

[::Go back to Oozie Documentation Index::](#)

Oozie Specification, a Hadoop Workflow System

(PROPOSAL v1.0-2009MAY21)

This goal of this document is to define a workflow engine system specialized in coordinating the execution of Hadoop Map/Reduce and Pig jobs.

- [Changelog](#)
- [0 Definitions](#)
- [1 Specification Highlights](#)
- [2 Workflow Definition](#)
- [2.1 Cycles in Workflow Definitions](#)
- [3 Workflow Nodes](#)
 - [3.1 Control Flow Nodes](#)
 - [3.1.1 Start Control Node](#)
 - [3.1.2 End Control Node](#)
 - [3.1.3 Kill Control Node](#)
 - [3.1.4 Decision Control Node](#)
 - [3.1.5 Fork and Join Control Nodes](#)
 - [3.2 Workflow Action Nodes](#)
 - [3.2.1 Action Basis](#)
 - [3.2.1.1 Action Computation/Processing Is Always Remote](#)
 - [3.2.1.2 Actions Are Asynchronous](#)
 - [3.2.1.3 Actions Have 2 Transitions, =ok= and =error=](#)
 - [3.2.1.4 Action Recovery](#)
 - [3.2.2 Map-Reduce Action](#)
 - [3.2.2.1 Adding Files and Archives for the Job](#)
 - [3.2.2.2 Streaming](#)
 - [3.2.2.3 Syntax](#)
 - [3.2.3 Pig Action](#)
 - [3.2.4 Fs action](#)
 - [3.2.5 Ssh Action](#)
 - [3.2.6 Sub-workflow Action](#)
 - [3.2.7 Http Action](#)
 - [3.2.8 Email Node](#)
- [4 Parameterization of Workflows](#)
 - [4.1 Workflow Job Properties \(or Parameters\)](#)
 - [4.2 Expression Language Functions](#)
 - [4.2.1 Basic EL Constants](#)
 - [4.2.2 Basic EL Functions](#)
 - [4.2.3 Workflow EL Functions](#)
 - [4.2.4 Hadoop EL Constants](#)
 - [4.2.5 Hadoop EL Functions](#)
 - [4.2.6 HDFS EL Functions](#)
- [5 Oozie Notifications](#)
 - [5.1 Workflow Job Status Notification](#)
 - [5.2 Node Start and End Notifications](#)
- [6 User Propagation](#)
- [7 Workflow Applications Packaging](#)
- [8 External Data Assumptions](#)
- [9 Workflow Jobs Lifecycle](#)
- [10 Workflow Jobs Recovery \(re-run\)](#)
- [11 Oozie Web Services API, V0](#)
 - [11.1 Versions End-Point](#)
 - [11.2 Admin End-Point](#)
 - [11.2.1 System Status](#)
 - [11.2.2 OS Environment](#)
 - [11.2.2 Java System Properties](#)
 - [11.2.2 Oozie Configuration](#)
 - [11.2.3 Oozie Instrumentation](#)
 - [11.3 Job and Jobs End-Points](#)
 - [11.3.1 Job Submission](#)
 - [11.3.1 Managing the Job](#)
 - [11.3.1 Job Information](#)
 - [11.3.2 Job Definition](#)
 - [11.3.3 Job Log](#)
 - [11.3.4 Jobs Information](#)
- [12 Client API](#)
- [13 Command Line Tools](#)
- [14 Web UI Console](#)
- [15 Customizing Oozie with Extensions](#)
- [16 Workflow Jobs Priority](#)
- [Appendixes](#)
 - [Appendix A, Oozie XML-Schema](#)
 - [Appendix B, Workflow Examples](#)

Changelog

2009MAY18

- #3.1.4 decision node, 'default' element, 'name' attribute changed to 'to'
- #3.1.5 fork node, 'transition' element changed to 'start', 'to' attribute change to 'path'
- #3.1.5 join node, 'transition' element remove, added 'to' attribute to 'join' element
- #3.2.1.4 Rewording on action recovery section
- #3.2.2 map-reduce action, added 'job-tracker', 'name-node' actions, 'file', 'cache-file' and 'cache-

archive' elements

- #3.2.2.1 map-reduce action, remove from 'streaming' element 'file', 'cache-file' and 'cache-archive' elements
- #3.2.2.2 map-reduce action, reorganized streaming section
- #3.2.3 pig action, removed information about implementation (SSH), changed elements names
- #3.2.4 fs action, removed 'fs-uri' and 'user-name' elements, file system URI is now specified in path, user is propagated
- #3.2.6 sub-workflow action, renamed elements 'oozie-uri' to 'oozie' and 'workflow-app' to 'app-path'
- #4 Properties that are valid Java identifiers can be used as \${NAME}
- #4.1 Renamed default properties file from 'configuration.xml' to 'default-configuration.xml'
- #4.2 Changes in EL Constants and Functions
- #5 Updated notification behavior and tokens
- #6 Changed user propagation behavior
- #7 Changed application packaging from ZIP to HDFS directory
- Removed application lifecycle and self containment model sections
- #10 Changed workflow job recovery, simplified recovery behavior
- #11 Detailed Web Services API
- #12 Updated Client API section
- #15 Updated Action Executor API section
- #Appendix A XML namespace updated to 'uri:oozie:workflow:0.1'
- #Appendix A Updated XML schema to changes in map-reduce/pig/fs/ssh actions
- #Appendix B Updated workflow example to schema changes

2009MAR25

- Changing all references of HWS to Oozie (project name)
- Typos, XML Formatting
- XML Schema URI correction

2009MAR09

- Changed CREATED job state to PREP to have same states as Hadoop
- Renamed 'hadoop-workflow' element to 'workflow-app'
- Decision syntax changed to be 'switch/case' with no transition indirection
- Action nodes common root element 'action', with the action type as sub-element (using a single built-in XML schema)
- Action nodes have 2 explicit transitions 'ok to' and 'error to' enforced by XML schema
- Renamed 'fail' action element to 'kill'
- Renamed 'hadoop' action element to 'map-reduce'
- Renamed 'hdfs' action element to 'fs'
- Updated all XML snippets and examples
- Made user propagation simpler and consistent
- Added Oozie XML schema to Appendix A
- Added workflow example to Appendix B

2009FEB22

- Opened [JIRA HADOOP-5303](#)

0 Definitions

Action: An execution/computation task (Map-Reduce job, Pig job, a shell command). It can also be referred as task or 'action node'.

Workflow: A collection of actions arranged in a control dependency DAG (Direct Acyclic Graph). "control dependency" from one action to another means that the second action can't run until the first action has completed.

Workflow Definition: A programmatic description of a workflow that can be executed.

Workflow Definition Language: The language used to define a Workflow Definition.

Workflow Job: An executable instance of a workflow definition.

Workflow Engine: A system that executes workflows jobs. It can also be referred as a DAG engine.

1 Specification Highlights

A Workflow application is DAG that coordinates the following types of actions: Hadoop, Pig, Ssh, Http, Email and sub-workflows.

Flow control operations within the workflow applications can be done using decision, fork and join nodes. Cycles in workflows are not supported.

Actions and decisions can be parameterized with job properties, actions output (i.e. Hadoop counters, Ssh key/value pairs output) and file information (file exists, file size, etc). Formal parameters are expressed in the workflow definition as `${VAR}` variables.

A Workflow application is a ZIP file that contains the workflow definition (an XML file), all the necessary files to run all the actions: JAR files for Map/Reduce jobs, shells for streaming Map/Reduce jobs, native libraries, Pig scripts, and other resource files.

Before running a workflow job, the corresponding workflow application must be deployed in Oozie.

Deploying workflow application and running workflow jobs can be done via command line tools, a WS API and

a Java API.

Monitoring the system and workflow jobs can be done via a web console, command line tools, a WS API and a Java API.

When submitting a workflow job, a set of properties resolving all the formal parameters in the workflow definitions must be provided. This set of properties is a Hadoop configuration.

Possible states for a workflow jobs are: `PREP` , `RUNNING` , `SUSPENDED` , `SUCCEEDED` , `KILLED` and `FAILED` .

In the case of a action failure in a workflow job, depending on the type of failure, Oozie will attempt automatic retries, it will request a manual retry or it will fail the workflow job.

Oozie can make HTTP callback notifications on action start/end/failure events and workflow end/failure events.

In the case of workflow job failure, the workflow job can be resubmitted skipping previously completed actions. Before doing a resubmission the workflow application could be updated with a patch to fix a problem in the workflow application code.

2 Workflow Definition

A workflow definition is a DAG with control flow nodes (`start`, `end`, `decision`, `fork`, `join`, `kill`) or action nodes (`map-reduce`, `pig`, etc.), nodes are connected by transitions arrows.

The workflow definition language is XML based and it is called hPDL (Hadoop Process Definition Language).

2.1 Cycles in Workflow Definitions

Oozie does not support cycles in workflow definitions, workflow definitions must be a strict DAG.

At workflow application deployment time, if Oozie detects a cycle in the workflow definition it must fail the deployment.

3 Workflow Nodes

Workflow nodes are classified in control flow nodes and action nodes:

- **Control flow nodes:** nodes that control the start and end of the workflow and workflow job execution path.
- **Action nodes:** nodes that trigger the execution of a computation/processing task.

Node names and transitions must be conform to the following pattern `=[a-zA-Z][\-_a-zA-Z0-0]*=`, of up to 20 characters long.

3.1 Control Flow Nodes

Control flow nodes define the beginning and the end of a workflow (the `start` , `end` and `kill` nodes) and provide a mechanism to control the workflow execution path (the `decision` , `fork` and `join` nodes).

3.1.1 Start Control Node

The `start` node is the entry point for a workflow job, it indicates the first workflow node the workflow job must transition to.

When a workflow is started, it automatically transitions to the node specified in the `start` .

