CA4MN Spark Streaming Wire Editor Configuration

|  |  |  |  |
| --- | --- | --- | --- |
|  | | |  |
| Author | Zhang Ou |
| Owner |  |
| Organization | Application & Analytics |
| Approver |  |
| Document ID |  |
| Document location |  |
| |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | Change History | | | | | | | | | | | Version | Status | Date | Author | Owner | Reviewed by | Reviewed date | Approver | Approval date | Description of changes | | | 1.0 | Draft | 18-07-2017 | Zhang Ou |  | He Shan | 01-09-2017 | He Shan | 01-09-2017 | RTNA architecture | | | 1.1 | Preliminary | 06-09-2017 | Zhang Ou |  | He Shan | 30-09-2017 | He Shan | 30-09-2017 | RTNA fwk changing | | | 1.2 | Standard | 30-10-2017 | He Shan |  |  |  |  |  | Rename to CA4MN and fwk changing | | | |  |
|  | | |  |

Contents

[1 Introduction 5](#_Toc489370062)

[1.1 Intended audience 5](#_Toc489370063)

[1.2 Scope 5](#_Toc489370064)

[1.3 Abbreviations / Acronyms 5](#_Toc489370065)

[1.4 Definitions 5](#_Toc489370066)

[2 Related Documentation 6](#_Toc489370067)

[2.1 Requirements Documents 6](#_Toc489370068)

[2.2 Other Applicable Documents 6](#_Toc489370069)

[2.3 Informative Documents 6](#_Toc489370070)

[3 General 7](#_Toc489370071)

[4 Source 8](#_Toc489370072)

[5 Sink 8](#_Toc489370073)

[6 Branch 9](#_Toc489370074)

[7 Processor 9](#_Toc489370075)

[7.1 Split 10](#_Toc489370076)

[7.2 Filter 11](#_Toc489370077)

[7.3 Transform 11](#_Toc489370078)

[7.4 Union 12](#_Toc489370079)

[7.5 Context 12](#_Toc489370080)

[7.6 Correlate 13](#_Toc489370081)

[8 SPEL 14](#_Toc489370082)

[8.1 NPC configuration data 14](#_Toc489370083)

[9 Example 14](#_Toc489370084)

**LIST OF FIGURES**

[Figure 1: RTNA Spark running chart 7](#_Toc489370086)

# Introduction

## Intended audience

The intended audience of this document is all persons involved in the specification, architecture, design and tests of RTNA project.

## Scope

This document is a description of the spark running chart xml configuration.

## Abbreviations / Acronyms

## Definitions

# Related Documentation

## Requirements Documents

## Other Applicable Documents

## Informative Documents

# General

RTNA Spark running configuration xml file is designed to ease the definition of spark running job dedicated for RTNA project.

A spark running chart contains three main parts: Input(sources), Processing Chain(branches) and Output(sinks).

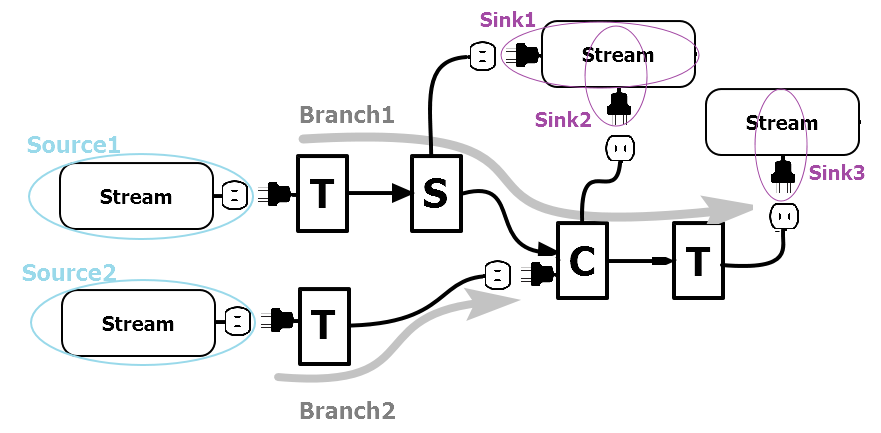


Figure 1: RTNA Spark running chart

The main xml structure is as follows:

