

## Chapter 11

# Object State Manager for the Web

This chapter describes the object state manager for the Web (OSMW); it contains the following sections:

Overview on page 137

Target URI on page 138

Observer URI on page 138

Process States on page 138

CorrelationData Tag on page 139

ContextData and ResultData Tags on page 139

Results and the Exception Codes on page 140

Operations Specifics on page 141

Architecture on page 142

Executing a Transaction Using the OSMW – Example on page 143

## Overview

---

The OSMW component provides an interface to the OSM through standard XML messaging over HTTP. This standard is endorsed by the Workflow Management Coalition (WfMC). It is one of the mappings of interface 4 [WFMC-REFMODEL] using XML. This chapter describes this mapping and the component. The messaging specification can be found in [WFMC-WFXML]. The only details described here are the ones that are left open by the specification, namely:

The content of the ContextData and ResultData tags

Which of the optional process states are used

The request and response URI formats

Use of the CorrelationData tag

The mapping between the exception error codes and the result codes

Some specifics about the operations

## Target URI

---

As specified in the standards, the Uniform Resource Indicator (URI) contained in the Key tag must be the same as the Uniform Resource Locator (URL) where the message is being sent. The format of such URI is:

```
http://<host>/<relative path to application>? procid=<process id>&
target=<targetDN>
```

where:

< host> is the hostname of the application host.

< relative path to application> is the file part of the URL pointing to the SSM-INT application.

< process id> is a unique string identifying the process instance.

< targetDN> is the DN of the target of the operation.

The process ID is specified at creation time in response to a CreateProcessInstance operation. This is the format of the ProcessInstanceKey. The ProcessDefinitionKey is the same, but without the parameters part.

## Observer URI

---

This URI is specified by the client to which the OSMW will send the ProcessInstanceStateChanged message. It is not bound by the format specified for the target URI, but it must be an accessible URL; that is, it must be possible to post a request to it.

## Process States

---

An OSM process can be in the following states specified by the standard:

open.notrunning – initial state of all transactions. This is the state immediately after the Prepare message was sent and acknowledged but before the Execute message. To move to the open.running state, an explicit ChangeProcessInstanceState message must be sent.

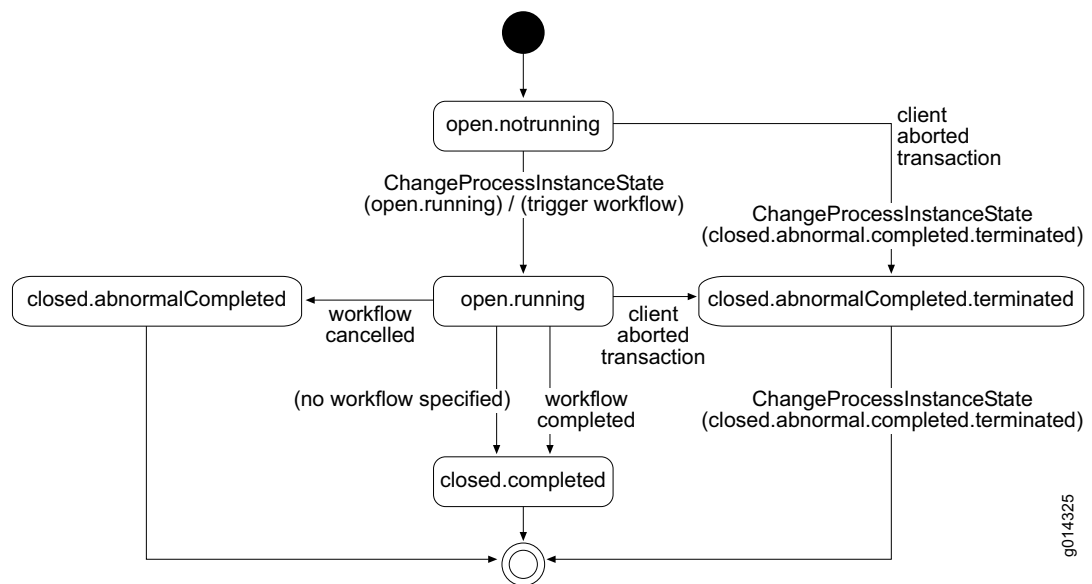
open.running – state of a transaction being executed (that is, after the Execute message was sent and acknowledged), which may mean that there is an active workflow associated with it. If there is no associated workflow, it moves immediately to the closed.completed state. Otherwise, it will move to one of the closed states on receipt of the report about the workflow execution.

closed.completed – the transaction was completed, and the target object changed its state accordingly. If a workflow was associated with the operation, the workflow was completed. There is no connection between the outcome of the workflow and this state, only with the state of the target object.

closed.abnormalCompleted – flags that the associated workflow has cancelled due to an internal error and that the transaction was successfully rolled back.

closed.abnormalCompleted.terminated – flags that the transaction was aborted as requested by the client and that it rolled back successfully.

