The Workflow Management Coalition Specification

Workflow Management Coalition Workflow Standard - Interoperability Wf-XML Binding

Document Number WFMC-TC-1023
Document Status – Draft 1.0 (Beta Status)

11-January-2000
Version 1.0 Beta

Send comments to : WORKGROUP4@FTPLIST.AIIM.ORG

Workflow Management Coalition
2 Crown Walk
Winchester
SO22 5XE
United Kingdom
Tel: +44 1962 873401
Fax: +44 1962 868111
Email: wfmc@wfmc.org
web: http://www.wfmc.org

First printing January 2000

Table of Content

1 Change History

Version 1.0 Beta – Editor: Effat Peyrovian (peyrovian@mail1.monmouth.army.mil)

- Enhanced naming conventions
- Structured XML body consisting of transport, header, and data.
- Concentrates on a minimum interoperability message set within ProcessDefinition, ProcessInstance, and Observer groups.

Version 1.0 Alpha – Editor: Michael Rossi (mrossi@crusher.jcals.csc.com)

- Wf-XML 1.0 (Alpha Status) was voted on August 1999 based on the initial submission. It defined seven groupings of operations: ProcessDefinition, ProcessInstance, Observer, Activity Observer, Work List, Work Item, and Entry Point.
- Initial version of the specification evolved from Simple Workflow Access Protocol (SWAP) design. It resulted from lessons learned on SWAP prototype to link IBM and U.S. Department of Defense's Joint Computer-aided Acquisition and Life-cycle Support (JCALS) workflow engine, developed by Computer Science Corporation (CSC). CSC and JCALS Program Manager jointly developed and submitted the initial submission of Wf-XML, in April 1999, to WfMC.

2 Introduction

This document represents a specification for an XML language designed to model the data transfer requirements set forth in the Workflow Management Coalition's Interoperability Abstract specification (WFMC-TC-1012) [1]. This language will be used as the basis for concrete implementations of the functionality described in the abstract in order to support the WfMC's Interface 4 (as defined by the workflow reference model [2]).

2.1 Purpose

It is the intention of this specification to describe a language that can be used to achieve the two basic types of interoperability defined in the abstract specification. Specifically, simple chained workflows and nested workflows. It will support these two types of interchange both synchronously and asynchronously. Furthermore, this specification will describe a language that is independent of any particular implementation mechanism, such as programming language, data transport mechanism, platform, hardware, etc. However, because of the fact that HTTP is considered as the most important data transport mechanism for Wf-XML, this specification provides a description on how Wf-XML interchanges are transferred using this protocol.

2.2 Scope

The scope of this specification is equivalent to that defined by the abstract specification of the interoperability standard (WFMC-TC-1012) [1].

2.3 Audience

This document should be reviewed by the Workflow Management Coalition and those interested in its efforts. Once finalized, this document should be utilized by vendors and other implementers of workflow systems concerned with interoperability.

2.4 Document Status

This document is a draft proposal for submission to the WfMC members for approval. It will form the basis for discussions, refinements, and further actions.

2.5 Documentation Convention

In several of the examples provided in this document ellipses (…) is used as a placeholder for other data. In certain contexts, this notation may also imply the optional repetition of previously indicated elements or content. In either case, it should not be interpreted as a literal part of the data stream.

2.6 Background

This specification has been generated based on previous work completed by Workflow Management Coalition, Object Management Group (OMG) and many vendor organizations in an effort to define the functionality required to achieve interoperability among workflow systems. The following documents have provided input to this specification:

- IF4 Abstract specification [1]
- OMG Workflow Management Facility specification [3]
- IF4 Internet e-mail MIME binding specification [4]
- Simple Workflow Access Protocol (SWAP) proposal [5]

2.7 Approach

At a high level, these are the goals of this specification:

- Support chained and nested workflows
- Provide for both synchronous and asynchronous interactions
- Remain implementation independent
- Define a light, easy-to-implement protocol

In order to achieve these goals, it is necessary to focus only on the common aspects of workflow implementations, which implies a specification based on data interchange. It is further necessary to describe this data interchange in an open, standards-based fashion that allows for the definition of a structured, robust and customizable communications format. For these reasons, this specification will utilize the Extensible Markup Language (XML) [6] to define the language with which workflow systems will interoperate.

3 Technical Specification

3.1 Logical Resource Model

It has been determined that the concepts of interoperability among workflow systems can be naturally extended to accomplish interaction among many other types of systems and services. These other systems are deemed “generic services” and can represent any identifiable resource with which an interaction can occur. A generic service may further be viewed as consisting of a number of different resources. These resources may be implemented in any fashion, so long as they are uniquely identifiable and can interact with other resources in a uniform fashion as specified in this document, receiving requests to enact services and sending appropriate responses to the requestors.

Individual interoperable functions are termed “operation”. Each operation may be passed a set of request parameters and return a set of response parameters. Operations are divided into different groups in order to better identify their context. The primary groups of operations required for interoperability are named ProcessDefinition, ProcessInstance and Observer. A resource implements a group of operations by supporting the operations defined to exist within that group. Furthermore, a resource may implement more than one group of operations such as ProcessInstance and Observer group. Conformance to the specification is discussed in the conformance section later in this document

The ProcessDefinition group is the most fundamental group of operations required for the interaction of generic services. It represents the description of a service’s most basic functions, and is the resource from which instances of a service will be created. Since every service to be enacted must be uniquely identifiable by an interoperating service or service requestor, the process definition will provide a resource identifier. When a service is to be enacted, this resource identifier will be used to reference the desired process to be executed.

The ProcessInstance group represents the actual enactment of a given process definition and will have its own resource identifier separate from the definition’s. When a service is to be enacted, a requestor will reference a process definition’s resource identifier and create an instance of that definition. Since a new instance will be created for each enactment, the process definition may be invoked (or instantiated) any number of times simultaneously. However, each process instance will be unique and exist only once. Once created, a process instance may be started and will eventually be completed or terminated.

The Observer group provides a means by which a process instance may communicate its completion or termination. In nested subprocesses, there must be a way for a requestor of a service enactment to determine or be informed when a subprocess completes. The Observer group will provide this information by giving a process instance the resource identifier of the requestor. The following diagram indicates the relationship between each group of operations explained above:

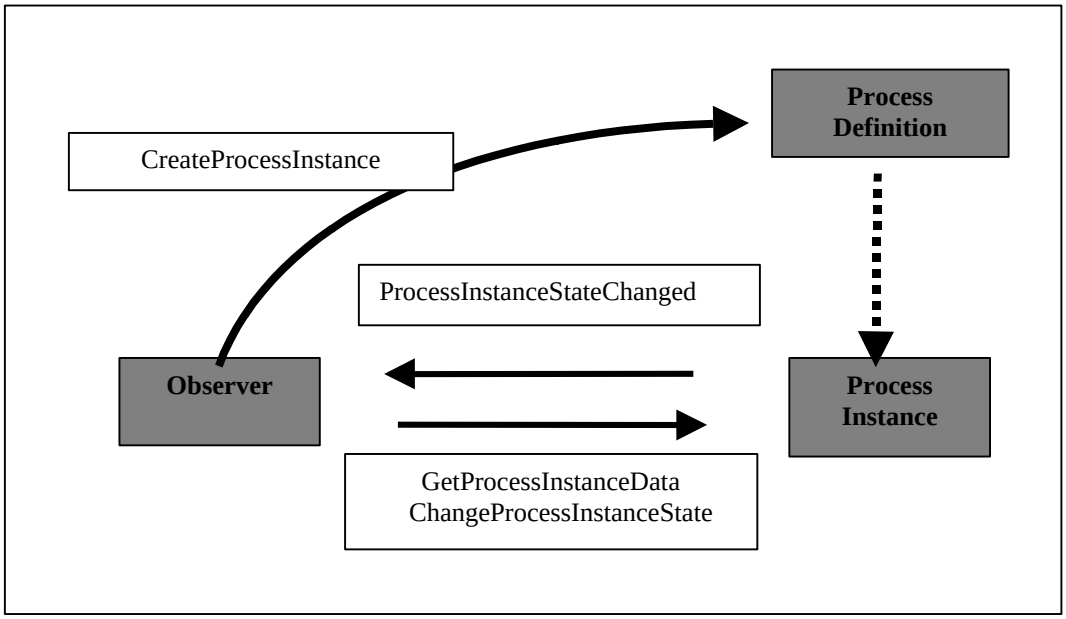

Diagram 1.

