Workflow Standard - Interoperability

XML-HTTP binding

Document Status: Submission 2.0

8 February, 2000

Authors:

Nobuyuki Kanaya, Fujitsu Laboratories LTD. (kanaya@flab.fujitsu.co.jp)

Shin'ichi Iwasaki, NEC Corporation (iwasaki@ccs.mt.nec.co.jp)

Akihiko Suzuki, NEC Corporation (ak-suzuk@ccs.mt.nec.co.jp)

Tsuyoshi Ebata, Fujitsu Limited (ebata@lp.nm.fujitsu.co.jp)

Hirotaka Hara, Fujitsu Laboratories LTD. (hara@flab.fujitsu.co.jp)

1. Introduction

1.1 Purpose

This document represents a workflow protocol that aims for interoperable, reliable, and practical interactions between services using HTTP protocol. This protocol is based on Wf-XML and extended for the sake of enterprise EDI applications.

Wf-XML defines more precise interfaces and data model than WfMC interface4, using URI to identify a resource that enables easy inquiries to a resource. Wf-XML provides the interfaces to search for process instances of each process definition in addition to the interface to create and terminate a process instance. We developed a practical enterprise EDI system using Wf-XML (to be exact, we used SWAP, origin of Wf-XML). As a result, Wf-XML is proved to be useful for developing EDI systems. However, we need to add some tags and interfaces to Wf-XML because Wf-XML falls short of some essential functions for EDI systems, that is, asynchronous messaging and multiple messaging. Enterprise interoperability will not be promoted if each system needs to add or modify a protocol respectively.

Another purpose of the document is to add some useful specifications of SWAP into the future Wf-XML. For example the activity observer interface provides the process tracking facility among several servers that is essential to EDI systems.

We propose this document because a practical workflow protocol that provides asynchronous messaging and multiple messaging using HTTP is indispensable in order to make workflow systems widespread into EDI applications.

1.2 Approach

We propose some interfaces and their protocols in order to solve some open issues of Wf-XML.

- Asynchronous messaging which ensures reliable transfers.

The result of a method request of Wf-XML is returned synchronously as a response page of HTTP. This quick response is a feature of Wf-XML compared with asynchronous transfers of the SMTP/POP implementations. However, asynchronous transfers are still necessary in EDI systems. This document defines an asynchronous messaging in which only an acknowledgement is returned in the response page of HTTP and the result of the method performed in a server is transferred in another HTTP request. A server can notify a client of errors occurred in performing the method and a client can inquiry the status of the method being performed. The synchronous interfaces can also be used together with the asynchronous interfaces.

- Multiple messaging

In EDI systems, it is often necessary to send a batch of requests in one message. A standard for the multiple messaging is required though Wf-XML does not define it. It is not sufficient to define a data format in which multiple Wf-XML requests are in one HTTP request because each EDI system should define and develop a different protocol or a different control mechanism when some errors occur in one of the requests. This document defines a multiple messaging mechanism in which a client can monitor and control a status of each request and a server can notify a client of an error of each request, in addition that a client can transfer multiple requests in one HTTP request.

- Workflow data format and DTD

This document provides two ways to specify workflow data in the ContextData element and the ResultData element. The one is used to specify external DTD for workflow data so that users can validate the data with it, and the other defines data type definitions for better interoperability for simple data exchange.

We also propose several specifications and facilities that are useful for implementing EDI systems.

- Activity Observer Interface

A process instance can contain activities. The WfMC client interface (Interface-2) provides an interface to access the activities. The interface is not only useful for workflow clients but also useful for the interoperability among workflow servers. SWAP provides the activity observer interface for tracking the status of subprocesses which is essential for EDI systems. The current Wf-XML does not, however, adopt it. This document includes the activity observer interface as a candidate of the next Wf-XML specifications.

- Addition of tracking and logging facilities

In EDI systems, it is important to get the status of a request quickly. Though Wf-XML provides basic reference, tracking, and logging facilities, it is difficult to manage a whole sequence of processes and get tracking information efficiently. For example, you need to query a sequence of processes in turn when you want to know which sub processes of the process you created are active. Also parent process may be completed before its child process is completed if it is a chained process. In this case, a simple tracking does not work well. In this document, two tags are added which provides tracking and logging facilities for a sequence of processes. The “GlobalKey” tag is inherited from a parent process to child processes in order to identify the sequence. The logging tag shows a resource for collecting logs.

- Validation check with DTD

This document defines a XML data format with DTD that enables the data validation check while allowing vendors extending the DTD.

1.3 Server Interaction Models and Messages

This specification defines an interaction model for exchanging Wf-XML request pages and result pages asynchronously or synchronously, such as “CreateProcessInstance”, “ProcessInstanceCompleted” and “SetData”. The model ensures reliable batch request, protocols with the formats of messages among workflow servers.

It is not necessary to define an interaction model for synchronous requests in HTTP binding, but it is necessary to define a reliable interaction model for EDI applications. The interaction model of the IF4 abstract specification that is based on reliable transfer protocols or infrastructures is too simple and unreliable to apply it to workflow servers on the Internet. For EDI applications, an interaction model is very important, because it is necessary to define a way to send requests and to retry to send them. The interaction model for batch requests in which thousands of Wf-XML requests are handled as one batch request must also include a way to refer the result of each request, to cancel the batch request, or to stop the batch request.

Our workflow server interaction model provides a reliable way to exchange Wf-XML request pages and Wf-XML result pages as a batch request. It defines a way to retry to send the pages, to refer the result of finished requests, and to control each request synchronously or asynchronously in a batch request. The interaction model is independent of the Wf-XML resource model. The model includes three ideas: “mission”, “operation”, and “phase”. The “mission” starts when a server makes a Wf-XML request, and it ends when the result of the request is returned. The “register operation” is one of the operations. It is used to send Wf-XML request pages as parameters and response pages as its result. The register operation starts and ends a mission. We also define two other operation types. There are three types of phases: “Register”, “Refer”, and “Control”. And an operation consists 4 phases: the “Register” phase, the “Acknowledge” phase (inbound), the “Response” phase, the “Acknowledge” phase (outbound). The following table is the summary of the operations, phases and directions in the interaction model.

This specification also defines a messaging protocol for workflow servers and its format based on the interaction model. In this protocol, servers exchange nine types (three types of phases by three types of operations) of messages to interact with other servers. A message has the “WfMessageHeader” part that contains information used for server interaction, and the ”WfMessageBody” part that contains Wf-XML request pages or Wf-XML response pages. For example, the simplest way for a server to make a Wf-XML request is sending a message. The message contains the “WfMessageHeader” that indicates that message is in the register operation and the request phase and the WfMessageBody that contains a Wf-XML request page.

There are a lot of ways to implement asynchronous request invocation, such as using an asynchronous transfer protocol (ex. SMTP) or using a special “Process Definition” that accepts Wf-XML request. But we believe that providing a definition of an interaction model for workflow servers is most important. The model must be independent both of workflow server model and of Wf-XML resource model. The protocol must be based on the model. Our protocol is mainly used for the HTTP, but the interaction model is also useful for other transfer protocols such as the SMTP or MOM. Thus we think the model and the protocol must become a standard of workflow server interoperability in the WfMC.

Table: Summary of operations, phases, directions.

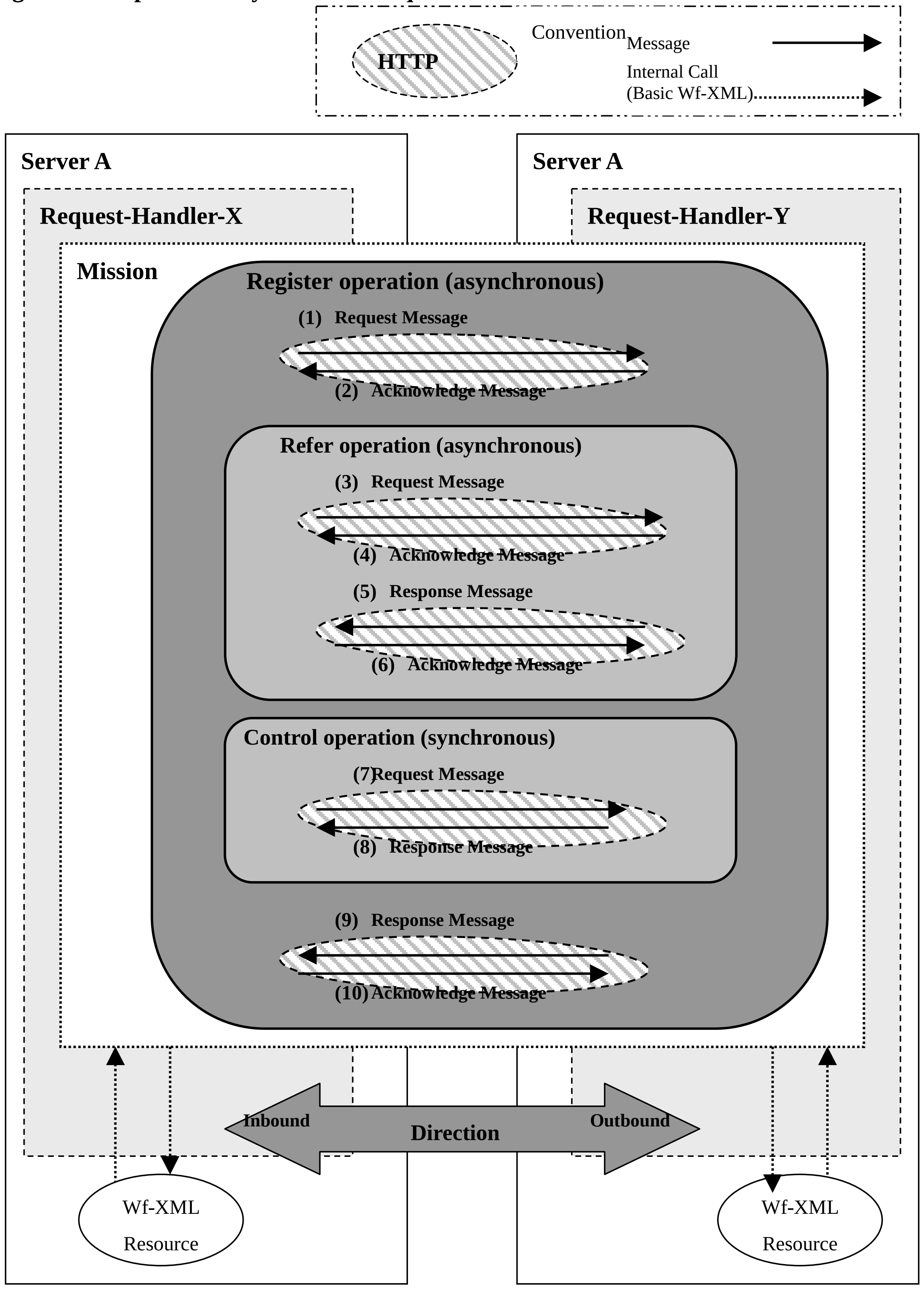
operation	phase	direction	Comment
Register	**Request**	outbound	**To send Wf-XML request pages.**
	Acknowledge	inbound	Used for an asynchronous invocation.
	Response	inbound	**To send Wf-XML response pages.**
	Acknowledge	outbound	Used for an asynchronous invocation.
Refer	Request	outbound	To ask to send the status of Wf-XML request invocation Used for an asynchronous invocation.
	Acknowledge	inbound	Used in an asynchronous invocation.
	Response	inbound	To send status of Wf-XML request invocation. Used for an asynchronous invocation.
	Acknowledge	outbound	Used in an asynchronous invocation.
Control	Request	outbound	To ask to cancel, stop, and resume Wf-XML invocation. Used for an asynchronous invocation.
	Acknowledge	inbound	Used in an asynchronous invocation.
	Response	inbound	To report the result of a control operation. Used for an asynchronous invocation.
	Acknowledge	outbound	Used for an asynchronous invocation.

1.3.1 Workflow Server Interaction Model

A Basic Interaction Model

The following figure is an example of a mission, operations and messages in an asynchronous Wf-XML request. The example describes how Server-A makes a Wf-XML request to Server-B, refers the status of the Wf-XML request execution, and controls Wf-the XML request execution. And it also describes how Server-B returns the result of the Wf-XML execution with the messages.

Figure: Example of an asynchronous request



The “Mission” starts when a server receives a Wf-XML request such as “ProcessDefinition::CreateProcessInstance” or “ActivityObserver::Complete”, and it ends when the receiver sends a Wf-XML response of the invocation. In the figure, the mission starts when the message No. 1 is received by “Request-Hander-Y” and it ends when the message No. 10 is sent. The “request handler” manages missions and interacts to the other request handers as an agent of Wf-XML resources in asynchronous requests. Messages are exchanged among request handlers in the asynchronous Wf-XML request. But the “request handler” is not necessary for synchronous requests because messages can be exchanged among Wf-XML resources in the synchronous Wf-XML request.

