The Workflow Management Coalition Specification

Workflow Management Coalition Workflow Standard - Interoperability Wf-XML Binding

Document Number WFMC-TC-1023
Document Status – Official

1-May-2000
Version 1.0

Send comments to: WORKGROUP4@FTPLIST.AIIM.ORG

Workflow Management Coalition (*Wf* MC)

2436 North Federal Highway #374,
Lighthouse Point,
FL 33064, USA

Phone +1 954 782 3376,
Fax +1 954 782 6365

email: wfmc@wfmc.org
http://www.wfmc.org/

First printing May 2000

Table of Contents

1 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]).

1.1 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 the most important data transport mechanism for Wf-XML, this specification provides a description of how Wf-XML interchanges are transferred using this protocol.

1.2 Scope

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

1.3 Audience

This specification is intended for use by software vendors, system integrators, consultants and any other individual or organization concerned with interoperability among workflow systems. Furthermore, it will be of value to those concerned with the design and implementation of integrated and/or distributed systems, as a protocol for the interaction of generic (possibly remote) services.

1.4 Document Status

This document is an official publication of the Workflow Management Coalition (WfMC), representing version 1.0 of the Wf-XML specification. It may be obtained via the Internet from: http://www.wfmc.org/standards/docs/Wf-XML, or by E-mailing a request to: mailto:wfmc@wfmc.org.

1.5 Documentation Conventions

- In several of the examples provided in this document, an ellipsis (…) 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.
- In addition to the examples that appear throughout this document, extracts from the DTD are provided in line with the descriptive text. These extracts are intended to highlight the particular markup constructs used throughout this specification. They will appear inside a box as in the following example:

```
<!ELEMENT foo (bar)>

<!ELEMENT xxx (baz)>
```

1.6 Background

This specification has been generated based on previous work completed by the Workflow Management Coalition (WfMC), the 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]

1.7 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.

2 Technical Specification

2.1 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.

An individual interoperable function is termed an "operation". Each operation may be passed a set of request parameters and return a set of response parameters. Operations are divided into different groups 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. 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:

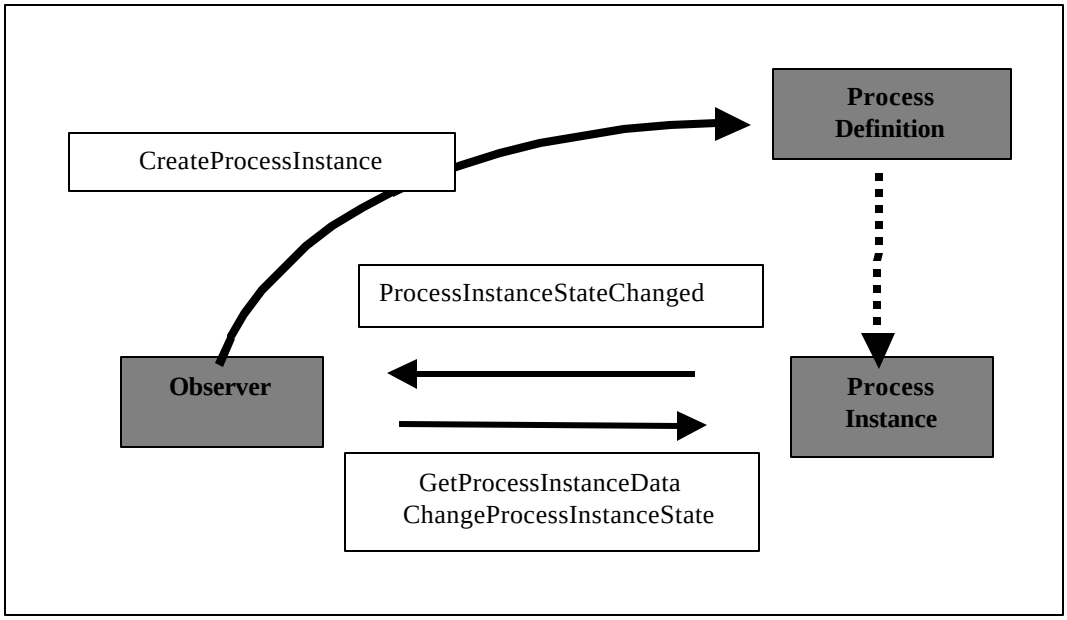

Diagram 1.