A workflow definition must have one `start` node.

Syntax:

```
<workflow-app name="[WF-DEF-NAME]" xmlns="uri:oozie:workflow:0.1">
  ...
  <start to="[NODE-NAME]" />
  ...
</workflow-app>
```

The `to` attribute is the name of first workflow node to execute.

Example:

```
<workflow-app name="foo-wf" xmlns="uri:oozie:workflow:0.1">
  ...
  <start to="firstHadoopJob" />
  ...
</workflow-app>
```

3.1.2 End Control Node

The `end` node is the end for a workflow job, it indicates that the workflow job has completed successfully.

When a workflow job reaches the `end` it finishes successfully (`SUCCEEDED`).

If one or more actions started by the workflow job are executing when the `end` node is reached, the actions will be killed. In this scenario the workflow job is still considered as successfully run.

A workflow definition must have one `end` node.

Syntax:

```
<workflow-app name="[WF-DEF-NAME]" xmlns="uri:oozie:workflow:0.1">
  ...
  <end name="[NODE-NAME]"/>
  ...
</workflow-app>
```

The name attribute is the name of the transition to do to end the workflow job.

Example:

```
<workflow-app name="foo-wf" xmlns="uri:oozie:workflow:0.1">
  ...
  <end name="end"/>
</workflow-app>
```

3.1.3 Kill Control Node

The kill node allows a workflow job to kill itself.

When a workflow job reaches the kill it finishes in error (KILLED).

If one or more actions started by the workflow job are executing when the kill node is reached, the actions will be killed.

A workflow definition may have zero or more kill nodes.

Syntax:

```
<workflow-app name="[WF-DEF-NAME]" xmlns="uri:oozie:workflow:0.1">
  ...
  <kill name="[NODE-NAME]">
    <message>[MESSAGE-TO-LOG]</message>
  </kill>
  ...
</workflow-app>
```

The name attribute in the kill node is the name of the Kill action node.

The content of the message element will be logged as the kill reason for the workflow job.

A kill node does not have transition elements because it ends the workflow job, as KILLED .

Example:

```
<workflow-app name="foo-wf" xmlns="uri:oozie:workflow:0.1">
  ...
  <kill name="killBecauseNoInput">
    <message>Input unavailable</message>
  </kill>
  ...
</workflow-app>
```

3.1.4 Decision Control Node

A decision node enables a workflow to make a selection on the execution path to follow.

The behavior of a decision node can be seen as a switch-case statement.

A decision node consists of a list of predicates-transition pairs plus a default transition. Predicates are evaluated in order or appearance until one of them evaluates to true and the corresponding transition is taken. If none of the predicates evaluates to true the default transition is taken.

Predicates are JSP Expression Language (EL) expressions (refer to section 4.2 of this document) that resolve into a boolean value, true or false . For example:

```

    ${wf:nodeStatus('myfirstjob') == 'OK'}
    ${hdfs:size('/usr/foo/myinputdir') gt 10 * GB}
```

Syntax:

```
<workflow-app name="[WF-DEF-NAME]" xmlns="uri:oozie:workflow:0.1">
  ...
  <decision name="[NODE-NAME]">
    <switch>
      <case to="[NODE_NAME]">[PREDICATE]</case>
      ...
      <case to="[NODE_NAME]">[PREDICATE]</case>
      <default to="[NODE_NAME]"/>
    </switch>
  </decision>
  ...
</workflow-app>
```

The name attribute in the decision node is the name of the decision node.

Each case elements contains a predicate an a transition name. The predicate ELs are evaluated in order until one returns true and the corresponding transition is taken.

The default element indicates the transition to take if none of the predicates evaluates to true .

All decision nodes must have a default element to avoid bringing the workflow into an error state if none of the predicates evaluates to true.

Example:

```
<workflow-app name="foo-wf" xmlns="uri:oozie:workflow:0.1">
  ...
  <decision name="mydecision">
    <switch>
      <case to="reconsolidatejob">
        ${hdfs:size(secondjobOutputDir) gt 10 * GB}
      </case>
      <case to="rexpandjob">${hdfs:size(secondjobOutputDir) lt 100 * MB}</eval>
      <case to="recomputejob">
    ${
      hadoop:counters('secondjob')['org.apache.hadoop.mapred.Task.Counter']['REDUCE_OUTPUT_RECORD']
      lt 1000000
    }
      </case>
      <default to="end"/>
    </switch>
  </decision>
  ...
</workflow-app>
```

3.1.5 Fork and Join Control Nodes

A `fork` node splits one path of execution into multiple concurrent paths of execution.

A `join` node waits until every concurrent execution path of a previous `fork` node arrives to it.

The `fork` and `join` nodes must be used in pairs. The `join` node assumes concurrent execution paths are children of the same `fork` node.

Syntax:

```
<workflow-app name="[WF-DEF-NAME]" xmlns="uri:oozie:workflow:0.1">
  ...
  <fork name="[FORK-NODE-NAME]">
    <path start="[NODE-NAME]" />
    ...
    <path start="[NODE-NAME]" />
  </fork>
  ...
  <join name="[JOIN-NODE-NAME]" to="[NODE-NAME]" />
</workflow-app>
```

The `name` attribute in the `fork` node is the name of the workflow fork node. The `start` attribute in the `path` elements in the `fork` node indicate the name of the workflow node that will be part of the concurrent execution paths.

The `name` attribute in the `join` node is the name of the workflow join node. The `to` attribute in the `join` node indicates the name of the workflow node that will be executed after all concurrent execution paths of the corresponding `fork` arrive to the `join` node.

Example:

```
<workflow-app name="sample-wf" xmlns="uri:oozie:workflow:0.1">
  ...
  <fork name="forking">
    <path start="firstparalleljob"/>
    <path start="secondparalleljob"/>
  </fork>
  <action name="firstparalleljob">
    <map-reduce>
      <job-tracker>foo:9001</job-tracker>
      <name-node>bar:9000</name-node>
      <job-xml>job1.xml</job-xml>
    </map-reduce>
    <ok to="joining"/>
    <error to="kill"/>
  </action>
  <map-reduce name="secondparalleljob">
    <map-reduce>
      <job-tracker>foo:9001</job-tracker>
      <name-node>bar:9000</name-node>
      <job-xml>job2.xml</job-xml>
    </map-reduce>
    <ok to="joining"/>
    <error to="kill"/>
  </map-reduce>
  <join name="joining" to="nextaction"/>
</workflow-app>
```

3.2 Workflow Action Nodes

Action nodes are the mechanism by which a workflow triggers the execution of a computation/processing task.

3.2.1 Action Basis

The following sub-sections define common behavior and capabilities for all action types.

3.2.1.1 Action Computation/Processing Is Always Remote

All computation/processing tasks triggered by an action node are remote to Oozie. No workflow application specific computation/processing task is executed within Oozie.

3.2.1.2 Actions Are Asynchronous

All computation/processing tasks triggered by an action node are executed asynchronously by Oozie. For most types of computation/processing tasks triggered by workflow action, the workflow job has to wait until the computation/processing task completes before transitioning to the following node in the workflow.

The exception is the `fs` action that is handled as a synchronous action.

Oozie can detect completion of computation/processing tasks by two different means, callbacks and polling.

When a computation/processing task is started by Oozie, Oozie provides a unique callback URL to the task, the task should invoke the given URL to notify its completion.

For cases that the task failed to invoke the callback URL for any reason (i.e. a transient network failure) or when the type of task cannot invoke the callback URL upon completion, Oozie has a mechanism to poll computation/processing tasks for completion.

3.2.1.3 Actions Have 2 Transitions, `=ok=` and `=error=`

If a computation/processing task -triggered by a workflow- completes successfully, it transitions to `ok`.

If a computation/processing task -triggered by a workflow- fails to complete successfully, it transitions to `error`.

If a computation/processing task exits in error, the computation/processing task must provide `error-code` and `error-message` information to Oozie. This information can be used from `decision` nodes to implement a fine grain error handling at workflow application level.

Each action type must clearly define all the error codes it can produce.

3.2.1.4 Action Recovery

Under certain conditions, Oozie will provide a recovery mechanism for computation/processing task triggered by a workflow action node before the action exits in error.

Depending on the nature of the action failure Oozie will have different recovery strategies.

If the action failure is of a recoverable transient nature, Oozie must perform retries after a pre-defined time interval. The number of retries and timer interval for a type of action must be pre-configured at Oozie level. Workflow jobs can override such configuration.

Examples of a transient failures are network problems or a remote system temporary unavailable.

If the action failure is of a recoverable non-transient nature, Oozie will suspend the workflow job until an manual or programmatic intervention resumes the workflow job and the action is retried. It is the responsibility of an administrator or an external managing system to perform any necessary cleanup before retrying the action.

If the action failure is of type error, Oozie will perform the error transition for the action.

3.2.2 Map-Reduce Action

The `map-reduce` action starts a Hadoop map/reduce job from a workflow. Hadoop jobs can be Java Map/Reduce jobs or streaming jobs.

A `map-reduce` action can be configured to perform file system cleanup and directory creation before starting the map reduce job. This capability enables Oozie to retry a Hadoop job in the situation of a transient failure (Hadoop checks the non-existence of the job output directory and then creates it when the Hadoop job is starting, thus a retry without cleanup of the job output directory would fail).

The workflow job will wait until the Hadoop map/reduce job completes before continuing to the next action in the workflow execution path.

The counters of the Hadoop job and job exit status (`=FAILED=`, `KILLED` or `SUCCEEDED`) must be available to the workflow job after the Hadoop jobs ends. This information can be used from within decision nodes and other actions configurations.

The `map-reduce` action has to be configured with all the necessary Hadoop JobConf properties to run the Hadoop map/reduce job.

Hadoop JobConf properties can be specified in a JobConf XML file bundled with the workflow application or they can be indicated inline in the `map-reduce` action configuration.