3.2 WF-XML Language Definition

Every Wf-XML message is an XML document instance, conforming to the XML 1.0 specification. Therefore, the first string that will appear in each message will be the XML declaration, which appears in the form of a PI as follows: '<?xml version="1.0"?>'. This implies that the supported XML character encoding UTF-8 or UTF-16 will be used. This section will describe each element used within Wf-XML messages and its purpose, also providing examples. For a consolidated table of all defined elements and the Wf-XML DTD, see Section 8.

3.2.1 Overall Message Structure

The following DTD segment defines the top-level structure of a Wf-XML message:

```
<!ELEMENT WfMessage (WfTransport?, WfMessageHeader,  WfMessageBody)>
<!ATTLIST WfMessage Version CDATA #REQUIRED>
```

The root element within a Wf-XML message is named "WfMessage". Each Wf-XML message contains a message header (tag WfMessageHeader), message body (tag WfMessageBody), and an optional section on transport mechanisms (tag WfTransport). The WfTransport section is not currently used in the specification, as our only proposed binding HTTP does not require information for this section. The message body (tag WfMessageBody) is where the operation specific information is placed. Therefore the skeleton of a message will appear as follows:

Example A:

```
<?xml version="1.0"?>
<WfMessage Version="1.0">
        <WfTransport/>
        <WfMessageHeader>
                …..
        </WfMessageHeader>
        <WfMessageBody>
                -----
        </WfMessageBody>
</WfMessage>
```

Each message can be either a request or a response message based on required parameters' values contained in the message header.

3.2.2 Message Transport Mechanism

Transport protocol specific information is described in WfTransprt section of WfMessage. This section is also used for user defined correlation information. (tag CorrelationData).

CorrelationData element is a user defined ID for correlating messages; set in request messages and returned without modification on response messages. An example might be a globally unique ID. (optional)

```
<!ELEMENT WfTransport (CorrelationData?) >
<!ELEMENT CorrelationData (#PCDATA)>
```

3.2.3 Message Header Definition

The message header (WfMessageHeader) contains more general and common tags that are generically useful to all messages. Separation of this information from the message body enables pre-porcessing of workflow messages without having to parse the request-specific information. It is used for a variety of session specific information – such as whether the message is a request or a response, and exception information. The message header must be provided in each message. Request and response message headers differ in the use of the ResponseRequired tag, in a request message and Exception element in a response message.(See Examples in Section 3.2.4)

```
<!ENTITY % OperationName "CreateProcessInstance | GetProcessInstanceData |
      ChangeProcessInstanceState | ProcessInstanceStateChanged">
<!ELEMENT WfMessageHeader ((Request | Response), Key, Language?,
      Operation)>
<!ELEMENT Request (ResponseRequired)>
<!ELEMENT ResponseRequired (#PCDATA)>
<!ELEMENT Response (Exception*)>
<!ELEMENT Exception (MainCode, SubCode?, Type, Subject, Description?)>
<!ELEMENT Key (#PCDATA)>
<!ELEMENT Language (#PCDATA)>
<!ELEMENT Operation (%OperationName)>
```


3.2.3.1 Message Header Elements

Request – indicates that this is a request message

ResponseRequired – This element is defined as numerated type { No | IfError | Yes }

Response – Indicates that this is a response message

Exception – Exception processing block which is used by the workflow engine, to return errors in the response message, which are related to system processing or header information errors to the requesting application. (See Section 3.2.12 Error Handling)

Key – Uniform Resource Identifier (URI) of the resource that is target of a Request or source of a Response

Language- Indicates what language is being used (English, German,…) in a language-specific data such as Subject or Description. This indicator is based on ISO Standards 639. If this element is not used, some default language is chosen. (optional)

Operation – The <*OperationName*> that identifies the information present in the body of the message. The Operation identifiers are provided both in the header and the body of the message to enable efficient pre-processing of Wf-XML messages.

3.2.4 Message Body Definition

The message body provides operation specific data. For a request message, it contains the <*OperationName.Request*> element, which further contains request parameters; for response messages it contains an <*OperationName.Response*> element that holds potential response parameters returned by an operation.

```
<!ENTITY % OperationRequest "CreateProcessInstance.Request |
       GetProcessInstanceData.Request |
       ChangeProcessInstanceState.Request |
       ProcessInstanceStateChanged.Request">

<!ENTITY % OperationResponse "CreateProcessInstance.Response |
       GetProcessInstanceData.Response |
       ChangeProcessInstanceState.Response |
       ProcessInstanceStateChanged.Response">

<!ELEMENT WfMessageBody (%OperationRequest; | %OperationResponse;)>
```

This model would have the following structure for a request message:

Example A:

```
<?xml version="1.0"?>
<WfMessage Version="1.0">
      <WfTransport/>
      <WfMessageHeader>
          <Request>
              <ResponseRequired>Yes</ResponseRequired>
          </Request>
           <Key>http://www.XYZcompany.com/wfprocess</Key>
          <Operation>CreateProcessInstance</Operation>
      </WfMessageHeader>
      <WfMessageBody>
              <CreateProcessInstance.Request>

                      <parameter/>
                      <parameter/>
              </CreateProcessInstance.Request>
      </WfMessageBody>
</WfMessage>
```


This model would have the following structure for a response message with an Exception:

Example B:

```
<?xml version=”1.0”?>
  <WfMessage Version=”1.0”>
     <WfTransport/>
     <WfMessageHeader>
          <Response>
               <Exception>
                    <MainCode>502</MainCode>
                    <Type>F</Type>
                    <Subject> Invalid process definition</Subject>
                    <Description> Cannot create instance</Description>
               </Exception>
          </Response>
          <Key>http://www.XYZcompany.com/wfprocess</Key>
          <Operation>CreateProcessInstance</Operation>
     </WfMessageHeader>
     <WfMessageBody/>
   </WfMessage>
```

Example C indicates the structure for a normal response message:

Example C:

```
<?xml version=”1.0”?>
<WfMessage Version=”1.0”>
     <WfTransport/>
     <WfMessageHeader>
          <Response/>
          <Key>http://www.XYZcompany.com/wfprocess</Key>
          <Operation>CreateProcessInstance</Operation>
     </WfMessageHeader>
     <WfMessageBody>
               <CreateProcessInstance.Response>
                      <parameter/>
                      <parameter/>
               </CreateProcessInstance.Response>
     </WfMessageBody>
</WfMessage>
```

3.2.5 Asynchronous Message Handling

Because the data transport mechanism of any given implementation is unknown to this specification, it is a stated goal of this document to support synchronous and asynchronous messaging. This implies a need to coordinate messages so that an implementation will be able to determine the request for which a response is being received. This will be accomplished by returning a user defined identifying information as CorrelationData within the transport specific section of the message (tag WfTransport).