<sparkrunningchart nativeRecord=”true” structuredStreaming=”false”>

<tuning>

<mapPartition>false</mapPartition> <!—default no use partition mapping-->

<partitionNum>0</partitionNum> <!-- 0 means no partition change in shuffle-->

<forcePartition>false</forcePartition> <!--for all processor can invoke repartition-->

<storageLevel>MEMORY\_ONLY\_SER</storageLevel> <!--for split/correlate-->

<asyncAction>false</asyncAction> <!--for sink-->

</tuning>

<sourceBuilderClass>com.nokia.rtna.framework.job.builder.DefaultRtnaSourceBuilder </sourceBuilderClass>

<processorBuilderClass>com.nokia.rtna.framework.job.builder.DefaultRtnaProcessorBuilder</processorBuilderClass>

<sinkBuilderClass>com.nokia.rtna.framework.job.builder.DefaultRtnaSinkBuilder</sinkBuilderClass>

**<sources>…</sources>**

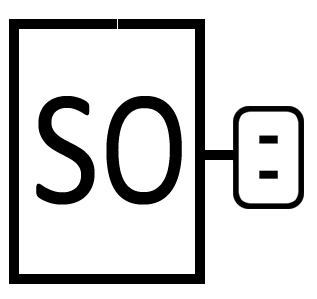
**<branches>…</branches>**

**<sinks>…</sinks>**

</sparkrunningchart>

Please note for all xml configuration part in this document, the grey colour means the corresponding tags are optional. The default value is provided in optional tag.

# Source

Source defines where the data comes from. Multiple sources can be defined and each source must provide an outlet for a branch plug to connect with.

If source data comes from Kafka, Kafka configuration should be provided. (Currently only support kafka source in production environment)

Customized data source can also be used by providing data source builder class name in <sourceBuilderClass>. If <sourceBuilderClass> not provided, a default Kafka source builder will be used.

Following sub tags are supported for source.

<source id=’’ keyedStream=’false’>

<outletId></outletId>

<Tuning>

<partitionNum>0</partitionNum> <forcePartition>false</forcePartition

</Tuning>

**<kafka>**

**<topics>**

**<topic></topic>**

<topic></topic>

…

**</topics>**

**<brokerList></brokerList>**

**<groupId></groupId>**

<localSchemaPath></localSchemaPath>

<avroSchemaRegistry>

**<schemaRegistryURL></schemaRegistryURL>**

<ids>

<id></id>

…

</ids>

</avroSchemaRegistry>

<properties>

<property>

**<name></name>**

**<value></value>**

</preoperty>

…

</properties>

<dumpRecords>false</dumpRecords>

<zookeeperURL></zookeeperURL>

<resilientConnectors></resilientConnectors>

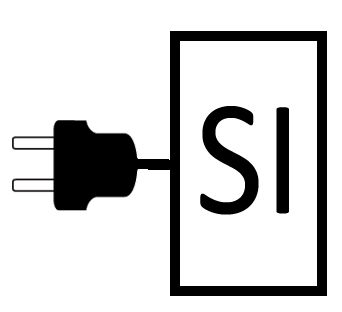
**</kafka>**

</source>

For GUI, Following fields should be provided:

|  |  |  |  |
| --- | --- | --- | --- |
| Label | Type | Rule (Checker) | Tooltip |
| Id \* | String | Should be unique in one job | Unique id of current processor |
| Description | String |  | Description of current processor |
| Topics \* | String List | Dynamic list, at least one provided | Kafka topics |
| Brokers \* | String | (IP:Port[,IP:Port[…]]) | Kafka brokers |
| Group \* | String |  | Kafka group id |
| Schema Definition \* | Radio button | “Local”, “Registry” | Avro schema definition source |
| Local Path | String | When “Local” selected for “Schema Definition” | Local schema definition file path |
| Registry URL | String | When “Registry” selected for “Schema Definition” | Schema registry server URL |

# Sink

Sink defines where the data goes to. Multiple sinks can be defined and each sink must provide a plug for a branch outlet to connect with.