The configuration properties are loaded in the following order, `streaming`, `job-xml` and `configuration`,

and later values override earlier values.

Streaming and inline property values can be parameterized (templated) using EL expressions.

The Hadoop `mapred.job.tracker` and `fs.default.name` properties must not be present in the `job-xml` and inline configuration.

3.2.2.1 Adding Files and Archives for the Job

The `file`, `archive` elements make available, to map-reduce jobs, files and archives. If the specified path is relative, it is assumed the file or archiver are within the application directory, in the corresponding sub-path. If the path is absolute, the file or archive it is expected in the given absolute path.

Files specified with the `file` element, will be symbolic links in the home directory of the task.

If a file is a native library (an `.so` or a `.so.#` file), it will be symlinked as and `.so` file in the task running directory, thus available to the task JVM.

To force a symlink for a file on the task running directory, use a `#` followed by the symlink name. For example `'mycat.sh#cat'`.

Refer to Hadoop distributed cache documentation for details more details on files and archives.

3.2.2.2 Streaming

Streaming information can be specified in the `streaming` element.

The `mapper` and `reducer` elements are used to specify the executable/script to be used as mapper and reducer.

User defined scripts must be bundled with the workflow application and they must be declared in the `files` element of the streaming configuration. If they are not declared in the `files` element of the configuration it is assumed they will be available (and in the command `PATH`) of the Hadoop slave machines.

Some streaming jobs require Files found on HDFS to be available to the mapper/reducer scripts. This is done using the `file` and `archive` elements described in the previous section.

The Mapper/Reducer can be overridden by a `mapred.mapper.class` or `mapred.reducer.class` properties in the `job-xml` file or `configuration` elements.

3.2.2.3 Syntax

```
<workflow-app name="[WF-DEF-NAME]" xmlns="uri:oozie:workflow:0.1">
  ...
  <action name="[NODE-NAME]">
    <map-reduce>
      <job-tracker>[JOB-TRACKER]</job-tracker>
      <name-node>[NAME-NODE]</name-node>
      <prepare>
        <delete path="[PATH]" />
        ...
        <mkdir path="[PATH]" />
        ...
      </prepare>
      <streaming>
        <mapper>[MAPPER-PROCESS]</mapper>
        <reducer>[REDUCER-PROCESS]</reducer>
        <record-reader>[RECORD-READER-CLASS]</record-reader>
        <record-reader-mapping>[NAME=VALUE]</record-reader-mapping>
        ...
        <env>[NAME=VALUE]</env>
        ...
      </streaming>
      <job-xml>[JOB-XML-FILE]</job-xml>
      <configuration>
        <property>
          <name>[PROPERTY-NAME]</name>
          <value>[PROPERTY-VALUE]</value>
        </property>
        ...
      </configuration>
      <file>[FILE-PATH]</file>
      ...
      <archive>[FILE-PATH]</archive>
      ...
    </map-reduce>
    <error to="[NODE-NAME]" />
  </action>
  ...
</workflow-app>
```

The `prepare` element, if present, indicates a list of path do delete before starting the job. This should be used exclusively for directory cleanup for the job to be executed. The delete operation will be performed in the `fs.default.name` filesystem.

The `job-xml` element, if present, must refer to a Hadoop JobConf `job.xml` file bundled in the workflow application. The `job-xml` element is optional and if present it can be only one.

The `configuration` element, if present, contains JobConf properties for the Hadoop job.

Properties specified in the configuration element override properties specified in the file specified in the job-xml element.

Example:

```
<workflow-app name="foo-wf" xmlns="uri:oozie:workflow:0.1">
  ...
  <action name="myfirstHadoopJob">
    <map-reduce>
      <job-tracker>foo:9001</job-tracker>
      <name-node>bar:9000</name-node>
      <prepare>
        <delete path="/usr/tucu/output-data"/>
      </prepare>
      <job-xml>/myfirstjob.xml</job-xml>
      <configuration>
        <property>
          <name>mapred.input.dir</name>
          <value>/usr/tucu/input-data</value>
        </property>
        <property>
          <name>mapred.output.dir</name>
          <value>/usr/tucu/input-data</value>
        </property>
        <property>
          <name>mapred.reduce.tasks</name>
          <value>${firstJobReducers}</value>
        </property>
      </configuration>
    </map-reduce>
    <ok to="myNextAction"/>
    <error to="errorCleanup"/>
  </action>
  ...
</workflow-app>
```

In the above example, the number of Reducers to be used by the Map/Reduce job has to be specified as a parameter of the workflow job configuration when creating the workflow job.

Streaming Example:

```
<workflow-app name="sample-wf" xmlns="uri:oozie:workflow:0.1">
  ...
  <action name="firstjob">
    <map-reduce>
      <job-tracker>foo:9001</job-tracker>
      <name-node>bar:9000</name-node>
      <prepare>
        <delete path="${output}"/>
      </prepare>
      <streaming>
        <mapper>/bin/bash testarchive/bin/mapper.sh testfile</mapper>
        <reducer>/bin/bash testarchive/bin/reducer.sh</reducer>
      </streaming>
      <configuration>
        <property>
          <name>mapred.input.dir</name>
          <value>${input}</value>
        </property>
        <property>
          <name>mapred.output.dir</name>
          <value>${output}</value>
        </property>
        <property>
          <name>stream.num.map.output.key.fields</name>
          <value>3</value>
        </property>
      </configuration>
      <file>/users/blabla/testfile.sh#testfile</file>
      <archive>/users/blabla/testarchive.jar#testarchive</archive>
    </map-reduce>
    <ok to="end"/>
    <error to="kill"/>
  </action>
  ...
</workflow-app>
```

3.2.3 Pig Action

The pig action starts a Pig job.

The workflow job will wait until the pig job completes before continuing to the next action.

The pig action has to be configured with the job-tracker, name-node, pig script and the necessary parameters and configuration to run the Pig job.

A pig action can be configured to perform HDFS files/directories cleanup before starting the Pig job. This capability enables Oozie to retry a Pig job in the situation of a transient failure (Pig creates temporary directories for intermediate data, thus a retry without cleanup would fail).

Hadoop JobConf properties can be specified in a JobConf XML file bundled with the workflow application or

they can be indicated inline in the `pig` action configuration.

The configuration properties are loaded in the following order, `job-xml` and `configuration`, and later values override earlier values.

Inline property values can be parameterized (templated) using EL expressions.

The Hadoop `mapred.job.tracker` and `fs.default.name` properties must not be present in the `job-xml` and inline configuration.

As with Hadoop map-reduce jobs, it is possible to add files and archives to be available to the Pig job, refer to section [\[#FilesArchives\]\[Adding Files and Archives for the Job\]](#).

Syntax:

```
<workflow-app name="[WF-DEF-NAME]" xmlns="uri:oozie:workflow:0.1">
  ...
  <action name="[NODE-NAME]">
    <pig>
      <job-tracker>[JOB-TRACKER]</job-tracker>
      <name-node>[NAME-NODE]</name-node>
      <prepare>
        <delete path="[PATH]"/>
        ...
        <mkdir path="[PATH]"/>
        ...
      </prepare>
      <job-xml>[JOB-XML-FILE]</job-xml>
      <configuration>
        <property>
          <name>[PROPERTY-NAME]</name>
          <value>[PROPERTY-VALUE]</value>
        </property>
        ...
      </configuration>
      <script>[PIG-SCRIPT]</script>
      <parameters>
        <param>[PARAM-VALUE]</param>
        ...
        <param>[PARAM-VALUE]</param>
      </parameters>
      <file>[FILE-PATH]</file>
      ...
      <archive>[FILE-PATH]</archive>
      ...
    </pig>
    <ok to="[NODE-NAME]"/>
    <error to="[NODE-NAME]"/>
  </action>
  ...
</workflow-app>
```

The `prepare` element, if present, indicates a list of path do delete before starting the job. This should be used exclusively for directory cleanup for the job to be executed.

The `job-xml` element, if present, must refer to a Hadoop JobConf `job.xml` file bundled in the workflow application. The `job-xml` element is optional and if present it can be only one.

The `configuration` element, if present, contains JobConf properties for the underlying Hadoop jobs.

Properties specified in the `configuration` element override properties specified in the file specified in the `job-xml` element.

The inline and `job-xml` configuration properties are passed to the Hadoop jobs submitted by Pig runtime.

The `script` element contains the pig script to execute. The pig script can be templated with variables of the form `${VARIABLE}`. The values of these variables can then be specified using the `params` element.

NOTE: Oozie will perform the parameter substitution before firing the pig job. This is different from the [parameter substitution mechanism provided by Pig](#), which has a few limitations.

The `params` element, if present, contains parameters to be passed to the pig script.

All the above elements can be parameterized (templated) using EL expressions.

Example:

```
<workflow-app name="sample-wf" xmlns="uri:oozie:workflow:0.1">
  ...
  <action name="myfirstpigjob">
    <pig>
      <job-tracker>foo:9001</job-tracker>
      <name-node>bar:9000</name-node>
      <prepare>
        <delete path="${jobOutput}"/>
      </prepare>
      <configuration>
        <property>
          <name>mapred.compress.map.output</name>
          <value>>true</value>
        </property>
      </configuration>
      <pig-script>/mypigscript.pig</pig-script>
      <parameters>
        <param>InputDir=/home/tucu/input-data</param>
      </parameters>
    </pig>
  </action>
</workflow-app>
```

```

        <param>OutputDir=${jobOutput}</param>
    </parameters>
</pig>
<ok to="myotherjob"/>
<error to="errorcleanup"/>
</action>
...
</workflow-app>

```

3.2.4 Fs action

The `fs` action allows to manipulate files and directories in HDFS from a workflow application. The supported commands are `move`, `delete` and `mkdir`.