3.2.6 Security

In general, security considerations are out of the scope of this document due to the fact that they are largely dependent upon transport mechanism. This applies to user identification and authorization, as well as data and functional access control. In many cases while security mechanisms such as SSL, PKI and LDAP may be sufficient for some applications, they may be viewed as insufficient or overkill by others. Therefore, security mechanism between two or more interoperating engines should be addressed in the interoperability contract between them.

3.2.7 Data Types

Instance specific data types may be of the following types:

- Binary
- Boolean
- Integer
- Unsigned
- Float
- Double
- String
- String [the maximum number of characters]
- Date
- URI (RFC 2396)
- XML

These data types are defined by WfMC Interface4 except for String[the maximum number of characters], URI, and XML. In case of any instance specific data types that are not defined in this specification, string data type will be assumed.

The following data types will be enforced for date and time values.

3.2.7.1 Date and Time Values

All date and time values shall be represented as Greenwich Mean Time (GMT) based timestamps to ensure interoperability between workflow engines that may not be in the same time zone. The DateTime format shall be represented as:

YYYY-MM-DD hh:mm:ssZ

where:

YYYY is the year in the Georgian calendar

MM is the month of the year (range 01 - 12)

DD is the day of the month (range 01 - 31)

hh is the hours of the day (range 00 - 24)

mm is the minutes of the hour (range 00 - 59)

ss is the seconds of the minute (range 00 - 59)

Z is the symbol that indicates Coordinated Universal Time (UTC) or GMT

A space separates the date and time string representations. A time of midnight may be expressed as 00:00:00 or 24:00:00.

All dates and times should be represented to the users of the system to meet their individual implementation requirements. This means, if all date/times are to be represented as "user" local times, they can be because the UTC time variable allows the conversion to local time, regardless of location in the world.

If a particular engine requires the dates/times to be represented as times local to their actions as opposed to local to the current user, then the engines will need to prepare additional context data fields to represent that originating timezone to enable a conversion to be performed locally within their system.

3.2.8 Representation of Process Context and Result Data

When a Process Definition is instantiated, the context of the resulting Process Instance is initialized; the ContextData element is used to represent the input parameters of a process known as context data. The ResultData element is used to represent the output parameters of a Process Instance. The default content model for ContextData and ResultData is specified as "ANY". The Wf-XML specification leaves it up to the parties involved in workflow interoperation to replace this default content model by their specific DTD definition of Context/ResultData.

```
<!ELEMENT ContextData  ANY>
<!ELEMENT ResultData  ANY>
```

The following example illustrates one possible encoding of a complex data structure as a nested XML document, representing nested data structures by nested XML tags and elemental data items by XML tags with content model CDATA:

```
<!ELEMENT ContextData (Vehicle, Furniture,…)>
<!ELEMENT Vehicle (VehicleType, Specification)>
<!ELEMENT Specification (Manufacturer, Model)>
<!ELEMENT Furniture (…)>
```

This may be refined by realization of the Wf-XML standard to support validation of the particular encoding of complex data. The example above could be represented as follows:

***Example A*:**

```
<?xml version="1.0"?>
<WfMessage Version="1.0">
    <WfMessageHeader>
        ……
    </WfMessageHeader>
    <WfMessageBody>
        <CreateProcessInstance.Request>
            <Key>xxx</Key>
            <ObserverKey>xxx</ObserverKey>
            <ContextData>
                <Vehicle>
                    <VehicleType>Car</VehicleType>
                    <Specification>

                    <Manufacturer>Mercedes</Manufacturer>
                    <Model>450SL</Model>
                    </Specification>
                </Vehicle>
                <Furniture>chair</Furniture>
                ……
            </ContextData>
        </CreateProcessInstance.Request>
    </WfMessageBody>
</WfMessage>
```

3.2.9 Wf-XML Namespace Definition

XML provides namespace declaration [11] for cases where vendors wish to support markup defined in another DTD, but this is not an approved standard yet. Namespace declaration for Wf-XML is intended to be defined as soon as this standard is approved. It may appear as:

```
<wf:WfMessage xmlns:wf="http://www.wfmc.org/standards/docs/Wf-XML">.
```

Using the above declaration, applications will be able to interpret elements defined elsewhere in order to achieve higher levels of interoperability without degrading conformance to this specification.

3.2.10 Process Status

The WfMC has defined a standard set of valid process instance states. States are organized into several levels of granularity. An implementation may choose to support states on any level of granularity, omit states or add additional states to the optional states defined below.

```
<!ENTITY % vstates "open.notrunning | open.notrunning.suspended |
       open.running | closed.completed | closed.abnormalCompleted
       |closed.abnormalCompleted.terminated |
       closed.abnormalCompleted.aborted">
<!ELEMENT open.notrunning (EMPTY)>
<!ELEMENT open.notrunning.suspended (EMPTY)>
<!ELEMENT open.running (EMPTY)>
<!ELEMENT closed.completed (EMPTY)>
<!ELEMENT closed.abnormalCompleted (EMPLTY)>
<!ELEMENT closed.abnormalCompleted.terminated (EMPTY)>
<!ELEMENT closed.abnormalCompleted.aborted (EMPTY)>
```

open.notrunning – A resource is active, but not running.

open.notrunning.suspended – A resource is in this state when it has initiated its participation in the enactment of a work process, but has been temporarily suspended. At this point, no resources contained within it may be started. (optional)

open.running – A resource is in this state when it is performing its part in the normal execution of a work process.

closed.completed – A resource is in this state when it has finished its task in the overall work process. All resources contained within it are assumed to be complete at this point.

closed.abnormalCompleted – A resource is in this state when it has completed abnormally. At this point the results for the completed tasks are returned

closed.abnormalCompleted.terminated – A resource is in this state when it has been terminated by the requesting resource before it completed its work process. At this point, all resources contained within it are assumed to be either completed or terminated. (optional)

closed.abnormalCompleted.aborted – A resource is in this state when the execution of its process has been abnormally ended before it completed its work process. At this point, no assumptions are made about the state of the resources contained within it. (optional)

3.2.11 Error Handling

Should any exception occur during the execution of a Wf-XML operation, that exception would have to be returned to the caller. Various types of exceptions can be anticipated, including temporary and fatal error types. Therefore, an “Exception” element has been defined to carry this information.

```
<!ELEMENT Exception (MainCode,  SubCode?, Type, Subject, Description?)>
<!ELEMENT MainCode (#PCDATA)>
<!ELEMENT SubCode (#PCDATA)>
<!ELEMENT Type (#PCDATA)>
<!ELEMENT Subject (#PCDATA)>
<!ELEMENT Description (#PCDATA)>
```


This exception element will be returned with the results from all operations, if an exception occurs. The exception information will contain the following tags:

3.2.11.1 Elements of an Exception

MainCode - This is a three digit positive integer defined in the operation specification. It is operation-specific and gives some indication on what went wrong. Programs can use that code to calculate what to do when that exception occurred. This specification defines main codes for all operations.

