



SPARROW.etl

Developer's Guide

Version 1.0

Saji Venugopalan
(vsaji17@yahoo.com)

Date	Version	Modified By	Description
13-Jul-2012	1.0	Saji Venugopalan	Initial version



Overview	3
Who should read this document	3
Introduction.....	3
Features	3
Prerequisite.....	4
OS.....	4
JRE	4
Dependency APIs.....	4
Component Model view.....	6
Usage Guideline	6
Core Components.....	6
DATA-EXTRACTOR Component.....	6
DB Extractor	7
CSV Extractor	8
Generic Extractor	8
JMS/MQ/MESSAGE Extractor.....	8
DATA-TRANSFORMER Component	11
Template Based Data Transformer	12
Inline Expression Transformer.....	13
DATA-WRITERS Component	16
DB writer	19
FILE CSV writer	20
JMS Writer	22
Zip Writer	23
Tar Writer.....	24
FTP Writer	24
SCP Writer	25
Email Writer	25
FILE-UTIL Writer	26
Supporter Components	28
MODULE Configuration	28
CYCLE-DEPENDENCY	30
MODULE-PARAM	30
RESOURCE Configuration	31
DATABASE	32
JMS	34
MQ.....	35
CONTEXT	35
SPRING.....	36
EJB	37
FTP	37
SCP	38
SMTP	39
SERVICE Configuration	39
DATA-PROVIDER Configuration	41
DB	42
CSV	44



PROC (Procedure)	46
DATA-LOOKUP Configuration	48
EXCEPTION-HANDLER Configuration	50
Implementation Guideline	51
Data Extractor Component	51
Data Transformer Component	52
Data Writer Component	54
Data Providers	56
Data look-ups	57
Load Balancer	58
Cycle Dependents	58
Resources	59
Pluggable Services	60
Schedulable Services	61
Exception Handler	61
Miscellaneous	63
Encryption	63
SPARROW Functions	63
Reading System/Application Properties	66
SPARROW variables	67
Global Variables	67
Appendix I	68
How to Run	68
Configuration Schema	68
Column Definition Template	68
Configuration template	69



Overview

This document focuses on how to use SPARROW components and comply with coding standards in developing application. SPARROW is a light weight high performance Java-based ETL which can work with single source and multiple destinations.

Who should read this document

This document is intended for developers who need to develop the application using SPARROW Framework. It also assumes that the developer has knowledge on the following area.

1. ETL Concepts
2. Java Transaction management
3. Java Threading concepts

Introduction

SPARROW is a Multithreaded JAVA-based Extraction, Transformation and Loading Framework. SPARROW supports a vast array of input and output formats, including text files, data sheets, and commercial and free database engines. Moreover, the transformation capabilities of SPARROW allow you to manipulate data with very few limitations. The user friendly xml based configuration allows the user to define their Extraction point - Transformation - Destination point without much complexity. The SPARROW architecture comprises of number of components which can be customized based on the user requirements, which includes DataExtractor, DataTransformer, DataWriter, LookupObject, DataProvider etc. The current version of the SPARROW Framework includes one or more implementations for most of the above Components.

Features

- Comprehensive data source and destination support.
 - MQ Messages
 - JMS
 - Flat file
 - Database via JDBC and proprietary database APIs
 - Define your own Extractor or Writer.
- Supports Template based transformations.
- Supports Store Procedure/SQL Block executions.
- Supports CS security policy in connecting DB and MQ connections.
- Flexible and user friendly OOD model XML configuration.
- Supports Transactions
 - Only for DB and Messaging
 - Supports exempt transaction.
- Supports Cache.
 - Static and Dynamic cache (Incremental cache)
- Supports Pluggable/Schedulable Service
 - Helps the developer to enhance / extend the core engine events.
 - Loosely coupled with core engine
- Exception Handling



- Declarative exception handling
- Supports Fatal Exception
- Support wildcard error description and code matching.
- Instance Locking
 - Prevents concurrent execution of the process.
 - User has the flexibility of customizing this features
- OS Supports
 - Supports Windows, Solaris and Linux Operating systems
- Limit number of records to be processed per cycle
- Limit number of cycles per run (process).
- Configurable cycle dependencies
- Configurable shutdown time
- Statistics monitor for Memory usage and Execution time

Prerequisite

OS

Tested on Sun Solaris, Windows, Linux

JRE

JRE 1.4.x

Dependency APIs

Jars	Presence	Description
jaxb-api.jar jaxb-libs.jar jaxb-ri.jar jaxb-xjc.jar	Mandatory	Used for reading the XML Configuration JAXB APIs
commons-dbcp-1.2.2.jar	Conditional	Used for Database connection pooling. API required only if RESOURCE TYPE=DB is used
log4j-1.2.12.jar	Mandatory	Used for logging
h2.jar	Conditional	Used for Caching data. API required only if DATA-PROVIDER>PARAM ("cache.type") is used
commons-pool-1.3.jar	Mandatory	Used for Object pooling.
csfb_utils.jar	Conditional	Used for enabling CS secure connection. API required only if RESOURCE TYPE=DB>PARAM ("password") is not used.
jta-1.1-classes.jar	Mandatory	Used for Transaction management.
jms.jar	Conditional	To Support JMS implementation API required only if RESOURCE TYPE=JMS or MQ is used.
activation.jar	Conditional	To support email functionality and Tar compression API required only if RESOURCE TYPE=SMTP or WRITER.TYPE="TAR" is used.
mail.jar	Conditional	To support email functionality. API required only if RESOURCE TYPE=SMTP

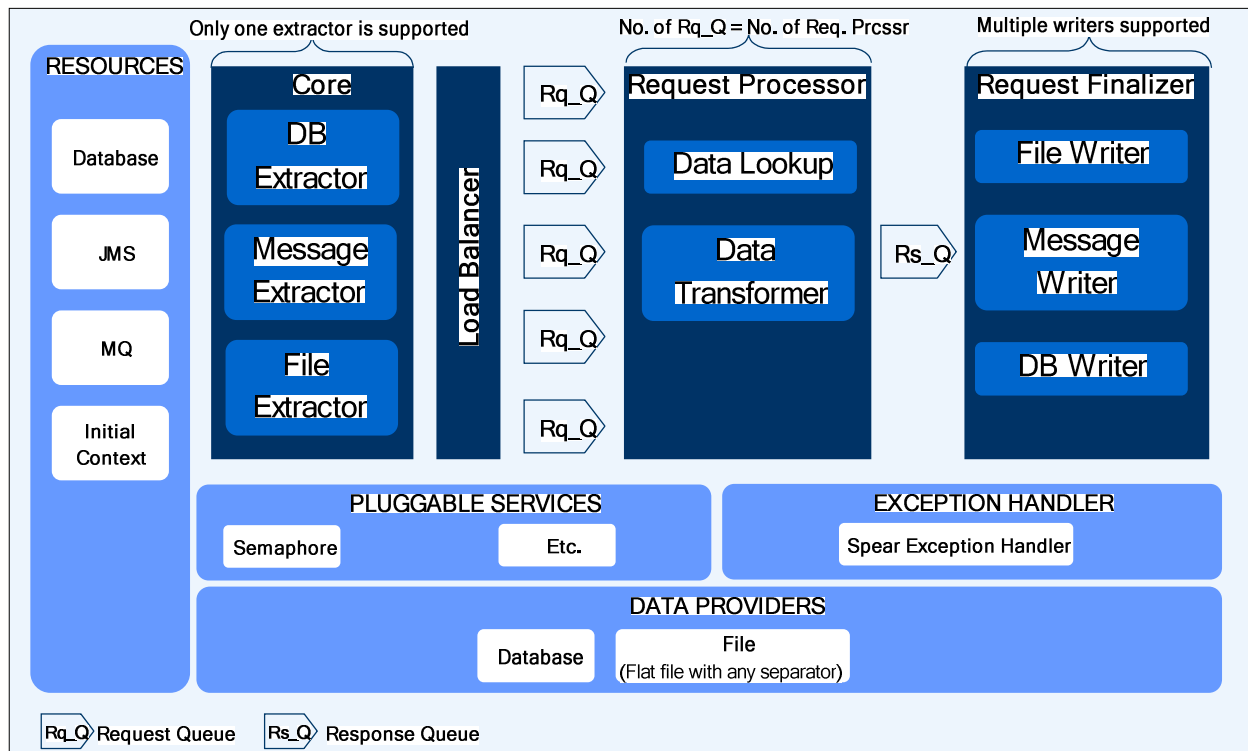


smtp.jar	Conditional	To support email functionality. API required only if RESOURCE TYPE=SMTP
mvel.jar	Conditional	API required only if <PARAM NAME="script.lang" VALUE="mvel">
janino.jar	Conditional	API required only if <PARAM NAME="script.lang" VALUE="java">
spring.jar	Conditional	API required only if RESOURCE TYPE=SPRING
commons-logging.jar	Conditional	spring.jar uses this API for logging purpose.
jsch.jar	Conditional	API required only if RESOURCE TYPE=FTP + PARAM(channel=secure) and RESOURCE TYPE=SCP
saxpath.jar	Conditional	API Require to read XML using XML path. Applicable only with DATA-EXTRACTOR[TYPE=JMS/MQ] + Default Message listener [sparrow.etl.impl.extractor.jms.XMLMessageListener]
jdom.jar	Conditional	API Require to read XML using XML path. Applicable only with DATA-EXTRACTOR[TYPE=JMS/MQ] + Default Message listener [sparrow.etl.impl.extractor.jms.XMLMessageListener]
jaxen-jdom.jar	Conditional	API Require to read XML using XML path. Applicable only with DATA-EXTRACTOR[TYPE=JMS/MQ] + Default Message listener [sparrow.etl.impl.extractor.jms.XMLMessageListener]
jaxen-core.jar	Conditional	API Require to read XML using XML path. Applicable only with DATA-EXTRACTOR[TYPE=JMS/MQ] + Default Message listener [sparrow.etl.impl.extractor.jms.XMLMessageListener]

Note: Vendor specific JDBC drivers or JMS APIs are not provided with SPARROW framework. You have to include them in you class path if you application need it.



Component Model view



Usage Guideline

SPARROW comes with default implementations for Extractors, Transformers and Writers which can be directly used without any modifications to the code. Based on your requirement either you can customize / optimize these components or you can write your own implementation.

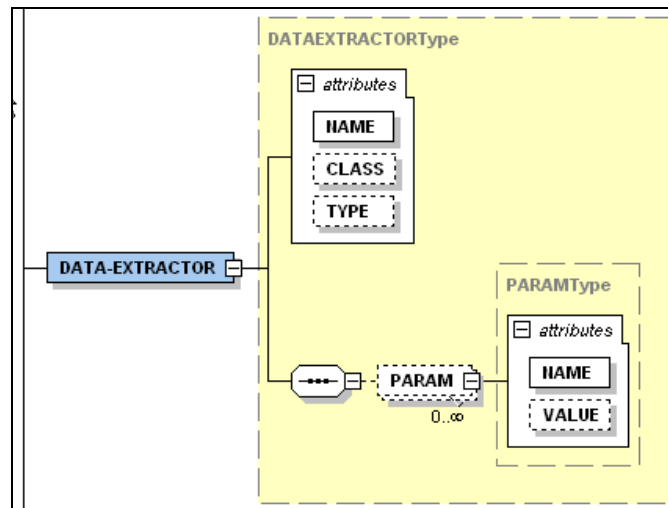
SPARROW components are categorized into 2 sections, 1) Core Components and 2) Supporter Components. Let's have a glance on these available components in SPARROW.

Core Components

Core components are domain components which represent the choice of Extractors, Transformers and Writers that you wish to configure thru SPARROW.

DATA-EXTRACTOR Component

SPARROW comes with 3 types of Data Extractors. Behavior of these extractors can be adjusted or customized by changing the parameter values in the configuration file. You can write your own writer components by implementing `sparrow.etl.core.extractor.DataExtractor` interface.



Argument	Presence	Description
NAME	Mandatory	Unique Identifier
CLASS	Optional	Class name - which implements DataExtractor interface
TYPE	Optional	DB - For Extract data from Database CSV - For Extract data from Flat file JMS - For Extract data from Message queues

DB Extractor

The class `sparrow.etl.impl.extractor.db.DBDataExtractor` provides the functionality of extracting the data from Database.

You can configure a data extractor in 3 ways.

1. By referring to an existing DATA-PROVIDER.

```
<DATA-EXTRACTOR NAME="driver" TYPE="DB">
  <PARAM NAME="data.provider" VALUE="tradefetch"/>
</DATA-EXTRACTOR>
```

2. Define the extractor parameters directly into the DATA-EXTRACTOR configuration.

```
<DATA-EXTRACTOR NAME="driver" TYPE="DB">
  <PARAM NAME="query"><![CDATA[select * from testable]]></PARAM>
  <PARAM NAME="resource" VALUE="someDBResource"/>
  <PARAM NAME="fetch.size" VALUE="500"/>
  <PARAM NAME="result.wrap" VALUE="disconnected"/>
</DATA-EXTRACTOR>
```

Note: if `data.provider` PARAM is specified, then the data extraction will happen thru the `DATA-PROVIDER` and the other parameters specified will be ignored.

Please refer to [DATA-PROVIDER](#) Section to see the explanation for the PARAM tags.



CSV Extractor

The class `sparrow.etl.impl.extractor.csv.CSVDataExtractor` provides the functionality of extracting the data from CSV file (any flat file format).

Similar to DB Extractor, you can configure the CSV Extractor in 2 ways.

- 1) By referring an existing DATA-PROVIDER name or
- 2) By configuring the parameters directly into the Extractor configuration. Explained below.

```
<DATA-EXTRACTOR NAME="driver" TYPE="CSV">
  <PARAM NAME="fetch.size" VALUE="500"/>
  <PARAM NAME="result.wrap" VALUE="disconnected"/>
  <PARAM NAME="file.path" VALUE="C:/app/temp/CSV"/>
  <PARAM NAME="file.name" VALUE="br_inv_invoice.csv"/>
  <PARAM NAME="file.delimiter" VALUE="comma"/>
  <PARAM NAME="column.definition" VALUE="csvcoldef_1.xml"/>
  <PARAM NAME="trim.value" VALUE="true"/>
  <PARAM NAME="start.line.number" VALUE="2"/>
  <PARAM NAME="validation.required" VALUE="false"/>
  <PARAM NAME="file.processor" VALUE="com.test.file.TestFileProcessor"/>
  <PARAM NAME="rejection.report.type" VALUE="file"/>
  <PARAM NAME="rejection.report.source" VALUE="C:/app/temp/CSV"/>
  <PARAM NAME="fail.dir" VALUE="C:/app/sparrownew/test/csv/fail"/>
  <PARAM NAME="post.process" VALUE="ignore"/>
  <PARAM NAME="post.process.dir" VALUE="C:/app/sparrownew/test/csv/processed"/>
  <PARAM NAME="wait.for.file" VALUE="true"/>
  <PARAM NAME="polling.interval" VALUE="5000"/>
  <PARAM NAME="polling.count" VALUE="5"/>
</DATA-EXTRACTOR>
```

Please refer to [DATA-PROVIDER](#) Section to see the explanation for the PARAMs.

Generic Extractor

Generic Extractor allows you to specify one or more data provider for a single data extraction. The data provider could be a CSV or DB or both.

Note : You should not use "TYPE" attribute while using Generic Extractor. If the TYPE attribute is not used, in default, the extractor becomes a Generic Extractor. Please see the example below.

```
<DATA-EXTRACTOR NAME="driver">
  <PARAM NAME="data.provider" VALUE="tradefetch1,tradefetch2"/>
</DATA-EXTRACTOR>
```

JMS/MQ/MESSAGE Extractor

The class `sparrow.etl.impl.extractor.jms.JMSDataExtractor` provides the functionality of extracting the data from Message Queue. The Messaging source can be anything that satisfies the standard JMS API.

```
<DATA-EXTRACTOR NAME="jmsextractor" TYPE="JMS">
  <PARAM NAME="message.listener" VALUE="com.test.extractor.jms.TestMessageListener"/>
```



```
<PARAM NAME="message.store.type" VALUE="file" />
<PARAM NAME="message.store.source" VALUE="C:/app/sparrownew/test/msg_store" />
<PARAM NAME="destination.name" VALUE="LOCALJMS@sparrow.test.Queue" />
<PARAM NAME="deadletter.queue" VALUE="LOCALJMS@errorJNDI" />
<PARAM NAME="column.definition" VALUE="csvcoldef.xml" />
<PARAM NAME="consumer.count" VALUE="5" />
<PARAM NAME="header.properties" VALUE="Topic" />
<PARAM NAME="retry.limit" VALUE="3" />
<PARAM NAME="error.log.dir" VALUE="C:/app/sparrownew/test/log" />
<PARAM NAME="message.type" VALUE="xml" />
<PARAM NAME="print.message" VALUE="true" />
</DATA-EXTRACTOR>
```

Argument	Presence	Description
message.listener	Optional	<p>The value for this argument must be a qualified class name which should implement <code>sparrow.etl.impl.extractor.jms.SPARROWMessageListener</code> or extends <code>AbstractSPARROWMessageListener</code> (Recommended)</p> <p>The message listener is similar to JMS message listener <code>javax.jms.MessageListener</code>, but the change in <code>SPARROWMessageListener</code> is, the <code>onMessage</code> method must return the <code>String[]</code> and it injects <code>sparrow.etl.impl.extractor.jms.SPARROWJMSMessage</code>. The reason behind this would be, the structure of the incoming message is unknown for the SPARROW Engine so the user must convert the message into <code>String[]</code> in order to the SPARROW to understand the structure.</p> <p>Default value : <code>sparrow.etl.impl.extractor.jms.GenericMessageListener</code></p> <p>This default implementation will allow you to read incoming String message without writing any message listener. Please see Appendix I : Column Definition Template section for how to map the incoming string message to a column.</p>
message.store.type	Optional	Type of the message store [<code>file DB</code>]. Current version supports only <code>file</code> .
message.store.source	Optional	If <code>message.store.type</code> is not present, this argument will be ignored. The path where the incoming messages should be stored. Stored messages will be removed once the process is successfully completed. This guarantees the recoverability of the message during the failure.
destination.name	Mandatory	<Resource Name>@<Queue Name> The Queues name on which the process suppose to be listen for messages.
deadletter.queue	Optional	<Resource Name>@<Queue Name> If the incoming message is failed during the processing, then the message will be redirected to this queue.
column.definition	Mandatory	XML file name. This should contain the data type definition for the <code>String[]</code> which would return by the <code>SPARROWMessageListener</code> implementation class. Please refer the Appendix I for the format.
consumer.count	Optional	Number of consumers that you wish to have for this Message source.

		Default value : 1
header.properties	Optional	<p>JMS header property keys to be read from the original JMS message - and to be available when you process the message. Within the your Message Listener implementation you can read these header properties thru</p> <pre>SPARROWMessage.getHeaderProperty("property name");</pre> <p>Ex:</p> <pre><PARAM NAME="header.properties" VALUE="Topic"/></pre> <p><u>Message Listener Implementation</u></p> <pre>public class TestMessageListener extends AbstractSPARROWMessageListener { public TestMessageListener(SPARROWDataExtractorConfig config) { super(config); } public String[] onMessage(SPARROWJMSMessage message) { String topic= message.getHeaderProperty("Topic"); } }</pre>
retry.limit	Optional	Number of retries for the failed message.
error.log.dir	Manadatory	<p>Directory path where you would like to log the errors/unprocessed inbound messages.</p> <p>The log file will be created when</p> <ul style="list-style-type: none"> - number of retries on the failure message is completed (or) - if there is no deadletter queue is defined. <p>File format :</p> <pre>error.log.dir + "/" + MODULE. PROCESS-ID + "_unprocessed_msg_" + "MMddyyyy" + ".log"</pre>
message.type	Optional	<p>xml : assumes the incoming message is in xml format and xpath has been provided in the column.definition file to map the tags directly to a column.</p> <p>string : assume the incoming message is in string format and one of the below token has been provided in the xpath attribute of column.definition file to map value to a column.</p> <p>@MSG , @MSGID , @INTRL_MSG_ID , @PROP:<header property key></p> <p>Please see Appendix I : Column Definition Template section for more details</p> <p>Default value : string</p>
print.message	Optional	<p>Set the value to true, if you want to print the incoming messages into the log file.</p> <p>Default value : false</p>



		Note: Enbaling this param would slowdown your process performance.
<code>on.failure</code>	Optional	<p>This parameter helps you to customize the post failure scenario, like whether you want to keep the failure messages in the store itself or remove it</p> <ul style="list-style-type: none">- after the specified number of retry is over- after publishing to dead-letter queue <p>direct.to.dq - If <code>deadletter.queue</code> is configured, then the failure message will be directed to this queue. This is the default value for <code>on.failure</code> param. If the dead-letter queue is not specified then the message will be logged to the location specified in the <code>error.log.dir</code> param</p> <p>preserve.in.store - The failed message will be stored in the message store directory specified in <code>message.store.source</code> param. These message will be picked-up for processing every time when the SPARROW process starts newly, and not on every cycle.</p>

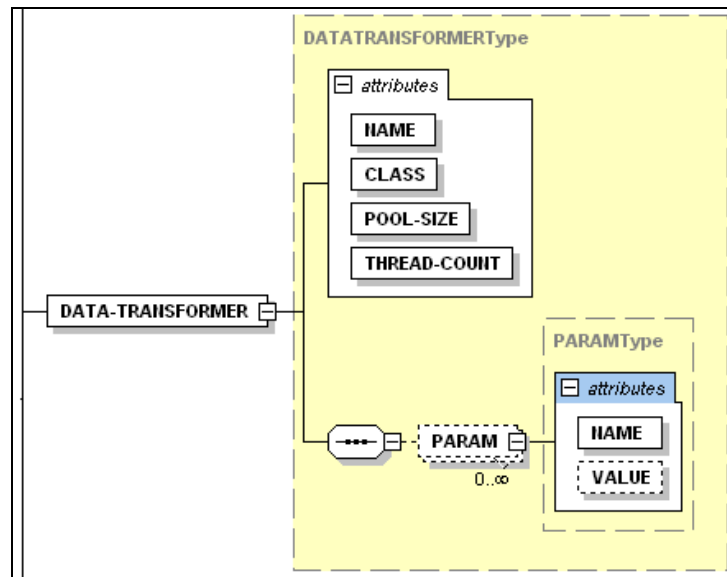
DATA-TRANSFORMER Component

Usage of this component is optional. If not used, a Dummy Data transformer will be used transparently with thread count as 5 and pool size as 50. The values can be customized as below.

```
<DATA-TRANSFORMER THREAD-COUNT="1" POOL-SIZE="10" />
```

You can write your own Transformer components by implementing `sparrow.etl.core.transformer.DataTransformer` interface or extending the abstract class `sparrow.etl.core.transformer.AbstractDataTransformer` (Recommended).