The FS commands are executed synchronously from within the FS action, the workflow job will wait until the specified file commands are completed before continuing to the next action.

Path names specified in the `fs` action can be parameterized (templated) using EL expressions.

Each file path must specify the file system URI, for move operations, the target must not specify the system URI.

IMPORTANT: All the commands within `fs` action do not happen atomically, if a `fs` action fails half way in the commands being executed, successfully executed commands are not rolled back. The `fs` action, before executing any command must check that source paths exist and target paths don't exist, thus failing before executing any command. Therefore the validity of all paths specified in one `fs` action are evaluated before any of the file operation are executed. Thus there is less chance of an error occurring while the `fs` action executes.

Syntax:

```

<workflow-app name="[WF-DEF-NAME]" xmlns="uri:oozie:workflow:0.1">
...
  <action name="[NODE-NAME]">
    <fs>
      <delete path='[PATH]'/>
      ...
      <mkdir path='[PATH]'/>
      ...
      <move source='[SOURCE-PATH]' target='[TARGET-PATH]'/>
      ...
    </fs>
    <ok to="[NODE-NAME]"/>
    <error to="[NODE-NAME]"/>
  </action>
...
</workflow-app>

```

The `delete` command deletes the specified path, if it is a directory it deletes recursively all its content and then deletes the directory.

The `mkdir` command creates the specified directory, it creates all missing directories in the path. If the directory already exist it does a no-op.

In the `move` command the `source` path must exist and the `target` path must not exist. The parent directory of the `target` path must exist, the `target` path must not specify the file system URI.

If relative paths are used it will be relative to the specified user home directory.

Example:

```

<workflow-app name="sample-wf" xmlns="uri:oozie:workflow:0.1">
...
  <action name="hdfscommands">
    <fs>
      <delete path='hdfs://foo:9000/usr/tucu/temp-data' />
      <mkdir path='archives/${wf:id()}' />
      <move source='${jobInput}' target='archives/${wf:id()}/processed-input' />
    </fs>
    <ok to="myotherjob"/>
    <error to="errorcleanup"/>
  </action>
...
</workflow-app>

```

In the above example, a directory named after the workflow job ID is created and the input of the job, passed as workflow configuration parameter, is archived under the previously created directory.

3.2.5 Ssh Action

The `ssh` action starts a shell command on a remote machine as a remote secure shell in background. The workflow job will wait until the remote shell command completes before continuing to the next action.

The shell command must be present in the remote machine and it must be available for execution via the command path.

The shell command is executed in the home directory of the specified user in the remote host.

The output (STDOUT) of the `ssh` job can be made available to the workflow job after the `ssh` job ends. This information could be used from within decision nodes. If the output of the `ssh` job is made available to the workflow job the shell command must follow the following requirements:

- The format of the output must be a valid Java Properties file.
- The size of the output must not exceed 2KB.

Syntax:

```
<workflow-app name="[WF-DEF-NAME]" xmlns="uri:oozie:workflow:0.1">
  ...
  <action name="[NODE-NAME]">
    <ssh>
      <host>[USER]@[HOST]</host>
      <command>[SHELL]</command>
      <args>[ARGUMENTS]</args>
      ...
    <capture-output/>
  </ssh>
  <ok to="[NODE-NAME]"/>
  <error to="[NODE-NAME]"/>
</action>
  ...
</workflow-app>
```

The `host` indicates the user and host where the shell will be executed.

The `command` element indicates the shell command to execute.

The `args` element, if present, contains parameters to be passed to the shell command. If more than one `args` element is present they are concatenated in order.

If the `capture-output` element is present, it indicates Oozie to capture output of the STDOUT of the `ssh` command execution. The `ssh` command output must be in Java Properties file format and it must not exceed 2KB. From within the workflow definition, the output of an `ssh` action node is accessible via the `String action:output(String node, String key)` function (Refer to section '4.2.6 Action EL Functions').

The configuration of the `ssh` action can be parameterized (templated) using EL expressions.

Example:

```
<workflow-app name="sample-wf" xmlns="uri:oozie:workflow:0.1">
  ...
  <action name="myssjob">
    </ssh>
    <host>foo@bar.com</host>
    <command>uploaddata</command>
    <args>jdbc:derby://bar.com:1527/myDB</args>
    <args>hdfs://foobar.com:9000/usr/tucu/myData</args>
  </ssh>
  <ok to="myotherjob"/>
  <error to="errorcleanup"/>
</action>
  ...
</workflow-app>
```

In the above example, the `uploaddata` shell command is executed with two arguments, `jdbc:derby://foo.com:1527/myDB` and `.hdfs://foobar.com:9000/usr/tucu/myData`.

The `uploaddata` shell must be available in the remote host and available in the command path.

The output of the command will be ignored because the `capture-output` element is not present.

3.2.6 Sub-workflow Action

The `sub-workflow` action runs a child workflow job, the child workflow job can be in the same Oozie system or in another Oozie system.

The parent workflow job will wait until the child workflow job has completed.

Syntax:

```
<workflow-app name="[WF-DEF-NAME]" xmlns="uri:oozie:workflow:0.1">
  ...
  <action name="[NODE-NAME]">
    <sub-workflow>
      <oozie>http://[HOST]:[PORT]/[Oozie-PATH]</oozie>
      <app-path>[WF-APPLICATION-PATH]</app-path>
      <configuration>
        <property>
          <name>[PROPERTY-NAME]</name>
          <value>[PROPERTY-VALUE]</value>
        </property>
        ...
      </configuration>
    </sub-workflow>
    <ok to="[NODE-NAME]"/>
    <error to="[NODE-NAME]"/>
  </action>
  ...
</workflow-app>
```

The `oozie` element can be used to specify the URL of the target Oozie system. If this tag is absent, it is assumed that the child workflow job runs in the same Oozie system instance where the parent workflow job is running.

The `app-path` element specifies the path to the workflow application of the child workflow job.

The `configuration` section can be used to specify the job properties that are required to run the child workflow job.

The configuration of the `sub-workflow` action can be parameterized (templated) using EL expressions.

Example:

```
<workflow-app name="sample-wf" xmlns="uri:oozie:workflow:0.1">
  ...
  <action name="a">
    <sub-workflow>
      <oozie>http://myhost:8080/oozie</oozie>
      <app-path>child-wf</app-path>
      <configuration>
        <property>
          <name>input.dir</name>
          <value>${wf:id()}/second-mr-output</value>
        </property>
      </configuration>
    </sub-workflow>
    <ok to="end"/>
    <error to="kill"/>
  </action>
  ...
</workflow-app>
```

In the above example, the workflow definition with the name `child-wf` will be run on the Oozie instance at `.http://myhost:8080/oozie`. The specified workflow application must be already deployed on the target Oozie instance.

A configuration parameter `input.dir` is being passed as job property to the child workflow job.

3.2.7 Http Action

The `http` action makes an HTTP GET or POST call from within a workflow job.

The `http` action can be configured to make the workflow job to wait on a callback notification or to continue immediately to the next node in the workflow job execution path.

The output of the `http` action can be made available to the workflow job after the `http` action ends. This information could be used from within decision nodes. If the output of the `http` action is made available to the workflow job, it must follow the following requirements:

- The format of the output must be a valid Java Properties file.
- The size of the output must not exceed 2KB.

Syntax:

```
<workflow-app name="[WF-DEF-NAME]" xmlns="uri:oozie:workflow:0.1">
  ...
  <action name="[NODE-NAME]">
    <http>
      <url>[EMAIL-ADDRESS, ...]</url>
      <post content-type="[CONTENT-TYPE]">
        [PAYLOAD]
      </post>
      <mode>CONTINUE|WAIT</mode>
      <capture-output/>
    </http>
    <ok to="[NODE-NAME]"/>
    <error to="[NODE-NAME]"/>
  </action>
  ...
</workflow-app>
```

The `url` element contains the URL to be used for the HTTP call.

If the `post` element is not present, a HTTP GET call will be done to the specified URL.

The `post` element is present, a HTTP POST call will be done to the specified URL using the content type and payload specified in the `post` element.

The `mode` element indicates the behavior of the `http` action.

If the mode is `CONTINUE`, the workflow job will transition to the next node in the execution path. If the `capture-output` element is present, it indicates Oozie to capture the payload of the HTTP response as the action output.

If the mode is `WAIT`, the workflow job will wait for a callback before completing the `http` action. The `http` action must communicate the target HTTP system of the callback URL, this must be done using the `wf:callbackUrl(String stateVar)` function. If the `capture-output` element is present, it indicates Oozie to capture HTTP callback request payload.

If the `capture-output` element is present, the HTTP call response or callback request payload (depending on the mode being `CONTINUE` or `WAIT`) will be stored in the workflow job. The payload must be in Java Properties file format, the content type must be 'x-application/java-properties' and it must not exceed 2KB. From within the workflow definition, this payload is accessible via the `String action:output(String node, String key)` function (Refer to section '4.2.6 Action EL Functions').

The `url` and `post` and `mode` elements can be parameterized (templated) using EL expressions.

Example:

```

<workflow-app name="sample-wf" xmlns="uri:oozie:workflow:0.1">
  ...
  <action name="httpNotification">
    <http>
      <url>http://foo.com/notify?callback=${urlEncode(wf:callback(' $EXITSTATE$ '))}</url>
      <mode>WAIT</mode>
    </http>
    <ok to="onOK" />
    <error to="onError" />
  </action>
  ...
</workflow-app>

```

In the above example, an HTTP GET call it will done to the `.http://foo.com/notify` URL and the query string of the URL will contain the callback URL for the external system to signal Oozie that the action has completed. The external system must replace the `$EXITSTATE$` token in the callback URL with `OK` or `ERROR` before making the callback. Because the `mode` is set to `WAIT`, Oozie will wait for the callback before completing the Http action.