There are two direction types: “inbound” and “outbound”. When Server-A makes a Wf-XML request (not operation) to “Server-B”, the direction of “outbound” indicates that the message is sent from Server-A to Server-B, while the inbound indicates that the message is sent from Server-B to Server-A. In a synchronous Wf-XML request on the HTTP protocol, an HTTP request is in the “outbound” direction, and the HTTP result is in the “inbound” direction.

A mission must have at least one operation. There are three operations in the example. Operations are used to make Wf-XML requests, to return the results, to refer the progress of the Wf-XML requests, or to control execution of the Wf-XML requests in a mission. There are three types of operations: “Register”, “Refer”, and “Control”. The register operation is a basic operation, and it is used to send a Wf-XML request page and a Wf-XML response page. In the figure, the message No. 1 includes Wf-XML request pages, and the message No. 9 includes response pages. The mission starts when a register operation is invoked and finishes when the register operation is over, so the mission must have exactly one register operation. The operation may be invoked synchronously or asynchronously. The synchronous Wf-XML request includes only one synchronous register operation.

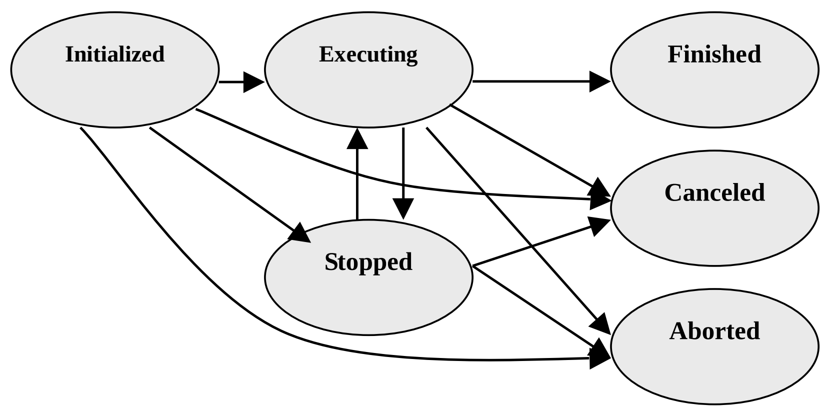
The “Refer” operation and the “Control” operation may be used only with asynchronous Wf-XML requests. These operations are optional. The “Refer” operation is used to ask the progress and result of a Wf-XML request that is finished. The “Refer” operation is especially useful when a server invokes a mission that includes thousands of various Wf-XML requests. The “Control” operation is used to cancel, stop, or resume a Wf-XML request in execution. Asynchronous “Control” operations and asynchronous “Refer” operations may overlap each other. Controlling a request doesn’t mean general rollback mechanism for Wf-XML requests. But a “Control” operation may be able to cancel the request before it is executed on a server. If a Wf-XML request is finished and the response page is returned to a requestor, the request can not be canceled by any “Control” operations. In some implementations, a “Control” operation can cancel only a Wf-XML request in a waiting queue and can not cancel a request in execution.

The synchronous operations consist of two phases: the “Request” phase and the “Response” phase like the message the No.7 and the message No.8 in the figure. The asynchronous operation consists of four phases: the “Request” phase like the message No.3, the inbound “Acknowledge phase like the message No.4, the “Response” phase like the message No. 5, and outbound “Acknowledge” phases like the message No. 7 in the figure. A server sends one message in one phase. We call a message in the request phase “request message”, and a message that is used in a request phase by a register operation “Register.Request” message, and so on. In the request phase a server starts an operation, and a request message is used to send a parameter of the operation in the phase. In the response phase a server returns the result of the operation, and a “Response” message is used to send the result in the response phase. In the acknowledge phase a server confirms whether it receives a “Request” or “Response” message in an asynchronous operation, and an “Acknowledge” message is used to return a receipt of a message in the phase.

The mission has one of the six states: “Initialized”, “Executing”, “Stopped”, “Canceled”, “Finished” and “Aborted”. A control operation can change the state of a mission.

The following figure is the state transition diagram of the mission.

Figure: State transition diagram of a mission



The details of the states are as follows

- **Initialized**

  “Initialized” means that a “Register.Request” message has been received, a mission has been created, the mission has started normally, and it is waiting to be executed. No Wf-XML requests in the mission are executed.

- **Executing**

  Wf-XML requests in this mission can be executed.

- **Stopped**

  Wf-XML requests in this mission can not be executed.

- **Finished**

  When all Wf-XML requests in this mission are finished, then a Register.Response message is returned.

- **Canceled**

  This state indicates that the mission is canceled. Some Wf-XML requests in this mission may not be finished.

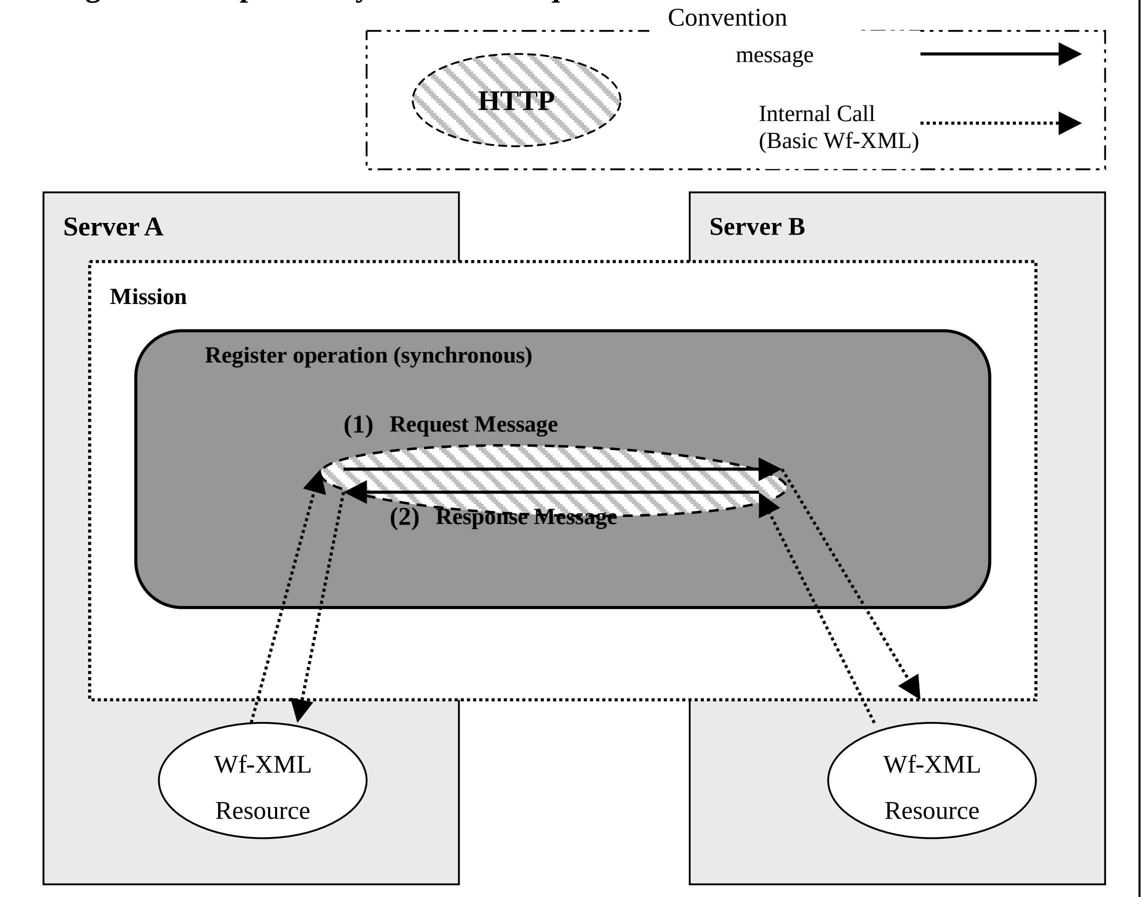
- **Aborted**

  The mission is aborted abnormally because of some internal errors. Some Wf-XML requests in this mission may not be finished.

Synchronous Interaction Model

The figure below is an example of a mission, operations and messages in a synchronous Wf-XML request. The example describes how Server-A makes a Wf-XML request to Server-B via an HTTP request.

Figure: Example of a synchronous request



8 February, 2000

In a synchronous Wf-XML request with our extension, the message format is similar to a Wf-XML request page and response page. A request message is sent to a Wf-XML resource and a response message is returned via a single HTTP connection. The request handlers are not necessary in the synchronous Wf-XML request. The mission has only one synchronous register operation and two messages. Each mission may have the MissionID, the MessageID or the RequestID to ensure reliable Wf-XML request invocation.

Interaction model for Reliability

Our interaction model has three types of identifiers assigned by a sender to prevent duplicated execution. They offer a safety method for retrying to send a Wf-XML request. Using these identifiers a receiver can detect duplicated execution.

- **RequestID**
  to prevent duplicated execution of Wf-XML requests
- **MissionID**
  to prevent duplicated execution of batch requests.
- **MessageID**
  to prevent duplicated execution of operations.

The following is an example for usage of the RequestID. After a server send a Wf-XML request, if the server doesn’t receive the response due to communication errors and so on, it is difficult to know whether the request has been completed. In such case, the server can send the request again with same RequestID. If the previous request has not reached the server and/or it has not started, then the request is accepted and the response of the request will be returned. If the previous request has reached the server and it has been completed, then the receiver will find the same RequestID, and the result of previous request will be returned.

1.3.2 Message Format Summary

The following XML is an example of a Register.Request message.

```
<?xml version="1.0" ?>
 <WfMessage version="1.0">
   <WfMessageHeader>
     <MessageID>i2o:msg:yyyymmdd:hhmmss007</MessageID>
     <Title>iwf-sample</Title>
     <Mission>
       <From>http://inbound/swap/reception</From>
       <To>http://outbound/iwf/RequestHandler</To>
       <MissionID>i2o:msn:yyyymmdd:hhmmss004</MissionID>
       <Register.Request>
         <ConnectionType>
           <Asynchronous/>
         </ConnectionType>
```

```
        <Parameters>parameters depend on each vender</Parameters>
      </Register.Request>
    </Mission>
  </WfMessageHeader>
  <WfMessageBody>
    <ProcessDefinition.GetData.Request Index="1">
      <Target>http://outbound/iwf?pd.001</Target>
      <!--
        Other tags are omitted. See "Wf-XML Message format" section.
      -->
    </ProcessDefinition.GetData.Request>

    <ProcessInstance.SetData.Request Index="2">
      <Target>http://outbound/iwf?001.10</Target>
      <!-- Other tags are omitted. -->
    </ProcessInstance.SetData.Request>

    <ActivityObserver.ProcessInstanceCompleted.Request Index="3">
      <Target>http://inbound/Wf-XML/ao?pid3.aid47</Target>
      <!--   Other tags are omitted. -->
    </ActivityObserver.ProcessInstanceCompleted.Request>
    <!--   *** Other Wf-XML message are omitted. ***    -->
  </WfMessageBody>
</WfMessage>
```

The top tag of the message, “WfMessage” tag, may contain of two elements: the “WfMessageHeader” and the “WfMessageBody”.

The “WfMessageHeader” may have the following tags.

- **MessageID**

  The “MessageID” is used to identify each message to prevent message duplication.
- **Title**

  The “Title” contains the description of this message.
- **Mission**

  The “Mission” contains information of this request.
- **From**

  The “From” specifies the request handler of the sender of this message. In an asynchronous Wf-XML request, the response is sent to this URI.
- **To**

  The “To” specifies the request handler of the receiver of the message.
- **MissionID**

  The “MissionID” is used to identify each mission. When a server invokes a new mission, it generates a new MissionID.
- **ConnectionType**

  The “ConnectionType “ indicates whether operation is synchronous or asynchronous.
- **Parameters**

The “Parameters” appears only in request messages. It contains parameters that depend for venders or implements.

- **Progress**

  The “Progress” appears only in inbound messages as a result of operations. It shows a status of the mission. It contains one of the states: ”Initialized”, ”Executing”, ”Stopped”, ”Canceled” and ”Aborted”.

The WfMessageHeader may also include other tags that vary depending on a phase and an operation. We call such tags "operation-phase dependent tags". The top tag of the operation-phase dependent tags in a message is expressed as a concatenation of the name of an operation and the name of a phase. For example, the "<Register.Request>" tag indicates that this message is used in the "Register" operation in the "Request" phase. The other operation-phase dependent tags that are contained by the top operation-phase dependent tag are different from one operation in one phase to another. In other words, the same top operation-phase dependent tags contain the same subordinate operation-phase dependent tags.

The WfMessageBody contains at least one Wf-XML request page or Wf-XML result page. The “Acknowledge” message must not include “WfMessageBody”. The “Request” messages and the “Response” message may include the WfMessageBody.

Wf-XML request page tags and Wf-XML result page tags contain the “Index” attribute and the “Target” tag.

- **Index**

  The “Index” is used to identify a Wf-XML request in the batch request. It must be unique in the mission. The response page of the request has the same index as the request page.