Argument	Presence	Description
NAME	Optional	Unique Identifier
CLASS	Optional	Class name - which implements <code>DataTransformer</code> or extends <code>AbstractDataTransformer</code> . Default value : DummyTransformer
POOL-SIZE	Optional	Number of active <code>DataTransformer</code> Object in a pool Default value : 50
THREAD-COUNT	Optional	No. of threads for processing the transformation request. For each request (driver record), the engine will borrow a transformation object from the pool and assign to the individual threads. Default value : 5



Template Based Data Transformer

Template based transformer can be used if your requirement is only to transform the data without any conditional operation on the extracted or look-up data. As the configuration shows below, the transformation tokens are wrapped within the dollar symbol forward by an open and closing curly brackets `${driver$INV_ID}`. During the runtime, the engine will replace these tokens with the records fetched by the extractor. You can also use the tokens which are part of look-up (reference) data. The tokens prefixed with "driver" are representing the data fetched by the extractor, whereas any token other than "driver" represent the respective look-up name. Ex: `${ccy$CUR_CODE}`, **ccy** = Lookup Name. You can also apply function on these tokens

Pattern : `${<driver/lookup name>${<column name>}`

Example : `${driver$INV_MOD_DATETIME}, ${ccy$CUR_CODE}, func_ucase(${ccy$CUR_DESC})`

Where the **driver** represents the record resulted by extractor and **ccy** represents the record from look-up. Function `func_ucase` will transform the token value to upper case. Please refer the section **FUNCTION** to see the functions supported by SPARROW Framework.

Note: "driver" is a reserved token in SPARROW framework so you cannot use this name for any look-ups.

```
<DATA-TRANSFORMER NAME="trans" TYPE="TEMPLATE">
  <PARAM NAME="expression"><![CDATA[
    ${driver$CNFRM_ID}, ${driver$CNFRM_REF_ID}, func_getdata(),
    func_lcase(${driver$DELV_TYPE}), func_ucase(${driver$DEST_ADDRESS})
  ]]></PARAM>
  <PARAM NAME="placeholder" VALUE="string"/>
  <PARAM NAME="rejection.report.type" VALUE="file"/>
  <PARAM NAME="rejection.report.source"
VALUE="C:/app/sparrownew/test/csv/report"/>
</DATA-TRANSFORMER>
```



Output:

```
1,I-1673348,17-OCT-2009,email,VENUGOPALAN.SAJI@CREDIT-SUISSE.COM
```

Argument	Presence	Description
expression	Mandatory	Use direct string expression which you would like to transform by SPARROW or Specify the file name of the template. Note: File extn. must end with .tmplt
placeholder	Optional	message : the transformed value will be automatically stored under the MessageHolder instance of the DataOutputHolder object. string : the transformed value will be automatically stored under the StringHolder instance of the DataOutputHolder object. object : is the default value if this arguments is not used , the transformed value will be automatically stored under the ObjectHolder instance of the DataOutputHolder object.
rejection.report.type	Optional	file : Log the rejection details to file. db : Log the rejection details to DB (not supported).
rejection.report.source	Optional	If rejection.report.type is file then this argument should carry the path where the file should be saved. If rejection.report.type is db then this argument should carry the Data provider name. Note : This is effective only if the rejection.report.type is present.
template.processor or	Optional	Currently, SPARROW Support 2 type of template processors. 1) Built-in template processor 2) Apache velocity template processor //Default <PARAM NAME="template.processor" VALUE="sparrow"/> (Or) <PARAM NAME="template.processor" VALUE="velocity"/>

Inline Expression Transformer

SPARROW allows you to write your data enrichment logic within your sparrow configuration file itself. This will help you in keeping all your business logic w.r.t enrichment in a single file. Some of the benefits you get out of this feature are

1. No compilation requires
2. Easily adaptable scripting language
3. Easy to test
4. A single file encapsulates all information.
5. Easy to release.

```
<DATA-TRANSFORMER NAME="trans" TYPE="script" THREAD-COUNT="5">  
  <PARAM NAME="script.lang" VALUE="mvel"/>
```



```
<PARAM NAME="script.input" VALUE="inline"/>
<PARAM NAME="script.value">
<![CDATA[
    import sparrow.etl.core.exception.*;
    import sparrow.etl.core.util.*;

    dset_keyval      = _dataset.getDataSetAsKeyValue();
    System.out.println("PBD = " + _dataset.getLookupResult("PBD").getRowCount());
    PDB= dset_keyval.get("PBD$business_date");
    System.out.println("PDB = " + PDB );
    _dataset.getDataSetAsKeyValue().put("PDB",PDB);

    System.out.println("citem_num = " +
    _dataset.getLookupResult("CITEM").getRowCount());
    citem_number = dset_keyval.get("CITEM$citem_num");
    System.out.println("citem is "+citem_number);
    ]]>
</PARAM>
<PARAM NAME="rejection.report.type" VALUE="file"/>
<PARAM NAME="rejection.report.source"
VALUE="C:/app/sparrownew/test/csv/report"/>
</DATA-TRANSFORMER>
```

Currently, SPARROW supports 3 scripting language, they are

1. MVEL (Java like scripting) - Reference : <http://mvel.codehaus.org>
2. Java (JIT - Just-In-Time compiler) - Reference : <http://www.janino.net>
3. XSLT

Following variables are made available within MVEL and Janino expressions (not in xslt) for you to access several SPARROW Resources.

Variable	Description
_context	Provides handle to the <code>sparrow.etl.core.context.SPARROWContext</code> instance.
_dataset	Returns <code>sparrow.etl.core.DataSet</code> instance, it holds current Driver Row and Look-up data w.r.t to the driver row.
_config	Provides handle to the configuration element of the data transformer <code>sparrow.etl.core.config.SPARROWDataTransformerConfig</code> .
_logger	SPARROWLogger instance to log your message.
_this	Returns the current instance of the <code>sparrow.etl.impl.transformer.CustomScriptDataTransformer</code> . This class provides handle to reject the records incase of any exception and access to all the method specified in the <code>sparrow.etl.core.transformer.DataTransformer</code>
_dataout	All your processing result must be stored in this object. <code>sparrow.etl.core.vo.DataOutputHolder</code>

MVEL usage

```
<DATA-TRANSFORMER NAME="trans" TYPE="script" THREAD-COUNT="5">
  <PARAM NAME="script.lang" VALUE="mvel"/>
  <PARAM NAME="script.input" VALUE="inline"/>
  <PARAM NAME="script.value">
  <![CDATA[
    import sparrow.etl.core.exception.*;
```



```
import sparrow.etl.core.util.*;

dset_keyval      = _dataset.getDataSetAsKeyValue();
System.out.println("PBD = " + _dataset.getLookupResult("PBD").getRowCount());
PDB= dset_keyval.get("PBD$business_date");
System.out.println("PDB = " + PDB );
_dataset.getDataSetAsKeyValue().put("PDB",PDB);

    ]]>
</PARAM>
<PARAM NAME="rejection.report.type" VALUE="file"/>
<PARAM NAME="rejection.report.source"
VALUE="C:/app/sparrownew/test/csv/report"/>
</DATA-TRANSFORMER>
```

JANINO usage

```
<DATA-TRANSFORMER NAME="trans" TYPE="script" THREAD-COUNT="5">
  <PARAM NAME="script.lang" VALUE="java"/>
  <PARAM NAME="script.input" VALUE="inline"/>
  <PARAM NAME="script.value">
    <![CDATA[
      import sparrow.etl.core.exception.*;
      import sparrow.etl.core.util.*;
      import java.util.*;

      Map dset_keyval      = _dataset.getDataSetAsKeyValue();
      System.out.println("PBD = " + _dataset.getLookupResult("PBD").getRowCount());
    ]]>
  </PARAM>
  <PARAM NAME="rejection.report.type" VALUE="file"/>
  <PARAM NAME="rejection.report.source"
VALUE="C:/app/sparrownew/test/csv/report"/>
</DATA-TRANSFORMER>
```

XSLT usage

```
<DATA-TRANSFORMER NAME="trans" TYPE="script" THREAD-COUNT="1">
  <PARAM NAME="script.input" VALUE="inline"/>
  <PARAM NAME="script.lang" VALUE="xslt"/>
  <PARAM NAME="script.value"><![CDATA[<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="text" omit-xml-declaration="yes"/>
  <xsl:template match="/"><xsl:value-of disable-output-
escaping="yes" select="dataset/driver/cnfrm_id"/>,<xsl:value-of
select="dataset/driver/cnfrm_destination_desc"/></xsl:template>
  </xsl:stylesheet>]]></PARAM>
  <PARAM NAME="rejection.report.type" VALUE="file"/>
  <PARAM NAME="rejection.report.source" VALUE="c:/app/temp/sparrow"/>
</DATA-TRANSFORMER>
```

For XSLT based transformation, the dataset structure will be converted into a xml string which intern used in the xslt transformation. Please see the example below.

```
<dataset>
  <driver>
    <column1>.....</column1>
    <column2>.....</column2>
```




```
</driver>
<lookup-name1>
  <column1>.....</column1>
  <column2>.....</column2>
</looku-pname1>
<lookup-name2>
  <column1>.....</column1>
  <column2>.....</column2>
</looku-pname2>
</dataset>
```

Argument	Presence	Description
script.input	Optional	file : Script will be referred from the file specified in the script.value param inline : Script will be directly read from script.value param
script.lang	Optional	java : is the default value for this param other possible values mvel & xslt
script.value	Mandatory	Value of this param must be a file name of the script or actual script embedded within this param.
rejection.report.type	Mandatory	file : Log the rejection details to file. db : Log the rejection details to DB (not supported).
rejection.report.source	Mandatory	If rejection.report.type is file then this argument should carry the path where the file should be saved. If rejection.report.type is db then this argument should carry the Data provider name. Note : This is effective only if the rejection.report.type is present.

DATA-WRITERS Component

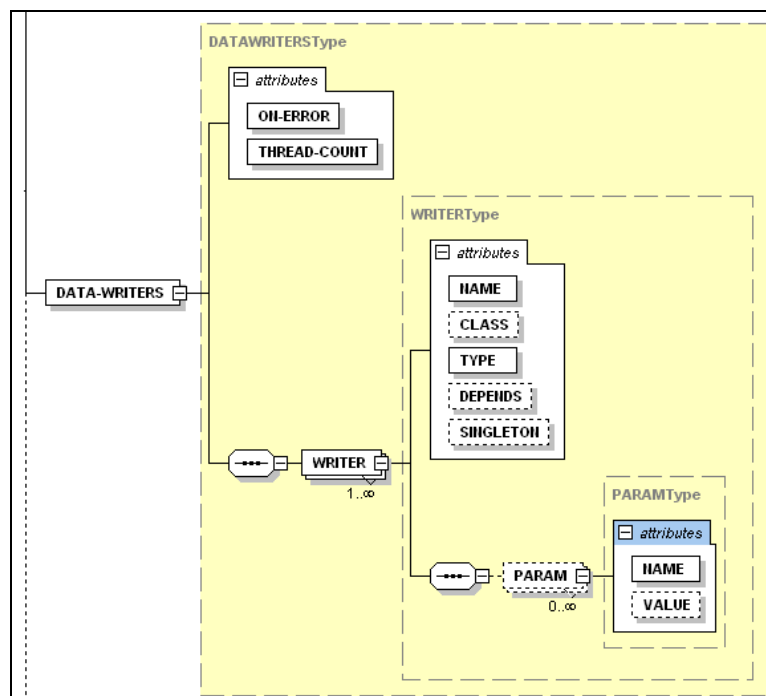
Data writers are pluggable java components which allow you to write/send/transfer data to any designated destinations. Currently, there are 7 types of data writers provided along with the SPARROW Framework. They are :

Data Writer Type	Description	Event
DB	To write data into database	Every Record, Begin Application, End Of Process, End of Application
FILE	To write data into a flat file. There is no classification of file type, because what you transform is written into a file.	Every Record
JMS/MQ	To write data into a JMS/MQ queue.	Every Record
ZIP	To zip the file(s) generated by a FILE writer or any existing file.	Begin Application, Begin Cycle, End Cycle, End Of Process, End Application



FTP	To Ftp the file(s) generated by a FILE/ZIP/TAR writer.	Begin Application, Begin Cycle, End Cycle, End Of Process, End Application
TAR	To Tar the files(s) generated by a FILE writer	Begin Application, Begin Cycle, End Cycle, End Of Process, End Application
EMAIL	To email the message with or without attachment	Begin Application, Begin Cycle, End Cycle, End Of Process, End Application
SCP	To securely copy the file from one location to another	Begin Application, Begin Cycle, End Cycle, End Of Process, End Application
FILE-UTIL	Helps you to do the file operations like rename, copy, delete, move, create file / directory etc.	Begin Application, Begin Cycle, End Cycle, End Of Process, End Application

Behavior of these extractors can be customized by changing the parameter values in the configuration file. You can write your own writer components by implementing `sparrow.etl.core.writer.DataWriter` interface or extending the abstract class `sparrow.etl.core.writer.AbstractDataWriter` (Recommended).



SPARROW allows you to configure more than one Data writer for each request. It also allows you to

- Enable transaction between the writers
- Define the number of writer threads
- Mark the writer as single instance
- Create dependency between writers.

The table below explains about each attribute defined in the DATA-WRITERS and WRITER element.



```
<DATA-WRITERS ON-ERROR="failall" THREAD-COUNT="5">
  <WRITER NAME="writer1" TYPE="DB" DEPENDS="insert1" SINGLETON="true">
    <PARAM NAME="....." VALUE="....." />
  </WRITER>
</DATA-WRITERS>
```

Argument	Presence	Description
DATA-WRITERS.ON-ERROR	Optional	<p>failall - Enables the transaction. It automatically enables the 2 phase commit if there are more than one writer are participated.</p> <p>ignore - Doesn't maintain any transaction during the writing process, hence if any one of the writer is failed it will go head with the next writer.</p> <p>Default value : ignore</p>
DATA-WRITERS.THREAD-COUNT	Optional	<p>Number of writer thread.</p> <p>Default value : 1</p>
WRITER.NAME	Mandatory	Unique identifier for the writer
WRITER.TYPE	Optional	<p>Please refer the table above.</p> <p>Default value : TEST</p>
WRITER.DEPENDS	Optional	Specify the existing writer name. If the existing writer is failed during the writing operation then the current writer will be skipped from the writing operation.
WRITER.CLASS	Optional	<p>Class name - which implements DataWriter or extends AbstractDataWriter.</p> <p>Default value : DummyWriter</p>
WRITER.SINGLETON	Optional	<p>true - Uses a single reference of this writer instance across all the threads. It automatically enables the synchronization in multi threading environment.</p> <p>false - Each thread will get a new instance of this writer.</p> <p>Default value : false</p>
WRITER.TRIGGER-ENEVT	Optional	<p>begin.app - Writer will get fired at the beginning of the application</p> <p>end.app - Writer will get fired at the end of the application</p> <p>request - Writer will get fired for all records</p> <p>begin.cycle - Writer will get fired everytime when a new cycle starts</p> <p>end.cycle - Writer will get fired everytime when a new cycle ends</p> <p>end.process - Writer will get fired if sparrow is configured with sparrow.cycle.count=-1 (i.e. the process will end when there is no record available to fetch by the data extractor)</p> <p>Default value</p>



		DB = request FILE=request TEST=begin.app JMS=request MQ=request ZIP=end.app FTP=end.app TAR=end.app EMAIL=end.app FILE-UTIL=end.app SCP=end.app
--	--	---

DB writer

The class `sparrow.etl.impl.writer.DBDataWriter` provides the functionality of writing the data into the configured DB. The DB Writer is designed to allow only update/insert/delete operation on database. It also supports SQL block and SQL Procedure execution.

As seen in DB Extractor, you can have 2 ways to configure the DB writer.

1. By referring an existing DATA-PROVIDER name or

```
<DATA-WRITERS ON-ERROR="failall" THREAD-COUNT="5">
  <WRITER NAME="writer1" TYPE="DB" DEPENDS="insert1" SINGLETON="true">
    <PARAM NAME="data.provider" VALUE="insert2"/>
    <PARAM NAME="batch.size" VALUE="25"/>
  </WRITER>
</DATA-WRITERS>
```

2) By configuring the parameters directly into the Writer configuration.

```
<DATA-WRITERS ON-ERROR="failall" THREAD-COUNT="5">
  <WRITER NAME="writer1" TYPE="DB" DEPENDS="insert1" SINGLETON="true">
    <PARAM NAME="query">
      <![CDATA[
        update testable set column_a='R' where
        column_b=${driver$INV_ID}
      ]]>
    </PARAM>
    <PARAM NAME="resource" VALUE="DSGINS50"/>
    <PARAM NAME="batch.size" VALUE="25"/>
  </WRITER>
</DATA-WRITERS>
```

Note: PARAM("query") supports sparrow function.

```
<DATA-WRITERS ON-ERROR="failall" THREAD-COUNT="5">
  <WRITER NAME="writer1" TYPE="DB" DEPENDS="insert1" SINGLETON="true">
    <PARAM NAME="query">
      <![CDATA[
        update testable set column_a='R' where
        column_b='func_trim(${driver$COLUMN})'
      ]]>
    </PARAM>
    <PARAM NAME="resource" VALUE="DSGINS50"/>
    <PARAM NAME="batch.size" VALUE="25"/>
  </WRITER>
</DATA-WRITERS>
```



```
</WRITER>
</DATA-WRITERS>
```

Argument	Presence	Description
query	Mandatory	Qualified SQL DML Note: Effective only if the <code>data.provider</code> is not used
resource	Mandatory	Name of the existing resource Note: Effective only if the <code>data.provider</code> is not used
batch.size	Optional	Value must be 2 or > 1. Enables batch updates/inserts/deletion.
data.provider	Mandatory	Existing data provider name. Note : If this parameter is specified then <code>query</code> and <code>resource</code> will be ignored.
batch.per.request	Optional	true - means, every request will contain a batch of records to be inserted/updated/deleted. false - is a default value. It denotes, there will be only one record per request.
exempt.trans	Optional	true - Writer will exempt from transaction. false - Writer will be participated in the transaction. Note: Effective only if <code>DATA-WRITERS.ON-ERROR</code> is set to <code>failall</code>

FILE | CSV writer

The class `sparrow.etl.impl.writer.FileWriter` provides the functionality of writing the data into a single file or multiple file.

```
<WRITER NAME="filewrite" TYPE="CSV">
  <PARAM NAME="file.close.event" VALUE="end.app"/>
  <PARAM NAME="file.path" VALUE="C:/app/temp/CSV"/>
  <PARAM NAME="file.name" VALUE="TEST_FILE.csv"/>
  <PARAM NAME="async.write" VALUE="true"/>
  <PARAM NAME="file.output" VALUE="multiple"/>
  <PARAM NAME="header.row.1" VALUE="HEADER,TEST,{CURRENT_DATE#yyyymmdd}" />
  <PARAM NAME="header.row.2" VALUE="col1,col1,col1,col1"/>
  <PARAM NAME="footer.row" VALUE="FOOTER,{RECORD_COUNT}" />
  <PARAM NAME="flush.type" VALUE="end.cycle"/>
  <PARAM NAME="key.name" VALUE="trans"/>
  <PARAM NAME="backup.on.exist" VALUE="true"/>
</WRITER>
```