SubCode - This is also a three digit positive integer. It details the main code, e.g., when a main code says "Invalid Key", the SubCode could say that it is the format that is wrong. This is where a vendor would specify errors, which are specific to their processing. (optional)

Type - The type of the error occurred. It can either be "F" for fatal error or "T" for temporary error.

Subject - This is a one-line text description of what went wrong.

Description - A several-line text description of what went wrong, which details the Subject. (optional)

These elements are used to structure an exception in such a way as to enable interpretation of application-specific error codes and translation of error messages independent of any context-specific information. An example is shown below.

Example A:

```
<?xml version="1.0"?>
<WfMessage Version="1.0">
     <WfMessageHeader>
        <Response>
              <Exception>
                   <MainCode>502</MainCode>
                   <Type>F</Type>
                   <Subject> Invalid process definition</Subject>
                   <Description> Cannot create instance</Description>
              </Exception>
        </Response>
     </WfMessageHeader>
     <WfMessageBody/>
</WfMessage>
```

3.2.11.2 Exception Codes

The following are a list of recommended MainCode three digit integer values, which can be used to report exceptions. Each MainCode category is listed below, with additional error information provided for that category. These are the exception codes that are used in the operations' specifications.

WfMessageHeader-specific	**100 Series**
These exceptions deal with missing or invalid parameters in the header.	
WF_PARSING_ERROR	100
WF_TAG_MISSING	101
WF_INVALID_VERSION	102
WF_INVALID_RESPONSE_REQUIRED_VALUE	103
WF_INVALID_KEY	104
WF_INVALID_OPERATION_SPECIFICATION	105

Data **200 Series**

These exceptions deal with incorrect context or result data

WF_INVALID_KEY 200

WF_INVALID_CONTEXT_DATA 201

WF_INVALID_RESULT_DATA 202

Authorization **300 Series**

A user may not be authorized to carry out this operation on a particular resource, e.g., may not create a process instance for that process definition.

WF_NO_AUTHORIZATION 300

Operation **400 Series**

The operation can not be accomplished because of some temporary internal error in the workflow engine. This error may occur even when the input data is syntactically correct and authorization is permitted.

WF_OPERATION_FAILED 400

Resource Access **500 Series**

A valid Key has been used, however this operation cannot be currently invoked on the specified resource.

WF_NO_ACCESS_TO_RESOURCE 500

WF_INVALID_PROCESS_DEFINITION 502

Operation-specific **600 Series**

These are the more operation specific exceptions. Typically, they are only used in a few operations, possibly a single one.

WF_INVALID_STATE_TRANSITION 600

WF_INVALID_OBSERVER_FOR_RESOURCE 601

Extensibility **800 Series**

An additional exception main code is provided to allow implementations of the WF-XML specification to return additional exceptions

WF_OTHER 800

3.3 Operation Definitions

The scope of the specification is limited to the operations shown in the following table. In brief, this section will discuss the collections of operations used for ProcessDefinition, ProcessInstance and Observer, as well as each of the operations in detail.

	Process Definition	**Process Instance**	**Observer**
CreateProcessInstance	X		
GetProcessInstanceData		X	
ChangeProcessInstanceState		X	
ProcessInstanceStateChanged			X

3.3.1 Process Definition Operations

This group of operations is used to create process instances and determine general information about process definitions. Currently it contains the operation CreateProcessInstance.

3.3.1.1 CreateProcessInstance

CreateProcessInstance is used to instantiate a known process definition. The instance will be created with context-specific data set according to the input data, and started.

```
<!ELEMENT Key (#PCDATA)>
<!ELEMENT ObserverKey (#PCDATA)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Subject (#PCDATA)>
<!ELEMENT Description (#PCDATA)>
<!ELEMENT ContextData ANY>
<!ELEMENT StartImmediately (CDATA)>
<!ELEMENT CreateProcessInstance.Request (Key, ObserverKey?, Name?,
      Subject?, Description?, ContextData,  StartImmediately)>
<!ELEMENT ProcessInstanceKey (#PCDATA)>
<!ELEMENT CreateProcessInstance.Response (ProcessInstanceKey)>
```

Request Parameters:

Key – URI of the process definition resource that is to be used to create this instance.

ObserverKey – URI of the resource that is to be the observer of the instance that is created by this operation. This observer resource (if it is specified) is to be notified of state changes to the instance, most notably the completion of the instance. (optional.)

Name – A human readable name requested to be assigned to the newly created instance. If this name is not unique, it may be modified to make it unique, or changed entirely. Therefore, the use of this name cannot be guaranteed. (optional)

Subject – A short description of the purpose of the new process instance. (optional)

Description – A longer description of the purpose of the newly created process instance. (optional)

ContextData – Context-specific data required to create this process instance. This information will be encoded according to the data encoding formalism agreed in the interoperability contract (see section on Process Context and Result Data above).

StartImmediately- A Boolean value ("Yes" or "No"), indicating whether the newly created instance should be started immediately upon creation. The value of this parameter is currently always "Yes".

***Example A*:**

```
<?xml version="1.0"?>
<WfMessage Version=”1.0”>
      <WfMessageHeader>
          <Request>
              <ResponseRequired>Yes</ResponseRequired>
          </Request>
          <Key>http://www.XYZcompany.com/wfprocess</Key>
          <Operation>CreateProcessInstance</Operation>
      </WfMessageHeader>
     <WfMessageBody>
          <CreateProcessInstance.Request>
              <Key>http://www.XYZcompany.com/wfprocess</Key>
              <ObserverKey>http://www.mycompany.com/mywork</ObserverKey>
              <ContextData>
                  <Vehicle>
                       <VehicleType>Car</VehicleType>
                       <Specification>
                            <Manufacturer>Mercedes</Manufacturer>
                            <Model>450SL</Model>
                       </Specification>
                  </Vehicle>
                  <Furniture>chair</Furniture>
              </ContextData>
              <StartImmediately>Yes</StartImmediately>
          </CreateProcessInstance.Request>
      </WfMessageBody>
</WfMessage>
```

Response Parameters:

ProcessInstanceKey – URI of the newly created process instance.

Exceptions:

The following exceptions are supported for this operation:

WF_INVALID_KEY

WF_INVALID_CONTEXT_DATA

WF_OPERATION_FAILED

WF_NO_AUTHORIZATION

***Example B*:**

```
<?xml version="1.0"?>
<WfMessage Version=”1.0”>
      <WfMessageHeader>
          ……..
      </WfMessageHeader>
      <WfMessageBody>
          <CreateProcessInstance.Response>
              <ProcessInstanceKey>http://www.XYZcompany.com/wfprocess/myprocess
              </ProcessInstanceKey>
          </CreateProcessInstance.Response>
     </WfMessageBody>
</WfMessage>
```


3.3.2 Process Instance Operations

This group of operations is used to communicate with a particular instance of a process definition (or enactment of a service), acquiring information about the instance and controlling it. Since a given instance may continue to execute for any amount of time, operations may be called on an instance while it is executing. These operations may obtain status information or obtain early results (although the results of a process instance are not final until the instance has been completed). This group contains the operations GetProcessInstanceData and ChangeProcessInstanceState.