- **Target**

  Each Wf-XML request page must include the ”Target” tag. It contains the URI of a Wf-XML resource on which the Wf-XML request executes.

1.3.3 Use cases and messaging examples

Use case in a batch request

In this section, we will show usage of the messages in an asynchronous batch Wf-XML request. In the batch request, messages are sent between two request handlers. A Request handler sends and receives Wf-XML requests and Wf-XML responses. The following is an example of interaction in the batch request that includes Wf-XML requests from Request-Handler-A in server A to Request-Handler-B in server B.

(1) Making a batch of Wf-XML requests

Request-Handler-A that gathers Wf-XML requests. And it makes a batch of them and numbers each request as an “Index”.

(2) Generating MissionID

Request-Handler-A generates a MissionID.

(3) Sending a batch request

The Request-Handler-A send a “register operation, request “ message to Request-Handler-B with a “MissionID” tag that is generated in STEP (1), a “From” tag that contains the URI of the request handler, and an empty “Asynchronous” tag that indicates an asynchronous operation via an HTTP request.

(4) Replying Acknowledgement

Request-Handler-B returns “acknowledge message” with MissionID via the HTTP result.

(5) Executing each Wf-XML requests in the batch

Request-Handler-B executes each Wf-XML requests in batch.

(6) Referring the progress of the batch request

When Request-Handler-A wants to know the progress and results of all finished Wf-XML requests, the response handler sends a “refer operation, request” with the MissionID passed in STEP (3) and <Synchronous/> that specifies synchronous operation via an HTTP request.

(7) Reporting progress

Request-Handler-B returns an “refer operation, response message” with progress and results reported by the HTTP result.

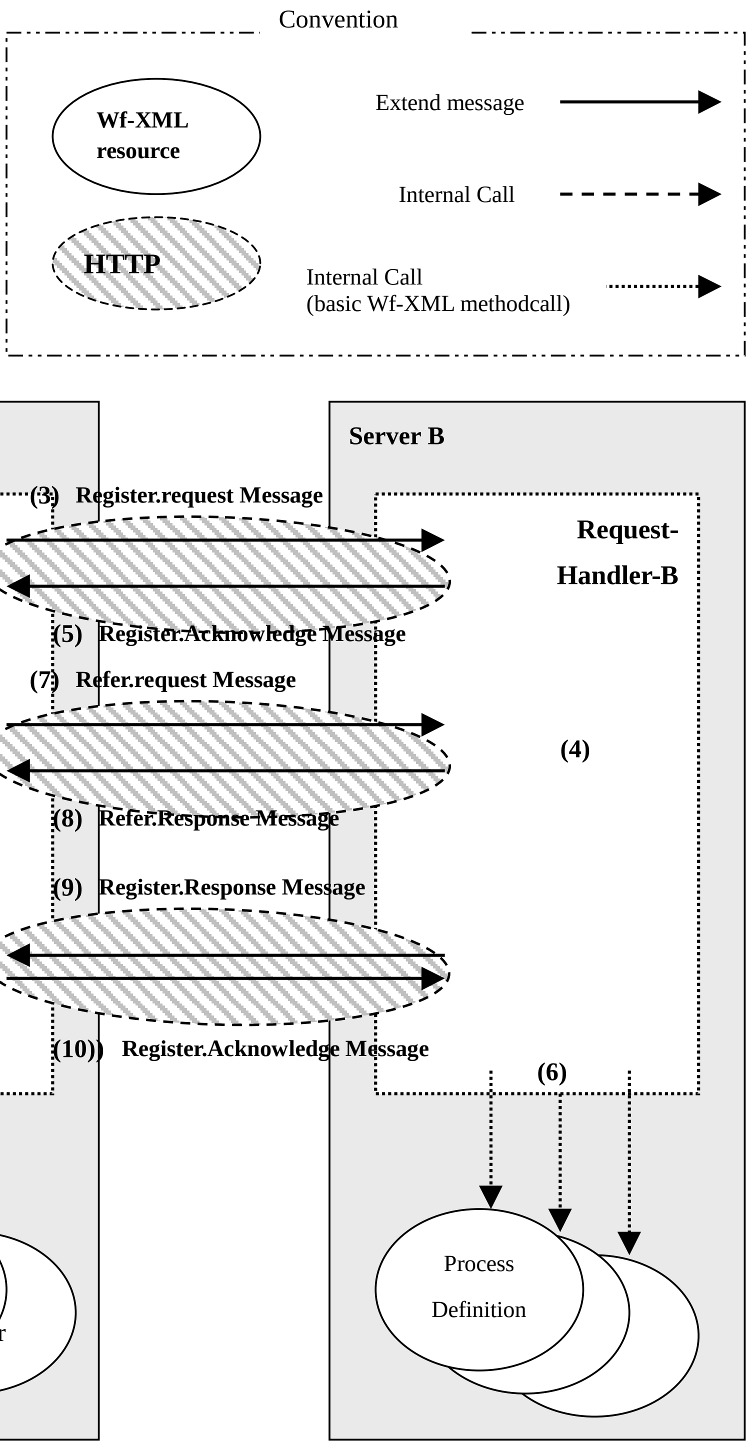
(8) Sending all results of the Wf-XML requests

After all Wf-XML requests in the batch are finished, Request-Handler-B sends a “register operation, response” message to the response handler with the results of Wf-XML requests that can also contain exceptions of requests, via an HTTP request.

(9) Replying acknowledgement

Request-Handler-A returns an “acknowledge message” by the HTTP result.

Figure: Example of Messaging



Convention
Wf-XML resource
HTTP
Extend message
Internal Call
Internal Call (basic Wf-XML methodcall)
Server A
Server B
Request-Handler-A
Request-Handler-B
(3) Register.request Message
(5) Register.Acknowledge Message
(7) Refer.request Message
(8) Refer.Response Message
(9) Register.Response Message
(10) Register.Acknowledge Message
(2)
(4)
(1)
(6)
Activity Observer
Process Definition

The following is an example of messages used in this use case.

An example of STEP (3)

```
<?xml version="1.0" ?>
 <WfMessage version="1.0">
    <WfMessageHeader>
       <MessageID>i2o:msg:yyyymmdd:hhmmss007</MessageID>
       <Title>iwf-sample</Title>
       <Mission>
          <From>http://inbound/swap/reception</From>
          <To>http://outbound/iwf/RequestHandler</To>
          <MissionID>i2o:msn:yyyymmdd:hhmmss004</MissionID>
          <Register.Request>
             <ConnectionType>
                <Asynchronous/>
             </ConnectionType>
             <Parameters>parameters depend on each vender</Parameters>
          </Register.Request>
       </Mission>
    </WfMessageHeader>
    <WfMessageBody>
       <ProcessDefinition.GetData.Request Index="1">
          <Target>http://outbound/iwf?pd.001</Target>
          <!--
            Other tags are omitted. See "Wf-XML Message format" section.
          -->
       </ProcessDefinition.GetData.Request>

       <ProcessInstance.SetData.Request Index="2">
          <Target>http://outbound/iwf?001.10</Target>
          <!-- Other tags are omitted. -->
       </ProcessInstance.SetData.Request>

       <ActivityObserver.ProcessInstanceCompleted.Request Index="3">
          <Target>http://inbound/Wf-XML/ao?pid3.aid47</Target>
          <!--   Other tags are omitted. -->
       </ActivityObserver.ProcessInstanceCompleted.Request>
       <!--   *** Other WfMessages are omitted. ***    -->
    </WfMessageBody>
 </WfMessage>
```

An example of STEP (5)

```
<?xml version="1.0" ?>
 < WfMessage version="1.0">
    <WfMessageHeader>
       <MessageID>o2i:msg:yyyymmdd:hhmmss006</MessageID>
       <Title>iwf-sample</Title>
       <Mission>
```

```
      <From>http://outbound/iwf/RequestHandler</From>
      <To>http://inbound/swap/reception</To>
      <MissionID>i2o:msn:yyyymmdd:hhmmss004</MissionID>
      <Register.Acknowledge>
        <ConnectionType>
          <Asynchronous/>
        </ConnectionType>
      </Register.Acknowledge>
    </Mission>
  </WfMessageHeader>
</WfMessage>
```

An example of STEP (7)

```
<?xml version="1.0" ?>
 <WfMessage version="1.0">
   <WfMessageHeader>
     <MessageID>i2o:msg:yyyymmdd:hhmmss_r01</MessageID>
     <Title>iwf-sample</Title>
     <Mission>
       <From>http://inbound/swap/reception</From>
       <To>http://outbound/iwf/RequestHandler</To>
       <MissionID>i2o:msn:yyyymmdd:hhmmss004</MissionID>
       <Refer.Request>
         <ConnectionType>
           <Synchronous/>
         </ConnectionType>
         <Parameters>parameters depend on each vender.</Parameters>
       </Refer.Request>
     </Mission>
   </WfMessageHeader>
 </WfMessage>
```

An example of STEP (8)

```
<?xml version="1.0" ?>
 <WfMessage version="1.0">
   <WfMessageHeader>
     <MessageID>o2i:msg:yyyymmdd:hhmmss_r01</MessageID>
     <Title>iwf-sample</Title>
     <Mission>
       <From>http://outbound/iwf/RequestHandler</From>
       <To>http://inbound/swap/reception</To>
       <MissionID>i2o:msn:yyyymmdd:hhmmss004</MissionID>
       <Refer.Response>
         <ConnectionType>
           <Synchronous/>
         </ConnectionType>
         <Parameters>parameters depend on each vender.</Parameters>
       </Refer.Response>
     </Mission>
   </WfMessageHeader>
   <WfMessageBody>
```

```
    <ProcessDefinition.GetData.Request Index="1">
      <Target>http://outbound/iwf?pd.001</Target>
      <!--
         Other tags are omitted.
         See "Wf-XML Message format" section.
       -->
    </processDefinition.GetData.Request>

    <ProcessInstance.SetData.Request Index="2">
      <Target>http://outbound/iwf?001.10</Target>
      <!-- Other tags are omitted. -->
    </ProcessInstance.SetData.Request>
    <!--   *** Other Wf-XML message are omitted. ***   -->
  </WfMessageBody>
</WfMessage>
```

An Example of STEP (9)

```
<?xml version="1.0" ?>
 <WfMessage version="1.0">
   <WfMessageHeader>
     <MessageID>o2i:msg:yyyymmdd:hhmmss007</MessageID>
     <Title>iwf-sample</Title>
     <Mission>
       <From>http://outbound/iwf/RequestHandler</From>
       <To>http://inbound/swap/reception</To>
       <MissionID>i2o:msn:yyyymmdd:hhmmss004</MissionID>
       <Register.Response>
         <ConnectionType>
           <Asynchronous/>
         </ConnectionType>
         <Parameters>parameters depend on each vender.</parameters>
       </Register.Response>
     </Mission>
   </WfMessageHeader>
   <WfMessageBody>
     <ProcessDefinition.GetData.Response Index="1">
       <Target>http://outbound/iwf?pd.001</Target>
       <!--
        Other tags are omitted. See "Wf-XML Message format" section.
       -->
     </ProcessDefinition.GetData.Response>

     <ProcessInstance.SetData.Response Index="2">
       <Target>http://outbound/iwf?001.10</Target>
       <!-- Other tags are omitted. -->
     </ProcessInstance.SetData.Response>

     <ActivityObserver.ProcessInstanceCompleted.Response Index="3">
       <Target>http://inbound/Wf-XML/ao?pid3.aid47</Target>
       <!-- Other tags are omitted. -->
     </ActivityObserver.ProcessInstanceCompleted.Response>
```

```
        <!--   *** Other Wf-XML message are omitted. ***   -->
     </WfMessageBody>
  </WfMessage>
```

An example of STEP (10)

```
<?xml version="1.0" ?>
  <WfMessage version="1.0">
     <WfMessageHeader>
        <MessageID>i2o:msg:yyyymmdd:hhmmss007</MessageID>
        <Title>iwf-sample</Title>
        <Mission>
           <From>http://inbound/swap/reception</From>
           <To>http://outbound/iwf/RequestHandler</To>
           <MissionID>i2o:msn:yyyymmdd:hhmmss004</MissionID>
           <Register.Acknowledge>
              <ConnectionType>
                 <Asynchronous/>
              </ConnectionType>
           </Register.Acknowledge>
        </Mission>
     </WfMessageHeader>
</WfMessage>
```

Examples of a synchronous Wf-XML request

The following is an example of a Register.Request message and a Register.Response message for the synchronous Wf-XML request via a single HTTP connection.