3.2.8 Email Node

The `email` action sends an email from within a workflow job.

Syntax:

```

<workflow-app name="[WF-DEF-NAME]" xmlns="uri:oozie:workflow:0.1">
  ...
  <action name="[NODE-NAME]">
    <email>
      <to>[EMAIL-ADDRESS, ...]</to>
      <subject>[SUBJECT]</subject>
      <message>[MESSAGE]</message>
    </email>
    <ok to="[NODE-NAME]" />
    <error to="[NODE-NAME]" />
  </action>
  ...
</workflow-app>

```

The `to` element contains the emails addresses to send the message, they must be separated by commas.

The `subject` element contains the subject for the message.

the `message` element contains the body of the message.

The `to`, `subject` and `message` elements can be parameterized (templated) using EL expressions.

Example:

```

<workflow-app name="sample-wf" xmlns="uri:oozie:workflow:0.1">
  ...
  <action name="notify">
    <email>
      <to>foo@bar.com, ${adminEmail}</to>
      <subject>Workflow ${wf:id()} finished second phase on ${timestamp()}</subject>
      <message/>
    </email>
    <ok to="next" />
    <error to="next" />
  </action>
  ...
</workflow-app>

```

4 Parameterization of Workflows

Workflow definitions can be parameterized.

When workflow node is executed by Oozie all the ELs are resolved into concrete values.

The parameterization of workflow definitions it done using JSP Expression Language syntax from the [JSP 2.0 Specification \(JSP.2.3\)](#), allowing not only to support variables as parameters but also functions and complex expressions.

EL expressions can be used in the configuration values of action and decision nodes. They can be used in XML attribute values and in XML element and attribute values.

They cannot be used in XML element and attribute names. They cannot be used in the name of a node and they cannot be used within the `transition` elements of a node.

4.1 Workflow Job Properties (or Parameters)

When a workflow job is submitted to Oozie, the submitter may specify as many workflow job properties as required (similar to Hadoop JobConf properties).

Workflow applications may define default values for the workflow job parameters. They must be defined in a `config-default.xml` file bundled with the workflow application archive (refer to section '7 Workflow Applications Packaging'). Workflow job properties have precedence over the default values.

Properties that are a valid Java identifier, `[A-Za-z_][0-9A-Za-z_]*`, are available as `'${NAME}'` variables

within the workflow definition.

Properties that are not valid Java Identifier, for example 'job.tracker', are available via the `String wf:conf(String name)` function. Valid identifier properties are available via this function as well.

Using properties that are valid Java identifiers result in a more readable and compact definition.

By using properties **Example:*

Parameterized Workflow definition:

```
<workflow-app name='hello-wf' xmlns="uri:oozie:workflow:0.1">
  ...
  <action name='firstjob'>
    <map-reduce>
      <job-tracker>${jobTracker}</job-tracker>
      <name-node>${nameNode}</name-node>
      <configuration>
        <property>
          <name>mapred.mapper.class</name>
          <property>com.foo.FirstMapper</property>
        </property>
        <property>
          <name>mapred.reducer.class</name>
          <property>com.foo.FirstReducer</property>
        </property>
        <property>
          <name>mapred.input.dir</name>
          <property>${inputDir}</property>
        </property>
        <property>
          <name>mapred.output.dir</name>
          <property>${outputDir}</property>
        </property>
      </configuration>
    </map-reduce>
    <ok to='secondjob' />
    <error to='killcleanup' />
  </action>
  ...
</workflow-app>
```

When submitting a workflow job for the workflow definition above, 3 workflow job properties must be specified:

- `jobTracker`:
- `inputDir`:
- `outputDir`:

4.2 Expression Language Functions

Oozie, besides allowing the use of workflow job properties to parameterize workflow jobs, it provides a set of build in EL functions that enable a more complex parameterization of workflow action nodes as well as the predicates in decision nodes.

4.2.1 Basic EL Constants

- **KB**: 1024, one kilobyte.
- **MB**: 1024 * KB, one megabyte.
- **GB**: 1024 * MB, one gigabyte.
- **TB**: 1024 * GB, one terabyte.
- **PB**: 1024 * TG, one petabyte.

All the above constants are of type `Long`.

4.2.2 Basic EL Functions

String firstNotNull(String value1, String value2)

It returns the first not `null` value, or `null` if both are `null`.

Note that if the output of this function is `null` and it is used as string, the EL library converts it to an empty string. This is the common behavior when using `firstNotNull()` in node configuration sections.

String concat(String s1, String s2)

It returns the concatenation of 2 strings. A string with `null` value is considered as an empty string.

String trim(String s)

It returns the trimmed value of the given string. A string with `null` value is considered as an empty string.

String urlEncode(String s)

It returns the URL UTF-8 encoded value of the given string. A string with `null` value is considered as an empty string.

String timestamp()

It returns the UTC current date and time in W3C format down to the second (YYYY-MM-DDThh:mm:ss.sZ). I.e.: 1997-07-16T19:20:30.45Z

4.2.3 Workflow EL Functions

String wf:id()

It returns the workflow job ID for the current workflow job.

String wf:name()

It returns the workflow application name for the current workflow job.

String wf:appPath()

It returns the workflow application path for the current workflow job.

String wf:conf(String name)

It returns the value of the workflow job configuration property for the current workflow job, or an empty string if undefined.

String wf:user()

It returns the user name that started the current workflow job.

String wf:group()

It returns the group name for the current workflow job.

String wf:callback(String stateVar)

It returns the callback URL for the current workflow action node, `stateVar` can be a valid exit state (=OK= or `ERROR`) for the action or a token to be replaced with the exit state by the remote system executing the task.

String wf:transition(String node)

It returns the transition taken by the specified workflow action node, or an empty string if the action has not been executed or it has not completed yet.

String wf:lastError()

It returns the name of the last workflow action node that exit with an `ERROR` exit state, or an empty string if no action has exited with `ERROR` state in the current workflow job.

String wf:errorCode(String node)

It returns the error code for the specified action node, or an empty string if the action node has not exited with `ERROR` state.

Each type of action node must define its complete error code list.

String wf:errorMessage(String message)

It returns the error message for the specified action node, or an empty string if no action node has not exited with `ERROR` state.

The error message can be useful for debugging and notification purposes.

int wf:run()

It returns the run number for the current workflow job, normally 1 unless the workflow job is re-run, in which case indicates the current run.

Map wf:actionData(String node)

This function is only applicable to action nodes that produce output data on completion.

Http and Ssh action node types are capable of producing output data if the tag is used. The output data is in a Java Properties format and via this EL function it is available as a `Map`.

int wf:actionExternalId(String node)

It returns the external Id for an action node, or an empty string if the action has not been executed or it has not completed yet.

int wf:actionTrackerUri(String node)

It returns the tracker URI for an action node, or an empty string if the action has not been executed or it has not completed yet.

int wf:actionExternalStatus(String node)

It returns the external status for an action node, or an empty string if the action has not been executed or it has not completed yet.

4.2.4 Hadoop EL Constants

- **RECORDS:** Hadoop record counters group name.
- **MAP_IN:** Hadoop mapper input records counter name.
- **MAP_OUT:** Hadoop mapper output records counter name.
- **REDUCE_IN:** Hadoop reducer input records counter name.
- **REDUCE_OUT:** Hadoop reducer input record counter name.
- **GROUPS:** 1024 * Hadoop mapper/reducer record groups counter name.

4.2.5 Hadoop EL Functions

Map < String, Map < String, Long > > hadoop:jobCounters(String node)

It returns the counters for a job submitted by a Hadoop action node. It returns 0 if the if the Hadoop job has not started yet and for undefined counters.

The outer Map key is a counter group name. The inner Map value is a Map of counter names and counter values.

The Hadoop EL constants defined in the previous section provide access to the Hadoop built in record counters.

4.2.6 HDFS EL Functions

For all the functions in this section the path must include the FS URI. For example

```
hdfs://foo:9000/user/tucu .
```

boolean hdfs:exists(String path)

It returns `true` or `false` depending if the specified path URI exists or not.

boolean hdfs:isDir(String path)

It returns `true` if the specified path URI exists and it is a directory, otherwise it returns `false` .

boolean hdfs:dirSize(String path)

It returns the size in bytes of all the files in the specified path. If the path is not a directory, or if it does not exist it returns -1. It does not work recursively, only computes the size of the files under the specified path.

boolean hdfs:fileSize(String path)

It returns the size in bytes of specified file. If the path is not a file, or if it does not exist it returns -1.

boolean hdfs:blockSize(String path)

It returns the block size in bytes of specified file. If the path is not a file, or if it does not exist it returns -1.

5 Oozie Notifications

Workflow jobs can be configured to make an HTTP GET notification upon start and end of a workflow action node and upon the completion of a workflow job.

Oozie will make a best effort to deliver the notifications, in case of failure it will retry the notification a pre-configured number of times at a pre-configured interval before giving up.

5.1 Workflow Job Status Notification

If the `oozie.workflow.notification.url` property is present in the workflow job properties when submitting the job, Oozie will make a notification to the provided URL when the workflow job changes its status.

If the URL contains any of the following tokens, they will be replaced with the actual values by Oozie before making the notification:

- `$jobId` : The workflow job ID
- `$status` : the workflow current state

5.2 Node Start and End Notifications

If the `oozie.action.notification.url` property is present in the workflow job properties when submitting the job, Oozie will make a notification to the provided URL every time the workflow job enters and exits an action node. For decision nodes, Oozie will send a single notification with the name of the first evaluation that resolved to `true` .

If the URL contains any of the following tokens, they will be replaced with the actual values by Oozie before making the notification:

- `$jobId` : The workflow job ID
- `$nodeName` : The name of the workflow node
- `$status` : If the action has not completed yet, it contains the action status 'S:'. If the action has ended, it contains the action transition 'T:'

6 User Propagation

When submitting a workflow job, the configuration must contain a `user.name` property. If security is enabled, Oozie must ensure that the value of the `user.name` property in the configuration match the user of the credential present in the protocol (web services) request.