If source data goes to Kafka, Kafka configuration should be provided. (Currently only support kafka source in production environment)

Customized sink can also be used by providing data sink builder class name in <sinkBuilderClass>. If <sinkBuilderClass> not provided, a default Kafka sink builder will be used.

Following sub tags are supported for sink.

<sink id=’’>

<plugId></plugId>

**<kafka>**

**<topics>**

**<topic></topic>**

<topic></topic>

…

**</topics>**

**<brokerList></brokerList>**

**<avroSchemaRegistry>**

**<schemaRegistryURL></schemaRegistryURL>**

**</avroSchemaRegistry>**

<properties>

<property>

**<name></name>**

**<value></value>**

</preoperty>

…

</properties>

<getTopicFromKey>false</getTopicFromKey>

<useTupleSchema>false</useTupleSchema>

**</kafka>**

</sink>

For GUI, Following fields should be provided:

|  |  |  |  |
| --- | --- | --- | --- |
| Label | Type | Rule (Checker) | Tooltip |
| Id \* | String | Should be unique in one job | Unique id of current processor |
| Description | String |  | Description of current processor |
| Brokers \* | String | (IP:Port[,IP:Port[…]]) | Kafka brokers |
| Registry URL | String |  | Schema registry server URL |

# Branch

Branch starts with a plug and ends with an outlet. Each branch is actually a processor chain. Processors in a branch should be defined in sequence of the processing. There should be no intersection between branches, which means one processor is dedicated for only one branch. The sequence of branch in branch list should follow the following principle: the input part (<plugId> or <plugs> part in the contained <processor>) must be defined in the previous branch output part (<outletId> or <outlet> part in the contained <processor>).

<branch id=*''*>

<plugId></plugId>

<outletId></outletId>

<processors>

<processor>…</processor>

<processor>…</processor>

</processors>

</branch>

# Processor

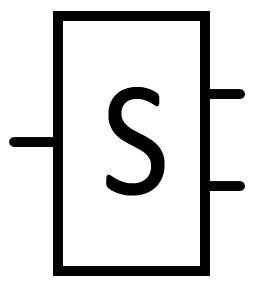
Processor is the smallest unit to handle stream processing. We try to design the processor as simple as possible. So each processor only implements one simple and dedicated function, which provides user with flexibility to assemble different processors together to implement complicated scenario cases.

Following 7 processor types are supported currently: **split, filter, transform, union, context, correlate and customized.**

The type should be specified in ‘actionType’ attribute.

For processors with multiple output, <outlets> should be defined for other branch or sink plug to connect with; for processors with multiple input, <plugs> should be defined for other branch or source outlet to connect with.

## Split

A splitter takes in one input stream and split them into several same output streams, among which, one output stream is on the branch, other streams should be defined in <outlets>.

<processor actionType=*'split'* id=’’ description=’’>

<outlets>

<outlet>

<id></id>

</outlet>

<outlet>

<id></id>

</outlet>

…

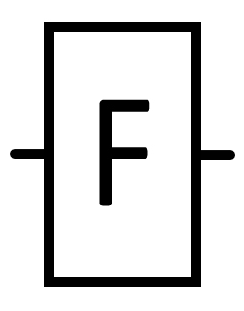
</outlets>

</processor>

For GUI, Following fields should be provided:

|  |  |  |  |
| --- | --- | --- | --- |
| Label | Type | Rule (Checker) | Tooltip |
| Id \* | String | Should be unique in one job | Unique id of current processor |
| Description | String |  | Description of current processor |

## Filter

A filter takes in the input stream and filters this stream according to the filter condition defined in <booleanExpression>.