An Example of Register.Request message.

```
<?xml version="1.0" ?>
<WfMessage version="1.0">
   <WfMessageBody>
   <ProcessDefinition.GetData.Request>
     <Target>http://outbound/iwf</Target>
     <RequestID>i2o:req:yyyymmdd:hhmmss002</RequestID>
     <!--
             Other tags are omitted. See "Wf-XML Message format" section.
     -->
   </ProcessDefinition.GetData.Request>
  </WfMessageBody>
</WfMessage>
```

An Example of Register.Request message.

```
<?xml version="1.0" ?>
<WfMessage version="1.0">
```

```
    <WfMessageBody>
      <ProcessDefinition.GetData.Response>
        <Target>http://outbound/iwf</Target>
        <RequestID>i2o:req:yyyymmdd:hhmmss002</RequestID>
    <!--
          Other tags are omitted. See "Wf-XML Message format" section.
    -->
      </ProcessDefinition.GetData.Response>
    </WfMessageBody>
  </WfMessage>
```

Example with Exception

The following is an example of an exception message. In this case, a server invokes a synchronous control operation, but a receiver doesn’t implement the synchronous control operation, so the receiver returns the following exception message.

```
<?xml version="1.0"?>
  <WfMessage version="1.0">
    <WfMessageHeader>
      <MessageID>bar:1999mmdd:hhmmss</MessageID>
      <Title>iwf-sample</Title>
      <Mission>
        <From>http://www.bar.com/inter-workflow-reception/</From>
        <To>http://www.foo.com/iwf-reception/</To>
        <MissionID>1999mmdd:hhmmss_mi001</MissionID>
        <Control.Response>
          <ConnectionType>
            <Synchronous/>
          </ConnectionType>
          <Parameters>parameters depend on each vender implementation.</parameters>
        </Control.Response>
      </Mission>
    </WfMessageHeader>
    <fatal.exception>
      <fatal>
        <ExceptionType>WfXML.Standard</ExceptionType>
        <MainCode>n</MainCode>
        <Subject>cannot synchronous processing</Subject>
      </fatal>
    </fatal.exception>
  </WfMessage>
```

The following is an example of a Register.Response message with a Wf-XML request exception. In this case, a server makes asynchronous Wf-XML requests, then a Wf-XML request causes an exception.

```
<?xml version="1.0" ?>
  <WfMessage version="1.0">
    <WfMessageHeader>
      <MessageID>o2i:msg:yyyymmdd:hhmmss007</MessageID>
      <Title>iwf-sample</Title>
      <Mission>
        <From>http://outbound/iwf/RequestHandler</From>
        <To>http://inbound/swap/reception</To>
        <MissionID>i2o:msn:yyyymmdd:hhmmss004</MissionID>
        <Register.Response>
          <ConnectionType>
            <Asynchronous/>
          </ConnectionType>
          <Parameters>parameters depend on each vender.</Parameters>
        </Register.Response>
      </Mission>
    </WfMessageHeader>
    <WfMessageBody>
       <fatal.exception>
         <Target> http://outbound/iwf?pd.001 </Target>
         <Fatal>
           <ExceptionType>Wf-XMLStandard</ExceptionType>
             <MainCode>7</MainCode>
             <Subject>cannot processing. </Subject>
         </Fatal>
       </fatal.exception>
      <!--   *** Other Wf-XML message are omitted. ***   -->
    </WfMessageBody>
  </WfMessage>
```

1.4 Other Extensions

1.4.1 Validation checking with DTD

DTD (Document Type Definition) and XML formats in this specification has some characteristics as follows:

- Validation:

This specification provides easy way to validate XML document. If validating processors don't report any violations of constrains in the XML message, it means that the message is valid e-form for inter-workflow system and it contains proper parameters to process corresponding operation. This verification could just be done with the DTD, no other docuuments are needed.

In this DTD, each element is designed to eliminate ambiguity and to be unique to each other elements. Each interface, method, request and response message has respective structure of elements.

- Independent from implementations:

XML formats are independent from implementations both existing workflow systems and inter-workflow gateway systems that is a WWW server extension or server plug-in or servlet.

This specification uses HTTP for transport layer, the DTD is designed to support synchronous/asynchronous interoperation and batch operation. It is independent to the transport mechanism or transport layer of the inter-workflow system.

The XML message is consist of header part and body part. The header part holds the particular dialogue information between two different workflow systems and the body part contains the actual messages. e.g. a resource URI is placed in both request and response messages besides the same URI may appears in the request-line as a target URL in the HTTP sessions.

- Structured documents:

DTD and XML formats have appropriate structure. The structure of element is simple and naming rule is easy.

- Extensible format:

Extensible feature is one of most important issues on common protocol. This specification provides the general extensible framework that keep consistency with DTD. The detail is described in section 2.2.

This document also defines detailed data type described in section 2.1, but there are so many XML standards and vendor specific data formats in the industry. To validate those existing XML data in the XML messages, this specification provides the following approaches:

1. Modifying the DTD specification.

   To transfer particular workflow data, vendors may modify some part of the DTD specification by using this DTD as an external subset or rewriting the DTD. This should be done with standard way defined in the XML specification.

2. Specifying additional DTD for the workflow data.

The "OpenContextData" element is designed to specify external DTDs with two attributes "name" and "dtdreference". The "name" attribute is a text string data that is used to specify DTD by name. The "dtdreference" attribute is used to refer specified DTD by its URL.

3. Using only this DTD specifications.

The "ClosedContextData" element is used to include specified XML data set that is conformed to the workflow data type definitions described in section 2.1.

- Others:
  - Document type declaration: This specification strongly recommends using "document type declaration".
  - Character code: This specification assumes to use UTF-8.
  - Namespace in XML: Namespace in XML is not required in this specification.
  - XSL: XSL is out of scope in this specification.
  - External entity: DTD in this specification does not use external entities. XML formats in this specification have one external entity that is XSWAP DTD and other external entities that are vender dependent DTD(s).

1.4.2 Extension for Tracking

This section describes the way for enhanced tracking and the principal additional tags: “logging” , and “GlobalKey”.

In ordinary workflow systems, it is very hard to retrieve all history of a sub-process tree. Because history logs of process instances is stored only in a server that creates or manages the process instances. For example, a process instance creates some sub-process instances on other servers, and each of the sub-process instances creates their sub-process instances on other servers in tern, and then the root process instance makes a sub-process tree. If the root process instance wants to know the progress, it needs to send lots of Wf-XML requests, such as “GetData” or “GetHistory”, to retrieve history logs on the other servers.

To make enhance tracking more efficient, we provide the “logging server” that gathers history of process instances. Each logging server must implement the “ActivityObserver” interface. Since the other servers send history logs to the logging server, to keep track of the current status and all history logs of a sub-process tree, in example above, becomes easy to retrieve from the logging server. Thus we introduce “logging” tag that is used to specify logging server(s).

To retrieve the current status or all history logs of a sub-process tree, it is necessary to provide a way to group process instances. Thus we introduce the “GlobalKey” tag that is used to specify process group. If all process instances in a sub-process tree has same identifier, it is easy to get the current state of a sub-process tree to look up processes that have the GlobalKey of the group.

logging: the tag to gather history logs

Major samples: ProcessDefinition::CreateProcessInstance, ProcessInstance::GetData, ActivityObserver::Notify

Each process instance may be assigned a logging URI when it is created. When the parent process instance generates its child process instances, it gives them its logging URI. Therefore all the related process instances have the same logging URI. Those process instances send events not only to their observer instances but also to the logging process instance.

GlobalKey: the tag to relate process instances

Major samples: ProcessDefinition::CreateProcessInstance, ProcessInstance::GetData, ProcessDefinition::Listinstances

The GlobalKey tag is used to assign a unique ID to the related process instances. It makes easy to retrieve all related process instances together. For example, a GlobalKey can be used as a filter parameter of the ProcessDefinition::Listinstances method.

When a parent process instance generates its child process instances, it gives them its GlobalKey. Then the child processes are created with the GlobalKey.

1.4.3 Error Contents

There are two types of errors, “Fatal” and “Warning”. A fatal exception means that the attempt is fail and a server can not execute more. In this case, the server returns not usual Wf-XML response page but a fatal-exception-page that contains some elements as follows.

- **Fatal.Exception**
  The top tag of the fatal-exception-page.
- **Target**
  URI of the processed resource.
- **Fatal**
  includes information of this exception. “Fatal” contains “Type” tag, “Code” tag, and “Msg” tag.
- **ExceptionType (String)**
  A string value that includes a type of error. The string value “Wf-XMLStandard” indicates a Wf-XML standard error. If it is an extended error by a vendor organization, its name will appear.
- **MainCode (Integer)**
  Three digit positive integer value that indicates an error code.
- **SubCode(Integer)**
  Three digit positive integer value that indicates an error code. It details the main code.
- **Subject (String)**
  This is one-line teext description of what went wrong.
- **Description (String, optional)**
  A string value with human readable description of this error.

In our format, a caller can easily distinguish between “Fatal” and “Warning” by the tag of exception information element, that is ether “Fatal.Exception” tag or “Warning” tag.

```
<Fatal.Exception>
  <Target>http://www.foo.com/inter-workflow-reception/990823/001</Target>
  <Fatal>
    <ExceptionType>Wf-XML.Standard</ExceptionType>
    <MainCode>9</MainCode>
    <SubCode>10</SubCode>
    <Subject>OperationFailed</Subject>
    <Description>cannot processing</Description>
  </Fatal>
</Fatal.Exception>
```

The following is example of a fatal page.

Standard Error Codes

This specification defines the standard error codes as follows.

- **InvalidFormat**
  The data format violates the DTD definition.
- **InvalidMethod**
  An attempt was made to execute a method. But the specified method is not allowed to execute in this resource.
- **UnimplementedMethod**
  An attempt was made to execute a method. But the specified method is not implemented.
- **InvalidKey**
  A method was passed an invalid or inappropriate URI as a key.
- **InvalidType**
  A method was passed an attribute value in an invalid or inappropriate type.
- **OperationFailed**
  The execution of a method was failed.
- **ServiceNotReady**
  The load of a server was too heavy to execute the specified method.
- **OperationTimeout**
  A request has not been completed within the specified time, and it was terminated.
- **AccessViolation**
  An attempt was made to execution an operation. But the operator is not allowed to execute the specified operation.
- **UnimplementedOperation**
  An attempt was made to execute a method. The specified method itself is implemented, but some operations for the tags are not implemented. This error occurs when an attempt was made to execute a meaningless operation, mainly SetData.

1.4.4 Detailed Definition of History

The history information defined by WfMC is classified into two types: the one defined by Inteface4 and the other defined by Interface5 (and Interface2). The former is used to record communication histories and the latter is used to record history logs of execution on servers. This specification gives more detailed event definitions for WfMC Interface 5.

A workflow server can refer the history logs that are recorded by other workflow servers as basic event objects and detailed event objects. This specification defines the XML formats for each event object.

WfMC Inteface5 classifies history logs into a several categories. Each category has a few history formats. The categories are further classified by eventCodes as shown in the following table. Each eventCode specifies the reason why the history logs are recorded. WfMC Interface5 categories correspond to basic events. And eventCodes correspond to detailed events.

This specification also defines the basic events and detailed events for the events that are recorded just as a change of an attribute value in Interface5. Because those events may have an important meaning in Wf-XML or include an important operation, such as Participant, or Data. Those event objects use the same formats for the NOTIFY method.

Table: The Relation between WfMC Interface5 and this specification

WfMC Interface 5		**Wf-XML**	
Category	**eventCode**	**Basic Event**	**Detailed Event**
Create/Start Process/Subprocess Instance Audit Data	**WMCreateProcessInstance**	**none**	**CreateProcessInstance**
	WMStartProcessInstance	**ChangedProcessInstanceState**	**StartProcessInstance**
Change Process/Subprocess InstanceState Audit Data	**WMChangedProcessInstanceState**		**ChangedProcessInstanceState**
	WMComplatedProcessInstance		**CompletedProcessInstance**
	WMTerminatedProcessInstance		**TerminatedProcessInstance**
	WMAbortedProcessInstance		**AbortedProcessInstance**
	WMWaintinOnEvent		**none**
	WMEventOccurred		**none**
	WMStartedSubprocess		**none**
	WMCompletedSubprocess		**none**
Assing Process Instance Atributes Audit Data	**WMAssignedProcessInstanceAttributes**	**AssingedProcessInstanceAttributes**	**ChangedProcessInstanceState**
			ChangedProcessInstanceData
			ChangedProcessInstanceParticipant
Change Activity Instance State Audit Data	**WMChangedActivityInstanceState**	**ChangeActivityObserverState**	**ChangeActivityInstanceState**
	WMCompletedActivityInstance		**CompletedActivityInstance**
	WMTerminatedActivytiInstance		**TerminatedActivityInstance**
	WMAbortedActivityInstance		**AbortedActivityInstance**
	WMWaitingOnEvent		**none**
	WMEventOccured		**none**
Assign Activity InstanceState Audit Data	**WMAssignActivityInstanceAttributes**	**AssignedActivityObserverAttributes**	

Event Descriptions

Every event must start with the common format "AbstructEvent object format". This format corresponds to CWAD Prefix Information and CWAD Suffix Information of Interface5.

- **CreateProcessInstanceEvent**

  This event indicates that a process instance has been created.

- **ChangedProcessIntanceStateEvent**

  This event indicates that the state of a process instance has been changed.
- **AssignedProcessInstanceAttributeEvent**

  This event indicates that an attribute of a process instance has been changed.
- **ChangedActivityInstanceStateEvent**

  This event indicates that the state of an Activity instance has been changed.
- **ChangedActivityInstanceAttributesEvent**

  This event indicates that an attribute of an Activity instance has been changed.

