts order from the recordset.

References

HPCC Systems (http://hpccsystems.com) ECL Language Reference (http://hpccsystems.com/download/docs/ecl-language-reference)

Examples

```
## Not run:
  topn <- ECLTOPN$new(name="T1", inDataset = dsRecords, count="5")
  topn$addField("-Rtot")
  ecl1$add(iterate)

## End(Not run)
```

ECLTOPN-class *Class* "ECLTOPN"

Description

Returns the first count number of records in the sorts order from the recordset.

Extends

Class "ECLDataset", directly.

Fields

`name`: Object of class `character` Class name.

`datasetType`: Object of class `ECLRecord` Input record name.

`logicalFileName`: Object of class `character` A string constant containing the logical file name.

`fileType`: Object of class `character` One of the following keywords, optionally followed by relevant options for that specific type of file: THOR/FLAT, CSV, XML, PIPE.

`def`: Object of class `character` ECl definition/code.

`inDataset`: Object of class `ECLDataset` The set of records to process.

`count`: Object of class `character` Expression defining the number of records to return.

Methods

`addField(value)`: Used to add ECL definitions.

`getName()`: Returns class name.

`print()`: Prints the ECL code.

The following methods are inherited (from the corresponding class): print ("ECLDataset"), getName ("ECLDataset"), getDatasetType ("ECLDataset"), addExpression ("ECLDataset")

References

HPCC Systems (http://hpccsystems.com) ECL Language Reference (http://hpccsystems.com/download/docs/ecl-language-reference)

Examples

```
## Not run:
  topn <- ECLTOPN$new(name="T1", inDataset = dsRecords, count="5")
  topn$addField("-Rtot")
  ecl1$add(iterate)

## End(Not run)
```

eclToUpperCase *The ToUpperCase functions return the source string with all lower case characters converted to upper case.*

Description

The ToUpperCase functions return the source string with all lower case characters converted to upper case.

Usage

```
eclToUpperCase(value)
```

Arguments

value	Input String

Examples

```
eclToUpperCase("testString")
```

ECLTransform *Creates an ECL "TRANSFORM" definition.*

Description

Creates an ECL "TRANSFORM" definition. A TRANSFORM defines the specific operations that must occur on a record-by-record basis.

References

HPCC Systems (http://hpccsystems.com) ECL Language Reference (http://hpccsystems.com/download/docs/ecl-language-reference)

Examples

```
## Not run:
  transfrm <- ECLTransform$new(name="transfrm", outECLRecord=rec_revenueDef);
  transfrm$addField("SELF.orderNumber", "RIGHT.orderNumber");
  transfrm$addField("SELF.prodCode", "LEFT.productCode");
  transfrm$addField("SELF.prodName", "LEFT.productName");
  transfrm$addField("SELF.revenue", "RIGHT.priceEach * RIGHT.quantityOrdered");

  joinCondition <- "LEFT.productCode=RIGHT.productCode"
  ds_revenue <- ECLJoin$new(name="ds_revenue", leftRecordSet= ds_products, rightRecordSet=ds_orderDetails
  ecl1$add(ds_revenue)
  output <- ECLOutput$new(name="output", def = ds_revenue$getName())
  ecl$add(output)
  ecl$print()
  xmlContent <- ecl$execute()
  data <- parseResults(xmlContent)
```

```
    data

## End(Not run)
```

ECLTransform-class *Class* "ECLTransform"

Description

Creates an ECL "TRANSFORM" definition. A TRANSFORM defines the specific operations that must occur on a record-by-record basis.

Fields

`name`: Object of class `character` Class name.

`outECLRecord`: Object of class `ECLRecord` Output record name.

`def`: Object of class `character` ECl definition/code.

Methods

`print()`: Prints the ECL code.

`addField(id, value)`: Used to add ECL definitions.

`getName()`: Returns class name.

References

HPCC Systems (http://hpccsystems.com) ECL Language Reference (http://hpccsystems.com/download/docs/ecl-language-reference)

Examples

```
## Not run:
  transfrm <- ECLTransform$new(name="transfrm", outECLRecord=rec_revenueDef);
  transfrm$addField("SELF.orderNumber", "RIGHT.orderNumber");
  transfrm$addField("SELF.prodCode", "LEFT.productCode");
  transfrm$addField("SELF.prodName", "LEFT.productName");
  transfrm$addField("SELF.revenue", "RIGHT.priceEach * RIGHT.quantityOrdered");

  joinCondition <- "LEFT.productCode=RIGHT.productCode"
  ds_revenue <- ECLJoin$new(name="ds_revenue", leftRecordSet= ds_products, rightRecordSet=ds_orderDetails
  ecl1$add(ds_revenue)
  output <- ECLOutput$new(name="output", def = ds_revenue$getName())
  ecl$add(output)
  ecl$print()
  xmlContent <- ecl$execute()
  data <- parseResults(xmlContent)
  data

## End(Not run)
```

parseResults *Parses the XML returned from eclDirectCall().*

Description

Parses the XML returned from eclDirectCall() and allows you to download the result in either CSV or XML format

Usage

```
parseResults(xmlResult, downloadPath, format)
```

Arguments

xmlResult	XML Output returned from eclDirectCall()
downloadPath	The path where you want to file. Do not specify the file name.
format	The format of the file to download. The file can be downloaded only in CSV and XML format. If no format is specified the file is downloaded in CSV format.