When submitting a workflow job, the configuration may contain a `group.name` property. If security is enabled, Oozie must ensure that the user of the request belongs to the specified group.

The specified user and group names are assigned to the created job.

Oozie must propagate the specified user and group to the system executing the actions.

It is not allowed for map-reduce, pig and fs actions to override user/group information.

For SSH actions, there must be a system configuration option in Oozie that disables user propagation. By default this switch must be turned off, meaning that Oozie uses the workflow job user name as the user for the SSH invocation. In this case, if the SSH node 'host' element contains a user name (i.e. 'tucu@foo.com') the job must fail. If the switch is turned on, Oozie will accept SSH node 'host' element values containing a user name, if the user name is not present, it will propagate the workflow job user name.

7 Workflow Applications Packaging

While Oozie encourages the use of self-contained applications (J2EE application model), it does not enforce it.

Workflow applications are installed in an HDFS directory. To submit a job for a workflow application the path to the HDFS directory where the application is must be specified.

The layout of a workflow application directory is:

```
- /workflow.xml
- /config-default.xml
|
- /lib/ (*.jar;*.so)
```

A workflow application must contain at least the workflow definition, the `workflow.xml` file.

All configuration files and scripts (Pig and shell) needed by the workflow action nodes should be under the application HDFS directory.

All the JAR files and native libraries within the application 'lib/' directory are automatically added to the map-reduce and pig jobs `classpath` and `LD_PATH`.

Additional JAR files and native libraries not present in the application 'lib/' directory can be specified in map-reduce and pig actions with the 'file' element (refer to the map-reduce and pig documentation).

The `config-default.xml` file defines, if any, default values for the workflow job parameters. This file must be use the Hadoop Configuration XML format. The `user.name` property cannot be specified in this file.

Any other resources like `job.xml` files referenced from a workflow action action node must be included under the corresponding path, relative paths always start from the root of the workflow application.

8 External Data Assumptions

Oozie runs workflow jobs under the assumption all necessary data to execute an action is readily available at the time the workflow job is about to executed the action.

In addition, it is assumed, but it is not the responsibility of Oozie, that all input data used by a workflow job is immutable for the duration of the workflow job.

9 Workflow Jobs Lifecycle

A workflow job can have be in any of the following states:

PREP: When a workflow job is first create it will be in **PREP** state. The workflow job is defined but it is not running.

RUNNING: When a **CREATED** workflow job is started it goes into **RUNNING** state, it will remain in **RUNNING** state while it does not reach its end state, ends in error or it is suspended.

SUSPENDED: A **RUNNING** workflow job can be suspended, it will remain in **SUSPENDED** state until the workflow job is resumed or it is killed.

SUCCEEDED: When a **RUNNING** workflow job reaches the `end` node it ends reaching the **SUCCEEDED** final state.

KILLED: When a **CREATED**, **RUNNING** or **SUSPENDED** workflow job is killed by an administrator or the owner via a request to Oozie the workflow job ends reaching the **KILLED** final state.

FAILED: When a **RUNNING** workflow job fails due to an unexpected error it ends reaching the **FAILED** final state.

Workflow job state valid transitions:

- --> PREP
- PREP --> RUNNING | KILLED
- RUNNING --> SUSPENDED | SUCCEEDED | KILLED | FAILED
- SUSPENDED --> RUNNING | KILLED

10 Workflow Jobs Recovery (re-run)

Oozie must provide a mechanism by which a a failed workflow job can be resubmitted and executed starting after any action node that has completed its execution in the prior run. This is specially useful when the already executed action of a workflow job are too expensive to be re-executed.

It is the responsibility of the user resubmitting the workflow job to do any necessary cleanup to ensure that the rerun will not fail due to not cleaned up data from the previous run.

When starting a workflow job in recovery mode, the user must indicate what workflow nodes in the workflow should be skipped. All workflow nodes indicated as skipped must have completed in the previous run. If a workflow node has not completed its execution in its previous run, and during the recovery submission is flagged as a node to be skipped, the recovery submission must fail.

The recovery workflow job will run under the same workflow job ID as the original workflow job.

To submit a recovery workflow job the target workflow job to recover must be in an end state (=SUCCEEDED=, FAILED or KILLED).

A recovery run could be done using a new workflow application path under certain constraints (see next paragraph). This is to allow users to do a one off patch for the workflow application without affecting other running jobs for the same application.

The workflow application use for a re-run must match the execution flow, node types, node names and node configuration for all executed nodes that will be skipped during recovery. This cannot be checked by Oozie, it is the responsibility of the user to ensure this is the case.

Oozie provides the `int wf:run()` EL function to returns the current run for a job, this function allows workflow applications to perform custom logic at workflow definition level (i.e. in a `decision` node) or at action node level (i.e. by passing the value of the `wf:run()` function as a parameter to the task).

11 Oozie Web Services API, V0

The Oozie Web Services API is a HTTP REST JSON API.

All responses are in UTF-8 .

Assuming Oozie is running at `OOZIE_URL` , the following web services end points are supported:

- /versions
- /v0/admin
- /v0/job
- /v0/jobs

11.1 Versions End-Point

This endpoint is for clients to perform protocol negotiation.

It support only HTTP GET request and not sub-resources.

It returns the supported Oozie protocol versions by the server.

Current returned value is 0 .

Request:

```
GET /oozie/versions
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
.
[0]
```

11.2 Admin End-Point

This endpoint is for obtaining Oozie system status and configuration information.

It supports the following sub-resources: `status` , `os-env` , `sys-props` , `configuration` , `instrumentation` .

11.2.1 System Status

A HTTP GET request returns the system status.

Request:

```
GET /oozie/v0/admin/status
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
.
{"safeMode":false}
```

With a HTTP PUT request it is possible to bring in and out the system from safemode.

Request:

```
PUT /oozie/v0/admin/status?safemode=true
```

Response:

```
HTTP/1.1 200 OK
```

11.2.2 OS Environment

A HTTP GET request returns the Oozie system OS environment.

Request:

```
GET /oozie/v0/admin/os-env
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
.
{
  TERM: "xterm",
  JAVA_HOME: "/usr/java/latest",
  XCURSOR_SIZE: "",
  SSH_CLIENT: "::ffff:127.0.0.1 49082 22",
  XCURSOR_THEME: "default",
  INPUTRC: "/etc/inputrc",
  HISTSIZE: "1000",
  PATH: "/usr/java/latest/bin"
  KDE_FULL_SESSION: "true",
  LANG: "en_US.UTF-8",
  ...
}
```

11.2.2 Java System Properties

A HTTP GET request returns the Oozie Java system properties.

Request:

```
GET /oozie/v0/admin/java-sys-properties
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
.
{
  java.vm.version: "11.0-b15",
  sun.jnu.encoding: "UTF-8",
  java.vendor.url: "http://java.sun.com/",
  java.vm.info: "mixed mode",
  ...
}
```

11.2.2 Oozie Configuration

A HTTP GET request returns the Oozie system configuration.

Request:

```
GET /oozie/v0/admin/configuration
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
.
{
  oozie.service.SchedulerService.threads: "5",
  oozie.service.ActionService.executor.classes: "
    org.apache.oozie.dag.action.decision.DecisionActionExecutor,
    org.apache.oozie.dag.action.hadoop.HadoopActionExecutor,
    org.apache.oozie.dag.action.hadoop.FsActionExecutor
  ",
  oozie.service.CallableQueueService.threads.min: "10",
  oozie.service.DBLiteWorkflowStoreService.oozie.autoinstall: "true",
  ...
}
```

11.2.3 Oozie Instrumentation

A HTTP GET request returns the Oozie instrumentation information.

Request:

```
GET /oozie/v0/admin/instrumentation
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
.
{
  timers: [
    {
      group: "webservices",
      name: "admin-PUT",
      ticks: 3,
      ownTimeAvg: 4,
      ownTimeStdDev: 0,
      ownMinTime: 0,
      ownMaxTime: 11,
      totalTimeAvg: 4,
    }
  ]
}
```

```

    totalTimeStdDev: 0,
    totalMinTime: 0,
    totalMaxTime: 11
  },
  ...
],
samplers: [
  {
    group: "webservices"
    name: "requests",
    value: 0.5,
  },
  ...
],
variables: [
  {
    group: "jvm"
    name: "max.memory",
    value: 506920960,
  },
  {
    group: "jvm"
    name: "total.memory",
    value: 32768000,
  },
  {
    group: "jvm"
    name: "free.memory",
    value: 22557896,
  },
  ...
]
}

```

11.3 Job and Jobs End-Points

These endpoints is for submitting, managing and retrieving information of jobs.

11.3.1 Job Submission

A HTTP POST request with an XML configuration as payload creates a job.

Request:

```

POST /oozie/v0/jobs
Content-Type: application/xml;charset=UTF-8
.
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <property>
    <name>user.name</name>
    <value>tucu</value>
  </property>
  <property>
    <name>oozie.wf.application.path</name>
    <value>hdfs://foo:9000/user/tucu/myapp</value>
  </property>
  ...
</configuration>

```

Response:

```

HTTP/1.1 201 CREATED
Content-Type: application/json;charset=UTF-8
.
{
  id: "job-3"
}

```

The created job is in PREP status.

If the query string parameter 'action=start' is provided in the POST URL, the job will be started immediatly and its status will be RUNNING.

11.3.1 Managing the Job

A HTTP PUT request starts, suspends, resumes or kills a job.

Request:

```
PUT /oozie/v0/job/job-3?action=start
```

Response:

```
HTTP/1.1 200 OK
```

Valid values for the 'action' parameter are 'start', 'suspend', 'resume' and 'kill'.