2.2 Security

In general, security considerations are out of the scope of this document because they are largely dependent upon the transport mechanism used by an implementation. This applies to user identification and authorization, encryption, and data/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, the security mechanisms used between two or more interoperating services should be identified in the interoperability contract between them.

2.3 WF-XML Language Definition

Every Wf-XML message is an XML document instance, conforming to the XML 1.0 specification. While not explicitly required by XML 1.0, each Wf-XML message will contain an XML declaration, for the sake of clarity and precision. The XML declaration will appear as follows: '<?xml version="1.0"?>'. This declaration contains no explicit encoding information, and therefore implies that the XML 1.0 supported character encodings of 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. The complete Wf-XML DTD can be found in Section 8.

2.3.1 Wf-XML Namespace Definition

One of the most important aspects of an XML-based interoperability specification is its ability to interact with other XML markup vocabularies, mixing elements from each as necessary. This capability will be crucial to Wf-XML, as much of the data exchanged between workflow systems will be specific to those systems, and is likely to be indicated by markup defined outside of this specification. It is for this reason, that the "Namespaces in XML" specification [11] was created, and should be used in conjunction with this specification.

In order to enable usage of this mechanism, the following namespace definition will be used for Wf-XML:

"http://www.wfmc.org/standards/docs/Wf-XML"

It should be noted that this namespace definition does not imply the existence or location of any DTD or XML Schema, as the purpose of a namespace declaration is simply to provide a unique identification for a set of XML elements. Within Wf-XML messages, this namespace should be declared as the default namespace for the document as follows:

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

Alternatively, this namespace may be explicitly declared on relevant elements within a message, as in the following:

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

Using the above declarations, applications will be able to distinguish elements defined by this specification from those defined elsewhere, in order to achieve higher levels of interoperability without degrading conformance to this specification. It should be noted however, that as of the time of the release of this specification, most XML parsers do not support validation of documents containing namespace-qualified elements. While Off-the-Shelf parsers can still be used to verify the well-formedness of these documents, validation against a DTD or Schema would require specialized preprocessing.

2.3.2 Data Types

Although DTD syntax does not support robust data typing, several required data types are provided for use with this specification. A future version of this specification will utilize the W3C's forthcoming XML Schema syntax, which will allow these types to be validated at the XML parser level. Where required, data fields 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 derived from the WfMC Interface4, except for String[the maximum number of characters], URI, and XML. Where no specific data type is indicated for a value, the type will default to String.

2.3.2.1 Date and Time Values

In addition to the data types described above, a specific Date/Time format is provided for data of type "Date". All date and time values shall be represented as Greenwich Mean Time (GMT) based timestamps to ensure interoperability between resources that may not be in the same time zone. The DateTime format shall be represented as:

YYYY-MM-DDThh:mm:ssZ

where:

YYYY is the year in the Gregorian calendar

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

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

T is the separator between the date and time portions of this timestamp

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 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 in a way that meets 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 resource requires dates/times to be represented locally (in the timezone of the resource), then it will need to perform the conversion from GMT to the local timezone.

2.3.3 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”. This element carries a required attribute (Version) that indicates the particular version of this specification with which the message conforms. The value of this attribute may be used by an implementation to determine whether this message can be processed. Each Wf-XML message contains an optional section for transport-specific information (WfTransport), a single, required message header (WfMessageHeader), and a single, required message body (WfMessageBody). If necessary, the WfTransport section will be used to convey information relevant to a particular implementation’s transport protocol. The message header will contain information relevant to routing and preprocessing of the message. The message body is where the operation specific information is placed. Therefore, the skeleton of a Wf-XML message will appear as follows:

Example 1:

```
<?xml version=”1.0”?>
<WfMessage Version=”1.0”>
        <WfTransport/>
        <WfMessageHeader>
                …
        </WfMessageHeader>
        <WfMessageBody>
                …
        </WfMessageBody>
</WfMessage>
```

2.3.4 Message Transport Mechanism

One of the goals of this specification is to provide an implementation-independent protocol. An important aspect of this independence is the ability to exchange Wf-XML messages over any transport mechanism. Therefore, this specification does not define any of the characteristics of supporting protocols and mechanisms used to exchange Wf-XML messages, such as details regarding message security, batch

or asynchronous processing, message identification, etc. However, it will often be necessary to provide this type of information in a Wf-XML message, and it is for this reason that the WfTransport section is provided. Details appearing in this section regarding the requirements of a particular transport should be specified by the binding protocol for that transport.