The followings are descriptions of detailed events. Their superordinate basic event objects are put in parentheses.

- **StartProcessInstanceEvent**

  (ChangedProcessInstanceStateEvent)

  This event indicates that a process instance has gotten into the "Open.running" state for the first time.
- **CompletedProcessInstanceEvent**

  (ChangedProcessInstanceStateEvent)

  This event indicates that a process instance has gotten into the "Closed.completed" state.
- **TerminatedProcessInstanceEvent**

  (ChangedProcessInstanceStateEvent)

  This event indicates that a process instance has gotten into the "Closed.Terminated" state.
- **AbortedProcessInstanceEvent**

  (ChangedProcessInstanceStateEvent)

  This event indicates that a process instance has gotten into the "Closed.Aborted" state.
- **StartedSubprocess**

  (ChagendProcessInstanceStateEvent)

  This event indicates that a process instance has generated its child process instance.
- **CompletedSubprocess**

  (ChangedActivityInstanceStateEvent)

  This event indicates that a child process instance that was generated by the parent process instance has been terminated.
- **ChangedProcessInstanceDataEvent**

  (ChangedProcessInstanceAttributesEvent)

  This process indicates that the data in a process instance has been changed.
- **ChangedProcessInstanceParticipantEvent**

  (ChangedProcessInstanceAttributesEvent)

This event indicates that the person in charge of a process instance has been changed.

- **CompletedActivityInstanceEvent**

  (ChangedActivityInstanceStateEvent)

  This event indicates that an Activity instance has gotten in to the “Closed.completed” state.

- **TerminatedActivityInstanceEvent**

  (ChangedActivityInstanceStateEvent)

  This event indicates that an Activity instance has gotten into the “Closed.Terminated” state.

- **AbortedActivityInstanceEvent**

  (ChangedActivityInstanceStateEvent)

  This event indicates that an Activity instance has gotten into the “Closed.Aborted” state.

2. Technical Specification

2.1 Documentation Convention

The data types used in this specification are as follows. These types are defined for explaining contents of the XML element, not defined for types for the “contextDataInfo” nor “resultDataInfo” tag.

- **URI**
  The URI of the server resource. This is globally unique. All URI values should be expressed in the standard HTTP (HTTPS) URL format as specified by RFC2068 (in HTTP bindings).
- **ID**
  A string value which is used as a key to identify a resource. This is unique within a server.
- **String**
  All String values must not include any keywords reserved by XML.
- **XML**
  The data that are structured by XML tags.
- **Data/time**
  Date and time values conformable to the data/time types of the UTC format.
- **BOOL**
  encoded with "1" representing true, and "0" representing false.
- **Integer**
  A 32-bit signed decimal numeric value expressed as [+-]?[0-9]+.

Tags in this specification are classified further into two types: required and optional. The optional tags are marked as “optional” in this specification. The optional tag in a request page means that the specified parameter can be omitted, while the one in a response page means that it may not be returned or may be assigned a default value.

2.2 Extended message format specification

This section describes the details of the extended message format. The message format depends on operations and phases. Only contents of WfMessageBody and “operation-phase dependent tag” are different, the other parts are the same.

In the following sections, first we describe common tags, secondly the format for synchronous Wf-XML request invocation and the formats of each messages used in asynchronous Wf-XML request invocation, lastly use cases of this extension.

2.2.1 Common message format

The top tag of a message may contain of two tags, “WfMessageHeader” and “WfMessageBody”.

A WfMessageHeader may have the following tags.

- **MessageID (optional)**

  The “MessageID” is used to identify each message to prevent message duplication. When a server sends a message the server must generate new MessageID for the message. The MessageID must be unique in a sender.
- **Title (optional)**

  The “Title” shows the description of this message. It can contain any strings.
- **Mission (optional in a synchronous request)**

  The “Mission” contains information on this request(s).
- **From (optional in a synchronous request)**

  The “From” specifies a request handler of the sender. In an asynchronous Wf-XML request, the response will be sent to this URI.
- **To (optional in a synchronous request)**

  The “To” specifies a request handler by which the message is handled.
- **ResponseRequired(optional)**

  The “ReponseRequired” specifies whether a target request handler must return response of the request or not. It can contain one of three empty elements: “No”, ”IfError”, ”Yes”.
- **MissionID(optional in a synchronous request)**

  The “MissionID” is used to identify each mission. When a server invokes a new mission by sending a “Register.Request” message, the server generates a new MissionID. A MissionID must be unique in the server. If a server that receives a “Register.Request” message detects duplicated MissionIDs from one server, the receiver must return an exception.
- **ConnectionType(optional in a synchronous request)**

  The “ConnectionType“ indicates whether the operation is performed synchronously or asynchronously. It can contain one of the three empty tags, “<Synchronous/>”,”<Asynchronous/>” and ”<Both/>”. Only request messages can specify “<Both/>”, it means that the receiver of this message can decide whether it executes this operation asynchronously or synchronously.

- **Parameters(optional)**

  The “Parameters” appears only in request messages. It contains parameters that depend on venders and implements.

- **Progress**

  The “Progress” appears only in an element of the “operation-phase dependent tag” in the inbound messages. It shows a status of the mission. It can contain one of the six states: ”Initialized”, ”Executing”, ”Stopped”, ”Canceled”, ”Aborted”, and ”Finished”.

Exception

An exception is returned as a “messaging exception” or a Wf-XML request exception. The “messaging exception” occurs when a mission, an operation, or messaging is failed in interaction among workflow servers. The Wf-XML request exception occurs when an execution of the Wf-XML method is failed. When a messaging exception occurs, exception information is included in the WfMessageBody with the “Fatal” tag. The messaging exception is returned only in the response phase and the acknowledge phase.

Exception information is an element of “WfMessageHeader” in Wf-XML. In our format, exception information is an element of “WfMessageBody”. It is natural that both results and exception information of one batch are listed in WfMessageBody when some exceptions occur in the execution.

The following is an example of the response message. The exception information of this message indicates that a synchronous control operation, but the server received can’t control operation synchronously.

```
<?xml version="1.0" ?>
 <WfMessage version="1.0">
   <WfMessageHeader>
     <MessageID>o2i:msg:yyyymmdd:hhmmss007</MessageID>
     <Title>iwf-sample</Title>
     <Mission>
       <From>http://outbound/iwf/RequestHandler</From>
       <To>http://inbound/iwf/reception</To>
       <MissionID>i2o:msn:yyyymmdd:hhmmss004</MissionID>
       <Register.Response>
         <ConnectionType> <Asynchronous/> </ConnectionType>
         <Parameters>parameters depend on each vender.</Parameters>
       </Register.Response>
     </Mission>
   </WfMessageHeader>
   <WfMessageBody>
      <Fatal.exception>
        <Target> http://outbound/iwf?pd.001 </Target>
        <Fatal>
          <ExceptionType>swapStandard</ExceptionType>
```

```
            <MainCode>7</MainCode>
            <Subject>cannot processing. </Subject>
        </Fatal>
      </fatal.exception>
     <!--   *** Other Wf-XML message are omitted. ***   -->
   </WfMessageBody>
 </WfMessage>
```

When an exception occurs in execution of Wf-XML requests, each Wf-XML exception appears in the fatal tag of the WfMessageBody with response message. See also example described in the later.

2.2.2 Format for a synchronous Wf-XML request invocation

For synchronous Wf-XML request invocation, two types of message formats are used: “Register.Request” and “Register.Response”. This section describes the specifications of those formats.

Register.Request

The “Register.Request” message must have the WfMessageBody that contains at least one Wf-XML request page. Most “Register.Request” messages don’t have the “WfMessageHeader”, but some of them may have it. Information in the “WfMessageHeader” is also useful in a synchronous request to ensure reliability.

- **MessageID**

  The “MessageID” is used to prevent message duplication.
- **Title(optional)**

  The “Title” contains the description of this message.
- **From(optional)**

  The “From” specifies the URI of the sender of this message. The “From” may contain the request-handler’s URI which can be used to retry to send the response page of this Wf-XML request when an HTTP connection is disconnected before the response page returns.

The following is an example of the “Register.Request” message.

```
<?xml version="1.0" ?>
<WfMessage version="1.0">
 <WfMessageHeader>
   <MessageID>i2o:msg:yyyymmdd:hhmmss001</MessageID>
   <Title>iwf-sample</Title>
 </WfMessageHeader>
 <WfMessageBody>
   <ProcessDefinition.GetData.Request>
     <Target>http://outbound/iwf</Target>
     <RequestID>i2o:req:yyyymmdd:hhmmss002</RequestID>
     <!-- Other tags are omitted. -->
   </ProcessDefinition.GetData.Request>
 </WfMessageBody>
```

</WfMessage>

In this example, the “WfMessageHeader“ is used to prevent message duplication and Wf-XML request duplication, but no tags for the mission, such as the MissionID or the “operation-phase dependent tag”.

Register.Response

The “Register.Request” message must have the “WfMessageBody” that contains at least one Wf-XML request page. It is allowed to contain any Wf-XML response pages in the “WfMessageBody”.

In case the correspondent message has the “WfMessageHeader”, the “Register.Response” may have the “WfMessageHeader”. Information in the “WfMessageHeader” is also useful in a synchronous request to ensure reliability.

The following is example of a Register.Response message.

```
<?xml version="1.0" ?>
 <WfMessage version="1.0">
   <WfMessageHeader>
     <MessageID>o2i:msg:yyyymmdd:hhmmss002</MessageID>
     <Title>iwf-sample</Title>
     <Mission>
       <From>http://outbound/iwf/RequestHandler</From>
       <To>http://inbound/swap/reception</To>
       <MissionID>i2o:msn:yyyymmdd:hhmmss001</MissionID>
       <Register.Response>
         <ConnectionType> <Synchronous/> </ConnectionType>
         <Progress><Complete/></Progress>
         <Parameters>parameters depend on each vender implementation.</Parameters>
       </Register.Response>
     </Mission>
   </WfMessageHeader>
   <WfMessageBody>
     <ProcessDefinition.GetData.Response>
       <Target>http://outbound/iwf</Target>
       <RequestID>i2o:req:yyyymmdd:hhmmss003</RequestID>
       <!-- Other tags are omitted. -->
     </ProcessDefinition.GetData.Response>
   </WfMessageBody>
 </WfMessage>
```

In this example, the “WfMessageHeader“ is used to prevent message duplication and Wf-XML request duplication, but no elements for information of the mission.

2.2.3 Format for an asynchronous Wf-XML request invocation

A message used for asynchronous Wf-XML request invocation must have the “WfMessageHeaderer”, and it includes “operation-phase dependent tag” that contains the mission information of the message. Some messages have the “WfMessageBody”. In this chapter we describe the message format used in each operations and phases.

Register.resquest message

The “Register.Request” messages include the following tags.

- **ConnectionType**

  The “ConnectionType“ indicates whether the operation performed synchronously or asynchronously. The requestor can specify one of the three empty tags: “<Synchronous/>”,”<Asynchronous/>” and ”<Both/>”. If a receiver doesn’t implement the “ConnectionType” specified by a sender, the receiver returns an exception.

- **WfMessageBody**

  The “Register.Request” message must have at least one “WfMessageBody” which contains Wf-XML request pages. If the “WfMessageBody” contains any Wf-XML response pages, a receiver returns an exception as an acknowledge message.

The following is an example of the “Register.Request” message.

```
<?xml version="1.0" ?>
 <WfMessage version="1.0">
   <WfMessageHeader>
     <MessageID>i2o:msg:yyyymmdd:hhmmss007</MessageID>
     <Title>iwf-sample</Title>
     <Mission>
       <From>http://inbound/swap/reception</From>
       <To>http://outbound/iwf/RequestHandler</To>
       <MissionID>i2o:msn:yyyymmdd:hhmmss004</MissionID>
       <Register.Request>
         <ConnectionType> <Asynchronous/> </ConnectionType>
         <Parameters>parameters depend on each vender</Parameters>
       </Register.Request>
     </Mission>
   </WfMessageHeader>
   <WfMessageBody>
     <ProcessDefinition.GetData.Request Index="1">
       <Target>http://outbound/iwf?pd.001</Target>
       <!--
         Other tags are omitted. See "Wf-XML Message format" section.
       -->
     </ProcessDefinition.GetData.Request>

     <ProcessInstance.SetData.Request Index="2">
       <Target>http://outbound/iwf?001.10</Target>
       <!-- Other tags are omitted. -->
     </ProcessInstance.SetData.Request>

     <ActivityObserver.ProcessInstanceCompleted.Request Index="3">
       <Target>http://inbound/Wf-XML/ao?pid3.aid47</Target>
       <!--    Other tags are omitted. -->
     </ActivityObserver.ProcessInstanceCompleted.Request>
```

```
        <!--    *** Other Wf-XML message are omitted. ***    -->
    </WfMessageBody>
  </WfMessage>
```

Register.Response message

The “Register.Response” message includes following tags.