Figure 32: OSM transaction states



In the closed states, there is no active associated workflow, and the state of the target was updated accordingly. The state diagram in Figure 32 summarizes the description.

## CorrelationData Tag

This tag is ignored.

## ContextData and ResultData Tags

As defined by the standard, these tags are to be agreed on the interoperability contract between the parts. They are declared of type ANY. In this specification, ContextData is defined by the following document type definition (DTD):

```
< !ELEMENT ContextData (TransactionID, Action, TargetObject)>
```

```
< !ELEMENT TransactionID (#PCDATA)>
```

```
< !ELEMENT Action (#PCDATA)>
```

```
< !ELEMENT TargetObject (#PCDATA)>
```

TransactionID contains the ID of the transaction. Action contains the name of the action to be executed upon the target. TargetObject contains the distinguished name (DN) of the object that is the target of the operation.

Similarly, ResultData is defined by the following DTD:

```
< !ELEMENT ResultData (SUCCEEDED |FAILED)>
```

```
< !ELEMENT SUCCEEDED EMPTY>
```

```
< !ELEMENT FAILED EMPTY>
```

The presence of the Succeeded tag means that the transaction was completed successfully. The presence of the Failed tag means it was completed with a failure. The result data is sent only when the state changes to closed.completed.

## Results and the Exception Codes

---

Exceptions are returned when an operation fails to execute. The standard specifies several exceptions that are thrown in various cases. They are supported for all operations. The standard leaves open a point to insert application-specific codes as subcodes. The only exception that makes sense, for this application, to be extended is the WF\_INVALID\_CONTEXT\_DATA, main code 201. Table 46 shows Result values mapped to subcodes.

**Table 46: Mapping from result values to subcodes of the WF\_INVALID\_CONTEXT\_DATA\_EXCEPTION**

ExecStatus	Subcode
FORBIDDEN	004
LOCKED	005
INVALID_DN	006
INVALID_DATA	007
BAD_ARGUMENT	008
BAD_TARGET	009

The Locked error is reported as a temporary error. The remaining ones are reported as fatal errors. The Transport Failure error is mapped to the WF Operation Failed error, main code 400, no subcode necessary, which is, by nature, a temporary error.

## Operations Specifics

---

This section describes the specifics about the operations defined by the standard.

Some remarks are common to all operations because they concern the message header:

The `ResponseRequired` tag is ignored. To every message there will always be a response because all the OSM operations return values that should be analyzed by the client.

The `ResponseLang` tag is ignored. The only supported language is English.

### ***CreateProcess Instance***

This operation is used to invoke service and subscriber management (SSM) operations. The process to be executed and the associated arguments are passed via the `ContextData` tag, as specified in *ContextData and ResultData Tags on page 139*. Some tags are not supported:

Name: ignored

Subject: ignored

Description: ignored

If the client wishes to provide such tags, there is no problem, but they are currently not used.

A response is provided to return a value for the OSM operation. If some of the errors specified in *Results and the Exception Codes on page 140* occur, an exception is returned, otherwise a regular response message.

The `ObserverKey` tag is marked as optional in the standard, but it should always be present in the request message. It is always used to report the completion of the process, even when a transaction does not trigger a workflow, thus returning immediately. The standard also says that the interoperability model of a nested subprocess is used. At this moment, every transaction exists by itself; that is, once it is finished, it cannot be reversed. Thus, these operations cannot be used as subprocesses.

### ***GetProcess InstanceData***

The standard specifies a set of properties of a process instance that can be retrieved, but not all of them are supported. A request message can specify any of the allowed values for the `ResultDataSet` tag; the unsupported ones will be returned as empty tags.

Some of the tags in the response message have special meanings:

`ValidStates` – returns the states specified in 5.3 regardless of the current state of the process instance.