3.3.2.1 GetProcessInstanceData

This operation is used to retrieve the values of all properties defined for the given process instance resource.

```
<!ENTITY %ProcessInstanceData “Name | Subject | Description | State |
      ValidStates | ObserverKey | ResultData | ProcessDefinitionKey |
      Priority |  LastModified”>

<!ENTITY %vstates “open.notrunning|open.notrunning.suspended|
      open.running|closed.completed|closed.abnormalCompleted.terminated|c
      losed.abnormalCompleted.aborted”>

<!ELEMENT ResultDataAttributes (%ProcessInstanceData;)*>

<!ELEMENT Key (#PCDATA)>

<!ELEMENT Name (#PCDATA)>

<!ELEMENT Subject (#PCDATA)>

<!ELEMENT Description (#PCDATA)>

<!ELEMENT State (#PCDATA)>

<!ELEMENT ValidStates (%vstates;)+>

<!ELEMENT ObserverKey (#PCDATA)>

<!ELEMENT ProcessDefinitionKey (#PCDATA)>

<!ELEMENT Priority (#PCDATA)>

<!ELEMENT LastModified (#PCDATA)>

<!ELEMENT GetProcessInstanceData.Request (Key, ResultDataAttributes?)>

<!ELEMENT GetProcessInstanceData.Response (%ProcessInstanceData;)+>
```

Request Parameters:

Key – URI of the ProcessInstance resource whose properties are to be returned.

ResultDataAttributes – if specified, this parameter contains a list of results to be returned where this list can be all of the results or a subset of all results. If empty or not specified, all results are returned. (optional)

The following example requests all data elements of a particular ProcessInstance:

Example A:

```
<?xml version="1.0"?>
<WfMessage Version=”1.0”>
    <WfMessageHeader>
        ……..
    </WfMessageHeader>
    <WfMessageBody>
       <GetProcessInstanceData.Request>
          <Key>http://www.XYZcompany.com/wfprocess/myprocess</Key>
      </GetProcessInstanceData.Request>
     </WfMessageBody>
</WfMessage>
```

The following example requests only the Name and Priority data elements of a particular Process Instance:

***Example B*:**

```
<?xml version="1.0"?>
<WfMessage Version=”1.0”>
     <WfMessageHeader>
                …
     </WfMessageHeader>
     <WfMessageBody>
           <GetProcessInstanceData.Request>
              <Key>http://www.XYZcompany.com/wfprocess/myprocess</Key>
              <ResultDataAttributes>
                        <Name/>
                        <Priority/>
              </ResultDataAttributes>
            </GetProcessInstanceData.Request>
     </WfMessageBody>
</WfMessage>
```

Response Parameters:

Name – A human readable identifier of the resource. This name may be nothing more than a number.

Subject – A short description of this process instance.

Description – A longer description of this process instance resource.

State – The current status of this resource.

ValidStates – A list of state values allowed by this resource. This is the list of states, which the current instance can transition to.

ProcessDefinitionKey – URI of the process definition resource from which this instance was created.

ObserverKey – URI of the registered observer of this process instance, if it exists.

ResultData – Context-specific data that represents the current result values. This information will be encoded as described in the section Process Context and Result Data above. If result data are not available (yet), the ResultData element is returned empty.

Priority – An indication of the relative importance of this process instance. This value will be an integer ranging from 1 to 5, 1 being the highest priority. The default value is 3.

LastModified – The date of the last modification of this instance, if available.

Exceptions:

The following Exceptions are supported for this operation:

WF_INVALID_KEY

WF_NO_ACCESS_TO_RESOURCE

WF_NO_AUTHORIZATION

The following is an example for a response message on a GetProcessInstanceData operation:

Example C:

```
<?xml version="1.0"?>
<WfMessage Version=”1.0”>
    <WfMessageHeader>
       ……..
    </WfMessageHeader>
    <WfMessageBody>
          <GetProcessInstanceData.Response>
              <Name>xxx</Name>
              <Subject>xxx</Subject>
              <ResultData>
                    <Vehicle>
                          <VehicleType>Car</VehicleType>
                          <Specification>
                                <Manufacturer>Mercedes</Manufacturer>
                                <Model>450SL</Model>
                          </Specification>
                    </Vehicle>
                    <Furniture>chair</Furniture>
              </ResultData>
          </GetProcessInstanceData.Response>
    </WfMessageBody>
</WfMessage>
```


3.3.2.2 ChangeProcessInstanceState

This operation is used to modify the process instance state; for example from open.running. to open.notrunning.suspended.

```
<!ELEMENT ChangeProcessInstanceState.Request ( Key, State )>
<!ELEMENT Key (#PCDATA)>
<!ELEMENT State (#PCDATA)>
<!ELEMENT ChangeProcessInstanceState.Response ( State )>
```

Request Parameters:

Key – URI of the process instance resource that is to be changed.

State – The State, which the process instance should transition to.

Example A:

```
<?xml version="1.0"?>
<WfMessage Version=”1.0”>
    <WfMessageHeader>
        …….
    </WfMessageHeader>
   <WfMessageBody>
      <ChangeProcessInstanceState.Request>
         <Key>http://www.XYZcompany.com/wfprocess/myprocess</Key>
         <State>open.notrunning.suspended</State>
     </ChangeProcessInstanceState.Request>
    </WfMessageBody>
</WfMessage>
```

Response Parameters:

State – The State resulting from the operation

Exceptions:

Exceptions which are supported for this operation:

WF_INVALID_KEY

WF_NO_AUTHORIZATION

WF_INVALID_STATE_TRANSITION

Example B:

```
<?xml version="1.0"?>
<WfMessage Version=”1.0”>
    <WfMessageHeader>
        ……
    </WfMessageHeader>
    <WfMessageBody>
        <ChangeProcessInstanceState.Response>
            <State>open.notrunning.suspended</State>
        </ChangeProcessInstanceState.Response>
    </WfMessageBody>
</WfMessage>
```


3.3.3 Observer Operations

This group of operations allows requesters of work, or other resources, to be notified upon status changes of a process instance. Once an instance's registered observer has been notified of status changes of a process instance, further actions taken by the resource responsible for that instance will depend on the new state of the process. Currently closed.completed and closed.abnormalCompleted status changes are notified. This group contains the operation ProcessInstanceStateChanged.

3.3.3.1 ProcessInstanceStateChanged

ProcessInstanceStateChanged is currently the only operation defined in the Observer group. This operation is used to support both closed.completed and closed.abnormalCompleted state changes.

```
<!ELEMENT ProcessInstanceStateChanged.Request (Key, ProcessInstanceKey,
      State, ResultData?,  LastModified?)>
<!ELEMENT Key (#PCDATA)>
<!ELEMENT ProcessInstanceKey (#PCDATA)>
<!ELEMENT State (#PCDATA)>
<!ELEMENT ResultData ANY>
<!ELEMENT LastModified (#PCDATA)>
<!ELEMENT ProcessInstanceStateChanged.Response EMPTY>
```

Request Parameters:

Key – URI of the observer resource that is to be notified when status changes occur.

ProcessInstanceKey – URI of the process instance resource that has been changed.

State – The new status of this resource.