- **WfMessageBody**

  The “Register.Request” message must have the “WfMessageBody” that contains at least one Wf-XML response page. If the WfMessageBody contains Wf-XML request pages, a receiver returns an exception as an acknowledge message.

The following is an example of the “Register.Response” message.

```
<?xml version="1.0" ?>
  <WfMessage version="1.0">
    <WfMessageHeader>
        <MessageID>o2i:msg:yyyymmdd:hhmmss007</MessageID>
        <Title>iwf-sample</Title>
        <Mission>
          <From>http://outbound/iwf/RequestHandler</From>
          <To>http://inbound/swap/reception</To>
          <MissionID>i2o:msn:yyyymmdd:hhmmss004</MissionID>
          <Register.Response>
            <ConnectionType> <Asynchronous/> </ConnectionType>
            <Parameters>parameters depend on each vender.</Parameters>
          </Register.Response>
        </Mission>
    </WfMessageHeader>
    <WfMessageBody>
        <ProcessDefinition.GetData.Response Index="1">
          <Target>http://outbound/iwf?pd.001</Target>
          <!--
            Other tags are omitted. See "Wf-XML Message format" section.
          -->
        </ProcessDefinition.GetData.Response>

        <ProcessInstance.SetData.Response Index="2">
          <Target>http://outbound/iwf?001.10</Target>
          <!-- Other tags are omitted. -->
        </ProcessInstance.SetData.Response>

        <ActivityObserver.ProcessInstanceCompleted.Response Index="3">
          <Target>http://inbound/Wf-XML/ao?pid3.aid47</Target>
          <!-- Other tags are omitted. -->
        </ActivityObserver.ProcessInstanceCompleted.Response>

        <!--    *** Other Wf-XML message are omitted. ***    -->
    </WfMessageBody>
  </WfMessage>
```

Refer.Request message

The “Refer.Request” message is optional in this extension.

The “Refer” message doesn’t have WfMessageBody.

```
<?xml version="1.0" ?>
 <WfMessage version="1.0">
   <WfMessageHeader>
     <MessageID>i2o:msg:yyyymmdd:hhmmss_r01</MessageID>
     <Title>iwf-sample</Title>
     <Mission>
       <From>http://inbound/swap/reception</From>
       <To>http://outbound/iwf/RequestHandler</To>
       <MissionID>i2o:msn:yyyymmdd:hhmmss004</MissionID>
       <Refer.Request>
         <ConnectionType> <Synchronous/> </ConnectionType>
         <Parameters>parameters depend on each vender.</Parameters>
       </Refer.Request>
     </Mission>
   </WfMessageHeader>
 </WfMessage>
```

Refer.Response message

The Refer.Response message is optional in this extension.

The “Register.Response” message includes following tags.

- **WfMessageBody**

  The “Refer.Request” message must have the “WfMessageBody.” The “WfMessageBody” contains Wf-XML response pages that are results of executed Wf-XML requests. If any of Wf-XML requests are not finished, the “WfMessageBody” is empty.

```
<?xml version="1.0" ?>
<WfMessage version="1.0">
   <WfMessageHeader>
     <MessageID>o2i:msg:yyyymmdd:hhmmss_r01</MessageID>
     <Title>iwf-sample</Title>
     <Mission>
       <From>http://outbound/iwf/RequestHandler</From>
       <To>http://inbound/swap/reception</To>
       <MissionID>i2o:msn:yyyymmdd:hhmmss004</MissionID>
       <Refer.Response>
         <ConnectionType> <Synchronous/> </ConnectionType>
         <Parameters>parameters depend on each vender.</parameters>
       </Refer.Response>
     </Mission>
   </WfMessageHeader>
```

```
    <WfMessageBody>
      <ProcessDefinition.GetData.Request Index="1">
        <Target>http://outbound/iwf?pd.001</Target>
        <!--
            Other tags are omitted. See "Wf-XML Message format" section.
          -->
      </ProcessDefinition.GetData.Request>

      <ProcessInstance.SetData.Request Index="2">
        <Target>http://outbound/iwf?001.10</Target>
        <!-- Other tags are omitted. -->
      </ProcessInstance.SetData.Request>
      <!--    *** Other Wf-XML message are omitted. ***    -->
    </WfMessageBody>
</WfMessage>
```

Control.Request message

The "Control.Request" message is optional in this extension. The "Control.Request" message must not have the "WfMessageBody". The "contorol.Request" message includes the following tags in the "WfMessageHeader".

- **ControlCommand**
  The "ControlCommand" tag is an element of the "Control.Request" tag and the "Control.RequestItem" tag. It must contain one of the three states: "stop", "resume" and "cancel". If the "ControlCommand" appears in the "Control.Request" tag, it is applied to the mission. If the "ControlCommand" tag is in the "Control.RequestItem", it is applied to a Wf-XML request that is indicated by the "Index" attribute.
- **Control.RequestItem(optional)**
  The "Control.RequestItem" is used to control each Wf-XML request. It contains the "Index" attribute and the "ControlCommand".
- **Index**
  The "Index" is used to identify a Wf-XML request to which "ControlCommand" apply.

The following is an example of the Control.Request message.

```
<?xml version="1.0"?>
<WfMessage version="1.0">
    <WfMessageHeader>
      <MessageID>i2o:msg:yyyymmdd:hhmmss_c02</MessageID>
      <Title>iwf-sample</Title>
      <Mission>
        <From>http://inbound/swap/reception</From>
        <To>http://outbound/iwf/RequestHandler</To>
        <MissionID>i2o:msn:yyyymmdd:hhmmss004</MissionID>
        <Control.Request>
          <ConnectionType> <Synchronous/> </ConnectionType>
```

```
            <Parameters>parameters depend on each vender implementation.</Parameters>
            <Control.RequestItem Target="1">
              <ControlCommand>terminate</ControlCommand>
            </Control.RequestItem>
            <Control.RequestItem Target="2">
              <ControlCommand>stop</ControlCommand>
            </Control.RequestItem>
            <Control.RequestItem Target="3">
              <ControlCommand>resume</ControlCommand>
            </Control.RequestItem>
          </Control.Request>
        </Mission>
      </WfMessageHeader>
    </WfMessage>
```

Control.Response message

The “Control.Request” messages is optional in this extension. The “Control.Response” message must not have WfMessageBody. The “Control.Response” message includes the following tags. The result of mission control is returned with a “Progress” tag.

- **Control.ResponseItem(optional)**
  The “Control.ResponseItem” is used to return the result of a control operation of a Wf-XML request. It contains the “Index” attribute, the “ControlCommand” tag and the “ControlResult” tag.
- **Index**
  The “Index” is used to identify a Wf-XML request to which “ControlCommand” applied to.
- **ControlCommand**
  The “ControlCommand” appears as an element of the “ControlItem” tag. It indicates which “ControlCommand” was specified in the “Control.Request” message.
- **ControlResult**
  The “ControlResult” indicates the result of control operation, it must contains either “success” or “failure”.

The following is an example of the “Control.Response” message.

```
<WfMessage version="1.0">
   <WfMessageHeader>
     <MessageID>o2i:msg:yyyymmdd:hhmmss_c03</MessageID>
     <Title>iwf-sample</Title>
     <Mission>
       <From>http://outbound/iwf/RequestHandler</From>
       <To>http://inbound/swap/reception</To>
       <MissionID>i2o:msn:yyyymmdd:hhmmss004</MissionID>
       <Control.Response>
         <ConnectionType> <Asynchronous/> </ConnectionType>
```

```
            <Progress>complete</Progress>
            <Parameters>parameters depend on each vender implementation.</Parameters>
            <Control.ResponseItem Target="1">
               <ControlCommand>teminate</ControlCommand>
               <ControlResult>success</ControlResult>
            </Control.ResponseItem>
            <Control.ResponseItem Target="2">
               <ControlCommand>change</ControlCommand>
               <ControlResult>success</ControlResult>
            </Control.ResponseItem>
            <Control.ResponseItem Target="3">
               <ControlCommand>resume</ControlCommand>
               <ControlResult>failure</ControlResult>
            </Control.ResponseItem>
         </Control.Response>
      </Mission>
   </WfMessageHeader>
</WfMessage>
```

Acknowledge message

All acknowledge message of all operations must not have WfMessageBody. The acknowledge message may contain exceptions.

The following is example of a Request.Acknowledge message

```
<?xml version="1.0" ?>
 <WfMessage version="1.0">
    <WfMessageHeader>
       <MessageID>o2i:msg:yyyymmdd:hhmmss006</MessageID>
       <Title>iwf-sample</Title>
       <Mission>
          <From>http://outbound/iwf/RequestHandler</From>
          <To>http://inbound/swap/reception</To>
          <MissionID>i2o:msn:yyyymmdd:hhmmss004</MissionID>
          <Register.Acknowledge>
             <ConnectionType> <Asynchronous/> </ConnectionType>
          </Register.Acknowledge>
       </Mission>
    </WfMessageHeader>
 </WfMessage>
```

2.3 ProcessDefinition Interface

2.3.1 GetData method

The GetData method is used to retrieve the value of all properties of the process definition resources.

Parameters:

parameters are specified in the “ProcessDefinition.GetData.Request” element.

[Some parameters are essential to process the method correctly and some are not. It is currentry under consideration.]

- Key (URI) – a target resource of this method.

Results:

results are contained in the “ProcessDefinition.GetData.Result” element.

- Key (URI) – a URI of the processed resource.
- Name (ID) – an identifier of this resource. The server must guarantee its uniqueness within the server’s all resources.
- ProcessDefinitionKey (URI) – a URI of this resource. The server must guarantee its uniqueness within the server’s all resources.
- Subject (String, optional) – a short description of this process definition.
- Description (String, optional) – a longer description of this process definition resource.
- Owner (String, optional) – an administrative user of this resource. This attribute may represented by an LDAP URL Format (RFC-1959) or vendor specific string.
- ContextData (XML) – input data set of this workflow process.
- ResultData (XML) – output data set of this workflow process.
- ValidStates.ProcessDefinition – a list of possible state values of this resource.
- State.ProcessDefinition – a state of this process definition. The value must be one of the ValidStates.
- UserInterface (URI, optional) – a URI of web page that enables user to start new work process from this process definition.
- Logging (URI, optional) – a URI of logging facility.
- Exception (optional) – indicates that an error has occurred while processing this method. If specified, some of above elements may be empty or omitted. The exception codes supported by this method as follows:
  - InvalidFormat
  - UnimplementedMethod
  - InvalidMethod
  - InvalidKey
  - OperationFailed(SystemError)

- ServiceNotReady
- OperationTimeout
- AccessViolation

Events:

Not specified.

2.3.2 CreateProcessInstance method

The CreateProcessInstance method is used to create a new process instance from the process definition.

Parameters:

parameters are specified in the "ProcessDefinition.CreateProcessInstance.Request" element.

- Key (URI) – a target resource of this method.
- RequestID (ID) – is used to check the request. If the same request is made more than twice, the server should return an error.
- ObserverKey (URI, optional) – a URI of the ActivityObserver resource that is to get the notification from the newly created process instance.
- GlobalKey (URI, optional) – a vendor specific globally unique identifier.
- Subject (String, optional) – a short description of the new process instance.If it is omitted, the server should use the process definition's one instead.
- Description (String, optional) – a longer description of the new process instance.
- Priority (Integer, optional) – value from 1 to 5 that indicates the importance of this process instance. If omitted, the server should use one that is provided by the process definition as a default value.
- ContextData (XML, optional) – data collection that is passed to the new process as input data.
- Comment (optional) – a comment that is logged to the event log.
- StartImmediately (optional) – if 'Yes' is specified, the newly created process starts immediately.
- Logging (optional) – a URI for logging facility.

Results:

results are contained in the "ProcessDefinition.CreateProcessInstance.Result" element.

- ProcessInstanceKey – a URI of the process instance that is newly created .
- Exception (optional) – indicates that an error has occurred while processing this method. If specified, some of above elements may be empty or omitted. The exception codes supported by this method as follows:
  - InvalidFormat

- UnimplementedMethod
- InvalidMethod
- InvalidKey
- InvalidAttributeName
- InvalidType
- OperationFailed(SystemError)
- ServiceNotReady
- OperationTimeout
- AccessViolation
- UnimplementedOperation
- InvalidRole
- InvalidData

Events:

The following events may be generated by the server while processing this method.

CreateProcessInstanceEvent

ChangeProcessInstanceStateEvent

StartProcessInstanceEvent

2.3.3 ListInstances method

The ListInstances method is used to retrieve a set of instances. The result can be filtered with given condition.

Parameters:

parameters are specified in the "ProcessDefinition.ListInstances.Request" element.

- Key (URI) – a target resource of this method.
- Filter (XML or String) – specifies conditions. If this entry is empty, the server should return all properties of all instances.
- FilterType (String) – specifies filter name.
- ResultAttributes (String list, optional) – specifies a list of concern property element. The server must return "Name", "Key" and "Priority" regardless of this value.