`ResultData` – always returns an empty tag.

Priority – always returns the default value of 3. Priorities are not supported.

LastModified – returns the creation date because this information is no longer available after the process is closed.

### ***ChangeProcess InstanceState***

This message is used in two cases to provoke the:

Abortion of a process – The target state in the request message is closed.abnormalCompleted.terminated.

Execution of the transaction – The target state is open.running.

The response message will then reflect that the state was changed to the specified target. Any other target states implicate in an exception of type WF\_INVALID\_STATE\_TRANSITION being thrown.

### ***Process InstanceState Changed***

This operation is used to report to the client (as specified by the ObserverKey tag in creation time) that the process has been completed, normally or not. The states reported are the closed ones, as specified in *Process States* on page 138. An empty response should be sent to acknowledge that the message was received and understood. The ResponseRequired tag always contains Yes as value. The return value of the OSM transaction is embedded in the ResultData tag in the case of a closed.completed state change.

## Architecture

---

The OSMW is implemented as a Java-based Web application (servlet). A Web server capable of running servlets is the host of the application, and thus is the part responsible for the connection management and other deployment details, such as server distribution for scalability and servlet aliasing.

The OSMW is a different packaging of the OSM that uses HTTP instead of plain sockets to communicate. Basically, it is a layer on top of the OSM that replaces the socket-based architecture.

### ***Servlet 2.2 API Compatible***

The OSMW component can be deployed to any Web server that is capable of running servlets compatible with the Servlet API version 2.2.

### ***Standard HTTP Authentication***

The OSMW does not implement any special authentication mechanism, relying on HTTP for that matter. Thus, it is the responsibility of the Web server to provide the security mechanisms to the application.

## Executing a Transaction Using the OSMW – Example

---

In this example, the server hosting the OSM for the Web is called shark. The Apache Tomcat application host is serving at port 8080. The servlet that is processing the requests is called *osmint*. The URL to which the message should be posted is <http://shark:8080/workflow/osmint> to create a new transaction. This is the *CreateProcessInstance* (request) message:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE WfMessage SYSTEM "http://shark:8080/workflow/Wf-XML-1_0.dtd">
<WfMessage Version="1.0">
  <WfTransport>
    <CorrelationData>OSMINTTEST_-1830803183 </CorrelationData>
  </WfTransport>
  <WfMessageHeader>
    <Request ResponseRequired="Yes"/>
    <Key>http://shark:8080/workflow/osmint</Key>
  </WfMessageHeader>
  <WfMessageBody>
    <CreateProcessInstance.Request StartImmediately="true">
      <ObserverKey>http://10.227.1.113:8000/osmintclient</ObserverKey>
      <ContextData>
        <TransactionID>REQ000123</TransactionID>
        <Action>subscribe_initial</Action>
      </ContextData>
    </CreateProcessInstance.Request>
  </WfMessageBody>
  <TargetObject>uniqueid=paulo,ou=ottawa,retailername=default,o=users,o=umc</TargetObject>
</WfMessage>
```

The host replies with a *CreateProcessInstance* (response) message, stating it was able to start a transaction, and returns the *ProcessInstanceKey* as <http://shark:8080/workflow/osmint?procid=REQ000123&target=uniqueid%3Dpaulo%2Co%3Dottawa%2Cretailername%3Ddefault%2Co%3Dusers%2Co%3Dumc>.



**NOTE:** The URL encoding is needed due to XML constraints. In the process instance key are encoded the DN of the target object and the action to be performed on it. Notice also that the correlation data is the same as the request message.

```
<?xml version="1.0" encoding="UTF-8"?>
<WfMessage Version="1.0">
  <WfTransport>
    <CorrelationData>OSMINTTEST_-1830803183</CorrelationData>
  </WfTransport>
  <WfMessageHeader>
    <Response/>
    <Key>http://shark:8080/workflow/osmint</Key>
  </WfMessageHeader>
  <WfMessageBody>
```