ResultData – Context-specific data that represents the current result values. This information will be encoded as described in the section on Process Context and Result Data above. If result data are not available (yet), the ResultData element is returned empty.(optional)

LastModified – The date of the last modification of this instance. (optional)

Example A:

```
<?xml version=”1.0”?>
<WfMessage Version=”1.0”>
     <WfMessageHeader>
                …
     </WfMessageHeader>
     <WfMessageBody>
           <ProcessInstanceStateChanged.Request>
              <Key>http://www.mycompany.com/mywork</Key>
              <ProcessInstanceKey>http://www.XYZcompany.com/wfprocess/myprocess
              </ProcessInstanceKey>
              <ResultData>
                   <Vehicle>
                        <VehicleType>Car</VehicleType>
                        <Specification>
                             <Manufacturer>Mercedes</Manufacturer>
                             <Model>450SL</Model>
                        </Specification>
                   </Vehicle>
                   <Furniture>chair</Furniture>
              </ResultData>
           </ProcessInstanceStateChanged.Request>
     </WfMessageBody>
</WfMessage>
```

Response Parameters:

None

Exceptions:

Exceptions which are supported for this operation:

WF_INVALID_KEY

WF_INVALID_RESULT_DATA

WF_OPERATION_FAILED

WF_NO_AUTHORIZATION

WF_INVALID_OBSERVER_FOR_THAT_RESOURCE

Example B:

```
<?xml version="1.0"?>
<WfMessage Version=”1.0”>
     <WfMessageHeader>
          <Response/>
          <Key>http://www.mycompany.com/mywork</Key>
          <Operation>ProcessInstantStateChanged</Operation>
     </WfMessageHeader>
     <WfMessageBody>
        <ProcessInstanceStateChanged.Response/>
      </WfMessageBody>
</WfMessage>
```


4 Relationship to other Standards

4.1 OMG Workflow Management Facility Standard (jointFlow)

The following discusses the mapping between the interfaces defined in the OMG Workflow Management Facility standard and the Wf-XML resources and operations. The Wf-XML standard uses the basic object model defined in the OMG Workflow Management Facility Standard specification. It supports a subset of the entities defined in this object model and it also combines operations that where separated in the OMG Workflow Management Facility interfaces into a single operation.

The OMG Workflow Management Facility WfProcessMgr interface corresponds to the Wf-XML ProcessDefinition resource type. The Wf-XML CreateProcessInstance operation combines the OMG Workflow Management Facility create_process operation on WfProcessMgr, the start and the set_context operation on WfProcess.

The OMG Workflow Management Facility WfProcess interface corresponds to the Wf-XML ProcessInstance resource type; the OMG Workflow Management Facility operation change_state on WfProcess (inherited from WfExecutionElement) corresponds to the Wf-XML operation ChangeProcessInstanceState. The WfProcess operation get_result in combination with the 'getter' functions for state variables on WfProcess correspond to the Wf-XML operation GetProcessInstanceData.

The OMG Workflow Management Facility WfRequester interface corresponds to the Wf-XML WfObserver resource type.

The OMG Workflow Management Facility specification defines a couple of interfaces that are not represented by Wf-XML: WfActivity, WfResource, WfAssignment.

5 Implementation Issues

5.1 Interoperability Contract

It is recognized that there may be additional requirements outside the scope of this specification that vendors may wish to fulfill in order to achieve basic interoperability. For this reason, it is recommended that an interoperability contract be established among vendors participating in interoperable workflows. This contract will clearly define each vendor's expectations and requirements in all areas that may impede interoperabilty. A list of topics to be included in the interoperability contract is provided here as an example, but this list should by no means be considered complete. Each interoperating vendor must ensure that all factors impacting their implementation are addressed completely.

Some of the topics that should be described in the interoperability contract are:

- Feature/Function Requirements – application-specific data required to be transferred in order to utilize basic or extended functionality.
- Data Types – any specific data types used by a vendor in context data or result data that an interoperating vendor must be aware of.
- Data Constraints – field lengths, allowable characters, character set encoding, overall message size, etc.
- Error Handling – application-specific error handling requirements such as SubCodes, descriptions, required actions, etc.
- Protocol Limitations – required header data, timeout values, buffer sizes, etc.
- Security Considerations – encryption methods, user verification, firewall configuration requirements, etc.

6 Conformance

For many product vendors and purchasers of workflow systems it will be highly desirable to have a means of ascertaining a system's conformance to this specification. This section outlines several factors involved in doing so. The most critical factor in determining conformance lies in a vendor's ability to implement the functionality described by the specification., and it is exactly this topic which is addressed in section 7 (Evaluation Criteria) of the Interoperability Abstract Specification [1]. The approaches described there concerning conformance statements and capabilities matrices are equally applicable to this specification and should be used to determine functional conformance to it. In addition to the topics discussed in the abstract, other XML-related factors are described here that may also impact conformance.

6.1 Validity vs. Well-Formedness

All XML document instances (in this case Wf-XML messages) may be in one of several states of "validity". They may be 'invalid' due to some syntactical error in their markup. They may be 'well-formed', meaning they are syntactically correct with regard to the XML specification. And finally, they may be 'valid', meaning they are not only syntactically correct (per spec), but are also fully compliant with a DTD. The XML specification imposes no requirement on a document instance to be valid, only well-formed. Therefore, if a document instance does not reference a DTD or doesn't fully comply with a referenced DTD, the data contained within it can still be processed successfully.

For this reason, this specification does not mandate validity of all document instances, rather it only requires that all Wf-XML messages be well-formed. However, there is an added measure of data integrity provided by validating a document instance via an XML parser. If an application should desire to do so, the DTD provided with this specification can be used for this purpose. Bear in mind though, that there will remain certain semantic constraints of this data that cannot currently be modeled in a DTD. These semantics will still have to be understood and handled by the implementing application.

6.2 Conformance vs. Extensibility

Another factor that can potentially impact conformance is extensibility. This topic has been addressed earlier in this document with regard to the provisions made within the constraints of the Wf-XML language. However, it is recognized that it may be desirable to extend an application's data exchange requirements beyond these limits. In cases where interoperating vendors have agreed upon functionality and message formats outside the definitions of this specification, or have simply utilized undefined markup that is to be ignored by their interoperating partners, they should be able to do so while maintaining conformance.

This specification only requires well-formed data. Therefore, interoperating vendors may exchange any data they wish so long as that data meets the syntactic requirements of the XML specification. Although it would obviously be best from a functional perspective if the vendors were able to agree upon this data's markup, if they cannot the recipient of the unknown markup should simply ignore it and return it to the sender upon request. Conformance will not be degraded unless the vendor fails to comply with the markup declarations provided here.

7 Transport Layer Bindings

Wf-XML messages for workflow interoperability can be communicated between interoperating workflow systems through different transport mechanisms. This section provides a specification for a Hyper Text Transfer Protocol (HTTP) binding , which is considered the most important transport mechanism, utilized to communicate Wf-XML request and response messages between workflow engines (as specified before, the communicating entities need not necessarily be workflow engines - in the same way as the processes could more generally be services; but for ease of explanation we always refer to this main usage).