References

HPCC Systems (http://hpccsystems.com) ECL Language Reference (http://hpccsystems.com/download/docs/ecl-language-reference)

Examples

```
## Not run:
  xmlContent <- eclDirectCall(hostName = "127.0.0.1", eclCode=ecl)
  data <- parseResults(xmlContent, downloadPath="C:/Temp", format="xml")

## End(Not run)
```

sprayECL *This class is used to Spray/Distribute the data across all nodes*

Description

This class contains fileds/methods that are used to upload the file(data) from the landing zone to the cluster.

References

HPCC Systems (http://hpccsystems.com) ECL Language Reference (http://hpccsystems.com/download/docs/ecl-language-reference)

Examples

```
## Not run:
  # CODE TO SPRAY FIXED LENGTH FILE
  host <- '192.168.217.132';
  spray <- sprayECL$new(name="SprayFIXED")
  spray$addField("fileType", "fixed")
  spray$addField("ip", host)
  spray$addField("recordSize", "301")
  spray$addField("clustername", "mythor")
  spray$addField("filePath", "/var/lib/HPCCSystems/mydropzone/Urinary_Cancer.txt")
  spray$addField("logicalFileName", "~seer::incidence::urinary_cancer")
  ecl <- ECL$new(hostName= host, port="8010")
  ecl$add(spray)
  ecl$print()
  ecl$execute()

  # CODE TO SPRAY VARIABLE LENGTH FILE

  host <- '192.168.217.132';
  spray <- sprayECL$new(name="SprayCSV")
  spray$addField("fileType", "csv")
  spray$addField("ip", host)
  spray$addField("clustername", "mythor")
  spray$addField("filePath", "/var/lib/HPCCSystems/mydropzone/State_FIPS_Codes.csv")
  spray$addField("logicalFileName", "~seer::statefipscode")
  ecl <- ECL$new(hostName= host, port="8010")
  ecl$add(spray)
  ecl$print()
  ecl$execute()

## End(Not run)
```

`sprayECL-class` *This class is used to Spray/Distribute the data across all nodes*

Description

This class contains fileds/methods that are used to upload the file(data) from the landing zone to the cluster.

Fields

`name`: Object of class `character` Class Name

`ip`: Object of class `character` HPCC server hostname

`port`: Object of class `character` HPCC server port

`clusterName`: Object of class `character` Cluster name on which the ECL code will execute

`filePath`: Object of class `character` File path in dropzone

`logicalFileName`: Object of class `character` Logical File Name

`fileType`: Object of class `character` CSV/FIXED/XML

`recordSize`: Object of class `character` Record Size for Fixed Length file

Methods

`print()`: Prints the ECL code.

`addField(fieldType, fieldName)`: Used to add member

`getName()`: Returns Class Name

References

HPCC Systems (http://hpccsystems.com) ECL Language Reference (http://hpccsystems.com/download/docs/ecl-language-reference)

Examples

```
## Not run:
  # CODE TO SPRAY FIXED LENGTH FILE
  host <- '192.168.217.132';
  spray <- sprayECL$new(name="SprayFIXED")
  spray$addField("fileType", "fixed")
  spray$addField("ip", host)
  spray$addField("recordSize", "301")
  spray$addField("clustername", "mythor")
  spray$addField("filePath", "/var/lib/HPCCSystems/mydropzone/Urinary_Cancer.txt")
  spray$addField("logicalFileName", "~seer::incidence::urinary_cancer")
  ecl <- ECL$new(hostName= host, port="8010")
  ecl$add(spray)
  ecl$print()
  ecl$execute()

  # CODE TO SPRAY VARIABLE LENGTH FILE

  host <- '192.168.217.132';
  spray <- sprayECL$new(name="SprayCSV")
  spray$addField("fileType", "csv")
  spray$addField("ip", host)
  spray$addField("clustername", "mythor")
  spray$addField("filePath", "/var/lib/HPCCSystems/mydropzone/State_FIPS_Codes.csv")
  spray$addField("logicalFileName", "~seer::statefipscode")
  ecl <- ECL$new(hostName= host, port="8010")
  ecl$add(spray)
  ecl$print()
  ecl$execute()

## End(Not run)
```

---

trim *This function is used to trim leading or trailing whitespaces*

Description

This function is used to trim leading or trailing whitespaces

Usage

```
trim(x)
```

Arguments

x	Input String

Examples

```
trim("testString ")
```

Index