```

    <CreateProcessInstance.Response>
    <ProcessInstanceKey>http://shark:8080/workflow/osmint?procid=REQ000123&
    amp;target=uniqueid%3Dpaulo%2Cou%3Dottawa%2Cretailname%3Ddefault%2C
    o%3Dusers%2Co%3Dumc</ProcessInstanceKey>
    </CreateProcessInstance.Response>
  </WfMessageBody>
</WfMessage>

```

This completes the transaction preparation cycle. At this point the transaction is created but not yet executing. The target object is backed up, and any changes made to it can be reverted if the transaction aborts. This is point where the object can be safely updated before a workflow is triggered, which is done by the next message. In this example, the target object is left intact.

The message that executes the transaction is the ChangeProcessInstanceState. It changes the state of the process from open.notrunning to open.running.

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE WfMessage SYSTEM "http://shark:8080/workflow/Wf-XML-1_0.dtd">
<WfMessage Version="1.0">
  <WfTransport>
    <CorrelationData>OSMINTTEST_2090215924</CorrelationData>
  </WfTransport>
  <WfMessageHeader>
    <Request ResponseRequired="Yes"/>
    <Key>http://shark:8080/workflow/osmint?procid=REQ000123&target=uniq
    ueid%3Dpaulo%2Cou%3Dottawa%2Cretailname%3Ddefault%2Co%3Dusers%2Co
    %3Dumc</Key>
  </WfMessageHeader>
  <WfMessageBody>
    <ChangeProcessInstanceState.Request>
      <State>
        <open.running/>
      </State>
    </ChangeProcessInstanceState.Request>
  </WfMessageBody>
</WfMessage>

```

The response to this message acknowledges the state change, which means that the transaction is executing and possibly also a workflow.

```

<?xml version="1.0" encoding="UTF-8"?>
<WfMessage Version="1.0">
  <WfTransport>
    <CorrelationData>OSMINTTEST_2090215924</CorrelationData>
  </WfTransport>
  <WfMessageHeader>
    <Response/>
    <Key>http://shark:8080/workflow/osmint?procid=REQ000123&target=uniq
    ueid%3Dpaulo%2Cou%3Dottawa%2Cretailname%3Ddefault%2Co%3Dusers%2Co
    %3Dumc</Key>
  </WfMessageHeader>
  <WfMessageBody>
    <ChangeProcessInstanceState.Response>
      <State>
        <open.running/>
      </State>
    </ChangeProcessInstanceState.Response>
  </WfMessageBody>
</WfMessage>

```

```

    </ChangeProcessInstanceState.Response>
  </WfMessageBody>
</WfMessage>

```

The client is now waiting for the transaction completion, which is notified by a `ProcessInstanceStateChanged` message sent to the Observer URL as specified in the `CreateProcessInstance` message by the `ObserverKey` tag. It includes the `ProcessInstanceKey`, when the process changed its state (`LastModified` tag), to which state and an optional `ResultData`, which in this case indicates the transaction was successful.

```

<?xml version="1.0" encoding="UTF-8"?>
<WfMessage Version="1.0">
  <WfTransport>
    <CorrelationData>REPORT_REQ000123</CorrelationData>
  </WfTransport>
  <WfMessageHeader>
    <Request ResponseRequired="Yes"/>
    <Key>http://10.227.1.113:8000/osmintclient</Key>
  </WfMessageHeader>
  <WfMessageBody>
    <ProcessInstanceStateChanged.Request>

      <ProcessInstanceKey>http://shark:8080/workflow/osmint?procid=REQ0001
      23&target=uniqueid%3Dpaulo%2Cou%3Ddottawa%2Cretailname%3Ddefault
      %2Co%3Duser
      s%2Co%3Dumc</ProcessInstanceKey>
      <State>
        <closed.completed/>
      </State>
      <ResultData>
        <SUCCEEDED/>
      </ResultData>
      <LastModified>2002-06-16 11:45:34EDT</LastModified>
    </ProcessInstanceStateChanged.Request>
  </WfMessageBody>
</WfMessage>

```

Because the above message requires a response as specified by the `ResponseRequired` attribute of the `Request` tag, the client sends this message:

```

<?xml version="1.0" encoding="UTF-8"?>
<WfMessage Version="1.0">
  <WfMessageHeader>
    <Response/>
    <Key>http:// 10.227.1.113:8000/osmintclient</Key>
  </WfMessageHeader>
  <WfMessageBody>
    <ProcessInstanceStateChanged.Response/>
  </WfMessageBody>
</WfMessage>

```

This completes the transaction execution cycle. The target object is properly updated in the directory to reflect its new state, and the backup copy and the lock in the object are removed, leaving it ready for another transaction.