Argument	Presence	Description
file.path	Mandatory	Location where the file needs to be created From 3.8.1 version onward, SPARROW functions can be used in this PARAM
file.name	Mandatory	Name of the output file Ex: <code>TEST_FILE.csv</code> or <code>TEST_FILE\${sometoken}.csv</code>



		From 3.8.1 version onward, SPARROW functions can be used in this PARAM
file.close.event	Optional	<p>Event when the file needs to be finalized.</p> <p>end.app - The file will be closed at the end of the application.</p> <p>end.process - The file will be closed when 0 record count return by the extractor.</p> <p>Default value : end.app</p>
async.write	Optional	<p>true - Data will be written in synchronized mode in multi threading environment.</p> <p>false - Data will be written in asynchronized mode in multi threading environment.</p>
file.output	Optional	<p>multiple - A new file will be created for each record.</p> <p>single - Single file will be used for capturing all the records.</p>
token.resolver	Optional	<p>Class name - which extends <code>sparrow.etl.core.util.GenericTokenResolver</code></p> <p>This helps you define your own tokens. Ex: If the <code>file.name</code> is given as <code>TEST_FILE\${sometoken}.csv</code> then, the value for <code>sometoken</code> can be resolved during the run time.</p> <p>The token is allowed to be used only on the following param. <code>file.name</code> <code>header.row.<n></code> <code>footer.row</code></p> <p>Default value : <code>sparrow.etl.core.util.GenericTokenResolver</code></p> <p>Reserved tokens: <code>\${CURRENT_DATE#<dateformat>}</code> <code>\${NEXT_DATE#<dateformat>}</code> <code>\${PREVIOUS_DATE#<dateformat>}</code> <code>\${RECORD_COUNT}</code> <code>\${LOCAL_HOST}</code> </p>
header.row.<n>	Optional	<p>Value will be printed directly on top of the file. Supports more than one header. Ex : <code>header.row.1</code> <code>header.row.2</code> <code>header.row.n</code> </p> <p>From 3.8.1 version onward, SPARROW functions can be used in this PARAM</p>
footer.row	Optional	<p>Value will be printed bottom of the file (last line).</p> <p>From 3.8.1 version onward, SPARROW functions can be used in this PARAM</p>
flush.type	Optional	<p>request - Data will be written to the file after every request.</p> <p>end.cycle - Data will be written to the file after getting the end.cycle notification.</p>



key.name	Mandatory	<p>Name, which used while storing the data into the DataOutputHolder instance.</p> <p>Ex : String st = "Abstc"; DataOutputHolder.addObject("abc",st);</p> <p>"abc" will be used here</p> <p>In other word, if the TemplateBasedTranformer used as Transformer then the transformer name has to be provided here as value.</p>
backup.on.exist	Optional	If the file name specified in the "file.name" is already exist, then it will be backed up before creating a new file.
create.preference	Optional	<p>always - The file will be created even if there is no record fetched resulted by the Data Extractor</p> <p>when.record.exist - The file will be created only if there any records resulted by the Data Extractor</p> <p>Default : always</p>

JMS Writer

The class `sparrow.etl.impl.writer.MessageWriter` provides the functionality of writing the data into the configured Destination.

```
<WRITER NAME="mqsend2" TYPE="JMS" DEPENDS="mqsend1">
  <PARAM NAME="key.name" VALUE="trade"/>
  <PARAM NAME="trade.msg.type" VALUE="1"/>
  <PARAM NAME="trade.msg.property.1" VALUE="Topic=/msg/sng/cash_security/jl1"/>
  <PARAM NAME="destination.name" VALUE="MQRESOURCE@FBOC_EQIT_BATS_VNE_JL01_P"/>
  <PARAM NAME="exempt.trans" VALUE="true"/>
</WRITER>
```

Argument	Presence	Description
key.name	Mandatory	<p>Name, which used while storing the data into the DataOutputHolder instance.</p> <p>Ex : MessageHolder msg = new MessageHolder(); DataOutputHolder.addMessage("trade",st);</p> <p>In other word, if the TemplateBasedTranformer used as Transformer then the transformer name has to be provided here as value.</p> <p>Note: Supports more than 1 key</p>
<key.name>.msg.type	Optional	<p>Type of the message.</p> <ul style="list-style-type: none"> 1 - Text 2 - Map 3 - Object 4 - Stream 5 - Byte



		Default value : 1
<key.name>.msg.property.<n>	Optional	Message properties. Will be set to Message Header. <code>javax.jms.Message.setStringProperty("Topic", "/msg/sng/cash_security/j11");</code>
destination.name	Mandatory	<Resource Name>@<Queue Name> Message destination where the messages need to be sent
exempt.trans	Optional	true - Writer will exempt from transaction. false - Writer will participate in the transaction. Note: Effective only if DATA-WRITERS.ON-ERROR is set to failall

Zip Writer

The class `sparrow.etl.impl.writer.ZipWriter` provides the functionality of compressing the file generated by any existing writer or any existing file. Please note that, zip writes will be invoked and executed only at the end of the process.

```
<WRITER NAME="zipwrite" TYPE="ZIP" DEPENDS="msgendl">
  <PARAM NAME="file.name" VALUE="c:/app/temp/output.zip"/>
  <PARAM NAME="backup.on.exist" VALUE="true"/>
  <PARAM NAME="preserve.file" VALUE="true"/>
  <PARAM NAME="file.list" VALUE="filewrite,c:/app/temp/sparrow-test.xml,filewrite2
"/>
</WRITER>
```

Argument	Presence	Description
file.name	Mandatory	Output file. Supports tokens, Ex: c:/app/temp/output_\${CURRENT_DATE#ddMMyyyy}.zip
backup.on.exist	Optional	If the file name specified in the "file.name" is already exist, then it will be backed up before creating a new file.
preserve.file	Optional	true - Preserves the files after zipping operation. false - Once the file is zipped the original file will be deleted. Default value : false
file.list	Mandatory	Hard coded file name or existing FILE/TAR/ZIP writer names to be picked for zip activity. Supports tokens, Ex: c:/app/temp/abc_\${CURRENT_DATE#ddMMyyyy}.csv
token.resolver	Optional	Class name - which extends <code>sparrow.etl.core.util.GenericTokenResolver</code> This helps you define your own tokens. Note : Tokens are allowed only in "file.name" & "file.list" param.



Tar Writer

The class `sparrow.etl.impl.writer.TarWriter` provides the functionality of compressing the file generated by any existing writer or any existing file. Please note that, Tar writer will be invoked and executed only at the end of the process. Tar writer inherits most of options available in the ZIP writer. In addition it has one more param called compress.

```
<WRITER NAME="tarwrite" TYPE="TAR" >
  <PARAM NAME="file.name" VALUE="c:/app/temp/output.tar"/>
  <PARAM NAME="backup.on.exist" VALUE="true"/>
  <PARAM NAME="preserve.file" VALUE="true"/>
  <PARAM NAME="compress" VALUE="true"/>
  <PARAM NAME="file.list" VALUE="filewrite,c:/app/temp/sparrow-test.xml,filewrite2
"/>
</WRITER>
```

Argument	Presence	Description
compress	optional	true - Compresses the tar content. false - Doesn't compress the content. Note: if the value is set to true , then the file.name should end with .gz . Default value : false

FTP Writer

The class `sparrow.etl.impl.writer.FTPWriter` provides the functionality of FTP the file to a configured destination. Please note that, FTP writer will be invoked and executed only at the end of the process.

```
<WRITER NAME="ftp" TYPE="FTP" DEPENDS="tar" >
  <PARAM NAME="destination.dir" VALUE="/home/abc/bindgen"/>
  <PARAM NAME="file.list" VALUE="filewrite,c:/app/temp/sparrow-test.xml,filewrite2
"/>
  <PARAM NAME="resource" VALUE="ftp" />
</WRITER>
```

Argument	Presence	Description
destination.dir	Mandatory	Directory path where your wish to place the file in the destination host
file.list	Mandatory	Hard coded file name or existing FILE/TAR/ZIP writer names to be picked for FTP. Supports tokens, Ex: <code>c:/app/temp/abc_\${CURRENT_DATE#ddMMyyyy}.csv</code>
resource	Mandatory	Existing FTP Resource Name.
token.resolver	Optional	Class name - which extends <code>sparrow.etl.core.util.GenericTokenResolver</code>



		This helps you define your own tokens. Note : Tokens are allowed only in "file.list" param.
--	--	--

SCP Writer

SCP Writer provides the functionality of transferring file thru the secure copy mode. Please note that, SCP writer will be invoked and executed only at the end of the process.

```
<WRITER NAME="scp" TYPE="SCP" DEPENDS="tar" >
  <PARAM NAME="destination.dir" VALUE="/home/abc/bindgen"/>
  <PARAM NAME="file.list" VALUE="filewrite,c:/app/temp/sparrow-test.xml,filewrite2"/>
  <PARAM NAME="resource" VALUE="myscp" />
</WRITER>
```

Argument	Presence	Description
destination.dir	Mandatory	Directory path where your wish to place the file in the destination host
file.list	Mandatory	Hard coded file name or existing FILE/TAR/ZIP writer names to be picked for FTP. Supports tokens, Ex: c:/app/temp/abc_\${CURRENT_DATE#ddMMyyyy}.csv
resource	Mandatory	Existing SCP Resource Name.
token.resolver	Optional	Class name - which extends sparrow.etl.core.util.GenericTokenResolver This helps you define your own tokens. Note : Tokens are allowed only in "file.list" param.

Email Writer

The class `sparrow.etl.impl.writer.EmailWriter` provides the functionality of sending email with/without attachment to the configured destination. Please note that, EMAIL writer will be invoked and executed only at the end of the process.

SMTP Email Example

```
<WRITER NAME="email" TYPE="EMAIL">
  <PARAM NAME="email.to" VALUE="venugopalan.saji@email.com" />
  <PARAM NAME="email.from" VALUE="venugopalan.saji@email.com" />
  <PARAM NAME="email.subject" VALUE="Test email from SPARROW" />
  <PARAM NAME="email.content" VALUE="Test email from SPARROW" />
  <PARAM NAME="email.mode" VALUE="smtp" />
  <PARAM NAME="attachment" VALUE="filewrite,c:/app/temp/sparrow-test.xml,filewrite2"/>
/>
</WRITER>
```

UNIX Email Example



```
<WRITER NAME="email" TYPE="EMAIL">
  <PARAM NAME="email.to" VALUE="venugopalan.saji@email.com" />
  <PARAM NAME="email.from" VALUE="venugopalan.saji@email.com" />
  <PARAM NAME="email.subject" VALUE="Test email from SPARROW" />
  <PARAM NAME="email.content" VALUE="Test email from SPARROW" />
  <PARAM NAME="email.mode" VALUE="unix" />
  <PARAM NAME="attachment" VALUE=" filewrite,c:/app/temp/sparrow-test.xml,filewrite2"
/>
</WRITER>
```

Note: Email Writer with **unix** mode has limitation on the number of character usage in `email.subject` and `email.content`

Argument	Presence	Description
destination.dir	Mandatory	true - Compresses the tar content. false - is the default value and doesn't compress the content.
attachment	Mandatory	Hard coded file name or existing FILE/TAR/ZIP writer names to be attached with the email. Supports tokens, Ex: <code>c:/app/temp/abc_\${CURRENT_DATE#ddMMyyyy}.csv</code>
resource	Optional	Existing SMTP Resource Name. Note: Effective only if <code>email.mode</code> is specified as SMTP
email.mode	Optional	<code>smtp</code> : Uses SMTP mode to send emails <code>unix</code> : Uses unix sendmail utility to send email Default value : unix
unix.shell	Optional	Specify which shell to be used to run the sendmail command. Default value : /bin/csh
token.resolver	Optional	Class name - which extends <code>sparrow.etl.core.util.GenericTokenResolver</code> This helps you define your own tokens. Note : Tokens are allowed only in "attachment" param.

FILE-UTIL Writer

File Util Writer contains a set of file util function like rename file, delete file, create file, create directory etc. File util writer will be invoked and executed only at the end of the process. Please note there is no separate rename and delete action for file and directory. Rename and Delete function works over file and directory.

```
<WRITER NAME="myfileutil" TYPE="FILE-UTIL">
  <PARAM NAME="action" VALUE="rename"/>
  <PARAM NAME="from.name" VALUE="c:/app/temp/sparrow-test.xml"/>
  <PARAM NAME="to.name" VALUE="c:/app/temp/sparrow-test1.xml" />
</WRITER>
```



```

<WRITER NAME="myfileutil" TYPE="FILE-UTIL">
  <PARAM NAME="action" VALUE="delete"/>
  <PARAM NAME="file.name" VALUE="c:/app/temp/sparrow-test.xml"/>
</WRITER>
<WRITER NAME="myfileutil" TYPE="FILE-UTIL">
  <PARAM NAME="action" VALUE="create.file"/>
  <PARAM NAME="file.name" VALUE="c:/app/temp/abc_feed_${CURRENT_DATE#ddMM}.flg"/>
</WRITER>
<WRITER NAME="myfileutil" TYPE="FILE-UTIL">
  <PARAM NAME="action" VALUE="copy.file"/>
  <PARAM NAME="from.name" VALUE="c:/app/temp/sparrow-test.xml"/>
  <PARAM NAME="to.name " VALUE="c:/app/temp/sparrow-test1.xml.bck" />
</WRITER>
<WRITER NAME="myfileutil" TYPE="FILE-UTIL">
  <PARAM NAME="action" VALUE="copy.file"/>
  <PARAM NAME="from.name" VALUE="c:/app/temp/sparrow-*.xml"/> //wildcard copy
  <PARAM NAME="to.name " VALUE="c:/app/temp " />
</WRITER>
<WRITER NAME="myfileutil" TYPE="FILE-UTIL">
  <PARAM NAME="action" VALUE="create.dir"/>
  <PARAM NAME="dir.name" VALUE="c:/app/temp/newdir_${CURRENT_DATE#ddMM}" />
</WRITER>
<WRITER NAME="myfileutil" TYPE="FILE-UTIL">
  <PARAM NAME="action" VALUE="copy.dir"/>
  <PARAM NAME="from.dir" VALUE="c:/app/temp/feed_${PREVIOUS_DATE#ddMM}" />
  <PARAM NAME="to.dir" VALUE="c:/app/temp/feed_${CURRENT_DATE#ddMM}" />
</WRITER>

```

Argument	Presence	Description
action	Mandatory	rename - To rename a existing file or directory delete - To delete an existing file or directory create.file - Create a new 0 byte file. copy.file - Copy file from one location to other. Supports Wild card file copying. create.dir - Create a new directory file. copy.dir - Copy a directory from one location to other. Supports recursive copying.
file.name	Conditional	File or directory name This PARAM is mandatory if the action is specified as create.file, delete
dir.name	Conditional	Directory name This PARAM is mandatory if the action is specified as create.dir
from.name	Conditional	Source file name This PARAM is mandatory if the action is specified as rename, copy.file
to.name	Conditional	Target file or directory name This PARAM is mandatory if the action is specified as rename, copy.file
from.dir	Conditional	Source directory name.



		This PARAM is mandatory if the action is specified as copy.dir
to.dir	Conditional	Target directory name This PARAM is mandatory if the action is specified as copy.dir
token.resolver	Optional	Class name - which extends <code>sparrow.etl.core.util.GenericTokenResolver</code> This helps you define your own tokens. Note : Tokens are allowed only in "file.name", "dir.name", "from.dir", "to.dir", "from.name" , "to.name" PARAM.

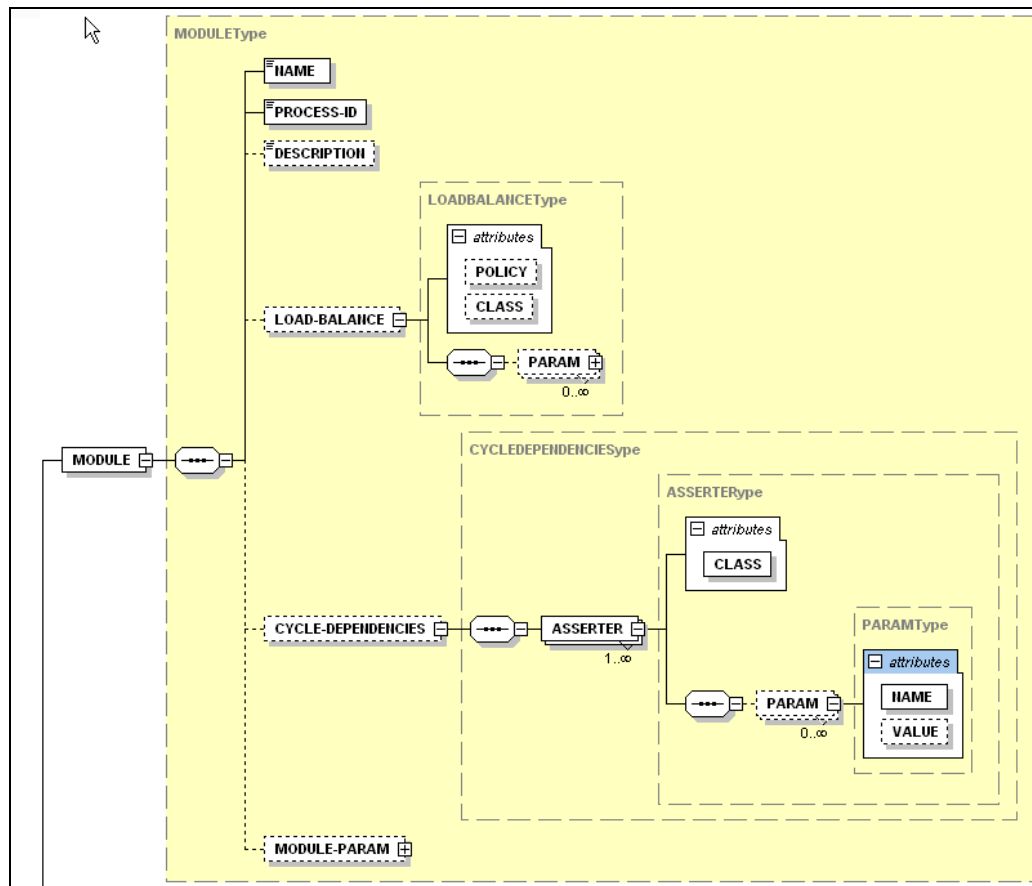
Supporter Components

Supporter components are helper components which provide additional and supportive functionality to the SPARROW Framework. Following are the key features offered by these components.

- ☞ Declarative Exception Handling,
- ☞ Pluggable services,
- ☞ Resource Management,
- ☞ Dynamic Data Look-up,
- ☞ Generic Data extraction
- ☞ Provides handle to access application parameter (Global Parameter).

MODULE Configuration

In MODULE configuration, we define the application header attributes and global parameters (which can be accessed anywhere in the application).



```

<MODULE>
  <NAME>Test Application</NAME>
  <PROCESS-ID>TEST</PROCESS-ID>
  <DESCRIPTION>For Testing</DESCRIPTION>
  <LOAD-BALANCE POLICY="ROUND-ROBIN" />
  <CYCLE-DEPENDENCIES>
    <ASSERTER CLASS="sparrow.etl.core.cycledependency.SystemGCForcer" />
    <ASSERTER CLASS="sparrow.etl.core.cycledependency.StatsCollector" />
  </CYCLE-DEPENDENCIES>
  <MODULE-PARAM>
    <PARAM NAME="sparrow.semaphore.check" VALUE="false" />
    <PARAM NAME="sparrow.cycle.interval" VALUE="5000" />
    <PARAM NAME="sparrow.shutdown" VALUE="Wed:21:58" />
    <PARAM NAME="driver.primary.keys" VALUE="INV_ID" />
    <PARAM NAME="sparrow.cycle.count" VALUE="0" />
  </MODULE-PARAM>
</MODULE>

```

Argument	Presence	Description
MODULE.NAME	Mandatory	Name of the application. This will be displayed while initializing the application.
MODULE.PROCESS-ID	Mandatory	Unique identifier for the application (Doesn't accept spaces)



MODULE.DESCRPTION	Optional	Description about the application. This value is not used anywhere in the framework
MODULE.LOAD-BALANCE	Optional	Load distribution policy.
MODULE.LOAD-BALANCE.POLICY	Optional	Currently supports only round robin model.
MODULE.LOAD-BALANCE.CLASS	Optional	You can write your implementation by using <code>sparrow.etl.core.loadbalance. RequestAssignerPolicy</code> interface Default value : ROUND-ROBIN
MODULE.CYCLE-DEPENDENCIES	Optional	Carries collection of cycle pre-dependents classes. Classes configured within this block will be notified when <code>end.cycle</code> and <code>begin.cycle</code> event are triggered You can write your implementation by using <code>sparrow.etl.core.cycledependency.CycleEventL istener</code> interface
MODULE.CYCLE-DEPENDENCIES.ASSERTER.CLASS	Optional	Class - which implements <code>sparrow.etl.core.cycledependency.CycleEventL istener</code>
MODULE.MODULE-PARAM	Optional	Carries collection of global parameters used by SPARROW Kernel. This also allows you to define your own global parameters.