<processor actionType=*'filter'* id=’’ description=’’>

<booleanExpression></booleanExpression>

</processor>

Filter condition must comply with SpEL. Eg.

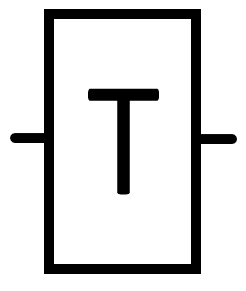
<booleanExpression>(get('M\_TMSI') ne null) and (#int(get('NumRRCFailureRecords')) gt 0)</booleanExpression>

For expression details, please refer to §8 Expression.

For GUI, Following fields should be provided:

|  |  |  |  |
| --- | --- | --- | --- |
| Label | Type | Rule (Checker) | Tooltip |
| Id \* | String | Should be unique in one job | Unique id of current processor |
| Description | String |  | Description of current processor |
| Expression \* | String | (Expression checker) | A Boolean expression based on GenericRecord, a record is evaluated with true can pass this filter |

## Transform

Transformer can transform a non-keyed stream to a keyed steam. The key expression should comply with SpEL.

If the transform is complex, we also provide a customized way to handle the stream transform.

A customized transform class name can be defined in ‘class’ attribute. If the transform involves with another Avro Schema, its URL should be defined in ‘avroSchemaURL’ attribute and if not provided, the schema of the stream on main branch will be used by default.

<processor actionType=*'transform'* id=’’ description=’’>

<key>

<expressions>

<expression></expression>

<expression></expression>

</expressions>

</key>

<avroDataTransform class=*''*>

<localSchemaPath></localSchemaPath>

<avroSchemaRegistry>

**<schemaRegistryURL></schemaRegistryURL>**

**<ids>**

**<id></id>**

…

**</ids>**

</avroSchemaRegistry>

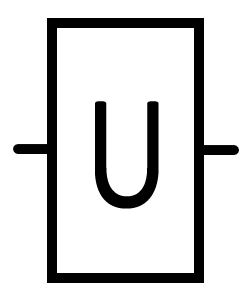
</avroDataTransform>

</processor>

For GUI, Following fields should be provided:

|  |  |  |  |
| --- | --- | --- | --- |
| Label | Type | Rule (Checker) | Tooltip |
| Id \* | String | Should be unique in one job | Unique id of current processor |
| Description | String |  | Description of current processor |
| Key Expression \* | String List | Dynamic list, at least one provided | A expression list, each of them based on GenericRecord, the union of all these expression value will be used as key |
| Schema Change Definition \* | Radio Button | “None”, “Local” | If want to change record schema and schema source definition |
| Local Path | String | When “Local” selected for “Schema Change Definition” | Local schema definition file path for target |
| Transform Class | String | When “Local” selected for “Schema Change Definition” | Transform java class name for complex avro schema changing, “data[0]” is input record |

## Union

Union processor will merge the multiple input streams into one stream. Except for the one stream on branch, other input streams should be defined in <plugs>.

Please note that the two input streams should follow the same Avro Schema.

<processor actionType=*’union’* id=’’ description=’’>

<plugs>

<plug>

<id></id>

</plug>

…

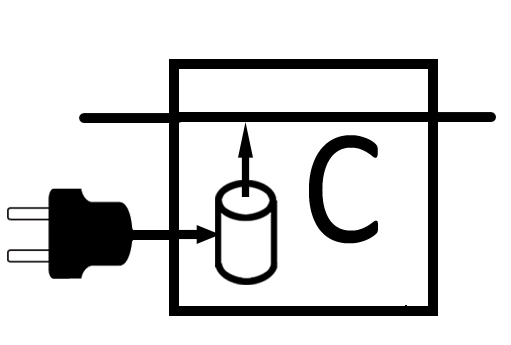
</plugs>

</processor>

For GUI, Following fields should be provided:

|  |  |  |  |
| --- | --- | --- | --- |
| Label | Type | Rule (Checker) | Tooltip |
| Id \* | String | Should be unique in one job | Unique id of current processor |
| Description | String |  | Description of current processor |

## Context

The input stream needed to be updated according to the context database. Because the updated record must be in the main branch, we name it as main input. The context input stream which provides the context should be provided in the plugs tag and timeout must be provided for how long the context should be cached.