11.3.1 Job Information

A HTTP GET request retrieves the job information.

Request:

```
GET /oozie/v0/job/job-3?show=info
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
.
{
  id: "0-200905191240-oozie-tucu",
  appName: "indexer-workflow",
  appPath: "hdfs://user/tucu/indexer.wf",
  user: "tucu",
  group: "other",
  status: "RUNNING",
  conf: "<configuration> ... </configuration>",
  createTime: "Thu, 01 Jan 2009 00:00:00 GMT",
  startTime: "Fri, 02 Jan 2009 00:00:00 GMT",
  endTime: null,
  run: 0,
  actions: [
    {
      id: "0-200905191240-oozie-tucu@indexer",
      name: "indexer",
      type: "map-reduce",
      conf: "<configuration> ...</configuration>",
      startTime: "Thu, 01 Jan 2009 00:00:00 GMT",
      endTime: "Fri, 02 Jan 2009 00:00:00 GMT",
      status: "OK",
      externalId: "job-123-200903101010",
      externalStatus: "SUCCEEDED",
      trackerUri: "foo:9001",
      consoleUrl: "http://foo:50040/jobdetailshistory.jsp?jobId=...",
      transition: "reporter",
      data: null,
      errorCode: null,
      errorMessage: null,
      retries: 0
    },
    ...
  ]
}
```

11.3.2 Job Definition

A HTTP GET request retrieves the workflow job definition file.

Request:

```
GET /oozie/v0/job/job-3?show=definition
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/xml;charset=UTF-8
.
<?xml version="1.0" encoding="UTF-8"?>
<workflow-app name='xyz-app' xmlns="uri:oozie:workflow:0.1">
  <start to='firstaction' />
  ...
  <end name='end' />
</workflow-app>
```

11.3.3 Job Log

A HTTP GET request retrieves the job log.

Request:

```
GET /oozie/v0/job/job-3?show=log
```

Response:

```
HTTP/1.1 200 OK
Content-Type: text/plain;charset=UTF-8
.
...
23:21:31,272 TRACE oozieapp:526 - USER[tucu] GROUP[other] TOKEN[-] APP[test-wf] JOB[0-20090518232130-oozie-tucu] ACTION[mr-1] Start
23:21:31,305 TRACE oozieapp:526 - USER[tucu] GROUP[other] TOKEN[-] APP[test-wf] JOB[0-20090518232130-oozie-tucu] ACTION[mr-1] End
...
```

11.3.4 Jobs Information

A HTTP GET request retrieves the jobs information.

Request:

```
GET /oozie/v0/jobs?filter=user=tucu
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
.
[
  {
    id: "0-200905191240-oozie-tucu",
    appName: "indexer-workflow",
    appPath: "hdfs://user/tucu/indexer.wf",
    user: "tucu",
    group: "other",
    status: "RUNNING",
    conf: "<configuration> ... </configuration>",
    createTime: "Thu, 01 Jan 2009 00:00:00 GMT",
    startTime: "Fri, 02 Jan 2009 00:00:00 GMT",
    endTime: null,
    run: 0,
    actions: [ ]
  },
  ...
]
```

No action information is returned when querying for multiple jobs.

The syntax for the filter is

```
[NAME=VALUE][;NAME=VALUE]*
```

Valid filter names are:

- name: the workflow application name from the workflow definition
- user: the user that submitted the job
- group: the group for the job
- status: the status of the job

The query will do an AND among all the filter names.

The query will do an OR among all the filter values for the same name. Multiple values must be specified as different name value pairs.

Additionally the `start` and `len` parameters can be used for pagination. The start parameter is base 1.

12 Client API

Oozie will provide a Java [Client API](#) that allows to perform all common workflow job operations.

The client API includes a [LocalOozie class](#) useful for testing a workflow from within an IDE and for unit testing purposes.

The Client API will be implemented as a client of the Web Services API.

13 Command Line Tools

Oozie will provide command line tool that allows to perform all common workflow job operations.

The command line tool will be implemented as a client of the Web Services API.

14 Web UI Console

Oozie will provide a read-only Web based console that allows to allow to monitor Oozie system status, workflow applications status and workflow jobs status.

The Web base console will be implemented as a client of the Web Services API.

15 Customizing Oozie with Extensions

Out of the box Oozie provides support for a predefined set of action node types and Expression Language functions.

Oozie will provide a well defined API, [Action executor API](#), to add support for additional action node types.

Extending Oozie should not require any code change to the Oozie codebase. It will require adding the JAR files providing the new functionality and declaring them in Oozie system configuration.

16 Workflow Jobs Priority

Oozie does not handle workflow jobs priority. As soon as a workflow job is ready to do a transition, Oozie will trigger the transition. Workflow transitions and action triggering are assumed to be fast and lightweight operations.

Oozie assumes that the remote systems are properly sized to handle the amount of remote jobs Oozie will

trigger.

Any prioritization of jobs in the remote systems is outside of the scope of Oozie.

Workflow applications can influence the remote systems priority via configuration if the remote systems support it.

Appendixes

Appendix A, Oozie XML-Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:workflow="uri:oozie:workflow:0.1"
  elementFormDefault="qualified" targetNamespace="uri:oozie:workflow:0.1">
  <xs:element name="workflow-app" type="workflow:WORKFLOW-APP"/>
  <xs:simpleType name="IDENTIFIER">
    <xs:restriction base="xs:string">
      <xs:pattern value="([a-zA-Z][\-_a-zA-Z0-9])*{1,19}"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:complexType name="WORKFLOW-APP">
    <xs:sequence>
      <xs:element name="start" type="workflow:START" minOccurs="1" maxOccurs="1"/>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element name="decision" type="workflow:DECISION" minOccurs="1" maxOccurs="1"/>
        <xs:element name="fork" type="workflow:FORK" minOccurs="1" maxOccurs="1"/>
        <xs:element name="join" type="workflow:JOIN" minOccurs="1" maxOccurs="1"/>
        <xs:element name="kill" type="workflow:KILL" minOccurs="1" maxOccurs="1"/>
        <xs:element name="action" type="workflow:ACTION" minOccurs="1" maxOccurs="1"/>
      </xs:choice>
      <xs:element name="end" type="workflow:END" minOccurs="1" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="name" type="workflow:IDENTIFIER" use="required"/>
  </xs:complexType>
  <xs:complexType name="START">
    <xs:attribute name="name" type="workflow:IDENTIFIER" use="required"/>
  </xs:complexType>
  <xs:complexType name="END">
    <xs:attribute name="name" type="workflow:IDENTIFIER" use="required"/>
  </xs:complexType>
  <xs:complexType name="DECISION">
    <xs:sequence>
      <xs:element name="switch" type="workflow:SWITCH" minOccurs="1" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="name" type="workflow:IDENTIFIER" use="required"/>
  </xs:complexType>
  <xs:element name="switch" type="workflow:SWITCH"/>
  <xs:complexType name="SWITCH">
    <xs:sequence>
      <xs:sequence>
        <xs:element name="case" type="workflow:CASE" minOccurs="1" maxOccurs="unbounded"/>
        <xs:element name="default" type="workflow:DEFAULT" minOccurs="1" maxOccurs="1"/>
      </xs:sequence>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="CASE">
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="to" type="workflow:IDENTIFIER" use="required"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
  <xs:complexType name="DEFAULT">
    <xs:attribute name="to" type="workflow:IDENTIFIER" use="required"/>
  </xs:complexType>
  <xs:complexType name="FORK_TRANSITION">
    <xs:attribute name="start" type="workflow:IDENTIFIER" use="required"/>
  </xs:complexType>
  <xs:complexType name="FORK">
    <xs:sequence>
      <xs:element name="path" type="workflow:FORK_TRANSITION" minOccurs="2" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="name" type="workflow:IDENTIFIER" use="required"/>
  </xs:complexType>
  <xs:complexType name="JOIN">
    <xs:attribute name="name" type="workflow:IDENTIFIER" use="required"/>
    <xs:attribute name="to" type="workflow:IDENTIFIER" use="required"/>
  </xs:complexType>
  <xs:element name="kill" type="workflow:KILL"/>
  <xs:complexType name="KILL">
    <xs:sequence>
      <xs:element name="message" type="xs:string" minOccurs="1" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="name" type="workflow:IDENTIFIER" use="required"/>
  </xs:complexType>
  <xs:complexType name="ACTION_TRANSITION">
    <xs:attribute name="to" type="workflow:IDENTIFIER" use="required"/>
  </xs:complexType>
  <xs:element name="map-reduce" type="workflow:MAP-REDUCE"/>
  <xs:element name="pig" type="workflow:PIG"/>

```