CYCLE-DEPENDENCY

Argument	Presence	Description
<code>sparrow.etl.core.cycledependency.SystemGCForcer</code>	Optional	Forces GC and Forces Object finalization
<code>sparrow.etl.core.cycledependency.StatsCollector</code>	Optional	Reports the time taken for each cycle.

MODULE-PARAM

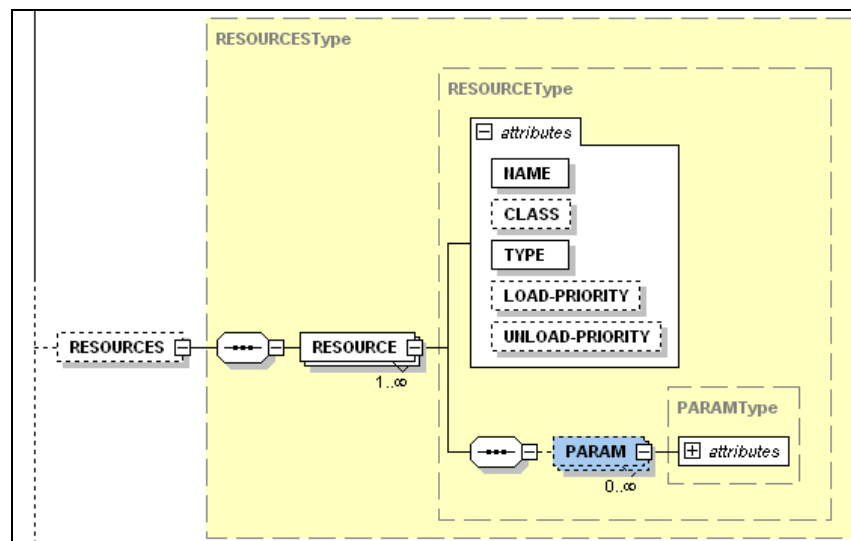
Argument	Presence	Description
<code>sparrow.semaphore.check</code>	Optional	true - Activates the semaphore check. This assumes that the SEMAPHORE pluggable service is configured/available. false - Ignores the semaphore check. Default value : false
<code>sparrow.cycle.interval</code>	Optional	Cycle intervals in milliseconds. Default value : 5000
<code>sparrow.shutdown</code>	Optional	When/what time the server should shutdown Supports <code>EEE:HH:MM</code> and <code>HH:MM</code> formats
<code>driver.primary.keys</code>	Optional	Column name(s) - contains in the Data Extractor



		<p>result. Multiple column names can be specified by (,).</p> <p>Value of the columns will be used by SPARROW logger.</p>
sparrow.cycle.count	Mandatory	<p>0 - Infinite run (Recommended If you are using sparrow.shutdown)</p> <p>-1 - Runs until the extractor results 0 record count</p> <p>0< <n> - Number of runs as defined <n>.</p>
sparrow.cache.pool.size	Optional	<p>Connection pool size for internal H2 DB</p> <p>Default value : 25</p>
sparrow.watcher.enable	Optional	<p>true - To Enable Watcher thread</p> <p>Default value : false</p>
sparrow.watcher.interval	Optional	<p>Interval in milliseconds.</p> <p>Note: This is effective only if sparrow.watcher.enable is used.</p> <p>Default value : 1000</p>

RESOURCE Configuration

SPARROW provides a functionality of centralizing the resources which are participating in an ETL process and allows you to borrow them whenever you needed it. Resources are automatically get pooled if they are pool able.



Some of the supported resource types are:

TYPE	Dependencies
Database	JDBC Driver
JMS	jms.jar, jndi.jar Vendor specific initial context factory (ex: weblogic.jar)
MQ	com.ibm.mq.jar, com.ibm.mqbind.jar, com.ibm.mqjms.jar, connector.jar, fscontext.jar, jndi.jar, jms.jar, mqcontext.jar, providerutil.jar



Initial context	jndi.jar Vendor specific initial context factory
SPRING	spring.jar + spring-<module>.jar if any
EJB	jndi.jar Vendor specific initial context factory
SMTP	mail.jar, smtp.jar, activation.jar
FTP	No Dependencies

Resource Configuration

```
<RESOURCE NAME="DSGINS50" TYPE="DB" LOAD-PRIORITY="2" UNLOAD-PRIORITY="1">  
  <PARAM NAME="abc" VALUE="....." />  
</RESOURCE>
```

Argument	Presence	Description
RESOURCE.NAME	Mandatory	Unique identifier for the resource
RESOURCE.TYPE	Mandatory	DB javax.sql.DataSource JMS javax.jms.Session MQ javax.jms.Session CONTEXT javax.naming.InitialContext SMTP javax.mail.Session FTP sparrow.etl.core.resource.FTPClientSession SCP sparrow.etl.core.resource.SCPClientSession SPRING org.springframework.context.ApplicationContext EJB javax.ejb.EJBHome
RESOURCE.LOAD-PRIORITY	Optional	Sequence when this resource should be loaded.
RESOURCE.UNLOAD-PRIORITY	Optional	Sequence when this resource should be unloaded.
RESOURCE.CLASS	Optional	Class name - which implements sparrow.etl.core.resource.GenericResourceInitializer

DATABASE

The configured Database will be initialized at the startup of the application. If the `pool.on.start` is not specified, the db connection will be established only when it is requested.

```
<RESOURCE NAME="DSGINS50" TYPE="DB" LOAD-PRIORITY="2" UNLOAD-PRIORITY="1">  
  <PARAM NAME="driver.classname" VALUE="oracle.jdbc.OracleDriver"/>  
  <PARAM NAME="connection.url" VALUE="jdbc:oracle:thin"/>  
  <PARAM NAME="user.name" VALUE="abc"/>  
  <PARAM NAME="password" VALUE="abc"/>  
  <PARAM NAME="pool.size" VALUE="25"/>  
  <PARAM NAME="pool.on.start" VALUE="true"/>  
  <PARAM NAME="max.wait" VALUE="5000"/>  
  <PARAM NAME="default.autocommit" VALUE="true"/>  
</RESOURCE>
```



Encrypt the sensitive information (Ex: clear text password) with SPARROW Encryption tool. Please see the [Encryption](#) section for more detail.

Usage:



```
Connection con = SPARROWContext.getDBConnection("DSGINS50");  
//Returns non-transactional connection  
(or)  
Connection con = (Connection)  
SPARROWContext.getResource("DSGINS50").getResource(Resource.NOT_IN_TRANSACTION);  
(or)  
Connection con = (Connection) SPARROWContext.getResource("DSGINS50").getResource()  
// If the thread is already in a transaction then it returns transaction enabled  
connection else non-transactional connection  
(or)  
Connection con = SPARROWContext.getTransactionEnabledDBConnection("DSGINS50")  
//to obtains a connection which is transactional.
```

SPARROWContext will be available thru out the application.

If the thread is already in a transaction then you can use `Resource.IN_TRANSACTION` to obtain a transaction enabled connection.

Argument	Presence	Description
driver.classname	Mandatory	The fully qualified Java class name of the JDBC driver to be used.
connection.url	Mandatory	The connection URL to be passed to our JDBC driver to establish a connection.
user.name	Mandatory	The connection username to be passed to our JDBC driver to establish a connection.
password	Optional/Mandatory	The connection password to be passed to our JDBC driver to establish a connection If this is not specified, CSFB secure connection will be enabled.
stream	Optional	Stream name for obtaining the database credential thru CSFB secure connection. <u>Note:</u> user.name will be used if this parameter is not used. This is effective only if the password is not supplied. Should use only if the DB is Oracle or Sybase
datasource	Optional	Datasource from application server Format: <Resource.Context Name>@<Datasource JNDI> <u>Note:</u> If this is used, all the other parameters will be ignored.
pool.size	Optional	The maximum number of active connections that can be allocated from this pool at the same time, or negative for no limit. Default value : 25
pool.on.start	Optional	Connection will be pooled in advance.
max.wait	Optional	The maximum number of milliseconds that the pool will wait (when there are no available connections) for a connection to be returned before throwing an



		exception, or ≤ 0 to wait indefinitely. Default value : 3000
min.idle	Optional	The minimum number of active connections that can remain idle in the pool, without extra ones being created, or 0 to create none Default value : 5
max.idle	Optional	The maximum numbers of connections that can remain idle in the pool, without extra ones being released, or negative for no limit. Default value : 10
default.autocommit	Optional	The default auto-commit state of connections created by this pool. Default value : false

JMS

The configured JMS Resource will be initialized at the startup of the application.

```
<RESOURCE NAME="LOCALJMS" TYPE="JMS" LOAD-PRIORITY="2" UNLOAD-PRIORITY="1">
  <PARAM NAME="jms.type" VALUE="queue"/>
  <PARAM NAME="provider.url" VALUE="t3://localhost:7001"/>
  <PARAM NAME="initialcontext.factory" VALUE="weblogic.jndi.WLInitialContextFactory"/>
  <PARAM NAME="connection.factory" VALUE="abc.queueconnectionfactory"/>
  <PARAM NAME="session.transacted" VALUE="true"/>
  <PARAM NAME="message.ack.type" VALUE="auto"/>
</RESOURCE>
```

Usage:

```
Session con = (Session)
SPARROWContext.getResource("LOCALJMS").getResource(Resource.NOT_IN_TRANSACTION);
```

Handler to SPARROWContext will be available thru out the application.

If the thread is already in a transaction then you can use `Resource.IN_TRANSACTION` to obtain a transaction enabled `Session` or you can use as shown below.

Argument	Presence	Description
provider.url	Mandatory	Provider URL to obtain Initial Context
initialcontext.factory	Mandatory	Initial Context Factory implementation Class name
connection.factory	Mandatory	JMS Connection Factory JNDI Name.
session.transacted	Optional	Please refer <code>javax.jms.Connection.createSession()</code> Default value : false
message.ack.type	Optional	Please refer <code>javax.jms.Connection.createSession()</code> Default value : Session.AUTO_ACKNOWLEDGE
jms.type	Optional	queue : Returns QueueSession topic : Returns TopicSession Default value : queue



context	Optional	Resource Name (Resource TYPE must be "CONTEXT" and the LOAD-PRIORITY should be less than this RESOURCE). If this is used then provider.url and initialcontext.factoy will be ignored.
---------	----------	--

MQ

MQ Resource configuration is basically inherits the parameters of JMS Resource except security.exit and security.exit.class.

```
<RESOURCE NAME="MQDEV" TYPE="MQ" LOAD-PRIORITY="2" UNLOAD-PRIORITY="1">
  <PARAM NAME="jms.type" VALUE="queue"/>
  <PARAM NAME="provider.url" VALUE="167.168.127.145:1414/SGUITD01_SVRCONN"/>
  <PARAM NAME="initialcontext.factoy" VALUE="
com.ibm.mq.jms.context.WMQInitialContextFactory"/>
  <PARAM NAME="connection.factory" VALUE="SGUITD01"/>
  <PARAM NAME="session.transacted" VALUE="true"/>
  <PARAM NAME="message.ack.type" VALUE="auto"/>
  <PARAM NAME="security.exit" VALUE="true"/>
  <PARAM NAME="security.exit.class" VALUE="
sparrow.etl.core.resource.SPARROWMQSecurityExit"/>
</RESOURCE>
```

Usage:

```
Session con = (Session)
SPARROWContext.getResource("MQDEV").getResource(Resource.NOT_IN_TRANSACTION);
```

If the thread is already in a transaction then you can use Resource.IN_TRANSACTION to obtain a transaction enabled Session or you can use as shown below.

Argument	Presence	Description
security.exit	Optional	CS MQ Security policy will be applied automatically. It uses MODULE.PROCESS_ID as the security string while communicating MQ. i.e. w3master(<PROCESS_ID>@168.158.12.15 true - Actives MQ Security Exit. false - Security exit will not be used. Default value : false
security.exit.class	Optional	Class name - which extends sparrow.etl.core.resource.SPARROWMQSecurityExit. This allows you to customize the security string. This is effective only if security.exit is set to true

CONTEXT

CONTEXT Resource type helps you to configure InitialContext once and use it where ever you need it.



```
<RESOURCE NAME="MQCTX" TYPE="CONTEXT" LOAD-PRIORITY="1" UNLOAD-PRIORITY="2">
  <PARAM NAME="provider.url" VALUE="167.168.127.145:1414/SGUITD01_SVRCONN"/>
  <PARAM NAME="initialcontext.factory"
VALUE="com.ibm.mq.jms.context.WMQInitialContextFactory"/>
</RESOURCE>
```

Usage:

```
InitialContext con = (InitialContext)
SPARROWContext.getResource("MQCTX").getResource();
```

Handle to SPARROWContext will be available thru out the application.

Argument	Presence	Description
provider.url	Mandatory	Provider URL to obtain Initial Context
initialcontext.factory	Mandatory	Initial Context Factory implementation Class name
security.principal	Optional	Username
security.credentials	Optional	Password



Encrypt the sensitive information (Ex: clear text password) with SPARROW Encryption tool. Please see the [Encryption](#) section for more detail.

Note: Always keep the LOAD-PRIORITY="1" and UNLOAD-PRIORITY as greater than the other resources for Context types. Also make sure that the LOAD-PRIORITY of the resources which uses this context is set to > 1.

SPRING

SPRING resource type allows to you to integrate the SPRING framework with SPARROW. It initializes the SPRING configuration at the start-up of the application along with the other resources.

```
<RESOURCE NAME="myspring" TYPE="SPRING" LOAD-PRIORITY="2" UNLOAD-PRIORITY="1">
  <PARAM NAME="context.file" VALUE="c:/app/temp/sparrow-spring-appcontext.xml"/>
  <PARAM NAME="context.location" VALUE="filesystem"/>
</RESOURCE>
```

Usage:

```
Object o = SPARROWContext.getBean("myspring", "mybean");
Or
ApplicationContext ac = (ApplicationContext)
SPARROWContext.getResource("myspring").getResource();
Object o = ac.getBean("mybean");
```

Where "object o" is the bean type defined against bean id="mybean" in you spring application context file.

Argument	Presence	Description
context.file	Mandatory	SPARROW Application context file. Note: context.location must be specified as filesystem, if absolute path (c:/ or /app) is



		provided for this param.
context.location	Optional	filesystem = file will be load from the absolute path of the file. classpath = file will be load from the class path. Default : classpath

EJB

EJB resource type allows you to look-up remote EJB bean thru SPARROW frame work.

```
<RESOURCE NAME="myejb" TYPE="EJB">
  <PARAM NAME="lookup" VALUE="myresource@my.jndi.name"/>
</RESOURCE>
```

Usage:

```
Object o = SPARROWContext.getBean("myejb"); (or)
Object o = SPARROWContext.getResource("myejb").getResource();
```

Where "Object o" is the javax.ejb.EJBHome instance obtained from the remote server.

Argument	Presence	Description
lookup	Mandatory	<resource>@<JNDI NAME> Where the value for the <resource> should be an existing CONTEXT resource.

FTP

FTP resource type allows you to define a FTP connection as a resource and obtain them as a connection thru SPARROWContext anywhere in the application to transfer your files.

```
<RESOURCE NAME="myftp" TYPE="FTP">
  <PARAM NAME="host.name" VALUE="sgs45a-125.sg.csfb.com"/>
  <PARAM NAME="user.name" VALUE="userftp"/>
  <PARAM NAME="password" VALUE="ftppassword"/>
</RESOURCE>
```

Usage 1:

```
sparrow.etl.core.resource.FTPClientSession o =
(sparrow.etl.core.resource.FTPClientSession )
SPARROWContext.getResource("myftp").getResource();
o.open(); // Opens FTP Connection
o.changeDirectory("/home/user/repository");
o.upload(new File("c:/app/temp/test1.csv"))
o.close();
```

Usage 2:

```
<WRITER NAME="ftp" TYPE="FTP" DEPENDS="tar" >
  <PARAM NAME="destination.dir" VALUE="/home/abc/bindgen"/>
  <PARAM NAME="file.list" VALUE="filewrite,c:/app/temp/sparrow-test.xml,filewrite2
"/>
  <PARAM NAME="resource" VALUE="myftp" />
```



</WRITER>

Argument	Presence	Description
host.name	Mandatory	FTP Host Name
user.name	Mandatory	FTP User name
password	Mandatory	FTP password
port	Optional	FTP Port Default : Non-Secure = 21 , Secure = 22
channel	Optional	secure : Enables secure File Transfer default : Default File Transfer (Non secure) Default : default



Encrypt the sensitive information (Ex: clear text password) with SPARROW Encryption tool. Please see the [Encryption](#) section for more detail.

SCP

The SCP resource type allows you to transfer file to any SCP (Secure copy) enabled UNIX destination.

```
<RESOURCE NAME="myscp" TYPE="SCP">
  <PARAM NAME="host.name" VALUE="sgs45a-125.sg.csfb.com" />
  <PARAM NAME="user.name" VALUE="userftp" />
  <PARAM NAME="password" VALUE="ftppassword" />
</RESOURCE>
```

Usage 1:

```
sparrow.etl.core.resource.SCPClientSession o =
(sparrow.etl.core.resource.SCPClientSession)
SPARROWContext.getResource("myscp").getResource();
o.open(); // Opens SCP Connection
o.upload("c:/app/temp/test1.csv", "/home/user/repository/test1.csv")
o.close();
```

Usage 2:

```
<WRITER NAME="scp" TYPE="SCP">
  <PARAM NAME="destination.dir" VALUE="/home/abc/bindgen" />
  <PARAM NAME="file.list" VALUE="filewrite,c:/app/temp/sparrow-test.xml,filewrite2
"/>
  <PARAM NAME="resource" VALUE="myscp" />
</WRITER>
```

Argument	Presence	Description
host.name	Mandatory	SCP Host Name
user.name	Mandatory	SCP User name
password	Mandatory	SCP password
port	Optional	FTP Port



		Default : 22
--	--	--------------



Encrypt the sensitive information (Ex: clear text password) with SPARROW Encryption tool. Please see the [Encryption](#) section for more detail.

SMTP

SMTP resource type allows you to define a SMTP based email connection as a resource and obtain them thru SPARROWContext anywhere in the application to send emails.

```
<RESOURCE NAME="myemail" TYPE="SMTP">
  <PARAM NAME="mail.smtp.host" VALUE="smtp-amg-ap.hedani.net"/>
  <PARAM NAME="mail.smtp.port" VALUE="25"/>
</RESOURCE>
```

Usage 1:

```
javax.mail.Session o = (javax.mail.Session)
SPARROWContext.getResource("myemail").getResource();
```

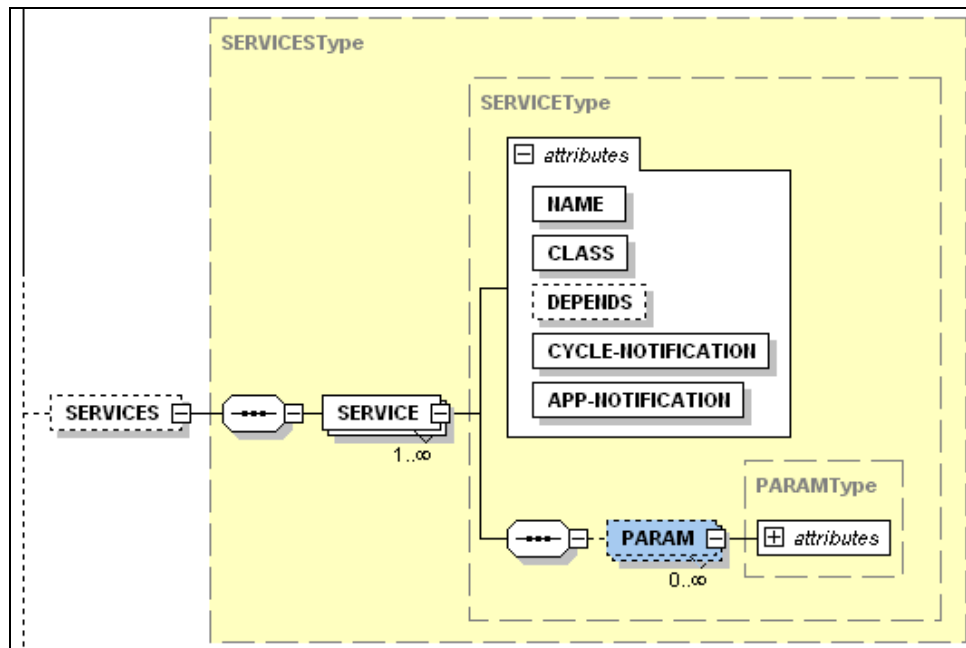
Usage 2:

```
<WRITER NAME="email" TYPE="EMAIL">
  <PARAM NAME="email.to" VALUE="venugopalan.saji@email.com" />
  <PARAM NAME="email.from" VALUE="venugopalan.saji@email.com" />
  <PARAM NAME="email.subject" VALUE="Test email from SPARROW" />
  <PARAM NAME="email.content" VALUE="Test email from SPARROW" />
  <PARAM NAME="resource" VALUE="myemail" />
  <PARAM NAME="attachment" VALUE="filewrite,c:/app/temp/sparrow-test.xml,filewrite2"
/>
</WRITER>
```

Argument	Presence	Description
mail.smtp.host	Mandatory	SMTP Host Name
mail.smtp.port	Optional	SMTP Port Number Default : 25

SERVICE Configuration

Pluggable services are optional/play 'n' play components which are useful if the user wants capture the SPARROW base events and write their requirements.