Except for timeout, the context cache purge could also be determined by a combination of timeout and remove condition. The context cache could be updated according to condition specified in <contextManagement> tag.

<processor actionType=*’context’* id=’’ description=’’>

<plugs>

<plug>

<id></id>

</plug>

</plugs>

<contextManagement>

<timeout></timeout>

<updateCondition>

<booleanExpression></booleanExpression>

</updateCondition>

<removeCondition>

<booleanExpression></booleanExpression>

</removeCondition>

<contextDatabaseTransform class=*’’*>

<localSchemaPath></localSchemaPath>

<avroSchemaRegistry>

**<schemaRegistryURL></schemaRegistryURL>**

**<ids>**

**<id></id>**

…

**</ids>**

</avroSchemaRegistry>

</contextDatabaseTransform>

</contextManagement>

<avroDataTransform class=’’>

<localSchemaPath></localSchemaPath>

<avroSchemaRegistry>

**<schemaRegistryURL></schemaRegistryURL>**

**<ids>**

**<id></id>**

…

**</ids>**

</avroSchemaRegistry>

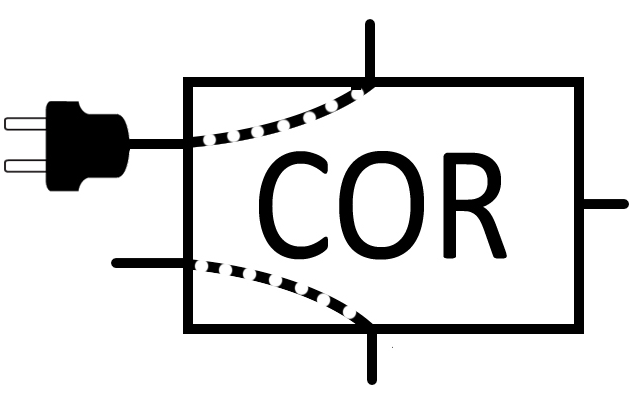
</avroDataTransform>

</processor>

For GUI, Following fields should be provided:

|  |  |  |  |
| --- | --- | --- | --- |
| Label | Type | Rule (Checker) | Tooltip |
| Id \* | String | Should be unique in one job | Unique id of current processor |
| Description | String |  | Description of current processor |
| Context Timeout \* | Long |  | Timeout milliseconds that context database record will be purged |
| Context Management By \* | Radio Button | “Expression”, “Code” | How to manage context database updating policy |
| Context Update Expression | String | When “Expression” selected for “Context Management By”, default “True” | “data1” is old GenericRecord in context DB, “data2” is new coming GenericRecord. Evaluate true will update context with “data2”. |
| Context Remove Condition | String | When “Expression” selected for “Context Management By”, default “False” | “data1” is old GenericRecord in context DB, “data2” is new coming GenericRecord. Evaluate true will make context DB empty. |
| Context Transform Class | String | When “Code” selected for “Context Management By” | “data[0]” is old record in context DB, “data[1]” is new coming record. Returned record will be stored in context DB. |
| Context Schema Local Path | String | When “Code” selected for “Context Management By” | Local schema definition file path for record in context DB, default same as context input |
| Output Merge Class \* | String |  | “data[0]” is record to be updated, “data[1]” is record in current context database. Returned record will be output. |
| Output Schema Local Path | String |  | Local schema definition file path for record will be output, default same as main input |

## Correlate

The correlate processor requires two input streams. The one on the main branch is called correlatend, and the one provided in the plugs tag is called correlator. After the correlation process, the correlated stream will be on the main stream outlet. The uncorrelated correlatend stream will be the first outlet in the outlets tag. And the uncorrelated correlator stream will be the second outlet in the outlets tag. For these two uncorrelated output streams, timeout policy could be defined.