7.1 HTTP

For HTTP, the communicating workflow engines are considered HTTP servers (Workflow engines may communicate directly via HTTP, or they may be combined with another program to enable them to send and receive HTTP methods.). Wf-XML messages for all the operations specified earlier are integrated as input data or output data with respect to HTTP interactions. In more detail:

An operation is encoded in the HTTP-method POST. POST is directed to some URI (Uniform Resource Identifier) and may have MIME (Multipurpose Internet Mail Extension) body parts for input and output. For Wf-XML, exactly one MIME body part is used for input and exactly one MIME body part is used for output:

- *The URI, to which a POST method is directed, is the key of the resource. This is vendor-specific and must either be known beforehand (e.g., in case of a process definition key) or it is known as the result of a response parameter returned by a previous operation (e.g., "ProcessInstanceKey"). Thus all the keys used in Wf-XML data need to be URIs that can be handled via HTTP, i.e., they must be of the form "http://..".*
- *The Wf-XML request is the one MIME body part for input.*
- *The Wf-XML response is the one MIME body part for output.*
- *Both of these body parts use the MIME content type "Content-type: text/xml" in the HTTP-method header, and the Content-length is set according to the length of the Wf-XML request or the response respectively.*
- *For authentication, the usual HTTP mechanisms are used. This includes usage of the respective HTTP header fields.*

8 Consolidated Tag Table and Document Type Definition (DTD)

This appendix provides a listing of all the XML elements defined for use in Wf-XML messages. In addition, the WF_XML DTD is provided for the purposes of implementation reference and optional data validation by an XML processor. The following lists indicate each tag name based on its purpose within a message.

8.1 General Message Elements:

- WfMessage
- WfMessageHeader
- WfTransport
- WfMessageBody
- Response
- Request
- ResponseRequired
- Key
- Language
- Exception
- MainCode
- SubCode
- Type
- Subject
- Description

8.2 Status Elements:

- open.notrunning
- open.notrunning.suspended
- open.running
- closed.completed
- closed.abnormalCompleted
- closed.abnormalCompleted.terminated
- closed.abnormalCompleted.aborted

8.3 Operation Name Elements:

- CreateProcessDefinition
- GetProcessInstanceData
- ChangeProcessInstanceState
- ProcessInstanceStateChanged

8.4 Operation Request Elements:

- CreateProcessDefinition.Request
- GetProcessInstanceData.Request
- ChangeProcessInstanceState.Request
- ProcessInstanceStateChaned.Request

8.5 Operation Response Elements:

- CreateProcessInstance.Response
- GetProcessInstanceData.Response
- ChangeProcessInstanceStateResponse
- ProcessInstanceStateChanged.Response

8.6 Request Parameter Elements:

- Key
- ObserverKey
- ProcessInstanceKey
- ResultDataAttributes
- Name
- Subject
- Description
- ContextData
- State
- ValidStates
- Priority
- LastModified
- StartImmediately
- ResultData

8.7 Response Parameter Elements:

- Name
- Key
- Subject
- Description
- State
- ValidStates
- ProcessDefinitionKey

- ObserverKey
- Exception
- ResultData
- Priority
- LastModified
- ProcessInstanceKey

8.8 Document Type Definition (DTD)

The DTD for Wf-XML messages is provided below. This DTD is designed with the intention of being simple and easy to implement, while supporting a robust and flexible structure.

```
<!--Copyright 1999 The HR-XML Consortium -->
<!-- Resume Content Model -->
<!-- ~~~~~~~~~~~~~~~~~~~ Entity Declarations ~~~~~~~~~~~~~~~~~~~ -->
<!ENTITY % OperationName "CreateProcessInstance | GetProcessInstanceData | ChangeProcessInstanceState
        | ProcessInstanceStateChanged">
<!ENTITY % OperationRequest "CreateProcessInstance.Request | GetProcessInstanceData.Request |
        ChangeProcessInstanceState.Request | ProcessInstanceStateChanged.Request">
<!ENTITY % OperationResponse "CreateProcessInstance.Response | GetProcessInstanceData.Response |
        ChangeProcessInstanceState.Response | ProcessInstanceStateChanged.Response">
<!ENTITY % ProcessInstanceData "Name | Subject | Description | State | ValidStates | ObserverKey |
        ResultData | ProcessDefinitionKey | Priority | LastModified">
<!ENTITY % vstates "open.notrunning | open.notrunning.suspended | open.running | closed.completed |
        closed.abnormalCompleted | closed.abnormalCompleted.terminated |
        closed.abnormalCompleted.aborted">


<!-- ~~~~~~~~~~~~~~~~~~~ Element Declarations ~~~~~~~~~~~~~~~~~~~ -->
<!ELEMENT WfMessage (WfTransport?, WfMessageHeader, WfMessageBody)>
<!ATTLIST WfMessage version CDATA #REQUIRED>


<!-- ~~~~~~~~~~~~~~~~~ WfTransport ~~~~~~~~~~~~~~~~~ -->
<!ELEMENT WfTransport (CorrelationData?)>
<!ELEMENT CorrelationData (#PCDATA)>


<!-- ~~~~~~~~~~~~~~~ WfMessageHeader ~~~~~~~~~~~~~~~~ -->
<!ELEMENT WfMessageHeader ((Request | Response), Key, Language?, Operation)>
<!ELEMENT Request (ResponseRequired)>
<!ELEMENT ResponseRequired (#PCDATA)>
<!ELEMENT Response (Exception)*>
<!ELEMENT Exception (MainCode, SubCode?, Type, Subject, Description?)>
<!ELEMENT MainCode (#PCDATA)>
```

```
<!ELEMENT SubCode (#PCDATA)>
<!ELEMENT Type (#PCDATA)>
<!ELEMENT Subject (#PCDATA)>
<!ELEMENT Description (#PCDATA)>
<!ELEMENT Key (#PCDATA)>
<!ELEMENT Language (#PCDATA)>
<!ELEMENT Operation (%OperationName;)>
<!ELEMENT CreateProcessInstance EMPTY>
<!ELEMENT GetProcessInstanceData EMPTY>
<!ELEMENT ChangeProcessInstanceState EMPTY>
<!ELEMENT ProcessInstanceStateChanged EMPTY>


<!-- ~~~~~~~~~~~~~~~ WfMessageBody ~~~~~~~~~~~~~~~~ -->
<!ELEMENT WfMessageBody (%OperationRequest; | %OperationResponse;)>
<!ELEMENT CreateProcessInstance.Request (Key, ObserverKey?, Name?, Subject?, Description?,
        ContextData, StartImmediately)>
<!ELEMENT ObserverKey (#PCDATA)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT ContextData ANY>
<!ELEMENT StartImmediately (#PCDATA)>
<!ELEMENT GetProcessInstanceData.Request (Key, ResultDataAttributes?)>
<!ELEMENT ResultDataAttributes (%ProcessInstanceData;)*>
<!ELEMENT State (#PCDATA)>
<!ELEMENT ValidStates (%vstates;)+>
<!ELEMENT open.notrunning EMPTY>
<!ELEMENT open.notrunning.suspended EMPTY>
<!ELEMENT open.running EMPTY>
<!ELEMENT closed.completed EMPTY>
<!ELEMENT closed.abnormalCompleted EMPTY>
<!ELEMENT closed.abnormalCompleted.terminated EMPTY>
<!ELEMENT closed.abnormalCompleted.aborted EMPTY>
<!ELEMENT ResultData ANY>
<!ELEMENT ProcessDefinitionKey (#PCDATA)>
<!ELEMENT Priority (#PCDATA)>
<!ELEMENT LastModified (#PCDATA)>
<!ELEMENT ChangeProcessInstanceState.Request (Key, State)>
<!ELEMENT ProcessInstanceStateChanged.Request (Key, ProcessInstanceKey, State, ResultData?,
        LastModified?)>
<!ELEMENT ProcessInstanceKey (#PCDATA)>
<!ELEMENT CreateProcessInstance.Response (ProcessInstanceKey)>
```



```
<!ELEMENT GetProcessInstanceData.Response (%ProcessInstanceData;)+>
<!ELEMENT ChangeProcessInstanceState.Response (State)>
<!ELEMENT ProcessInstanceStateChanged.Response EMPTY>
```


Appendix A -- Terminology

Throughout this document various terms, acronyms and abbreviations are used that may have ambiguous or conflicting definitions for those that have been exposed to similar terminology in other industries. In order to make certain that these terms are properly understood several key terms and definitions are provided here. In large part, the terms used herein and their meanings are as stated in the Workflow Management Coalition Glossary (WfMC000) [9].