Argument	Presence	Description
SERVICE.NAME	Mandatory	Unique Identifier
SERVICE.CLASS	Mandatory	Class name - which implements <code>sparrow.etl.core.services.PluggableService</code>
SERVICE.CYCLE-NOTIFICATION	Mandatory	true -begin/end cycles events will be notified. false - No events will be notified.
SERVICE.APP-NOTIFICATION	Mandatory	true -begin/end application events will be notified. false - No events will be notified.
SERVICE.DEPENDS	Optional	An existing SERVICE.NAME

Configuring Schedulable Service:

```

<SERVICES>
<SERVICE NAME="timeservice" CLASS="com.abc.text.Service" CYCLE-NOTIFICATION="true"
APP-NOTIFICATION="true">
  <PARAM NAME="....." VALUE="....."/>
  <PARAM NAME="service.interval" VALUE="2000"/>
</SERVICE>
</SERVICES>
  
```

`com.abc.text.Service` Class will be invoked every 2000 milliseconds

Arguments:

Argument	Presence	Description
----------	----------	-------------



service.interval	Mandatory	Interval in milliseconds.
------------------	-----------	---------------------------

Example:

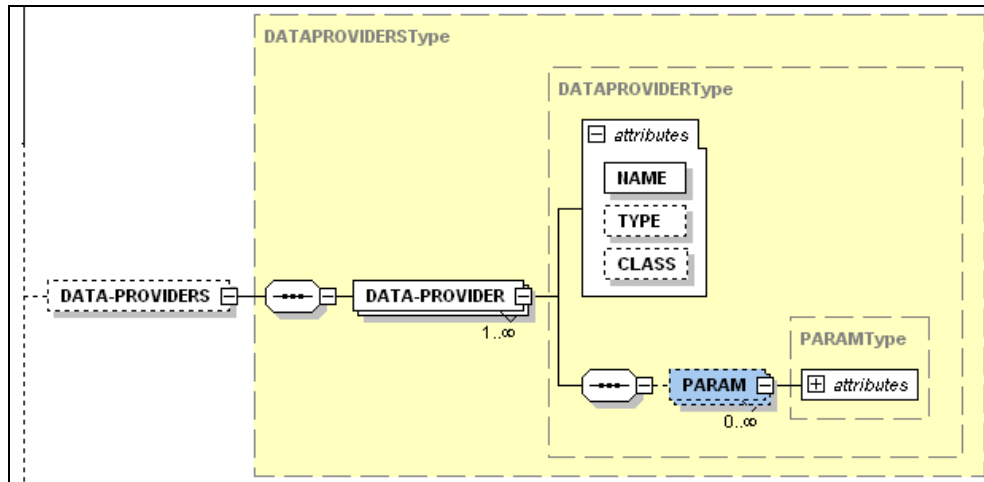
SPARROW offers a SEMAPHORE service which provides the functionality of preventing more than one SPARROW instance running in parallel. The implementation uses the database based status management to ensure the instance locking.

```
<SERVICES>
<SERVICE NAME="semaphore" CLASS="sparrow.etl.impl.services.DBSemaphoreService" CYCLE-
NOTIFICATION="true" APP-NOTIFICATION="true" DEPENDS="s1">
  <PARAM NAME="check.status" VALUE="semaphore_select"/>
  <PARAM NAME="update.status" VALUE="semaphore_update"/>
  <PARAM NAME="pre.start.status" VALUE="C"/>
  <PARAM NAME="post.start.status" VALUE="P"/>
  <PARAM NAME="end.status" VALUE="C"/>
  <PARAM NAME="termination.status" VALUE="K"/>
</SERVICE>
</SERVICES>
```

Argument	Presence	Description
check.status	Mandatory	DATA-PROVIDER name Where the status selection SQL is maintained.
update.status	Mandatory	DATA-PROVIDER name Where the status updation SQL is maintained.
pre.start.status	Mandatory	A valid status which allow the application to run
post.start.status	Mandatory	Status which should be updated after the process kicked-off
end.status	Mandatory	Status which tells the process is completed.
termination.status	Mandatory	Status which notifies the kernel to initiate a graceful shutdown

DATA-PROVIDER Configuration

Data Providers offer a functionality of extracting data from various sources. Currently it supports only DB and csv as the extraction point, but it allows you to write and plug your own data providers by implementing `sparrow.etl.core.dao.provider.DataProvider` interface.



Argument	Presence	Description
DATA-PROVIDER.NAME	Mandatory	Unique Identifier
DATA-PROVIDER.CLASS	Optional	Class name - which implements <code>sparrow.etl.core.dao.provider.DataProvider</code>
DATA-PROVIDER.TYPE	Optional	DB - Extracts the data based on the SQL provided. For DB, it also allows you to do insert/update/deletion + store proc invocation. csv - Extracts the data from the flat file.

DB

The DB Data Provider element provides the following functionality.

1. Data extraction/insertion/updation/deletion based on the SQL provided.
2. Supports store procedure call
3. Supports SPARROW Functions
4. Configurable data source (`resource`)
5. Control over fetching number of records - (Dynamically modifies the query based on the DB)
6. Choice of selecting Connected or Disconnected ResultSet
7. Supports Incremental, Event and timer based cache refreshes.

```

<DATA-PROVIDERS>
  <DATA-PROVIDER NAME="entity" TYPE="DB">
    <PARAM NAME="query"><![CDATA[select * from testable]]></PARAM>
    <PARAM NAME="resource" VALUE="DSGINS50"/>
    <PARAM NAME="fetch.size" VALUE="200"/>
    <PARAM NAME="result.wrap" VALUE="disconnected"/>
    <PARAM NAME="cache.type" VALUE="begin.app"/>
  </DATA-PROVIDER>
</DATA-PROVIDERS>

```

Argument	Presence	Description
query	Mandatory	SQL query

		<p>Note: You can use the following 2 types of tokens to pass argument to the query.</p> <p><code>\${<driver/lookup name>\${<column name>}}</code> - the value for the respective tokens will be bind as an argument in JDBC prepared statement.</p> <p>Ex: select * from table where column1=<code>'\${driver\$status_code}'</code> Result: select * from table where column1=?</p> <p><code>@{<driver/lookup name>\${<column name>}}</code> - the value for the respective tokens will be directly replaced in the query.</p> <p>Ex: select * from table where column1=<code>'@{driver\$status_code}'</code> Result: select * from table where column1=<code>'FAILED'</code></p>
resource	Mandatory	Resource name (Must be a RESOURCE with TYPE="DB")
fetch.size	Optional	<p>Number of records to be fetched.</p> <p>0 - Fetch full.</p> <p>0<<n - Fetches only <n> of records.</p>
result.wrap	Optional	<p>connected - Connected record set.</p> <p>Disconnected - Disconnected record set.</p> <p>Default value : connected</p>
use.db	Optional	<p>This param help you to specify the DB which you want to fire the query. This prevents prefixing db name in the query.</p> <p>Note : This feature works only on SQL and SYBASE resources.</p>
cache.type	Optional	<p>begin.cycle - Refreshes the cache in beginning of every cycle.</p> <p>end.cycle - Refreshes the cache in end of every cycle.</p> <p>begin.app - Refreshes the cache at the start of the application.</p> <p>Incremental - Record will be cached automatically if it doesn't exist in the cache store.</p> <p>timer.refresh - Refreshes in every specified interval.</p>
cache.flush.event	Optional	<p>end.cycle - Clears the cache in end of every cycle.</p> <p>begin.cycle - Clears the cache in beginning of every cycle.</p> <p>Note: Effective only with cache.type= Incremental</p>
lookup.keys	Optional	Specify the column names (separated by comma) to use as a unique key for the record. Comparatively, key based cache is faster than non-key based. Behind the scene, a java.util.Map based cache will be created for key based caches, for non-key based; cached records will be stored in



		a H2 memory Database. Note: Effective only if <code>cache.type</code> is specified
<code>cache.refresh.interval</code>	Optional	Refresh interval in milliseconds. Note: Effective only with <code>cache.type= timer.refresh</code> Default value : 1000

CSV

CSV Data provider provides the functionality of reading data from any type of the flat file including fixed length. The CSV Data Provider inherits all the parameters from DB Data provider except `query` and `resource`.

```
<DATA-PROVIDER NAME="abcfile" TYPE="CSV">
  <PARAM NAME="fetch.size" VALUE="500"/>
  <PARAM NAME="result.wrap" VALUE="connected"/>
  <PARAM NAME="file.path" VALUE="C:/app/temp"/>
  <PARAM NAME="file.name" VALUE="br_inv_invoice.csv"/>
  <PARAM NAME="file.delimiter" VALUE="comma"/>
  <PARAM NAME="column.definition " VALUE="csvcoldef_1.xml"/>
  <PARAM NAME="trim.value" VALUE="true"/>
  <PARAM NAME="start.line.number" VALUE="2"/>
  <PARAM NAME="validation.required" VALUE="false"/>
  <PARAM NAME="rejection.report.type" VALUE="file"/>
  <PARAM NAME="file.processor" VALUE="com.test.file.TestFileProcessor"/>
  <PARAM NAME="rejection.report.source" VALUE="C:/ report"/>
  <PARAM NAME="fail.dir" VALUE="C:/fail"/>
  <PARAM NAME="post.process" VALUE="ignore"/>
  <PARAM NAME="post.process.dir" VALUE="C:/processed"/>
  <PARAM NAME="wait.for.file" VALUE="true"/>
  <PARAM NAME="polling.interval" VALUE="5000"/>
  <PARAM NAME="polling.count" VALUE="5"/>
</DATA-PROVIDER>
```

Argument	Presence	Description
<code>file.path</code>	Mandatory	Location where the file needs to be looked-up
<code>file.name</code>	Mandatory	File name
<code>file.delimiter</code>	Mandatory	COMMA or (,) TAB or (\t) PIPE or () SEMICOLON or (;) COLON or (:) TILDE or (~) FIXEDLENGTH or (@) or Any Single Character delimiter
<code>column.definition</code>	Mandatory	XML File name - which holds Data type definition
<code>trim.value</code>	Optional	true - Trims the white space from the value false - Ignores the white space.

		Ideal to use this feature if you are reading the data from fixed length file.
start.line.number	Mandatory	Line number from where the actual records starting
file.processor	Optional	<p>Class Name - which extends FileProcessor Or CSVFileProcessor.</p> <p>This allows you to define your own TokenResolver and File validation.</p> <p>You need to override the getTokenResolver method and return your own TokenResolver implementation.</p>
validation.required	Optional	<p>true - Invokes validate method in the FileProcessor class</p> <p>false - Ignores the validation.</p> <p>Note: Process will exit if the file validation is failed.</p>
rejection.report.type	Optional	<p>file: Log the rejection details to file.</p> <p>db : Log the rejection details to DB. (Not Supported)</p>
rejection.report.source	Optional	<p>If rejection.report.type is file then this argument should carry the path where the file should be saved.</p> <p>If rejection.report.type is db then this argument should carry the Data provider name.</p> <p>Note: This is effective only if the rejection.report.type is present.</p>
fail.dir	Mandatory	Location where the file should be moved in the event of validation failure.
post.process	Optional	<p>ignore: Post process the file will be stay in file.path.</p> <p>movefile: Post process the file will be moved to post.process.dir.</p> <p>deletefile: Post process the file will be deleted.</p> <p>Default value : ignore</p>
post.process.dir	Optional	<p>Location where the file should be moved after the process completion.</p> <p>Note : Effective only with post.process= movefile</p>
wait.for.file	Optional	<p>true - If the file doesn't exist in the file.path then it will poll the directory as per the polling.interval and polling.count</p> <p>false - If the file doesn't exist in the file.path then the process will fail.</p>
polling.interval	Optional	<p>Polling interval in milliseconds</p> <p>Note : Effective only with wait.for.file=true</p>
polling.count	Optional	<p>Number of time the file needs to be looked-up</p> <p>Note : Effective only with wait.for.file=true</p>
file.processor	Optional	<p>Class name - which extends sparrow.etl.core.dao.provider.impl.CSVFileProcessor</p> <p>.</p> <p>If the user wish to validate the file then they have to create</p>



		their own class which overrides validate method.
--	--	--

PROC (Procedure)

Procedure type of data provider can be used to

- intercept the data retrieved from an existing data provider
- manipulate data for testing purpose

Procedures are written with java language with JIT (Just-In-Time) compiler help. So `janino.jar` file Note: must be present in the class path

Argument	Presence	Description
procedure	Mandatory	Write a java based procedure. The procedure must return a instance of <code>sparrow.etl.core.dao.impl.RecordSet</code> class. Please see the example below.

Usage

```
<DATA-PROVIDER NAME="records" TYPE="PROC">
  <PARAM NAME="procedure"><![CDATA[
    DisconnectedRecordSet rs1 = new DisconnectedRecordSet();
    try{
      ColumnHeader ch = new ColumnHeader(new String[]{"tablename"});
      rs1.addRow(ch,new String[]{"CONDITION"});
      rs1.addRow(ch,new String[]{"EMBARCADERO_EXPLAIN_PLAN"});
      rs1.addRow(ch,new String[]{"EXCEPTION_ATTRIBUTES"});
      rs1.addRow(ch,new String[]{"EXCEPTION_MAPPING"});
      rs1.addRow(ch,new String[]{"EXMAN_M1_JMSSTATE"});
      rs1.addRow(ch,new String[]{"EXMAN_M1_JMSSTORE"});
    }catch(DataException ex){
      throw ex;
    }
    return rs1;
  ]]></PARAM>
</DATA-PROVIDER>
```

Note: Following java import statements are included automatically, so you need not specify them explicitly.

- `sparrow.etl.core.dao.impl.RecordSet`
- `sparrow.etl.core.exception.DataException`
- `sparrow.etl.core.dao.impl.ResultRow`
- `sparrow.etl.core.dao.impl.ResultRowImpl`
- `sparrow.etl.core.dao.impl.DisconnectedRecordSet`
- `sparrow.etl.core.dao.impl.ColumnHeader`

In order to allow access to several SPARROW resource instance, following variables are available within procedure block to use.

Variable	Description
<code>_context</code>	Handle to the <code>sparrow.etl.core.context.SPARROWContext</code> instance.
<code>_dataset</code>	<code>java.util.Map</code> instance to store values for other data provider. Kind of passing argument to another "proc" data provider.



<code>_datahandler</code>	Handle to the current instance of the <code>sparrow.etl.impl.dao.ProceduralDataProvider</code> . This allows you to execute other data provider. Following methods are available for your use. <code>getData(String dpName)</code> <code>getData(String dpName, Map mapParam, List listParam)</code> <code>executeQuery(String dpName)</code> <code>executeQuery(String dpName, Map mapParam, List listParam)</code>
---------------------------	--

Usage

```
<DATA-PROVIDER NAME="procdp" TYPE="PROC">
  <PARAM NAME="procedure"><![CDATA[
    import java.util.*;

    RecordSet rs1 = null;

    try{
      rs1 = _datahandler.getData("dp1");

      Map argMap = new HashMap();
      argMap.put("token","some value");

      RecordSet rs2 = _datahandler.getData("dp2", argMap,null);

      List argLs = new ArrayList();
      argLs.add(new Integer(3));

      RecordSet rs3 = _datahandler.getData("dp3", argMap, argLs);

      rs1.merge(rs2);
      rs1.merge(rs3);

    }catch(DataException ex){
      throw ex;
    }
    return rs1;
  ]]></PARAM>
</DATA-PROVIDER>
<DATA-PROVIDER NAME="dp1" TYPE="DB">
  <PARAM NAME="query"><![CDATA[ select * from table1 ]]></PARAM>
  <PARAM NAME="resource" VALUE="DSGINS50"/>
</DATA-PROVIDER>
<DATA-PROVIDER NAME="dp2" TYPE="DB">
  <PARAM NAME="query"><![CDATA[ select * from table1 where
column1=${token1} ]]></PARAM>
  <PARAM NAME="resource" VALUE="DSGINS50"/>
</DATA-PROVIDER>
<DATA-PROVIDER NAME="dp3" TYPE="DB">
  <PARAM NAME="query"><![CDATA[ select * from table1 where column1=${token1} and
column2=? ]]></PARAM>
  <PARAM NAME="resource" VALUE="DSGINS50"/>
</DATA-PROVIDER>
```

Using Procedure Data Provider in DATA-EXTRACTOR

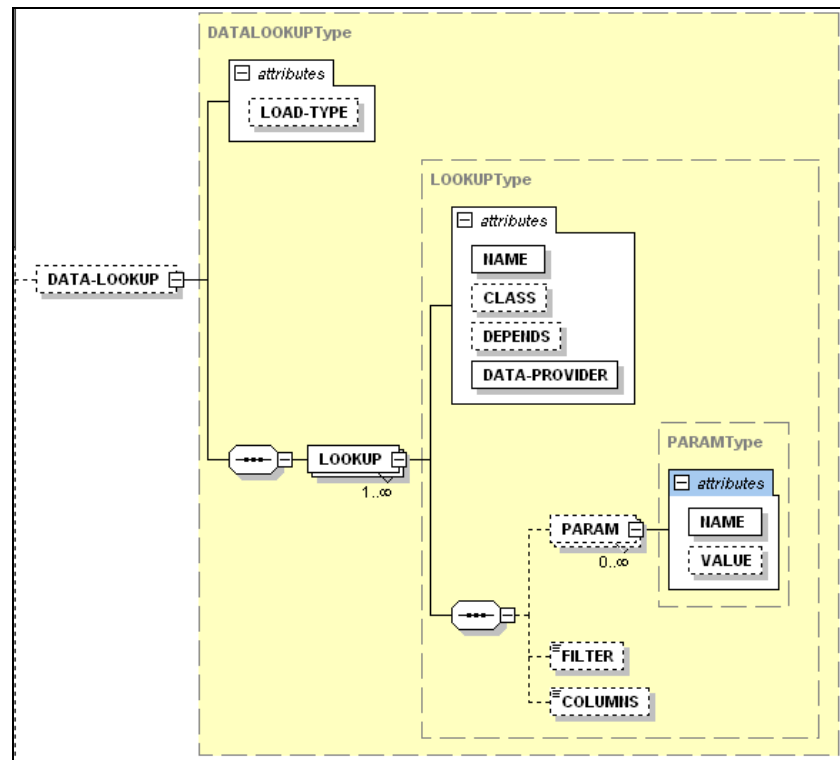
```
<DATA-EXTRACTOR NAME="driver">
  <PARAM NAME="data.provider" VALUE="procdp"/>
</DATA-EXTRACTOR>
```




Note: TYPE attribute should not be specified in the DATA=EXTRACTOR tag while using Procedure data provider.

DATA-LOOKUP Configuration

Data Look-up component provides a functionality of externalizing your data look-up operation and declare it in the configuration file.



It offers the following features.

1. Supports SQL type filter criteria. Please refer the below URL to get more idea about the supported SQL functions and features. <http://www.h2database.com/html/functions.html>
2. Supports AUTO and LAZY Lookup.
3. Allows you to choose columns that you wish to return after the filterization.
4. Supports dependencies over the look-ups
5. Supports tokens.
6. Supports SPARROW Functions.
7. Supports dynamic binding.
8. Transparent look-up over the DB and Cache.

```
<DATA-LOOKUP LOAD-TYPE="LAZY">
  <LOOKUP NAME="ccy" DATA-PROVIDER="currency" DEPENDS="cntry">
    <FILTER>
      <![CDATA[
        CUR_COU_ID=${cntry$COU_ID}
      ]]>
    </FILTER>
    <COLUMNS>
      <![CDATA[
```



```
        CUR_CODE,CUR_ID
    ]]>
</COLUMNS>
</LOOKUP>
<LOOKUP NAME="cntry" DATA-PROVIDER="country">
    <FILTER>
        <![CDATA[
            COU_ID=${driver$INV_COU_ID} AND COU_REGION_CODE='EU'
        ]]>
    </FILTER>
    <COLUMNS>
        <![CDATA[
            COU_CODE,COU_ID
        ]]>
    </COLUMNS>
</LOOKUP>
<DATA-LOOKUP>
```

Argument	Presence	Description
DATA-LOOKUP.LOAD-TYPE	Optional	auto: Automatically loads the look-up results before calling the <code>DataTransformer.enrichData</code> method. lazy: Loads only when the <code>DataSet.getLookupResult("")</code> is invoked. Ex: In <code>DataTransformer</code> implementation class. <pre>public DataOutputHolder enrichData(DataSet dataSet) { RecordSet rs = dataSet.getLookupResult("ccy"); }</pre> Default value : auto
LOOKUP.NAME	Mandatory	Unique identifier
LOOKUP.DATA-PROVIDER	Mandatory	Data provider name
LOOKUP.CLASS	Optional	Class name - which implements <code>sparrow.etl.core.lookup.LookupObject</code> This helps you to write your own implementation for the lookup operation.
LOOKUP.DEPENDS	Optional	An exiting Data look-up name
FILTER	Optional	Filter criteria. Supports SQL style where condition Supports SPARROW Functions
COLUMN	Optional	Column Names

Note: FILTER and COLUMN tags are effective only if the DATA-PROVIDER assigned to this LOOKUP is cache enabled (ie. DATA-PROVIDER->PARAM ["cache.type"] is used). An `InitializationException` will be thrown if this case is not satisfied.