<processor actionType=*'correlate'* id=’’ description=’’>

<plugs>

<plug>

<id></id>

</plug>

</plugs>

<correlateManagement>

<correlateCondition></correlateCondition>

<scanCorrelatendFromOldToNew>false</scanCorrelatendFromOldToNew>

<scanCorrelatorFromOldToNew>false</scanCorrelatorFromOldToNew>

</correlateManagement>

<avroDataTransform class='' avroSchemaURL=''>

<localSchemaPath></localSchemaPath>

<avroSchemaRegistry>

**<schemaRegistryURL></schemaRegistryURL>**

**<ids>**

**<id></id>**

…

**</ids>**

</avroSchemaRegistry>

</avroDataTransform>

<outlets>

<outlet>

<id></id>

<timeoutPolicy>

<longExpression></longExpression>

<timeout></timeout>

</timeoutPolicy>

</outlet>

<outlet>

<id></id>

<timeoutPolicy>

<longExpression></longExpression>

<timeout></timeout>

</timeoutPolicy>

</outlet>

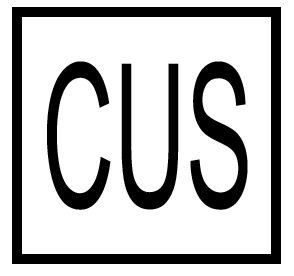
</outlets>

</processor>

For GUI, Following fields should be provided:

|  |  |  |  |
| --- | --- | --- | --- |
| Label | Type | Rule (Checker) | Tooltip |
| Id \* | String | Should be unique in one job | Unique id of current processor |
| Description | String |  | Description of current processor |
| Correlate Expression \* | String |  | “data1” is correlatend GenericRecord, “data2” is correlator GenericRecord, return True means correlated successfully |
| Correlation Cache Management \* | Radio Button | “Cache Correlatend”, “Cache Correlator”, “Cache Both” | How to cache history record to do correlation |
| Correlatend Time Extractor | String | When “Cache Correlatend” or “Cache Both” selected for “Correlation Cache Management” | A Long expression based on correlatend GenericRecord, the result will be treat as time of current record |
| Correlatend Timeout | Long | When “Cache Correlatend” or “Cache Both” selected for “Correlation Cache Management” | Timeout milliseconds that correlatend record will be purged |
| Correlator Time Extractor | String | When “Cache Correlator” or “Cache Both” selected for “Correlation Cache Management” | A Long expression based on correlator GenericRecord, the result will be treat as time of current record |
| Correlator Timeout | Long | When “Cache Correlator” or “Cache Both” selected for “Correlation Cache Management” | Timeout milliseconds that correlator record will be purged |
| Correlatend Scan Method \* | Radio Button | “Old to New”, “New to Old” | Scan method defines which record has high priority to be correlated |
| Correlator Scan Method \* | Radio Button | “Old to New”, “New to Old” | Scan method defines which record has high priority to be correlated |
| Output Merge Class \* | String |  | “data[0]” is correlatend record, “data[1]” is correlator record. Returned record will be output. |
| Output Schema Local Path | String |  | Local schema definition file path for record will be output, default same as correlatend |

## Customize

If current processor types provided cannot fulfil the customer requirements, a customized processor can be implemented according to customer need by providing a processor class name in class tag. The input streams and output streams can be specified using the same mechanism as other pre-defined processors with tags <plugs> and <outlets>. Customized properties can be ingested into processor with <properties> tag.

