Deploying your Application Using SABRE

1. Writing an Application that uses 'SABRE'

Step 1 : Understanding 'FlowContext'

Every rule that is to be executed inside the rule chain will get a flow context as a parameter for its 'executeRule()' method. The data required for processing of rule can be obtained from this flowcontext using a key. When the data processing completes you can place the data that any other rule may use as input inside its 'executeRule()' method using a key.

Step 2 : Writing an Input data handler

You need to write an input data handler which will take in requests and forward the data to the rule-engine. The input data handler must extend the *'InputProcessor' or 'LinkedProcessor'* of Sabre. Sabre will start processing the input data only when the '*process()*' method on input data handler is called.

Note : Without calling the 'process()' method Sabre will not start working.

Step 3: Writing data processors

The data processor has to extend the SingletonRule or CloneableRule class as per the requirement whether you want the rule to behave as Singleton or Cloneable respectively. You can write the business logic inside the '*executeRule()*' method. Whatever data is required by the processor can be obtained from the FlowContext received as a parameter of '*executeRule()*' method.

Note : The 'executeRule()' method returns a RuleDecisionKey on which the further decision of data processing is taken by Sabre.

Step 4: Creating the '*Rules.xml*' file

This file is a mapper for rule-names to rule class
A sample rule file will looks as ::

```
<RulesConfig>
	<rule name="LogFileName" classname="com.nb.sabre.demo.FileNameLogger" />
	<rule name="LogFileDetails" classname="com.nb.sabre.demo.FileDetailsLogger" />
	<rule name="FindFileType" classname="com.nb.sabre.demo.FileTypeFinder" />
	<rule name="ListFileContentSummary" classname="com.nb.sabre.demo.FileContentSummarizer" />
	<rule name="ListZipContents" classname="com.nb.sabre.demo.ZipContentLister" />
	<rule name="ProcessUnknownFile" classname="com.nb.sabre.demo.UnknownFileProcessor" />
	<rule name="MoveFileOnSuccess" classname="com.nb.sabre.demo.FileMover">
		<parameter value="files/succeeded" classname="java.lang.String" />
	</rule>
	<rule name="MoveFileOnFail" classname="com.nb.sabre.demo.FileMover">
		<parameter value="files/failed" classname="java.lang.String" />
	</rule>
</RulesConfig>
```

The 'rule' tag specifies a single rule. Actual rule name is stated as an attribute of 'rule' tag. The class to be instantiated when this rule comes is stated in attribute 'classname'. If any parameters are needed to be passed to the constructor for instantiation of class object the parameter tag needs to be defined for the same. The classname attribute of parameter tag denoted the data type of parameter. Currently only string parameters are supported

For eg: for rule 'LogAction' the class that will be instantiated is 'org.sabre.logging.ReportLogger' with parameters 'logActionLogger' and 'Debug' passed to the construntor.

Step 5: Creating the '*RulesChain.xml*' file

This file defines the sequence in which the rule-chains.
A sample rules-chain file looks as:

```
<InitializeRuleEngine>

	<chain-def name="FileOperationsChainStart">
		<rule name="LogFileName">
			<on-success>
				<rule name="LogFileDetails">
					<on-success>
						<rule name="FindFileType">
							<if-txt>
								<rule name="ListFileContentSummary">
									<chain-link name="FinishProcessing" />
								</rule>
							</if-txt>
							<if-zip>
								<rule name="ListZipContents">
									<chain-link name="FinishProcessing" />
								</rule>
							</if-zip>
							<if-bak>
								<rule name="ListFileContentSummary">
									<chain-link name="FinishProcessing" />
								</rule>
							</if-bak>

							<if-log>
								<rule name="ListFileContentSummary">
									<chain-link name="FinishProcessing" />
								</rule>
							</if-log>
							<on-fail>
								<chain-link name="ErrorChain" />
							</on-fail>
							<rule name="ProcessUnknownFile">
								<chain-link name="FinishProcessing" />
							</rule>
						</rule>
					</on-success>
					<on-fail>
						<chain-link name="ErrorChain" />
					</on-fail>
				</rule>
			</on-success>
			<on-fail>
				<chain-link name="ErrorChain" />
			</on-fail>
		</rule>
	</chain-def>

	<chain-def name="FinishProcessing">
		<rule name="MoveFileOnSuccess">
		</rule>
	</chain-def>

	<chain-def name="ErrorChain">
		<rule name="MoveFileOnFail">
		</rule>
	</chain-def>

</InitializeRuleEngine>
```

The chain-def tags define a single rule chain. The decision-key tag follows the rule tag specifying what the next decision is. The action to be taken for this key is specified as the next rule.

Step 6: Properties file for Sabre

This file tells Sabre from where it will get its inputdata, on which the rules will be applied, the rule.xml , ruleschain.xml file paths, the rule-chain name from which the rule processing must begin and the output processor to which Sabre will send its output.

```
#logger configuration file
sabre.logger.logConfigFile = conf/log4j.properties

#processor configuration file
sabre.processor.configs = conf/processors.xml

#rules configuration file
sabre.ruleengine.rules.filename = conf/Rules.xml

#database configuration file
sabre.dbconfig.filename = conf/DbConnectionConfig.xml

#event record configuration file
sabre.erconfig.filename = conf/er.xml

#rule chain configuration file
sabre.ruleengine.rulechains.filename = conf/RuleChains.xml

#configuration for deciding the start chain in call flow
sabre.ruleengine.rulechains.startchain = FileOperationsChainStart

#thread configurations
sabre.ruleengine.threadpool.minThreads = 5
sabre.ruleengine.threadpool.maxThreads = 10
sabre.ruleengine.threadpool.queueSize = 1000

#license file configuration
sabre.license.filename = conf/sabre.lic
```

2. Starting the application

- You need to call an 'install()' method on an instance of '*Install*' passing it the properties file from which Sabre will take its configuration.
- Calling the install method will start up the application.

Note :: For the application to work you need to include the following jars in your application :

1. bsh-2.0b4.jar
2. commons-beanutils.jar
3. commons-collections-3.2.jar
4. commons-digester-1.7.jar
5. commons-logging-1.1.jar
6. jsr173_1.0_api.jar
7. jsr173_1.0_ri.jar
8. log4j-1.2.9.jar