If your DATA-PROVIDER is not cache enabled, then you can specify the filter criteria as part of your query itself. Please see the example below.

DATA-PROVIDER

```
<DATA-PROVIDER NAME="currencyTest" TYPE="DB">
    <PARAM NAME="query"><![CDATA[select * from testable where CURRENCY_CODE=${ccy$
```



```
CUR_CODE}}]></PARAM>
  <PARAM NAME="resource" VALUE="DSGINS50"/>
  <PARAM NAME="fetch.size" VALUE="0"/>
  <PARAM NAME="result.wrap" VALUE="disconnected"/>
</DATA-PROVIDER>
```

DATA-LOOKUP

```
<DATA-LOOKUP LOAD-TYPE="LAZY">
<LOOKUP NAME="test" DATA-PROVIDER="currentyTest" DEPENDS="ccy"/>
</DATA-LOOKUP>
```



SPARROW allows you to use custom token in DATA-LOOKUP.

Note : Supports only with LOAD-TYPE="LAZY"

```
<DATA-PROVIDER NAME="currentyTest" TYPE="DB">
  <PARAM NAME="query"><![CDATA[select * from testable where
CURRENCY_CODE=${abc}]]></PARAM>
  <PARAM NAME="resource" VALUE="DSGINS50"/>
  <PARAM NAME="fetch.size" VALUE="0"/>
  <PARAM NAME="result.wrap" VALUE="disconnected"/>
</DATA-PROVIDER>
<DATA-LOOKUP>
<LOOKUP NAME="test" DATA-PROVIDER="currentyTest"/>
</DATA-LOOKUP>
```

In you DataTransformer implementation

```
public DataOutputHolder enrichData(DataSet dataSet){
    RecordSet rs = dataSet.getLookupResult("test",new
    HashMap(){{put("abc", "USD")}}});
}
```

EXCEPTION-HANDLER Configuration

Exception Handler in SPARROW Framework helps you to declare the severity of the exception based on their error code and description. Usage of EXCEPTION-HANDLER in sparrow framework is optional. However, a default exception handler (sparrow.etl.core.exception.SPARROWExceptionHandler) will be initialized during the start-up of the application to handle the following types of exceptions. All the below exceptions are fatal in nature.

Exception	Error code
sparrow.etl.core.exception.InitializationException	1. MANDATORY_PARAM_MISSING 2. *_INIT_EXP 3. APP_INITIALIZATION_EXP
sparrow.etl.core.exception.ResourceException	* (All)
sparrow.etl.core.exception.EventNotifierException	1. JMS_INIT_BEGIN_APP_JMS_EXP 2. JMS_INIT_BEGIN_APP_EXP 3. SEMAPHORE_BA_LOCKED 4. SEMAPHORE_UNKNOWN 5. SEMAPHORE_BA_PROVIDER_NOT_FOUND
sparrow.etl.core.exception.ParserException	* (All)
java.sql.SQLException	9* Error code start with 9 (mainly for Oracle)



Currently it supports 2 types of severities: fatal and ignore.

FATAL: The process will get terminated immediately.

IGNORE: The program ignores the exception and will proceed with the next step.

```
<EXCEPTION-HANDLER>
  <HANDLER CLASS="sparrow.etl.core.exception.SPARROWExceptionHandler"
NAME="default.exception.handler" />
  <HANDLE EXCEPTION="java.sql.SQLException" HANDLER="default.exception.handler">
    <ERROR CODE="9*34" VALUE="Invalid column name" TYPE="fatal"/>
    <ERROR CODE="*03" VALUE="invalid * identifier" TYPE="ignore"/>
    <ERROR CODE="89903" VALUE="* unique constraint" TYPE="fatal"/>
  </HANDLE>
</EXCEPTION-HANDLER>
```

Argument	Presence	Description
HANDLER	Optional	HANDLER tag is optional. Default : <HANDLER CLASS="sparrow.etl.core.exception.SPARROWExceptionHandler" NAME="default.exception.handler"/>
HANDLER.CLASS	Optional	Class Name - which extends sparrow.etl.core.exception.ExceptionHandler Default : sparrow.etl.core.exception.SPARROWExceptionHandler
HANDLER.NAME	Optional	Name of the Handler Default : default.exception.handler
HANDLE.EXCEPTION	Mandatory	Exception class name - Exception that you wish to handle.
HANDLE.HANDLER	Optional	Handler name - that need to be used to handle this exception. Default : default.exception.handler
ERROR.CODE	Mandatory	Error Code Note : Supports wild card (ex: 9*233, *_NOT_FOUND etc)
ERROR.VALUE	Mandatory	Error Description Note : Supports wild card (ex: 9*233, *_NOT_FOUND etc)
ERROR.TYPE	Mandatory	fatal : Terminates the process immediately ignore : Ignore the exception

Note: EXCEPTION handler can handle only exceptions defined within SPARROW or any exception wrapped with in an exception class which extends
sparrow.etl.core.exception.SPARROWRuntimeException or
sparrow.etl.core.exception.SPARROWException.

Implementation Guideline

Data Extractor Component

To develop your own Data Extractor component, the first step is to create a class that implements
sparrow.etl.core.extractor.DataExtractor interface. The DataExtractor Interface defines 3
methods and out of 2 are component life cycle methods.



```
public interface DataExtractor {  
    public abstract void initialize(); //life cycle method  
    public abstract DataHolder loadData() throws DataException;  
    public abstract void destroy() throws DataException; //life cycle method  
}
```

Your DataExtractor implementation must have a constructor with SPARROWDataExtractorConfig class as argument.

```
public class TestDataExtractor implements DataExtractor {  
    public TestDataExtractor (SPARROWDataExtractorConfig config)  
    {  
    }  
}
```

The argument class SPARROWDataExtractorConfig will provide a handle to access the SPARROWContext and read the parameter which you have configured for this extractor and access the SPARROWContext.

Return Type	Method	Description
void	initialize();	This method will be Invoked by SPARROWCoreEngine class before the first cycle. This method is invoked only once in the Process
DataHolder	loadData()	This method will be invoked for every cycle by SPARROWCoreEngine
void	destroy()	This method will be Invoked by SPARROWCoreEngine class before the application shutting down.

It is mandatory to transform the extracted data to a RecordSet class.

Once you have created your DataExtractor class, you will need to inform the SPARROW framework to use your class for extraction.

```
<DATA-EXTRACTOR NAME="test" CLASS="com.csfb.test.TestDataExtractor">  
    <PARAM NAME="....." VALUE="....." />  
</DATA-EXTRACTOR>
```



You can create your own TYPE by introducing a key=value in the user_impl.properties file.

Ex: extractor.**TEST**=com.csfb.test.TestDataExtractor

```
<DATA-EXTRACTOR NAME="test" TYPE="TEST">
```

Data Transformer Component

Data Transformer component is driven by `sparrow.etl.core.transformer.DataTransformer` interface. You can develop your Data Transformer Object by implementing this Interface or extending the abstract class `sparrow.etl.core.transformer.AbstractDataTransformer` (Recommended).



DataTransformer.java

```
public interface DataTransformer extends TransformerLCEvent {  
  
    public abstract DataOutputHolder enrichData(DataSet dataSet) throws  
    EnrichDataException;  
  
    public abstract void setDriverRowEventListener(DriverRowEventListener listener);  
  
    public abstract DriverRowEventListener getDriverRowEventListener();  
  
    public abstract void markForRejection(String reason) throws RejectionException;  
  
    public abstract boolean isMarkedForRejection();  
  
    public abstract void setDriverRow(ResultRow row);  
  
}
```

Return Type	Method	Description
DataOutputHolder	enrichData(DataSet dataSet)	This method will be invoked by AbstractRequestProcessor for each row of the Extracted data. The argument DataSet carries the results of the look-ups made as part of the current driver row.
void	setDriverRowEventListener(DriverRowEventListener listener) <i>Default Implementation is available in AbstractDataTransformer.</i>	DriverRowEventListener class helps you to handle the following driver row events preLookup, postLookup, preQueue, preWrite, postWrite, preFinalize, preFilter and postFinalize
DriverRowEventListener	getDriverRowEventListener() <i>Default Implementation is available in AbstractDataTransformer.</i>	Returns the Listener which was set thru setDriverRowEventListener
void	markForRejection(String reason) <i>Default Implementation is available in AbstractDataTransformer.</i>	This utility method helps you report the rejection of a driver row. If the rejection.report.source is configured, the "reason" will print to the configured source.
boolean	isMarkedForRejection() <i>Implementation is available in AbstractDataTransformer.</i>	AbstractRequestProcessor class invokes this method to check whether the request has been rejected by the user implementation
void	setDriverRow(ResultRow row)	AbstractRequestProcessor invokes this method to set the current driver row

TransformerLCEvent.java

```
public interface TransformerLCEvent {  
  
    public abstract void setOLC(IObjectPoolLifeCycle olc);  
  
    public abstract void finalizeObject();  
  
}
```



```
public abstract void returnObject();

public abstract void initialize();

public abstract void destroy();
}
```

TransfromerLCEvent interface is used for managing the life cycle of the DataTransformer Object.

Return Type	Method	Description
void	setOLC(IObjectPoolLifeCycle olc) <i>Implementation is available in AbstractDataTransformer.</i>	Sets IObjectPoolLifeCycle interface (instance of DataTransformerPoolManger)
void	finalizeObject() <i>Implementation is available in AbstractDataTransformer.</i>	Destroys the object from the pool
void	returnObject() <i>Implementation is available in AbstractDataTransformer.</i>	Returns the current object to the pool.
void	initialize()	This method will be called before the current instance is ready to process a new request.
void	destroy()	This method will be called before the instance returns to the pool.

The class that implements DataTransformer interface or extends AbstractDataTransformer class must have a constructor with SPARROWDataTransformerConfig class as argument.

```
public class TestDataTransformer implements DataTransformer {

    public TestDataTransformer (SPARROWDataTransformerConfig config)
    {
    }
}
```

```
public class TestDataTransformer extends AbstractDataTransformer { //Recommended

    public TestDataTransformer (SPARROWDataTransformerConfig config)
    {
        super(config);
    }
}
```

The argument SPARROWDataTransformerConfig helps you to access the SPARROWContext and read the parameter which you have configured for this DataTransformer.

Data Writer Component

Data Writer component is driven by **sparrow.etl.core.writer.DataWriter** interface. You can develop your Data writer by implementing this Interface or extending the abstract class **sparrow.etl.core.writer.AbstractDataWriter** (Recommended).



The DataWriter interface internally extends another three more interfaces namely Cloneable, EOPObserver and CycleObserver. Where EOPObserver helps you to capture the “End Of process” event and CycleObserver for capturing begin and end cycle events.

DataWriter.java

```
public interface DataWriter extends Cloneable, EOPObserver, CycleObserver {  
  
    public abstract void initialize();  
  
    public abstract int writeData(DataOutputHolder data, int statusCode) throws  
        DataWriterException;  
  
    public abstract void onError(String writerName, Exception ex) throws  
        DataWriterException;  
  
    public abstract boolean isInTransaction();  
  
    public abstract void setInTransaction(boolean trans);  
  
    public abstract void destroy();  
  
    public abstract boolean startRequest() throws DataWriterException;  
  
    public abstract boolean endRequest() throws DataWriterException;  
  
    public abstract SPARROWDataWriterConfig getWriterConfig();  
}
```

Return Type	Method	Description
void	initialize()	This method will be invoked once for each instance. In the other word, if the INGLETON is configured as false then sparrow kernel will create instances which are equal in number of threads.
int	writeData(DataOutputHolder data, int statusCode)	This is the key method which carries the DataOutputHolder (retuned by DataTransformer.enrichData ()) and statusCode of previous writer.
void	onError(String writerName, Exception ex)	Notifies if any of the writer throws Exception during the writing operation.
void	setInTransaction(boolean trans)	Set true if the writer can participate in transaction.
boolean	isInTransaction() <i>Implementation is available in AbstractDataTransformer.</i>	SPARROW kernel invokes this method to check whether the writer can participate in transaction.
boolean	startRequest()	This method will be called prior to the writeData method
Boolean	endRequest()	This method will be called once all the writers are completed their writing operation.
SPARROWDataWriterConfig	getWriterConfig() <i>Implementation is available in AbstractDataTransformer.</i>	Return current writers configuration object.

Your DataWriter implementation must have a constructor with SPARROWDataWriterConfig Class as argument.



```
public class TestDataWriter implements DataWriter {  
    public TestDataWriter (SPARROWDataWriterConfig config)  
    {  
    }  
}
```

```
public class TestDataWriter extends AbstractDataWriter { //Recommended  
    public TestDataWriter (SPARROWDataWriterConfig config)  
    {  
        super(config);  
    }  
}
```



You can create your own TYPE by introducing a key=value in the user_impl.properties file.

Ex: writer.**TEST**=com.csfb.test.TestDataWriter

<WRITER NAME="test" TYPE="**TEST**">

Data Providers

Data Provider component is driven by `sparrow.etl.core.dao.provider.DataProvider` interface. You can develop your Data provider by implementing this Interface.

```
public interface DataProvider extends Cloneable {  
    public abstract void initialize();  
    public abstract RecordSet getData() throws DataException;  
    public abstract String getName();  
    public abstract boolean isCacheable();  
    public abstract void destory();  
    public abstract Object clone() throws CloneNotSupportedException;  
}
```

Return Type	Method	Description
void	initialize()	This method will be invoked once for each instance.
RecordSet	getData()	Should return the extracted data in the form of RecordSet.
String	getName()	Returns the name of the data provider.
void	destory()	This method will be invoked once the usage of this instance is over.
Object	clone()	Should return super.clone() + the objects that to be deep cloned.



Note: A Data Provider components need not be aware of cache mechanism of SPARROW.

Your `DataProvider` implementation must have a constructor with `SPARROWDataProviderConfig` class as argument.

```
public class TestDataProvider implements DataProvider {  
    public TestDataProvider (SPARROWDataProviderConfig config)  
    {  
    }  
}
```

You can enable your `DataProvider` to support `DataExtractor` request by creating an additional constructor with `SPARROWDataExtractorConfig` class as argument.

```
public class TestDataProvider implements DataProvider {  
    public TestDataProvider (SPARROWDataProviderConfig config) {  
    }  
    public TestDataProvider (SPARROWDataExtractorConfig config) {  
    }  
}
```



You can create your own TYPE by introducing a key=value in the `user_impl.properties` file.

Ex: `provider.TEST=com.csfb.test.TestDataProvider`

`<DATA-PROVIDER NAME="test" TYPE="TEST">`

Data look-ups

Data Look-up component is driven by `sparrow.etl.core.lookup.LookupObject` interface. You can develop your Data look-up object by implementing this Interface.

```
public interface LookupObject {  
    public abstract RecordSet getLookupData(ResultRow row, Map resultMap,  
                                           DriverRowEventListener eventListener) throws  
        DataException;  
}
```

Return Type	Method	Description
RecordSet	getLookupData(ResultRow row, Map resultMap, DriverRowEventListener eventListener)	Should return the looked-up data in the form of RecordSet.

Note: Configured Data-lookup components will be invoked sequentially (if dependencies are not specified) before the enrichment call made to `DataTransformer` component.



Your Data look-up implementation must have a constructor with `SPARROWDataLookupConfig` class as argument.

```
public class TestDataLookup implements LookupObject {  
    public TestDataLookup (SPARROWDataLookupConfig config)  
    {  
    }  
}
```

Load Balancer

Load Balancer component is driven by `sparrow.etl.core.loadbalance.RequestAssignerPolicy` interface. You can develop your Load Balancer by implementing this Interface.

```
public interface RequestAssignerPolicy {  
    abstract void assign(ResultRow row);  
}
```

Return Type	Method	Description
void	assign(ResultRow row)	This method will be invoked with each row of the extracted data. Prior to this, list of queues will be provided thru the constructor.

Your Load Balancer implementation must have a constructor with `Map` and `SPARROWContext` class as argument.

Where the `Map` carries the name of the queues and respective instance of the `FIFO` classes.

```
public class TestRequestAssignerPolicy implements RequestAssignerPolicy{  
    public TestRequestAssignerPolicy (Map fifos, SPARROWContext context) {  
    }  
}
```



You can create your own POLICY by introducing a key=value in the `user_impl.properties` file.

Ex: `lbpolicy.TEST=com.csfb.test.TestRequestAssignerPolicy`

`<LOAD-BALANCE POLICY="TEST">`

Cycle Dependents

Cycle Dependent component is driven by `sparrow.etl.core.cycledependency.CycleEventListener` interface. You can develop your Cycle Dependency object by implementing this Interface.



```
public interface CycleEventListener extends CycleObserver {  
    public abstract String getName();  
  
    public abstract boolean checkDependency() throws DependencyCheckException;  
  
    public abstract boolean isProcessTerminationRequired();  
  
    public abstract String getStatusDescription();  
}
```

Return Type	Method	Description
void	getName()	Return a unique name of the dependent. Used for logging purpose
boolean	checkDependency()	This method will be called in every Cycle interval. If all the configured dependents returns false, then the Core engine will fetch the record for new cycle.
boolean	isProcessTerminationRequired()	If this method returns true then the process will be terminated.
String	getStatusDescription();	Returns description about this Dependent. Used for logging purpose

Your Cycle Dependent implementation must have a constructor with `ConfigParam`, `SPARROWContext` classes as argument.

Where the `ConfigParam` carries arguments configured for `ASSERTER` tag and `SPARROWContext` for getting access into various `SPARROW` components.

```
public class TestCycleEventListener implements CycleEventListener{  
    public TestCycleEventListener (ConfigParam param,  
                                   SPARROWContext context) {  
    }  
}
```

Resources

Resource component is driven by `sparrow.etl.core.resource.GenericResourceInitializer` interface. You can develop your Resource Object by implementing this Interface.

```
public interface GenericResourceInitializer extends AppObserver {  
    abstract void initializeResource();  
  
    abstract Resource getResource();  
}
```

Return Type	Method	Description
void	initializeResource()	This method will be called by the ResourceInitializer class during the startup of the application



Resource	getResource()	This method can be accessed only thru SPARROWContext.getResource("nameOfTheResource") ;
----------	---------------	---

Resource.java

```
public interface Resource {  
  
    public static final int IN_TRANSACTION = 1;  
    public static final int NOT_IN_TRANSACTION = 2;  
  
    public abstract String getName();  
    public abstract Object getResource(int transFlag) throws ResourceException;  
    public abstract Object getResource() throws ResourceException;  
    public abstract ConfigParam getParam();  
    public abstract ResourceType getType();  
  
}
```

Return Type	Method	Description
String	getName()	Return a unique name of the resource.
Object	getResource(int transFlag)	If transFlag== IN_TRANSACTION the this method has be to return a Transaction Enabled resource. Else any Object which can be shared across the application.
Object	getResource()	This method should automatically determine the current thread is participated in a transaction, if yes, this should return a Transaction enabled resource
ConfigParam	getParam()	Returns arguments configured for this RESOURCE
ResourceType	getType()	Returns the name and class associated to this resource.

Your GenericResourceInitializer implementation must have a constructor with SPARROWResourceConfig class as argument.

```
public class TestResourceInitilializer implements GenericResourceInitializer {  
    public TestResourceInitilializer (SPARROWResourceConfig config) {  
    }  
}
```

Pluggable Services

Pluggable Service component is driven by **sparrow.etl.core.services.PluggableService** interface. You can develop your service by implementing this Interface. PluggableService receives notification for all Cycle Events (begin/end) and App events (begin/end), provided, the APP-NOTIFICATION and CYCLE-NOTIFICATION is set to true. SPARROW Kernel creates a single instance of this class and uses it thru-out the process.



```
public interface PluggableService extends AppObserver, CycleObserver {  
  
    public void initialize(SPARROWServiceConfig config) throws  
        ServiceInitializationException;  
  
    public Object getService() throws ServiceUnavailableException;  
  
    public Object getService(Object serviceName) throws  
        ServiceUnavailableException;  
  
}
```

Return Type	Method	Description
void	initialize(SPARROWServiceConfig config)	This method will be called only once in the process.
Object	getService()	This method can be accessed only thru SPARROWContext.getService(String name);
Object	getService(Object serviceName)	Additional method which you can be used a factory pattern to return your own object

Schedulable Services

SPARROW framework supports schedule components which enables you to write your own component and schedule them to run in a defined interval. It extends `PluggableService` so you get the benefit of registering your call to receive events applicable for Pluggable service.

```
public interface SchedulerService extends PluggableService{  
  
    public abstract void executeTask() throws SchedulerException;  
  
}
```

Return Type	Method	Description
void	executeTask()	This method will be called every <n> milliseconds configured in "service.interval" param.