Results:

results are contained in the "ProcessDefinition.ListInstances.Result" element.

- Target (URI) – a URI of the processed resource.
- Instances(XML) – a list of process instance that matches the condition.
- Exception (optional) – indicates that an error has occurred while processing this method. If specified, some of above elements may be empty or omitted. The exception codes supported by this method as follows:

- InvalidFormat
- UnimplementedMethod
- InvalidMethod
- InvalidKey
- InvalidType
- OperationFailed(SystemError)
- ServiceNotReady
- OperationTimeout
- AccessViolation
- UnimplementedOperation
- InvalidFilter
- InvalidAttributeName

Events:

Not specified

2.4 ProcessInstance Interface

2.4.1 GetData method

The GetData method is used to retrieve the properties of the process instance.

Parameters:

parameters are specified in the “ProcessDefinition.GetData.Request” element.

- Key (URI) – a target resource of this method.
- ResultAttributes (optional) – specifies a list of properties to be obtained. If empty or not specified, all results are returned. The sequence of the result elements may differ from this list.

Results:

results are contained in the “ProcessInstance.GetData.Result” element.

- Key (URI) – a URI of the processed resource.
- Name (ID) – an identifier of this resource. The server must guarantee its uniqueness within the server’s all resources.
- ProcessInstanceKey (URI) – a URI of this resource. The server must guarantee its uniqueness within the server’s all resources.
- GlobalKey(URI) – a vendor specific globally unique identifier.
- Subject (String, optional) – a short description of this process definition.
- Description (String, optional) – a longer description of this process definition resource.
- State.ProcessInstance (String) – state of this process instance. The value must be one of the ValidStates.ProcessInstance.
- ValidStates.ProcessInstance (String) – a list of possible state values of this resource.
- ProcessDefinitionKey (URI) – A process definition of this instance.
- Activities (XML) – A list of activities. If “ResultAttributes” element is omitted, each “Activity” element contains following elements:
  - Key
  - Name
  - Subject
  - ExpirationDate
  - Assignees
  - CreationDate
  - HasExpired
- ObserverKey (URI, optional) – an activity observer resource.
- ResultData – Output data set of this workflow process.
- Priority – value from 1 to 5 that indicates the importance of this process instance. If

omitted, the server should use one that is provided by the process definition as a default value.

- UserInterface (URI, optional) – a URI of web page that enables user to start new work process from this process definition.
- Creator (ID, optional) – an administrator user of this resource. This attribute may represented by an LDAP URL Format(RFC-1959) or vendor specific string.
- CreationDate (optional) – the date of creation.
- LastModified (optional) – the date of the last modification.
- Logging (optional) – a URI of logging facility.
- Exception (optional) – indicates that an error has occurred while processing this method. If specified, some of above elements may be empty or omitted. The exception codes supported by this method as follows:
  - InvalidFormat
  - UnimplementedMethod
  - InvalidMethod
  - InvalidKey
  - OperationFailed(SystemError)
  - ServiceNotReady
  - OperationTimeout
  - AccessViolation

Events:

Not specified.

2.4.2 SetData method

The SetData method is used to modify the properties of the process instances such as “subject”, “description” and “priority”. It is also used to change the process instance state and workflow data. If a valid state value is specified, the process instance must transit its state to the specifed one.

Parameters:

parameters are specified in the “ProcessDefinition.SetData.Request” element.

- Key (URI) – a target resource of this method.
- Subject (optional) – new subject of this process definition.
- Description (String, optional) – new description of the process instance.
- State (String, optional) – specifies new state value.
- Priority (Integer, optional) – specify new priority value (1 to 5) of this resource.
- ContextDataData (XML, optional) – initial workflow data set for this process instance. (OpenContextData or ClosedContextData.)

- ResultData (XML, optional) – a set of data to override current workflow data. (OpenContextData or ClosedContextData)
- Comment (String, optional) – the “Comment” is used to set comment text to an event object that is associated with this method calling. If omitted, empty string should recorded in the event log.
- Exception (optional) – indicates that an error has occurred while processing this method. If specified, some of above elements may be empty or omitted. The exception codes supported by this method as follows:
  - InvalidFormat
  - UnimplementedMethod
  - InvalidMethod
  - InvalidKey
  - InvalidType
  - OperationFailed(SystemError)
  - ServiceNotReady
  - OperationTimeout
  - AccessViolation
  - UnimplementedOperation
  - InvalidRole
  - InvalidState
  - InvalidData

Results:

results are contained in the “ProcessInstance.SetData.Result” element.

- Key (URI) – a URI of the processed resource.
- Name (ID) – an identifier of this resource. The server must guarantee its uniqueness within the server’s all resources.
- ProcessInstanceKey (URI) – a URI of this resource. The server must guarantee its uniqueness within the server’s all resources.
- GlobalKey(URI) – vendor specific identifier.
- Subject (String, optional) – a short description of this process instance.
- Description (String, optional) – a longer description of this process instance resource.
- State.ProcessInstance (String) – state of this process instance. The value must be one of the ValidStates.ProcessInstance.
- ValidStates.ProcessInstance (String) – a list of possible state values of this resource.
- DefinitionKey (URI) – a process definition of this instance.
- Activities (XML) – a list of activities. If “ResultAttributes” element is omitted, each

“Activity” element contains following elements:

- Key
- Name
- Subject
- ExpirationDate
- Assignees
- CreationDate
- HasExpired

- Observer (URI, optional) – an activity observer resource.
- ResultData – output data set of this workflow process.
- Priority – value from 1 to 5 that indicates the importance of this process instance. If omitted, the server should use one that is provided by the process definition as a default value.
- UserInterface (optional) – a URI of web page that enables user to start new work process from this process definition.
- Creator (ID, optional) – an administrator user of this resource. This attribute may represented by an LDAP URL Format(RFC-1959) or vendor specific string.
- CreationDate (optional) – the date of creation.
- LastModified (optional) – the date of the last modification.
- Logging (optional) – a URI of logging facility.
- Exception (optional) – indicates that an error has occurred while processing this method. If specified, some of above elements may be empty or omitted.
- The exception codes supported by this method as follows:
  - InvalidFormat
  - UnimplementedMethod
  - InvalidMethod
  - InvalidKey
  - InvalidType
  - OperationFailed (SystemError)
  - ServiceNotReady
  - OperationTimeout
  - AccessViolation
  - UnimplementedOperation
  - InvalidRole
  - InvalidState
  - InvalidData

Events:

The following events may be generated by the server while processing this method.

ChangedProcessIntanceStateEvent

AssignedProcessInstanceAttributeEvent

ChangeActivityInstanceStateEvent

StartProcessInstanceEvent

TerminatedProcessInstanceEvent

ChangedProcessInstanceDataEvent

ChangedProcessInstanceParticipantEvent

TerminatedActivityInstanceEvent

2.4.3 TerminateProcessInstance method

The TerminateProcessInstance method is used to terminate an execution of the workflow process.

Parameters:

parameters are specified in the “ProcessInstance.TerminateProcessInstance.Request” element.

- Key (URI) – a target resource of this method.
- Reason (String, optional) – a description about the process termination.

Results:

results are contained in the “ProcessInstance.TerminateProcessInstance.Result” element.

- Key (URI) – a URI of the processed resource.
- Exception (optional) – indicates that an error has occurred while processing this method. If specified, some of above elements may be empty or omitted. The exception codes supported by this method as follows:
  - InvalidFormat
  - UnimplementedMethod
  - InvalidMethod
  - InvalidKey
  - InvalidType
  - OperationFailed (SystemError)
  - ServiceNotReady
  - OperationTimeout
  - AccessViolation
  - UnimplementedOperation

Events:

The following events may be generated while processing this method.

TerminatedProcessInstanceEvent

TerminatedActivityObserverEvent

AbortedActivityInstanceEvent

2.4.4 Subscribe method

The Subscribe method is used to register an ActivityObserver resource.

Parameters:

parameters are specified in the "ProcessInstance.Subscribe.Request" element.

- Key (URI) – a target resource of this method.
- Observer (URI) – an ActivityObserver resource which is to be notified events of this resource .

Results:

results are contained in the "ProcessInstance.Subscribe.Result" element.

- Key (URI) – a URI of the processed resource.
- Exception (optional) – indicates that an error has occurred while processing this method. If specified, some of above elements may be empty or omitted. The exception codes supported by this method as follows:
  - InvalidFormat
  - UnimplementedMethod
  - InvalidMethod
  - InvalidKey
  - InvalidType
  - OperationFailed(SystemError)
  - ServiceNotReady
  - OperationTimeout
  - AccessViolation

Events:

Not specified.

2.4.5 Unsubscribe method

The Unsubscribe method is used to stop notification from the workflow system.

parameters:

parameters are specified in the "ProcessInstance.Unsubscribe.Request" element.

- Key (URI) – a target resource of this method.
- Observer (URI) – an ActivityObserver resource.

Results:

results are contained in the “ProcessInstance.Unsubscribe.Result” element.

- Key (URI) – a URI of the processed resource.
- Exception (optional) – indicates that an error has occurred while processing this method. If specified, some of above elements may be empty or omitted. The exception codes supported by this method as follows:
  - InvalidFormat
  - UnimplementedMethod
  - InvalidMethod
  - InvalidKey
  - InvalidType
  - OperationFailed(SystemError)
  - ServiceNotReady
  - OperationTimeout
  - AccessViolation

Events:

Not specified.

2.4.6 GetHistory method

The GetHistory method is used to get a list of events. The list can be filtered by given condition.

parameters:

parameters are specified in the “ProcessInstance.GetHistory.Request” element.

- Key (URI) – a target resource of this method.
- Filter (XML or String, optional) – specifies filter.
- FilterType (optional) – specifies filter name.

Results:

results are contained in the “ProcessInstance.GetHistory.Result” element.

- Key (URI) – a URI of the processed resource.
- History (XML:eventObject) – a list of events. each event object contains following data:
  eventObject
    - Timestamp – the time at which this event was occurred.
    - Code (Integer) – event code.
    - Type (String) – a text string that represents the type of event.
    - Name (optional) – name of event.
    - Responsible (String) – a corresponding user of this event.
    - Source (XML) – a corresponding resource of this event.
      Key (URI) – a URI of the resource.

Name (ID) – an ID of the resource.

- ContainerKey (URI, optional)
- RequestID (ID) –
- OldState (String, optional) event occurred.
- NewState (String, optional)
- Transition (String, optional)
- ChangeInfo (XML, optional)
- PreviousInfo (XML, optional)

● Exception (optional) – indicates that an error has occurred while processing this method. If specified, some of above elements may be empty or omitted. The exception codes supported by this method as follows:
  - InvalidFormat
  - UnimplementedMethod
  - InvalidMethod
  - InvalidKey
  - InvalidType
  - OperationFailed(SystemError)
  - ServiceNotReady
  - OperationTimeout
  - AccessViolation
  - UnimplementedOperation

Events:

Not specified.

2.5 ActivityObserver Interface

2.5.1 GetData method

The GetData method is used to retrieve the all properties of the ActivityObserver resources.This method returns the all properties.

Parameters:

parameters are specified in the "ActivityObserver.GetData.Request" element.

- Key (URI) – a target resource of this method.

Results:

results are contained in the "ActivityObserver.GetData.Result" element.

- Key (URI) – a URI of the processed resource.
- Name (ID) – an identifier of this resource. The server must guarantee its uniqueness within the server's all resources.
- ActivityObserver Key (URI) – a URI of this resource. The server must guarantee its uniqueness within the server's all resources.
- Subject (String, optional) – a short description of this resource.
- Description (String, optional) – a longer description of this resource.
- State.ActivityObserver (String) – current state of this resource. The value must be one of the "ValidStates. actibityObserver".
- ValidStates.ActibityObserver (String) – a list of possible state values of this resource.
- ProcessInstance (URI, optional) – a URI of the sub process.
- Container (URI) – a URI of the ProcessInstance resource.
- Priority (Integer) – value from 1 to 5 that indicates the importance of corresponding activity.
- ContextData (XML) – data collection that is passed to the new process as input data.
- ResultData (XML, optional) – result data collection of the corresponding workflow process.
- Assignees (String list, optional) – a list of users assigned to this activity.
- Conclusions(String) – a list of the result status of the corresponding process.
- UserInterface (optional) – a URI of web page corresponds to this resources.
- ExpirationDate (Date, optional) – indicates the term of validity.
- HasExpired (BOOL, optional) – 'False' means that this resource has already expired. The "HasExpired" is valid if "ExpirationDate" is specified.
- CreationDate (Date) – the date of creation.
- ProcessInfo (XML, optional) – the "ProcessInfo" element contains the following element tags:

- Name
- GlobalKey
- Subject
- Description
- Definition
- Creator

specification of each element is the same as that of the ProcessInstance interface.

- Exception (optional) – indicates that an error has occurred while processing this method. If specified, some of above elements may be empty or omitted. The exception codes supported by this method as follows:
  - InvalidFormat
  - UnimplementedMethod
  - InvalidMethod
  - InvalidKey
  - InvalidType
  - OperationFailed(SystemError)
  - ServiceNotReady
  - OperationTimeout
  - AccessViolation

Events:

Not specified.