```

<xs:element name="ssh" type="workflow:SSH"/>
<xs:element name="sub-workflow" type="workflow:SUB-WORKFLOW"/>
<xs:element name="fs" type="workflow:FS"/>
<xs:complexType name="ACTION">
  <xs:sequence>
    <xs:choice minOccurs="1" maxOccurs="1">
      <xs:element name="map-reduce" type="workflow:MAP-REDUCE" minOccurs="1" maxOccurs="1"/>
      <xs:element name="pig" type="workflow:PIG" minOccurs="1" maxOccurs="1"/>
      <xs:element name="ssh" type="workflow:SSH" minOccurs="1" maxOccurs="1"/>
      <xs:element name="sub-workflow" type="workflow:SUB-WORKFLOW" minOccurs="1" maxOccurs="1"/>
      <xs:element name="fs" type="workflow:FS" minOccurs="1" maxOccurs="1"/>
      <xs:element name="http" type="workflow:HTTP" minOccurs="1" maxOccurs="1"/>
      <xs:element name="email" type="workflow:EMAIL" minOccurs="1" maxOccurs="1"/>
      <xs:any namespace="##other" minOccurs="1" maxOccurs="1"/>
    </xs:choice>
    <xs:element name="ok" type="workflow:ACTION_TRANSITION" minOccurs="1" maxOccurs="1"/>
    <xs:element name="error" type="workflow:ACTION_TRANSITION" minOccurs="1" maxOccurs="1"/>
  </xs:sequence>
  <xs:attribute name="name" type="workflow:IDENTIFIER" use="required"/>
</xs:complexType>
<xs:complexType name="MAP-REDUCE">
  <xs:sequence>
    <xs:element name="job-tracker" type="xs:string" minOccurs="1" maxOccurs="1"/>
    <xs:element name="name-node" type="xs:string" minOccurs="1" maxOccurs="1"/>
    <xs:element name="prepare" type="workflow:PREPARE" minOccurs="0" maxOccurs="1"/>
    <xs:element name="streaming" type="workflow:STREAMING" minOccurs="0" maxOccurs="1"/>
    <xs:element name="job-xml" type="xs:string" minOccurs="0" maxOccurs="1"/>
    <xs:element name="configuration" type="workflow:CONFIGURATION" minOccurs="0" maxOccurs="1"/>
    <xs:element name="file" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="cache-file" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="cache-archive" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="PIG">
  <xs:sequence>
    <xs:element name="job-tracker" type="xs:string" minOccurs="1" maxOccurs="1"/>
    <xs:element name="name-node" type="xs:string" minOccurs="1" maxOccurs="1"/>
    <xs:element name="prepare" type="workflow:PREPARE" minOccurs="0" maxOccurs="1"/>
    <xs:element name="host" type="xs:string" minOccurs="1" maxOccurs="1"/>
    <xs:element name="job-xml" type="xs:string" minOccurs="0" maxOccurs="1"/>
    <xs:element name="configuration" type="workflow:CONFIGURATION" minOccurs="0" maxOccurs="1"/>
    <xs:element name="script" type="xs:string" minOccurs="1" maxOccurs="1"/>
    <xs:element name="param" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="file" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="cache-file" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="cache-archive" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="SSH">
  <xs:sequence>
    <xs:element name="host" type="xs:string" minOccurs="1" maxOccurs="1"/>
    <xs:element name="command" type="xs:string" minOccurs="1" maxOccurs="1"/>
    <xs:element name="args" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="capture-output" type="workflow:FLAG" minOccurs="0" maxOccurs="1"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="SUB-WORKFLOW">
  <xs:sequence>
    <xs:element name="oozie" type="xs:string" minOccurs="0" maxOccurs="1"/>
    <xs:element name="app-path" type="xs:string" minOccurs="1" maxOccurs="1"/>
    <xs:element name="configuration" type="workflow:CONFIGURATION" minOccurs="0" maxOccurs="1"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="FS">
  <xs:sequence>
    <xs:element name="delete" type="workflow:DELETE" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="mkdir" type="workflow:MKDIR" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="move" type="workflow:MOVE" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="HTTP">
  <xs:sequence>
    <xs:element name="http-url" type="xs:string" minOccurs="1" maxOccurs="1"/>
    <xs:element name="post" type="workflow:HTTP-POST" minOccurs="0" maxOccurs="1"/>
    <xs:choice minOccurs="1" maxOccurs="1">
      <xs:element name="wait" type="workflow:FLAG" minOccurs="1" maxOccurs="1"/>
      <xs:element name="continue" type="workflow:FLAG" minOccurs="1" maxOccurs="1"/>
    </xs:choice>
    <xs:element name="capture-output" type="workflow:FLAG"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="EMAIL">
  <xs:sequence>
    <xs:element name="to" type="xs:string" minOccurs="1" maxOccurs="1"/>
    <xs:element name="subject" type="xs:string" minOccurs="1" maxOccurs="1"/>
    <xs:element name="message" type="xs:string" minOccurs="0" maxOccurs="1"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="HTTP-POST">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="content-type" type="xs:string" use="required"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

```

```

</xs:simpleContent>
</xs:complexType>
<xs:complexType name="FLAG" />
<xs:complexType name="CONFIGURATION">
  <xs:sequence>
    <xs:element name="property" minOccurs="1" maxOccurs="unbounded">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="name" minOccurs="1" maxOccurs="1" type="xs:string"/>
          <xs:element name="value" minOccurs="1" maxOccurs="1" type="xs:string"/>
          <xs:element name="description" minOccurs="0" maxOccurs="1" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="STREAMING">
  <xs:sequence>
    <xs:element name="mapper" type="xs:string" minOccurs="0" maxOccurs="1"/>
    <xs:element name="reducer" type="xs:string" minOccurs="0" maxOccurs="1"/>
    <xs:element name="env" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="record-reader" type="xs:string" minOccurs="0" maxOccurs="1"/>
    <xs:element name="record-reader-mapping" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="PREPARE">
  <xs:sequence>
    <xs:element name="delete" type="workflow:DELETE" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="mkdir" type="workflow:MKDIR" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="DELETE">
  <xs:attribute name="path" type="xs:string" use="required"/>
</xs:complexType>
<xs:complexType name="MKDIR">
  <xs:attribute name="path" type="xs:string" use="required"/>
</xs:complexType>
<xs:complexType name="MOVE">
  <xs:attribute name="source" type="xs:string" use="required"/>
  <xs:attribute name="target" type="xs:string" use="required"/>
</xs:complexType>
</xs:schema>

```

Appendix B, Workflow Examples

Fork and Join Example

The following workflow definition example executes 4 Map-Reduce jobs in 3 steps, 1 job, 2 jobs in parallel and 1 job.

The output of the jobs in the previous step are use as input for the next jobs.

Required workflow job parameters:

- jobtracker : JobTracker HOST:PORT
- namenode : NameNode HOST:PORT
- input : input directory
- output : output directory

```

<workflow-app name='example-forkjoinwf' xmlns="uri:oozie:workflow:0.1">
  <start to='firstjob' />
  <action name="firstjob">
    <map-reduce>
      <job-tracker>${jobtracker}</job-tracker>
      <name-node>${namenode}</name-node>
      <configuration>
        <property>
          <name>mapred.mapper.class</name>
          <value>org.apache.hadoop.example.IdMapper</value>
        </property>
        <property>
          <name>mapred.reducer.class</name>
          <value>org.apache.hadoop.example.IdReducer</value>
        </property>
        <property>
          <name>mapred.map.tasks</name>
          <value>1</value>
        </property>
        <property>
          <name>mapred.input.dir</name>
          <value>${input}</value>
        </property>
        <property>
          <name>mapred.output.dir</name>
          <value>/usr/foo/${wf:id()}/templ</value>
        </property>
      </configuration>
    </map-reduce>
    <ok to="fork" />
    <error to="kill" />
  </action>

```

```

</action>
<fork name='fork'>
  <path start='secondjob' />
  <path start='thirdjob' />
</fork>
<action name="secondjob">
  <map-reduce>
    <job-tracker>${jobtracker}</job-tracker>
    <name-node>${namenode}</name-node>
    <configuration>
      <property>
        <name>mapred.mapper.class</name>
        <value>org.apache.hadoop.example.IdMapper</value>
      </property>
      <property>
        <name>mapred.reducer.class</name>
        <value>org.apache.hadoop.example.IdReducer</value>
      </property>
      <property>
        <name>mapred.map.tasks</name>
        <value>1</value>
      </property>
      <property>
        <name>mapred.input.dir</name>
        <value>/usr/foo/${wf:id()}/temp1</value>
      </property>
      <property>
        <name>mapred.output.dir</name>
        <value>/usr/foo/${wf:id()}/temp2</value>
      </property>
    </configuration>
  </map-reduce>
  <ok to="join" />
  <error to="kill" />
</action>
<action name="thirdjob">
  <map-reduce>
    <job-tracker>${jobtracker}</job-tracker>
    <name-node>${namenode}</name-node>
    <configuration>
      <property>
        <name>mapred.mapper.class</name>
        <value>org.apache.hadoop.example.IdMapper</value>
      </property>
      <property>
        <name>mapred.reducer.class</name>
        <value>org.apache.hadoop.example.IdReducer</value>
      </property>
      <property>
        <name>mapred.map.tasks</name>
        <value>1</value>
      </property>
      <property>
        <name>mapred.input.dir</name>
        <value>/usr/foo/${wf:id()}/temp1</value>
      </property>
      <property>
        <name>mapred.output.dir</name>
        <value>/usr/foo/${wf:id()}/temp3</value>
      </property>
    </configuration>
  </map-reduce>
  <ok to="join" />
  <error to="kill" />
</action>
<join name='join' to='finalejob' />
<action name="finaljob">
  <map-reduce>
    <job-tracker>${jobtracker}</job-tracker>
    <name-node>${namenode}</name-node>
    <configuration>
      <property>
        <name>mapred.mapper.class</name>
        <value>org.apache.hadoop.example.IdMapper</value>
      </property>
      <property>
        <name>mapred.reducer.class</name>
        <value>org.apache.hadoop.example.IdReducer</value>
      </property>
      <property>
        <name>mapred.map.tasks</name>
        <value>1</value>
      </property>
      <property>
        <name>mapred.input.dir</name>
        <value>/usr/foo/${wf:id()}/temp2,/usr/foo/${wf:id()}/temp3
        </value>
      </property>
      <property>
        <name>mapred.output.dir</name>
        <value>${output}</value>
      </property>
    </configuration>
  </map-reduce>
  <ok to="join" />
  <error to="kill" />
</action>

```

```
</map-reduce>
  <ok to="end" />
  <ok to="kill" />
</action>
<kill name="kill">
  <message>Map/Reduce failed, error message[${wf.errorMessage}]</message>
</kill>
<end name='end' />
</workflow-app>
```

[::Go back to Oozie Documentation Index::](#)