Exception Handler

Exception Handler component is driven by `sparrow.etl.core.exception.ExceptionHandlerAdapter` abstract class. You can develop your ExceptionHandler by extending this class or extending `SPARROWExceptionHandler`. In order to handle your own exception you have to create a method named `handle` in your class which takes the exception class as argument. You can have 'n' number of overloaded `handle` methods for handling different types of exceptions.

```
public class TestExceptionHandler extends ExceptionHandlerAdapter {  
  
    public TestExceptionHandler(ErrorConfig config) {  
        super(config);  
    }  
  
}
```



```
public void handle(TestException e) {
    super.setErrorCode(e.getErrorCode());
    super.setErrorDesc(e.getErrorDescription());
}

public void handle(ABCEException e) {
    super.setErrorCode(e.getErrorCode());
    super.setErrorDesc(e.getErrorDescription());
}
}
```

Once you have created your class, you have to plug it into the configuration as showing below.

```
<HANDLER CLASS="com.csfb.test.TestExceptionHandler" NAME="test.exception.handler"/>
```

Now you have to inform the SPARROW engine which exception to be handled by this handler.

```
<HANDLE EXCEPTION="com.csfb.test.TestException" HANDLER="test.exception.handler">
    <ERROR CODE="FILE_NOT_ARRIVED" VALUE="" TYPE="fatal"/>
</HANDLE>
<HANDLE EXCEPTION="com.csfb.test.ABCEException" HANDLER="test.exception.handler">
    <ERROR CODE="FILE_NOT_ARRIVED" VALUE="" TYPE="fatal"/>
    <ERROR CODE="LESS_IMPACT_EXP" VALUE="" TYPE="ignore"/>
</HANDLE>
```

If the error code or description is marked as `fatal` then the process will be terminated immediately (force shutdown mode) whereas `ignore` type will ignore the exception and continue the process.



Miscellaneous

Encryption

From 3.0.7 version onwards SPARROW supports encryption on all PARAM tag. Please follow the instruction below to implement the same.

Step 1: Run the following command with 2 arguments.

- arg 1 - String that you wish to encrypt
- arg 2 - Process ID (MODULE.PROCESS-ID)

```
java -cp sparrow.<version>.jar sparrow.etl.core.security.Encryptor <arg1> <arg2>
```

Example: arg 1 = password, arg 2 = SPR_PROCESS

```
C:\app\temp>java -cp sparrow.3.0.7.jar sparrow.etl.core.security.Encryptor password
SPR_PROCESS
[INPUT]==> password
[ENC  ]==> KO7dhIHnLEDnz5gxgi*RO2XbFIBvs4oQ==
[DEC  ]==> password
[NOTE ]==> In the param value, pre-fix "{ENC}" with the encrypted value.
Ex.{ENC}KO7dhIHnLEDnz5gxgi*RO2XbFIBvs4oQ==
```

Step 2: As specified in the output of the command. Use the encrypted value anywhere in the PARAM tag.

Example:

```
<RESOURCE NAME="DSGINS50" TYPE="DB" LOAD-PRIORITY="2" UNLOAD-PRIORITY="1">
  <PARAM NAME="driver.classname" VALUE="oracle.jdbc.OracleDriver"/>
  <PARAM NAME="connection.url" VALUE="jdbc:oracle:thin"/>
  <PARAM NAME="user.name" VALUE="abc"/>
  <PARAM NAME="password" VALUE="{ENC}KO7dhIHnLEDnz5gxgi*RO2XbFIBvs4oQ==" />
  <PARAM NAME="pool.size" VALUE="25"/>
  <PARAM NAME="pool.on.start" VALUE="true"/>
  <PARAM NAME="max.wait" VALUE="5000"/>
  <PARAM NAME="default.autocommit" VALUE="true"/>
</RESOURCE>
```

Step 3: Make sure that the same process Id (which you have provided during the encryption) is used in the MODULE.PROCESS-ID tag in the header.

SPARROW Functions

SPARROW Functions are introduced to ease the data enrichment operation and optionally replace the Database function calls in the data provider and look-up filter. The table below describes the functions supported by SPARROW.

Function	Description	Example
func_trim	Trims the string value	func_trim(" some text ") Result : "some text"
func_substring	Returns the substring of the	func_substring("establishment",2,6) Result: "tabl"

	first argument starting at the position specified in the second argument and the length specified in the third argument.	
func_lcase	Make a string lowercase	func_lcase("SPARROW") Result: "sparrow"
func_ucase	Make a string uppercase	func_ucase("sparrow") Result: "SPARROW"
func_replace	Returns a new string resulting from replacing all occurrences of oldString in this string with newString	func_replace("something", "some", "no") Result: "nothing"
func_length	Returns the length of this string.	func_length("SPARROW") Result: 5
func_lpad	Pads the left-side of a string with a specific set of characters	func_lpad("tech", 8, '0'); Result: "0000tech"
func_rpad	Pads the right-side of a string with a specific set of characters	func_rpad("tech", 8, '0'); Result: "tech0000"
func_nullval	Will let you substitute a value when a null value is encountered	func_nullval(null, "new value");
func_ternary	Helps to perform ternary operation.	Ex:1 func_ternary((1==2), "yes", "no") Result: "no" Ex:2 func_ternary((SPARROW==SPARROW), "yes", "no") Result: "yes" Ex:3 func_ternary((func_substring("SPARROW", 0, 2)=="SP") , "yes", "no") Result: "yes"
func_eval	Evaluate Number (int/long) expressions	func_eval(((2+3)+(3+2))*(3+2)) Result: 50
func_evaldouble	Evaluate Number (float/double) expressions	func_evaldouble((2.345+3.322)+(3.55+2.78)*(3.34+2.1223)) Result: 65.5312131
func_getdate	No argument : returns current date With argument : returns Data Object of the string	func_getdate() Result : Current Date (06-01-2010) func_getdate("06-12-2009") Result : Date Object of 06-12-2009
func_formatdate	Formats the date into the input pattern	func_formatdate(func_getdate(12-10-2009), "EEE, MMM d, yy") Result : Mon, Oct 12, 09
func_formatdecimal	Formats the double value into the input pattern	func_formatdecimal(func_evaldouble(85.545+78.25855) , ##.000) Result : 163.804
func_math	Supports Floor, Ceil, Absolute and Round. F=Floor, C=Ceil, A=Absolute,	func_math(func_evaldouble(85.545+78.25855), F) Result : 163.0 func_evaldouble(85.545+78.25855), A)



	R=Round	Result : 163.80355 func_evaldouble(85.545+78.25855),R) Result : 164
func_convertdouble	Converts string into double	func_evaldouble(func_convertdouble(\"78.25855\")+34) Result : 112.25855
func_convertnumber	Converts string into number	func_eval(func_convertnumber (\"78\")+34) Result : 112
func_indexof	Returns the index within this string of the first occurrence of the specified character	func_indexof("SPARROW","E") Result : 2
func_isnull	Returns true if the argument is null	Assume \${driver\$column}=null. func_isnull(\${driver\$column}) returns true
func_isnotnull	Returns true if the argument is not null	Assume \${driver\$column}="SPARROW". func_isnotnull(\${driver\$column}) returns true
func_concatenate	Concatenate string and variable. Use (" ") pipe symbol to separate the values.	Assume \${today} = "Sunday" func_concatenate("Today is \${today}") returns "Today is Sunday"
func_isempty	To check if the value of the variable is empty (i.e. 0 space + not null)	Assume \${today} = "" func_isempty(\${today}) returns true
func_getlocalhost	To print the local host IP or Name	func_getlocalhost("IP") returns 168.165.258.12 func_getlocalhost("NAME") returns sgl45-588.sg.csfb.com

Following are some real time usage scenarios.

Example 1: WRITER

```
<WRITER NAME="writer1" TYPE="DB" >
  <PARAM NAME="query"><![CDATA[
    update table_name set add_user = 'func_ucase(${driver$user})', add_datetime='
func_formatdate(func_getdata(),"dd-MMM-yyyy")'
  ]]></PARAM>
  <PARAM NAME="resource" VALUE="SQL"/>
</WRITER>
```

Example 2: DATA-PROVIDER

```
<DATA-PROVIDER NAME="dp" TYPE="DB">
  <PARAM NAME="query"><![CDATA[
    select * from table_name where
column1='func_substring(${driver$status},0,func_length(${driver$status})-1)'
  ]]></PARAM>
  <PARAM NAME="resource" VALUE="SQL"/>
  <PARAM NAME="fetch.size" VALUE="5000"/>
</DATA-PROVIDER>
```

Example 3: LOOK-UP

```
<LOOKUP NAME="LKP" DATA-PROVIDER="ABC">
  <FILTER> COLUMN1='func_ucase(func_trim(${driver$status}))'</FILTER>
```



```
</LOOKUP>
```

Example 3: DATA-TRANSFORMER

```
<DATA-TRANSFORMER NAME="trans" TYPE="template" THREAD-COUNT="1">
<PARAM NAME="expression">
<![CDATA[ ${driver$CNFRM_ID}, ${driver$CNFRM_REF_ID}, ${driver$TMPLT_PRFRNC_ID},
${driver$TMPLT_PRFRNC_VERS_ID}, ${driver$DELIVERY_CHANNEL_CD},
func_length(${driver$CNFRM_DESTINATION_DESC}), ${driver$CNFRM_DESTINATION_DESC},
func_ternary((func_evaldouble(2.4+2.4)==func_convertdouble(4.3)), (3+3)),
func_replace(${driver$CNFRM_STATUS_CD},_,*), ${driver$CNFRM_CONSLDTN_IND},
func_nullval(${driver$USER_ADD_ID},null), func_nullval(${driver$PROCESS_ADD_ID},
null), ${driver$STAMP_ADD_DTIME}, func_nullval(${driver$USER_UPDATE_ID},null),
func_nullval(${driver$PROCESS_UPDATE_ID},null), ${driver$STAMP_UPDATE_DTIME}
]]></PARAM>
<PARAM NAME="rejection.report.type" VALUE="file"/>
<PARAM NAME="rejection.report.source" VALUE="c:/app/temp/sparrow"/>
</DATA-TRANSFORMER>
<WRITER NAME="filewriter1" TYPE="FILE">
<PARAM NAME="file.path" VALUE="c:/app/temp/sparrow" />
<PARAM NAME="file.name" VALUE="TEST_FILE1_${CURRENT_DATE#ddMM}.csv" />
<PARAM NAME="all.or.none" VALUE="true" />
<PARAM NAME="async.write" VALUE="true" />
<PARAM NAME="file.output" VALUE="single" />
<PARAM NAME="header.row.1" VALUE="HEADER,TEST,${CURRENT_DATE#yyyyddMM}" />
<PARAM NAME="key.name" VALUE="trans"/>
<PARAM NAME="footer.row" VALUE="FOOTER,${RECORD_COUNT}" />
</WRITER>
```

Reading System/Application Properties

SPARROW allows you to read & access system & application properties which are injected thru the command line. Properties must be covered with "@@" before and after. Please see the example below.

Note: You can use these properties anywhere in the sparrow configuration.

Example : System properties

```
<MODULE-PARAM>
<PARAM NAME="sparrow.semaphore.check" VALUE="false"/>
<PARAM NAME="sparrow.cycle.interval" VALUE="5000"/>
<PARAM NAME="sparrow.shutdown" VALUE="Wed:21:58"/>
<PARAM NAME="sparrow.cycle.count" VALUE="1"/>
<PARAM NAME="sparrow.cache.pool.size" VALUE="10"/>
<PARAM NAME="sparrow.java.version" VALUE="@@java.version@@" />
</MODULE-PARAM>
<DATA-EXTRACTOR NAME="@@user.name@@">
<PARAM NAME="data.provider" VALUE="records"/>
</DATA-EXTRACTOR>
```

Example: Application properties

Application properties are injected thru the command line with -D option (Ex: app.name=SPARROW -Dtoken1=somevalue)

```
<MODULE-PARAM>
<PARAM NAME="sparrow.semaphore.check" VALUE="false"/>
<PARAM NAME="sparrow.cycle.interval" VALUE="5000"/>
```



```
<PARAM NAME="sparrow.shutdown" VALUE="Wed:21:58"/>
<PARAM NAME="sparrow.cycle.count" VALUE="1"/>
<PARAM NAME="sparrow.cache.pool.size" VALUE="10"/>
<PARAM NAME="sparrow.java.version" VALUE="@@app.name@" />
</MODULE-PARAM>
<DATA-EXTRACTOR NAME="@@token1@">
  <PARAM NAME="data.provider" VALUE="records" />
</DATA-EXTRACTOR>
```

SPARROW variables

SPARROW allows you define variable and use them in any PARAM.VALUE attribute. The variables must be defined with in the <MODULE-PARAM> tag. The following variable convention must be followed to refer them in another PARAM tag \${module@<variable name>}.

Example

```
<MODULE-PARAM>
  <PARAM NAME="sparrow.semaphore.check" VALUE="false"/>
  <PARAM NAME="sparrow.cycle.interval" VALUE="5000"/>
  <PARAM NAME="sparrow.shutdown" VALUE="Wed:21:58"/>
  <PARAM NAME="sparrow.cycle.count" VALUE="1"/>
  <PARAM NAME="sparrow.cache.pool.size" VALUE="10"/>
  <PARAM NAME="sparrow.test.dir" VALUE="/tmp/test/sparrow"/>
</MODULE-PARAM>
<DATA-WRITERS ON-ERROR="ignore" THREAD-COUNT="1">
  <WRITER NAME="filewrite" TYPE="FILE">
    <PARAM NAME="file.path" VALUE="${module@sparrow.test.dir}" />
    <PARAM NAME="file.name" VALUE="DMF_${LOCAL_HOST}.csv" />
    <PARAM NAME="all.or.none" VALUE="true" />
    <PARAM NAME="async.write" VALUE="true" />
    <PARAM NAME="file.output" VALUE="single" />
    <PARAM NAME="header.row.1" VALUE="HEADER,TEST,${CURRENT_DATE#yyyyddMM}" />
    <PARAM NAME="key.name" VALUE="trans"/>
    <PARAM NAME="footer.row" VALUE="FOOTER,${RECORD_COUNT}" />
  </WRITER>
</DATA-WRITERS>
```

Global Variables

SPARROW allows you to define global variable and access them thru the getter and setter attributes provided in SPARROWContext instance. These variables can be read in any place where the token based access is supported.

For Ex: Within your programe you have introduced a variable abc="Some value" (i.e. context.setAttribute("abc","Some Value")). This variable can be read in the File Writer PARAM <PARAM NAME="footer.row" VALUE="FOOTER,\${RECORD_COUNT},\${abc}"/> or header.row or file.path or file.name.




Appendix I

How to Run

```
java -classpath <all dependency jar>;sparrow.x.x.x.jar -Dsparrow.config=sparrow-  
config.xml [-Dapp.properties=your.app.properties] [-Dinstance.name=your.app.name] [-  
Dlog4j.configuration=log4j.properties] sparrow.etl.core.SPARROWStarter
```

Configuration Schema

@@\pacbo\PACBO_SPARROW\Source_SPARROW_ADV\config\schema\sparrow.xsd


C:\app\spear\vsaji\
SPEAR2.vsaji\pacbo\

Column Definition Template

Tag	Presence	Description
column-set.name	Mandatory	Unique Identifier. This name should match the requested component name. Ex: <DATA-EXTRACTOR NAME="abc" TYPE="CSV"> //or JMS <PARAM NAME="column.definition" VALUE="csvcoldef.xml"/> <PARAM NAME="....." VALUE="....."/> </DATA-EXTRACTOR> So the value for the name attribute should assign with "abc" <column-set name="abc"> This rule should be used wherever you use column.definition parameter.
column.name	Mandatory	Column name that you wish to use. Should be distinct and non-spaced value.
column.type	Mandatory	Data type of the column. string,date,int,long,double,float,java_object
column.default-value	Optional	If the column resulted a null value then the value specified in this attribute will be used (replaced).
column.size	Optional	This is effective only for fixed-length files.
column.format	Conditional	This is mandatory if the column.type is set to "date". You have to specify the date format (java date format) which is expected for the column.
column.exclude-column	Optional	true : The column will be ignored while parsing the line. false : The column will be used. Default value : false.
column.xpath	Optional	This is mandatory if MQ/JMS Extractor is used with param message.tye=xml. Following are the few pre-defined tokens which can be used apart from the standard XPath



		<p>@MSG = Assign the actual (incoming + unparsed) message as it is to a particular column</p> <p>@MSGID = Assign the Unique message id produced by the source messaging system.</p> <p>@INTRL_MSG_ID = If you the message.store.type=File then this attribute carries the file name (with out extention) associated to the message.</p> <p>@PROP:<header property key> = Header property that you would like to read from the incoming message. Please note, you will be able to read only those properties that you have specified in the "header.properties" PARAM.</p>
--	--	---