- DTD – Document Type Definition.
- PI - Processing Instructions
- Element – A Component of a document consisting of markup and the text contained within the markup.
- Root Element – The outermost element of a document instance, such that the element does not appear in the content of any other element in the instance.
- Attribute - Additional items that are added to elements to provide clarity to the element.
- Entity – A unit of storage in which the contents of storage unit are associated with a name.
- Tag – Instructions enclosed within angle brackets placed at the beginning and end of each document item.
- CDATA – Non-parsable character data.
- Document Instance – An instance of a document type (or class of documents).

Appendix B -- Additional Operations Definitions (Non-Normative)

The following operations are not within the scope of the initial specification but are reserved for future use. This information will be retained for review and comment after the operations are implemented. The following table summarizes the operations covered in this section.

	Process Definition	Process Instance	Observer
ListInstances	X		
SetData			
Subscribe		X	
Unsubscribe		X	
GetHistory		X	
Notify			X

B.1 Process Definition Operations

B.1.1 ListInstances

This operation is used to retrieve a list of instances of the given process definition. Each instance in the returned list is identified with its key, name and priority.

Request Parameters:

Key – Identifier of the process definition resource whose instances are to be listed.

Filter – A filter to specify the type of instances that are to be returned.

FilterType – An indication of the language used to specify the filter.

Response Parameters:

Instances – A list of process instances, optionally based on the provided filtering requirements. Each returned instance will contain its:

- Key – the resource identifier of the process instance
- Name – the display name of the instance
- Priority – the assigned priority of the instance

Exception –

B.2 Process Instance Operations

B.2.1 SetData

This operation is used to set the values of any number of properties of the given process instance resource. This operation allows all of the settable properties of a resource as parameters, dependent on the interface in which it is invoked. At least one parameter must be provided in order for the operation to have any effect, but all parameters are optional. Current values of all the properties of the resource are returned.

Request Parameters:

Key – Identifier of the process instance resource whose properties are to be set.

Subject – A short description of this process instance.

Description – A longer description of this process instance resource.

State – The name of a new state to which the instance's status should be changed. This must be one of the allowed states of the resource, and must be reachable from the current state, otherwise an exception will be returned.

Priority – An optional indication of the relative importance of this process instance. This value will be an integer ranging from 1 to 5, 1 being the highest priority.

ContextData – Context-specific data relevant to this process instance. A partial set of values may be submitted for update, as the resulting context is considered to be the union of the previous context and these values. This information will be encoded in data pairs containing the name and value of each item.

Response Parameters:

B.2.2 Subscribe

This operation is used to register a resource with another resource, as a party interested in status changes and events that occur. If this particular resource does not support other observers, an exception will be returned to the caller.

Request Parameters:

Key – Identifier of the process instance resource for which the calling resource wishes to become an observer.

Observer – Resource id of the caller, which will receive event notifications (and must therefore implement the Observer interface).

Response Parameters:

Exception – If an exception occurs during execution of this method it will be returned here, and the above information may be omitted. Exceptions which are supported for this method:

List Exceptions and/or return codes here.

B.2.3 Unsubscribe

This operation is used to remove a resource from the list of registered observers of a resource. The calling resource will no longer receive event notifications after executing this operation.

Request Parameters:

Key – Identifier of the process instance resource for which the calling resource no longer wishes to be an observer.

Observer – The resource id of the caller. This id must be on the registered observers list of the given process instance. If it is not there, will be no change to the process instance, and an exception will be returned.

Response Parameters:

Exception –

B.2.4 GetHistory

This operation is used to retrieve the list of events that have occurred on this resource. If the service implementing this resource has not kept a transaction log, there may not be any history available. But if there is, it will be returned by this method.

Request Parameters

Key – Identifier of the process instance resource whose history is to be retrieved.

Filter – An optional filter to specify the set of history items to be returned. If no filter is provided, all history items will be returned.

FilterType – An indication of the language used to specify the filter.

Response Parameters

History – A list of event objects. Each returned event will contain its:

- Timestamp – the time of the event
- EventCode – an integer event code value representing the actual event
- EventType – a human readable name of the event. A listing of predefined event types can be found in the Interface 5 Audit Data Specification. [10]
- Responsible – the name of the resource or participant that caused the event
- SourceKey – the identifier of the resource to which the event refers
- SourceName – the name of the resource to which the event refers
- ContainerKey – the identifier of the resource containing the event, if any
- OldState – the former status of the resource, if this was a state changed event
- NewState – the current status of the resource, if this was a state changed event
- Transition – the name of the transition taken, if this was a state changed event

- ChangedData – a list of data items that were changed, if this was a data changed event. This information will be encoded in data pairs containing the name and value of each item.
- ChangedRole – the name of the role that changed, if this was a participant changed event
- Participants – a list of names of the participants of a role, if this was a participant changed event

Exception –

B.3 Observer Operations

B.3.1 Notify

This operation is used to send an event notification to a registered observer resource of a process instance.

Request Parameters

Key – Identifier of the resource on which some event has occurred.

EventObject – A list of information about the event:

- Timestamp – the time of the event
- EventCode – an integer event code value representing the actual event
- EventType – a human readable name of the event. A listing of predefined event types can be found in the Interface 5 Audit Data Specification. [10]
- Responsible – the name of the resource or participant that caused the event
- SourceKey – the identifier of the resource to which the event refers
- SourceName – the name of the resource to which the event refers
- ContainerKey – the identifier of the resource containing the event, if any
- OldState – the former status of the resource, if this was a state changed event
- NewState – the current status of the resource, if this was a state changed event
- Transition – the name of the transition taken, if this was a state changed event
- ChangedData – a list of data items that were changed, if this was a data changed event. This information will be encoded in data pairs containing the name and value of each item.
- ChangedRole – the name of the role that changed, if this was a participant changed event
- Participants – a list of names of the participants of a role, if this was a participant changed event

Response Parameters

Exception –

Appendix C References

[1] "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

[2] "The Workflow Reference Model", The Workflow Management Coalition, WfMC-TC-1003, 1.1, 19 Jan 1995. http://www.wfmc.org/standards/docs/tc003v11-16.pdf

[3] "Workflow Management Facility", Joint Submission, bom/98-06-07, revised submission, 4 July 1998. ftp://ftp.omg.org/pub/docs/bom/98-06-07.pdf

[4] "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

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

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

[7] Open Database Connectivity (ODBC), Microsoft Corporation. http://www.microsoft.com/data/odbc

[8] "Data elements and interchange formats -- Information interchange -- Representation of dates and times", International Standards Organization, ISO 8601:1988, 1, 4 March 1999. http://www.iso.ch/cate/d15903.html

[9] "Terminology & Glossary", The Workflow Management Coalition, WfMC-TC-1011, 3.0, Feb. 1999 http://www.wfmc.org/standards/docs/glossy3.pdf

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

[11] "Namespaces in XML", W3C, REC-xml-names-19990114, 1.0, 14 Jan. 1999. http://www.w3.org/TR/REC-xml-names