<processor actionType=*'customize'* class=*’’* id=’’ description=’’>

<plugs>

<plug>

<id></id>

</plug>

…

</plugs>

<outlets>

<outlet>

<id></id>

</outlet>

…

</outlets>

<properties>

<property>

<name></name>

<value></value>

</property>

…

<properties>

</processor>

For GUI, Following fields should be provided:

|  |  |  |  |
| --- | --- | --- | --- |
| Label | Type | Rule (Checker) | Tooltip |
| Id \* | String | Should be unique in one job | Unique id of current processor |
| Description | String |  | Description of current processor |
| Properties \* | Pair List | Dynamic list with “name”, “value” pair, can be empty | Customized properties used by current processor |

# Expression

## ‘get’ operator

Currently, we support three tags to fill SPEL expression: <longExpression> <booleanExpression> and <expression>. All expression must comply with Avro Data Schema. We provide ‘get’ method operator to retrieve data defined in Avro Data Schema.

If the expression contains definition in Avro Data Schema, the evaluation must get into type ‘field’.

For example, if the schema is as follows:

{"namespace": "nokia.rtna.pcmd",

"type": "record",

"name": "primary",

"fields": [

{"name": "PCMDid", "type": "string"},

{"name": "secondary", "type": {

"type": "array",

"items":

{"namespace": "nokia.rtna.pcmd",

"type": "record",

"name": "secondary",

"fields": [

{"name": "PGWid", "type": "string"}]

}

}

}

]

}

The validity of the expressions:

<expression>get(PCMDid)</expression> OK

<expression>get(secondary)</expression> NOK

<expression>get(secondary).get(0)</expression> NOK

<expression>get(secondary).get(0).get(PGWid)</expression> OK

## ‘data1’ and ‘data2’ operator

For processors with two input streams, we use ‘data1’ to represent the stream on main branch, ‘data2’ to represent the stream ingested from <plug> tag.

For a correlator processor, data1 refers to correlatend and data2 refers to correlator. For a context processor, data1 refers to stream to be updated and data2 refers to stream providing context.

A valid expression for correlate processor could be like this:

<booleanExpression>data2.get('eNBstopCollectionTime') ge data1.get('ProcedureStartTime')

</booleanExpression>

## Predefined methods

For expression tag with type specified, like <booleanExpression> and <longExpreesion>, the expression evaluation result should be a boolean or long.

For common supported relation and logical operators that could be used in <booleanExpression> tag, please refer to [SpEL documentation](http://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/expressions.html).

We also predefined the following methods to do the type conversion and number conversion.

* #int(String str)
* #long(String str)
* #float(String str)
* #double(String str)
* #boolean(String str)
* #hexToDecimal(String str)

For example, the <longExpression> requires the evaluation result to be Long type, we can use predefined ‘long’ method to convert a result of String type to Long type.

<longExpression>#long(get(‘eNBstopCollectionTime’))</longExpression>

## Predefined variables

For some configuration, we may want to reconfigure them without changing xml each time (eg. Kafka parameters). For this purpose, we provide a mechanism by utilizing SpEL predefined variables to ingest configuration values as arguments when launching the application.

For example, you can prefix ‘#’ to variable and provide its value in launch arguments by prefixing ‘-V’

<kafka>

<topic>#topic</topic>

<brokerList>#brokerList</brokerList>

<groupId>#groupId</groupId>

<avroSchemaURL>#avroSchema</avroSchemaURL>

</kafka>

Application launch arguments:

-Vtopic=mme\_topic –VbrokerList=135.252.169.7:9092,135.252.169.8:9092 –VgroupId=mme –V avroSchema=/MMEPCMD\_13.asvc

Another predefined variable is #batchDuration. This is not related to xml configuration, but a parameter for spark streaming.

Application launch arguments:

-b 6000L

# Example

Here we provide a complex example for the PCMD correlation, which is a practical user case.