Configuration template

<pre><SPARROW-CONFIG xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"></pre>		
		<pre> <MODULE> <NAME>Test Application</NAME> <PROCESS-ID>TEST</PROCESS-ID> <DESCRIPTION>For Testing</DESCRIPTION> <MODULE-PARAM> <PARAM NAME="sparrow.cycle.interval" VALUE="5000"/> <PARAM NAME="sparrow.shutdown" VALUE="Wed:21:58"/> <PARAM NAME="sparrow.cycle.count" VALUE="1"/> <PARAM NAME="base.dir" VALUE="c:/app/temp"/> <PARAM NAME="config.dir" VALUE="c:/app/abc/config"/> </MODULE-PARAM> </MODULE> </pre>
		<pre> <DATA-EXTRACTOR NAME="driver" TYPE="DB"> <PARAM NAME="query"><![CDATA[select column1,column2 from table]]></PARAM> <PARAM NAME="resource" VALUE="resource name"/> <PARAM NAME="fetch.size" VALUE="500"/> <PARAM NAME="result.wrap" VALUE="disconnected"/> <PARAM NAME="trim.value" VALUE="true"/> </DATA-EXTRACTOR> </pre>
		<pre> <DATA-TRANSFORMER NAME="abc" TYPE="script" POOL-SIZE="10" THREAD-COUNT="5"> <PARAM NAME="script.input" VALUE="inline"/> <PARAM NAME="script.lang" VALUE="mvel"/> <PARAM NAME="script.value"> <![CDATA[System.out.println("Access to Dataset:"+_dataset); System.out.println("Access to Lookup:"+_dataset.getLookup("lookupabc").getRowCount()); System.out.println("Access to Driver:"+_dataset.getDriverRow().getValue("column1")); varabc = "This string will be written into the file"; _dataout.addObject("token1",varabc);]]> </PARAM> <PARAM NAME="rejection.report.type" VALUE="file"/> <PARAM NAME="rejection.report.source" VALUE="\${module@base.dir}/logs"/> </DATA-TRANSFORMER> </pre>
		<pre> <DATA-WRITERS ON-ERROR="ignore" THREAD-COUNT="1"> <WRITER NAME="filewrite" TYPE="FILE"> </pre>



```
<PARAM NAME="file.path" VALUE="${module@base.dir}/fileout" />
<PARAM NAME="file.name" VALUE="DMF_${CURRENT_DATE#ddMM}.csv" />
<PARAM NAME="all.or.none" VALUE="true" />
<PARAM NAME="async.write" VALUE="true" />
<PARAM NAME="file.output" VALUE="single" />
<PARAM NAME="header.row.1"
VALUE="HEADER,TEST,${CURRENT_DATE#yyyyddMM}" />
<PARAM NAME="key.name" VALUE="token1"/>
<PARAM NAME="footer.row" VALUE="FOOTER,${RECORD_COUNT}" />
</WRITER>
<WRITER NAME="emailwrite" TYPE="EMAIL" DEPENDS="filewrite">
  <PARAM NAME="email.to" VALUE="abc@email.com" />
  <PARAM NAME="email.from" VALUE="xyz@email.com" />
  <PARAM NAME="email.subject" VALUE="${LOCAL_HOST}:Test email from
SPARROW" />
  <PARAM NAME="email.content" VALUE="Test email from SPARROW" />
  <PARAM NAME="email.mode" VALUE="unix" />
  <PARAM NAME="attachment" VALUE="filewrite" />
</WRITER>
<WRITER NAME="dbwrite1" TYPE="DB" >
  <PARAM NAME="query"><![CDATA[ insert into tabel123
values(${driver$column1},?)]]>
  </PARAM>
  <PARAM NAME="resource" VALUE="TAXSYB"/>
  <PARAM NAME="batch.size" VALUE="1"/>
  <PARAM NAME="batch.per.request" VALUE="true"/>
  <PARAM NAME="key.name" VALUE="inputparam"/>
</WRITER>
<WRITER NAME="dbwrite2" TYPE="DB" >
  <PARAM NAME="data.provider" VALUE="dp4"/>
  <PARAM NAME="batch.size" VALUE="200"/>
</WRITER>
</DATA-WRITERS>

<RESOURCES>
  <RESOURCE NAME="qsgcsar1" TYPE="DB" LOAD-PRIORITY="2" UNLOAD-PRIORITY="1">
    <PARAM NAME="driver.classname" VALUE="oracle.jdbc.OracleDriver"/>
    <PARAM NAME="connection.url" VALUE="jdbc:oracle:thin:@sgl23h-
055d.sg.csfb.com:1530:QSGCSAR1"/>
    <PARAM NAME="user.name" VALUE="abc"/>
    <PARAM NAME="password" VALUE="abc"/>
    <PARAM NAME="pool.size" VALUE="10"/>
    <PARAM NAME="pool.on.start" VALUE="false"/>
    <PARAM NAME="max.wait" VALUE="5000"/>
    <PARAM NAME="default.autocommit" VALUE="true"/>
  </RESOURCE>
  <RESOURCE NAME="TAXSYB" TYPE="DB" LOAD-PRIORITY="2" UNLOAD-PRIORITY="1">
    <PARAM NAME="driver.classname" VALUE="com.sybase.jdbc3.jdbc.SybDriver"/>
    <PARAM NAME="connection.url"
VALUE="jdbc:sybase:Tds:${module@taxman.syb.server}/${module@taxman.syb.tempdb}" />
    <PARAM NAME="user.name" VALUE="${module@taxman.syb.tempdbuname}" />
    <PARAM NAME="password" VALUE="${module@taxman.syb.tempdbpwd}" />
    <PARAM NAME="pool.size" VALUE="10"/>
    <PARAM NAME="pool.on.start" VALUE="false"/>
    <PARAM NAME="max.wait" VALUE="5000"/>
    <PARAM NAME="default.autocommit" VALUE="true"/>
  </RESOURCE>
  <RESOURCE NAME="dsgcsar2" TYPE="DB" LOAD-PRIORITY="2" UNLOAD-PRIORITY="1">
    <PARAM NAME="datasource" VALUE="ctx@DataSourceJNDI"/>
  </RESOURCE>
  <RESOURCE NAME="spring" TYPE="SPRING" LOAD-PRIORITY="2" UNLOAD-PRIORITY="1">
    <PARAM NAME="context.file" VALUE="${module@config.dir}/sparrow-spring-
```




```
appcontext.xml"/>
    <PARAM NAME="context.location" VALUE="filesystem"/>
</RESOURCE>
<RESOURCE NAME="ejb" TYPE="EJB" LOAD-PRIORITY="2" UNLOAD-PRIORITY="1">
    <PARAM NAME="lookup" VALUE="ctx@EJBJndi"/>
</RESOURCE>
<RESOURCE NAME="queue" TYPE="EJB" LOAD-PRIORITY="2" UNLOAD-PRIORITY="1">
    <PARAM NAME="lookup" VALUE="ctx@QueueJNDI"/>
</RESOURCE>
<RESOURCE NAME="ctx" TYPE="CONTEXT" LOAD-PRIORITY="1" UNLOAD-PRIORITY="2">
    <PARAM NAME="provider.url" VALUE="t3://sgl33a-9503.sg.csfb.com:2001"/>
    <PARAM NAME="initialcontext.factory"
VALUE="weblogic.jndi.WLInitialContextFactory"/>
</RESOURCE>
<RESOURCE NAME="email" TYPE="SMTP" LOAD-PRIORITY="2" UNLOAD-PRIORITY="1">
    <PARAM NAME="mail.smtp.host" VALUE="smtp-amg-ap.hedani.net"/>
</RESOURCE>
<RESOURCE NAME="ftp" TYPE="FTP" LOAD-PRIORITY="2" UNLOAD-PRIORITY="1">
    <PARAM NAME="host.name" VALUE="sgs33a-124.sg.csfb.com"/>
    <PARAM NAME="user.name" VALUE="abc"/>
    <PARAM NAME="password" VALUE="abc"/>
</RESOURCE>
<RESOURCE NAME="sftp" TYPE="FTP" LOAD-PRIORITY="2" UNLOAD-PRIORITY="1">
    <PARAM NAME="host.name" VALUE="sgs33a-124.sg.csfb.com"/>
    <PARAM NAME="user.name" VALUE="abc"/>
    <PARAM NAME="password" VALUE="abc"/>
    <PARAM NAME="channel" VALUE="secure"/>
</RESOURCE>
<RESOURCE NAME="scp" TYPE="SCP" LOAD-PRIORITY="2" UNLOAD-PRIORITY="1">
    <PARAM NAME="host.name" VALUE="sgs33a-124.sg.csfb.com"/>
    <PARAM NAME="user.name" VALUE="abc"/>
    <PARAM NAME="password" VALUE="abc"/>
</RESOURCE>
<RESOURCE NAME="TIB" TYPE="JMS" LOAD-PRIORITY="3" UNLOAD-PRIORITY="1">
    <PARAM NAME="jms.type" VALUE="queue" />
    <PARAM NAME="context" VALUE="tibcoctx" />
    <PARAM NAME="connection.factory" VALUE="saji" />
    <PARAM NAME="secure.connection" VALUE="true" />
    <PARAM NAME="session.transacted" VALUE="true" />
    <PARAM NAME="message.ack.type" VALUE="auto" />
</RESOURCE>
<RESOURCE NAME="tibcoctx" TYPE="CONTEXT" LOAD-PRIORITY="1" UNLOAD-PRIORITY="3">
    <PARAM NAME="provider.url" VALUE="tibjmsnaming://sgs65d-0788.sg.csfb.com:33012"
/>
    <PARAM NAME="initialcontext.factory"
VALUE="com.tibco.tibjms.naming.TibjmsInitialContextFactory" />
    <PARAM NAME="security.principal" VALUE="vsaji" />
    <PARAM NAME="security.credentials" VALUE="vsaji" />
</RESOURCE>
</RESOURCES>

<SERVICES>
    <SERVICE NAME="scheduler" CLASS="com.test.TestScheduler" CYCLE-
NOTIFICATION="false" APP-NOTIFICATION="true">
        <PARAM NAME="service.interval" VALUE="2000"/>
    </SERVICE>
</SERVICES>

<DATA-PROVIDERS>
    <DATA-PROVIDER NAME="dpl" TYPE="DB">
        <PARAM NAME="query"><![CDATA[ select * from abc where
col='@{driver$column2}' ]]></PARAM>
```




```
<PARAM NAME="resource" VALUE="dsgcsar2"/>
<PARAM NAME="fetch.size" VALUE="200"/>
</DATA-PROVIDER>
<DATA-PROVIDER NAME="dp2" TYPE="DB">
  <PARAM NAME="query"><![CDATA[ select * from currency]]></PARAM>
  <PARAM NAME="resource" VALUE="dsgcsar2"/>
  <PARAM NAME="cache.type" VALUE="begin.app"/>
</DATA-PROVIDER>
<DATA-PROVIDER NAME="dp3" TYPE="DB">
  <PARAM NAME="query"><![CDATA[package.invoice_insert(
${driver$column1},${driver$column2},${sp.out}]]></PARAM>
  <PARAM NAME="resource" VALUE="dsgcsar2"/>
  <PARAM NAME="sp.out.param" VALUE="3:VARCHAR:cbd" />
</DATA-PROVIDER>
<DATA-PROVIDER NAME="dp4" TYPE="DB">
  <PARAM NAME="query"><![CDATA[insert into abc
values(${lkp1$column1},${driver$column2},${lkp2$column1}]]></PARAM>
  <PARAM NAME="resource" VALUE="TAXSYB"/>
</DATA-PROVIDER>

<DATA-LOOKUP>
  <LOOKUP NAME="currency" DATA-PROVIDER="dp2">
    <FILTER>IssueCurrency =${driver$column3}</FILTER>
  </LOOKUP>
  <LOOKUP NAME="DTA_RATE" DATA-PROVIDER="dp1"/>
</DATA-LOOKUP>

<EXCEPTION-HANDLER>
  <HANDLE EXCEPTION="sparrow.etl.core.exception.ProcessException">
    <ERROR CODE="DBINIT_EA_SQL_EXP" VALUE="" TYPE="fatal"/>
    <ERROR CODE="SEMAPHORE_BA_LOCKED" VALUE="" TYPE="fatal"/>
    <ERROR CODE="SEMAPHORE_UNKNOWN" VALUE="" TYPE="fatal"/>
    <ERROR CODE="SEMAPHORE_BA_PROVIDER_NOT_FOUND" VALUE="" TYPE="fatal"/>
  </HANDLE>
  <HANDLE EXCEPTION="sparrow.etl.core.exception.ValidatorException">
    <ERROR CODE="TOKEN_DBPARAM_EXIST" VALUE="" TYPE="ignore"/>
    <ERROR CODE="APP_INITIALIZATION_EXP" VALUE="" TYPE="fatal"/>
    <ERROR CODE="LOCALJMS_INIT_EXP" VALUE="" TYPE="ignore"/>
  </HANDLE>
</EXCEPTION-HANDLER>
</SPARROW-CONFIG>
```

impl.properties Template

```
#
sparrow.build.version=${sparrow.label$

# Cycledependency Implementations.
cycledependency.ShutdownTimeCheckDependant = sparrow.etl.core.cycledependency.ShutdownTimeListener
cycledependency.EndCycleCheckDependant = sparrow.etl.core.cycledependency.EndCycleListener
cycledependency.ProcessTerminationCheckDependant = sparrow.etl.core.cycledependency.ProcessTerminationListener
cycledependency.CycleCountCheckDependant = sparrow.etl.core.cycledependency.CycleCountListener

# Resource Implementations
resources.DB=sparrow.etl.core.resource.DefaultDBSourceInitializer
resources.MQ=sparrow.etl.core.resource.JMSResourceInitializer
resources.JMS=sparrow.etl.core.resource.JMSResourceInitializer
resources.SMTP=sparrow.etl.core.resource.EmailResourceInitializer
resources.EMAIL=sparrow.etl.core.resource.EmailResourceInitializer
resources.FTP=sparrow.etl.core.resource.FTPResourceInitializer
resources.SCP=sparrow.etl.core.resource.SCPResourceInitializer
resources.SPRING=sparrow.etl.core.resource.SpringResourceInitializer
resources.EJB=sparrow.etl.core.resource.EJBResourceInitializer
```



```
resources.CONTEXT=sparrow.etl.core.resource.InitialContextInitializer
resources.OTHER=null

ftp.default=sparrow.etl.core.resource.SunFTPClientSession
ftp.sun=sparrow.etl.core.resource.SunFTPClientSession
ftp.secure=sparrow.etl.core.resource.SFTPClientSession

# Writer Implementations
writer.DB=sparrow.etl.impl.writer.DBDataWriter
writer.FILE=sparrow.etl.impl.writer.FileWriter
writer.TEST=sparrow.etl.impl.writer.DummyDataWriter
writer.JMS=sparrow.etl.impl.writer.MessageWriter
writer.MQ=sparrow.etl.impl.writer.MessageWriter
writer.ZIP=sparrow.etl.impl.writer.ZipWriter
writer.FTP=sparrow.etl.impl.writer.FTPWriter
writer.TAR=sparrow.etl.impl.writer.TarWriter
writer.EMAIL=sparrow.etl.impl.writer.EmailWriter
writer.FILE-UTIL=sparrow.etl.impl.writer.FileUtilWriter
writer.SCP=sparrow.etl.impl.writer.SCPWriter
writer.OTHER=null

# Writer Extention
writer_extn.failall@batch/true/bpr/false=sparrow.etl.impl.writer.DBDataWriterSupportBatch
writer_extn.ignore@batch/true/bpr/false=sparrow.etl.impl.writer.DBDataWriterSupportBatch
writer_extn.ignore@batch/false/bpr/false=sparrow.etl.impl.writer.DBDataWriterSupportSelfCommit
writer_extn.failall@batch/false/bpr/false=sparrow.etl.impl.writer.DBDataWriterSupportAllorNone
writer_extn.failall@batch/true/bpr/true=sparrow.etl.impl.writer.DBDWDoubeBatchSupport
writer_extn.ignore@batch/true/bpr/true=sparrow.etl.impl.writer.DBDWDoubeBatchSupport
writer_extn.ignore@batch/false/bpr/true=sparrow.etl.impl.writer.DBDWSupportDBSelfCommit
writer_extn.failall@batch/false/bpr/true=sparrow.etl.impl.writer.DBDWDBSupportAllorNone

# Notifier Injection
notifier.watcher=sparrow.etl.core.notifier.WatcherNotifier

#Event Injection
event.exception=sparrow.etl.core.notifier.event.ExceptionEventEvaluator

# DataProvider Implementations
provider.DB =
sparrow.etl.core.dao.provider.impl.DBDataProviderElement,sparrow.etl.core.dao.provider.impl.DBDataPro
provider.CSV =
sparrow.etl.core.dao.provider.BaseDataProviderElement,sparrow.etl.core.dao.provider.impl.CSVDataPro
provider.PROC =
sparrow.etl.core.dao.provider.impl.DBDataProviderElement,sparrow.etl.impl.dao.ProceduralDataProvide

# Extractor Implementation
extractor.DEFAULT = sparrow.etl.impl.extractor.GenericDataExtractor
extractor.PROC = sparrow.etl.impl.extractor.DefaultDataExtractor
extractor.CSV = sparrow.etl.impl.extractor.csv.CSVDataExtractor
extractor.DB =sparrow.etl.impl.extractor.db.DBDataExtractor
extractor.MESSAGE = sparrow.etl.impl.extractor.jms.JMSDataExtractor
extractor.MQ = sparrow.etl.impl.extractor.jms.JMSDataExtractor
extractor.JMS = sparrow.etl.impl.extractor.jms.JMSDataExtractor

#Transfomer Implementation
transformer.DEFAULT=sparrow.etl.impl.transformer.DummyDataTransformer
transformer.TEST=sparrow.etl.impl.transformer.DummyDataTransformer
transformer.SCRIPTLET-MVEL=sparrow.etl.impl.transformer.MVELBasedDataTransformer
transformer.SCRIPT=sparrow.etl.impl.transformer.CustomScriptDataTransformer
transformer.TEMPLATE=sparrow.etl.impl.transformer.ExpressionBasedTransformer

# Services Implementation
```



```
service.DB-SEMAPHORE=sparrow.etl.impl.services.DBSemaphoreService

# CSV Data Provider - File Processor
fileprocessor.CSV=sparrow.etl.core.dao.provider.impl.CSVFileProcessor

#Load Banalancer Default policy Impl
lbpolicy.ROUND-ROBIN=sparrow.etl.impl.loadbalance.RoundRobinRequestAssignerPolicy

#JDBC
dbdialect.jdbc.oracle.thin=sparrow.etl.core.dao.dialect.OracleDBDialect
dbdialect.jdbc.jtds.sybase=sparrow.etl.core.dao.dialect.SybaseDBDialect
dbdialect.jdbc.sybase.Tds=sparrow.etl.core.dao.dialect.SybaseDBDialect
dbdialect.jdbc.microsoft.sqlserver=sparrow.etl.core.dao.dialect.MSSQLServerDBDialect
dbdialect.jdbc.sqlserver=sparrow.etl.core.dao.dialect.SybaseDBDialect
dbdialect.com.microsoft.jdbc=sparrow.etl.core.dao.dialect.MSSQLServerDBDialect
dbdialect.com.microsoft.sqlserver=sparrow.etl.core.dao.dialect.MSSQLServerDBDialect
dbdialect.com.microsoft.sqlserver.jdbc.SQLServerDriver=sparrow.etl.core.dao.dialect.MSSQLServerDBDialect
dbdialect.oracle.jdbc.driver=sparrow.etl.core.dao.dialect.OracleDBDialect
dbdialect.com.sybase=sparrow.etl.core.dao.dialect.SybaseDBDialect
dbdialect.oracle.jdbc.OracleDriver=sparrow.etl.core.dao.dialect.OracleDBDialect
dbdialect.com.sybase.jdbc.SybDriver=sparrow.etl.core.dao.dialect.SybaseDBDialect
dbdialect.com.sybase.jdbc2.jdbc.SybDriver=sparrow.etl.core.dao.dialect.SybaseDBDialect

dbdialect.com.microsoft.sqlserver.jdbc.SQLServerDriver=sparrow.etl.core.dao.dialect.MSSQLServerDBDialect

#Script Support
script.mvel=sparrow.etl.impl.script.MVELScriptEngine
script.java=sparrow.etl.impl.script.JaninoScriptEngine
script.java.cbe=sparrow.etl.impl.script.JaninoScriptEngineCBEImpl
script.xslt=sparrow.etl.impl.script.XSLTScriptEngine

import.1=sparrow.etl.core.dao.impl.RecordSet
import.2=sparrow.etl.core.exception.DataException
import.3=sparrow.etl.core.dao.impl.ResultRow
import.4=sparrow.etl.core.dao.impl.ResultRowImpl
import.5=sparrow.etl.core.dao.impl.DisconnectedRecordSet
import.6=sparrow.etl.core.dao.impl.ColumnHeader
import.7=sparrow.etl.core.dao.impl.RecordSetImpl_Disconnected

dml.tokens=SELECT ,INSERT ,DELETE ,UPDATE ,BEGIN ,BEGIN\n

functions.func_substring=sparrow.etl.core.lang.function.SubstringFunction
functions.func_lcase=sparrow.etl.core.lang.function.LCaseFunction
functions.func_trim=sparrow.etl.core.lang.function.TrimFunction
functions.func_ucase=sparrow.etl.core.lang.function.UCaseFunction
functions.func_replace=sparrow.etl.core.lang.function.ReplaceFunction
functions.func_length=sparrow.etl.core.lang.function.LengthFunction
functions.func_lpad=sparrow.etl.core.lang.function.LPadFunction
functions.func_rpad=sparrow.etl.core.lang.function.RPadFunction
functions.func_nullval=sparrow.etl.core.lang.function.NullValFunction
functions.func_ternary=sparrow.etl.core.lang.function.TernaryFunction
functions.func_evaldouble=sparrow.etl.core.lang.function.EvalDoubleFunction
functions.func_eval=sparrow.etl.core.lang.function.EvalNumberFunction
functions.func_getdate=sparrow.etl.core.lang.function.DateFunction
functions.func_formatdate=sparrow.etl.core.lang.function.FormatDateFunction
functions.func_formatdecimal=sparrow.etl.core.lang.function.FormatDecimalFunction
functions.func_math=sparrow.etl.core.lang.function.MathFunction
functions.func_convertdouble=sparrow.etl.core.lang.function.ConvertToDoubleFunction
functions.func_convertnumber=sparrow.etl.core.lang.function.ConvertToNumberFunction
functions.func_indexof=sparrow.etl.core.lang.function.IndexOfFunction
```



```
# Exception Handlers
handler.1.name=default.exception.handler
handler.1.class=sparrow.etl.core.exception.SPARROWExceptionHandler

# Handle Excetion for sparrow.etl.core.exception.InitializationException
handle.1.exception.class=sparrow.etl.core.exception.InitializationException
handle.1.hander.name=default.exception.handler
handle.1.error.1.code=MANDATORY_PARAM_MISSING
handle.1.error.1.desc=
handle.1.error.1.type=fatal
handle.1.error.2.code=*_INIT_EXP
handle.1.error.2.desc=
handle.1.error.2.type=fatal
handle.1.error.3.code=APP_INITIALIZATION_EXP
handle.1.error.3.desc=
handle.1.error.3.type=fatal

# Handle Excetion for sparrow.etl.core.exception.ResourceException
handle.2.exception.class=sparrow.etl.core.exception.ResourceException
handle.2.hander.name=default.exception.handler
handle.2.error.1.code=DB_DATASOURCE_CREATION
handle.2.error.1.desc=
handle.2.error.1.type=fatal
handle.2.error.2.code=*_EXCEPTION_CON
handle.2.error.2.desc=
handle.2.error.2.type=fatal
handle.2.error.3.code=DB_BATCH_NOT_SUPPORT
handle.2.error.3.desc=
handle.2.error.3.type=fatal
handle.2.error.4.code=*_INIT_EXP
handle.2.error.4.desc=
handle.2.error.4.type=fatal
handle.2.error.5.code=*
handle.2.error.5.desc=
handle.2.error.5.type=fatal
# Handle Excetion for sparrow.etl.core.exception.ResourceException
handle.3.exception.class=sparrow.etl.core.exception.EventNotifierException
handle.3.hander.name=default.exception.handler
handle.3.error.1.code=JMS_INIT_BEGIN_APP_JMS_EXP
handle.3.error.1.desc=
handle.3.error.1.type=fatal
handle.3.error.2.code=JMS_INIT_BEGIN_APP_EXP
handle.3.error.2.desc=
handle.3.error.2.type=fatal
handle.3.error.3.code=SEMAPHORE_BA_LOCKED
handle.3.error.3.desc=
handle.3.error.3.type=fatal
handle.3.error.4.code=SEMAPHORE_UNKNOWN
handle.3.error.4.desc=
handle.3.error.4.type=fatal
handle.3.error.5.code=SEMAPHORE_BA_PROVIDER_NOT_FOUND
handle.3.error.5.desc=
handle.3.error.5.type=fatal
# *****
handle.4.exception.class=sparrow.etl.core.exception.ParserException
handle.4.hander.name=default.exception.handler
handle.4.error.1.code=*
handle.4.error.1.desc=
handle.4.error.1.type=fatal
# *****
handle.5.exception.class=java.sql.SQLException
```



```
handle.5.hander.name=default.exception.handler  
handle.5.error.1.code=9*  
handle.5.error.1.desc=  
handle.5.error.1.type=fatal
```