2.5.2 SetData method

The SetData method is used to modify the attributes of the Activity Observer resource.

Parameters:

parameters are specified in the “ActivityObserver.SetData.Request” element.

- Key (URI) – a target resource of this method.
- Priority (Integer, optional) – value from 1 to 5 that indicates the importance of corresponding activity.
- ResultData (XML, optional) – result data collection of the corresponding workflow process.

Results:

results are contained in the “ActivityObserver.SetData.Result” element.

- Key (URI) – a URI of the processed resource.
- Name (ID) – an identifier of this resource. The server must guarantee its uniqueness within the server’s all resources.

- Key (URI) – a URI of this resource. The server must guarantee its uniqueness within the server’s all resources.
- Subject (String, optional) – a short description of this resource.
- Description (String, optional) – a longer description of this resource.
- State.ActivityObserver(String) – a state of this resource. The value must be one of the “ValidStates. actibityObserver”.
- ValidStates.ActibityObserver (String) – a list of possible state values of this resource.
- ProcessInstance (URI, optional) – a URI of the sub process.
- Container (URI) – a URI of the PrrocessInstance resource.
- Priority (Integer) – value from 1 to 5 that indicates the importance of this process instance.
- ContextData (XML) – data collection that is passed to the new process as input data.
- ResultData (XML) – result data collection of the corresponding process.
- Assignees (String list, optional) – a list of user assigned to this activity.
- Conclusions (String) – a list of the result status of the corresponding process.
- UserInterface (optional) – a URI of web page corresponds to this resources.
- ExpirationDate (Date, optional) – indicates the term of validity.
- HasExpired (BOOL, optional) – ‘False’ means that this resource has already expired. The “HasExpired” is valid if “ExpirationDate” is specified.
- CreationDate (Date) – the date of creation.
- ProcessInfo (XML, optional) – the “ProcessInfo” element contains the following element tags:
  - name
  - GlobalKey
  - Subject
  - Description
  - Definition
  - Creator

  specification of each element is the same as that of the ProcessInstance interface.
- Exception (optional) – indicates that an error has occurred while processing this method. If specified, some of above elements may be empty or omitted. The exception codes supported by this method as follows:
  - InvalidFormat
  - UnimplementedMethod
  - InvalidMethod
  - InvalidKey

- InvalidType
- OperationFailed(SystemError)
- ServiceNotReady
- OperationTimeout
- AccessViolation
- UnimplementedOperation
- InvalidData

Events:

The following events may be generated by the server while processing this method.

ChangeActivityInstanceAttributesEvent

ChangeActivityInstanceDataEvent

2.5.3 ProcessInstanceCompleted method

The ProcessInstanceCompleted method is used to notify that the process has been completed normally. After this notification, the state value of the activity observer must turn into "Closed.Terminated".

Parameters:

parameters are specified in the "ActivityObserver.ProcessInstanceCompleted.Request" element.

- Key (URI) – target resource of this method.
- ResultData (optional) – output data of the corresponding process.
- Option – is used to specifies the result status of the process.

Results:

results are contained in the "ActivityObserver.complete.Result" element.

- Key (URI) – a URI of the processed resource.
- Exception (eventObject, optional) – indicates that an error has occurred while processing this method. If specified, some of above elements may be empty or omitted. The exception codes supported by this method as follows:
  - InvalidFormat
  - UnimplementedMethod
  - InvalidMethod
  - InvalidKey
  - InvalidType
  - OperationFailed(SystemError)
  - ServiceNotReady
  - OperationTimeout
  - AccessViolation

- InvalidOption
- InvalidData

Events:

The following events may be generated by the server while processing this method.

ChangeActivityInstanceAttributesEvent

CompletedActivityInstanceEvent

2.5.4 ProcessInstanceTerminated method

The ProcessInstanceTerminated method is used to notify that the process has been terminated. After this notification, the state of the resource must transit to ”Closed.Terminated”.

Parameters:

parameters are specified in the “ActivityObserver.ProcessInstanceTerminated.Request” element.

- Key (URI) – a target resource of this method.
- Reason (String, optional) – a description of the process termination.

Results:

parameters are contained in the “ActivityObserver.ProcessInstanceTerminated.Result” element.

- Key (URI) – a URI of the processed resource.
- Exception (optional) – indicates that an error has occurred while processing this method.

  If specified, some of above elements may be empty or omitted.

  The exception codes supported by this method as follows:

  - InvalidFormat
  - UnimplementedMethod
  - InvalidMethod
  - InvalidKey
  - InvalidType
  - OperationFailed(SystemError)
  - ServiceNotReady
  - OperationTimeout
  - AccessViolation

Events:

The following events may be generated by the server while processing this method.

TerminatedActivityInstanceEvent

2.5.5 Notify method

The Notify method is used to notify that an event has occurred.

Parameters:

parameters are specified in the “ActivityObserver.Notify.Request” element.

- Key (URI) – a target resource of this method.
- EventObject – an event object that contains the information about this notification.

Results:

results are contained in the “ActivityObserver.Notify.Result” element.

- Key (URI) – a URI of the processed resource.
- Exception (optional) – indicates that an error has occurred while processing this method. If specified, some of above elements may be empty or omitted. The exception codes supported by this method as follows:
  - InvalidFormat
  - UnimplementedMethod
  - InvalidMethod
  - InvalidKey
  - InvalidType
  - OperationFailed(SystemError)
  - ServiceNotReady
  - OperationTimeout
  - AccessViolation
  - UnimplementedOperation

Events:

Not specified.

2.6 Additional specifications

2.6.1 HTTP Protocol Handling

Post Method Request

The HTTP/1.1 POST method is used to send or receive a Wf-XML request. A request page is sent as an HTTP request, and a response page is returned as an HTTP response.

In synchronous request, URI of a Wf-XML resource that performs the Wf-XML request is specified as the Request-URI in HTTP header of an HTTP request. The URI must also be specified with the "Target" tag in a request page. If URI of a Wf-XML resource were specified both in the Request-URI in HTTP header and in the Target tag, the URI of the Target tag would be used.

In synchronous request, URI of a request handler that performs the mission is specified as the Request-URI in HTTP header of an HTTP request. The URI must also be specified with the "to" tag in a WfMessageHeader. If URI of a request handlers URI were specified both in the Request-URI in HTTP header and in the to tag, the URI of the to tag would be used.

Other request-header fields conform to HTTP/1.1.

Other HTTP Requests

When a Wf-XML server receives any other normal HTTP request, the server must act conformity with the HTTP/1.1.

- When the GET method is invoked, return the result of invoking GetData without parameters.
- When the HEAD method is invoked, return the entity-header fields corresponding to the requested resource to tell the existence of the resource. All responses to the HEAD request method must not include a message-body, even though the presence of entity-header fields might lead one to believe they do.
- When any other method is invoked, return an error.

HTTP Status Codes and Wf-XML Exceptions

Once an HTTP connection is established and a workflow server receives an HTTP request, the server must return the HTTP status code "200 OK". Any other HTTP status codes must not be returned. If an error occurs inside the server, the server must described it as a Wf-XML exception in the response page.

Only if an HTTP connection is not established successfully or a server can not understand a request, a Wf-XML server may return an HTTP client error code "4xx".

2.6.2 Parameters for searches

FilterType

This specification will not provide the definition of the filiterType. It will be defined by vendors. Vendors will also define the original definition of the tags that are used for searches.

- **The structure of the filterType is**

  "VendorName.fFlterType(.Version)".

  The version may or may not be included.

  The VenderName includes the name of the vendor that support this filter.
- All string values of the filterType must consist of single byte characters: a-Z, 0-9, and "_".

  ex. fujitsu.
- **The structure of the filter**

  The filter is expressed in the XML format.

Standard Filter

We consider that standard filter format and its "filterType" is required to ensure interoperability. We define standard filter format, and its "FilterType" is "Wf-XML.starndardFilter.1.0".

Conditional expression filters consist of the following conditional expressions. All filter items for conditional expressions must not include any tags or keywords reserved by XML.

In the following descriptions, the *name* indicates the name of a filter item, the *value* indicates a value to compare, and the *conditional expression* indicates one of the following conditional expressions.

- **Equality**

<eq><name>*name***</name><value>***value***</value></eq>**

If a value of the filter item "*name*" is equal to the *value*, this conditional expression becomes true.

- **Comparison**

<Gt><Name>*name***</Name><Value>***value***</Value></Gt>**

The Gt tag indicates "greater than". If a value of the filter item "*name*" is greater than the *value*, this conditional expression becomes true. Similarly the lt tag indicates "less than", the Ge tag indicates "greater than or equal", and the Le tag indicates "less than or equal".

- **Not Equal**

<Not>*conditional expression***</Not>**

If the *conditional expression* is true, the whole conditional expression becomes false.

- **Logical AND**

<And>*conditional expression …***</And>**

The and tag indicates a logical AND operation among the listed *conditional expression*s.

- **Logical OR**

```
<Or>conditional expression …</Or>
```

The and tag indicates a logical OR operation among the listed *conditional expression*s.

2.6.3 TYPES

In the element of contextDataInfo and the element of resultDataInfo, the following types are can be specified. Some of the types are defined by WfMC Interface4 and the others are defined by this specificaion.

- **binary**
- **boolean**
- **int**
- **unsigned**
- **float**
- **double**
- **string**
- **string[the maximum number of characters] (This is not defined by Interface4)**
- **Date**
- **URI (This is not defined by Interface4)**
- **XML (This is not defined by Interface4)**

If contextDataInfo or resultDataInfo information includes the types that are not defined by this specification, those type will be regarded as string.

3. Change History

1. **Section 1.1**

   A paragraph regarding the adoption of some SWAP specifications into the next Wf-XML is added

2. **Section 1.2**

   “Workflow data format and DTD” and “Activity Observer Interface” are added.

3. **Section 2 and 3**

   Some terms are changed with alignment to the current Wf-XML specification listed below.

- **Method names**

ProcessDefinition interfaces

・GetData

・SetData

・CreateProcessInstance

・ListInstances

ProcessInstance interfaces

・GetData

・SetData

・TerminateProcessInstance

・Subscribe

・Unsubscribe

・GetHistory

ActivityObserver interfaces

・GetData

・SetData

・ProcessInstanceCompleted

・ProcessInstanceTreminated

・Notify

- **Names of the top tag and its elements**

The “XWfM” tag is renamed “WfMessage”.

The “XWfM.head” tag is renamed “WfMessageHeader”.

The “XWfM.body” tag is renamed “WfMessageBody”.

List expression

In the last proposal, the tags of list elements end with " ...List" such as "ValidStateList". We renamed these tags with alignment to Wf-XML such as "ValidStates".

- **Capitalize**

All tags are changed to begin with a capital letter with alignment to Wf-XML.

- **“ResponseRequired” tag**

The “ResponseRequired” tag is added as an element of the “Mission” tag. We believe that the “ResponseRequired” tag is useful especially in asynchronous requests. In our interaction model, if a response of a request is not required, a caller set the “ResponseRequired” tag “No”, then a response phase of the request is omitted

- **Exception format**

The format of exception information was changed with alignment to Wf-XML. But some format is still different from Wf-XML format in order to realize asynchronous batch request.

- **Renaming other tags**

We changed some tags with alignment to Wf-XML. For example, “target” is renamed “Key”. The dot-connected notation is still remained to ensure easy validation with DTD.

4. References

[1] Simple Workflow Access Protocol (SWAP)", Keith Swenson, Internet-Draft, 7 Aug. 1998. http://www.ics.uci.edu/~ietfswap

[2] Workflow Standard – Interoperability Wf-XML binding, The Workflow Management Coalition, WfMC-TC-1023, 1.0, 20 April 1999.

[3] "Workflow Standard – Interoperability Abstract Specification", The Workflow Management Coalition, WfMC-TC-1012, 1.0, 20 Oct. 1996. http://www.wfmc.org/standards/docs/if4-a.pdf

[4] "The Workflow Reference Model", The Workflow Management Coalition, WfMC-TC-1003, 1.1, 29 Nov. 1994. http://www.wfmc.org/standards/docs/rmv1-16.pdf

[5] "Workflow Standard - Interoperability Internet e-mail MIME Binding", The Workflow Management Coalition, WfMC-TC-1018, 1.1, 25 Sep. 1998. http://www.wfmc.org/standards/docs/I4Mime1x.pdf

[6] "Extensible Markup Language (XML)", W3C, REC-xml-19980210, 1.0, 10 Feb. 1998. http://www.w3.org/TR/1998/REC-xml-19980210

[7] "Terminology & Glossary", The Workflow Management Coalition, WfMC-TC-1011, 2.0, June 1996. http://www.wfmc.org/standards/docs/glossary.pdf

[8] "Audit Data Specification", The Workflow Management Coalition, WfMC-TC-1015, 1.1, 22 Sep. 1998. http://www.wfmc.org/standards/docs/if5v11b.pdf