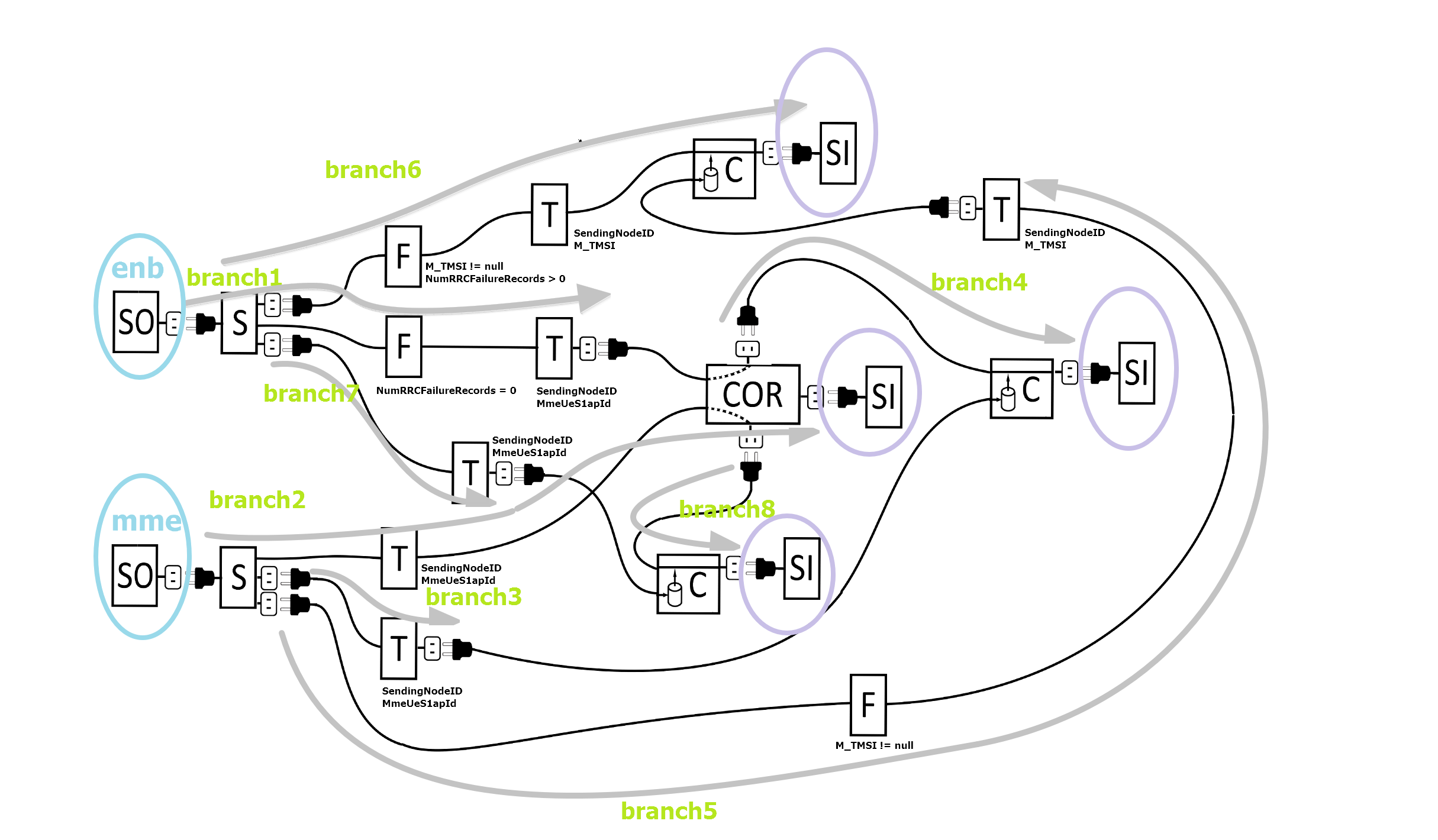


Figure 2: RTNA Spark running chart



🙣 End of DOCUMENT 🙡