As it can be expected that many implementations will require some means of identifying and/or synchronizing messages (e.g. – when an asynchronous transport protocol is used), an optional element called CorrelationData is predefined in this section. This is purely a convenience, and may be used or ignored by an implementation as appropriate.

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

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

2.3.5 Message Type

There are two types of Wf-XML messages, called “Request” and “Response”. A Request message is used to initiate an operation in a remote resource, and/or to provide input to that resource. A Response message is used to send the results of an operation to its requesting resource, providing output. Although these two message types are clearly complimentary, there is no requirement that they always be used in conjunction. That is to say, that unlike the model used by HTTP (with the same names), not every Wf-XML request requires a response. These names are used only as a way of differentiating the type of message being processed.

For example, a requestor may wish to initiate a process and continue it’s own normal processing, at that point becoming an observer of the remote process. Therefore, the initiating resource sends a request to the remote resource, which may or may not send back a response. When the remote resource completes the requested operation, it sends a request message to the initiating resource to inform it of the completion. This message requires no response, as it is merely a notification of an event, but is a “Request” type of message.

2.3.6 Message Header Definition

The message header (WfMessageHeader) contains information that is generically useful to all messages, such as resource identifiers, operation names, etc. Separation of this information from the message body enables pre-processing of workflow messages without having to parse the operation-specific information. The message header is defined as follows:

```
<!ENTITY % ISOLangs
"(aa|ab|af|am|ar|as|ay|az|ba|be|bg|bh|bi|bn|bo|br|ca|co|cs|cy|da|de|dz|el|en|eo|es|et|eu|fa|fi|fj|fo|fr|fy|ga|gd|gl|
gn|gu|ha|hi|hr|hu|hy|ia|ie|ik|in|is|it|iw|ja|ji|jw|ka|kk|kl|km|kn|ko|ks|ku|ky|la|ln|lo|lt|lv|mg|mi|mk|ml|mn|mo|mr|ms|
mt|my|na|ne|nl|no|oc|om|or|pa|pl|ps|pt|qu|rm|rn|ro|ru|rw|sa|sd|sg|sh|si|sk|sl|sm|sn|so|sq|sr|ss|st|su|sv|sw|ta|t
e|tg|th|ti|tk|tl|tn|to|tr|ts|tt|tw|uk|ur|uz|vi|vo|wo|xh|yo|zh|zu)">

<!ELEMENT WfMessageHeader ((Request | Response), Key)>

<!ELEMENT Request EMPTY>

<!ATTLIST Request ResponseRequired (Yes | No | IfError) #REQUIRED

        ResponseLang %ISOLangs; #IMPLIED>

<!ELEMENT Response EMPTY>

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

The elements and attributes defined in the message header have the following meanings:

Request – The presence of this element indicates that this is a request message

ResponseRequired – This attribute may contain the following values: Yes, No, or IfError. If the value specified is Yes, a response must be returned for this request in all cases, and it must be processed by the requesting resource. If the value specified is No, a response does not need to be returned for this request, and if one is returned it may be ignored by the requesting resource. If the value specified is IfError, a response only needs to be returned for this request in the case where an error has occurred processing it, and the requesting resource must process the response.

ResponseLang – The value of this attribute indicates the spoken language to be used (English, German, Japanese, etc.) in language-specific data elements such as Subject or Description when that information is returned in the response to this request. The value of this attribute is chosen from a list of language identifiers defined in the ISO 639 standard [13] for language identifiers. If this element is not used, no assumption can be made about the language used in the response returned for this request.

Response – The presence of this element indicates that this is a response message

Key – The Uniform Resource Identifier (URI) [14] of the resource that is the target of this request, or the source of this response, is contained in this element. For Process Definition related operations (CreateProcessInstance), this is the URI of the appropriate process definition. For Process Instance related operations, it is the URI of the associated process instance. It should be noted that this is not required to be an absolute URI. An implementation may wish to maintain base URIs internally, thereby only requiring a relative URI within the message header. In these cases, semantics and mechanisms for processing these relative URIs should be agreed upon in the interoperability contract.

For example, a request message would appear as follows:

Example 2:

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

A Response message would appear as follows:

Example 3:

```
<?xml version="1.0"?>
<WfMessage Version="1.0">
       <WfTransport/>
       <WfMessageHeader>
              <Response/>
              <Key>http://www.XYZcompany.com/wfprocess/foo</Key>
       </WfMessageHeader>
       <WfMessageBody>
              …
       </WfMessageBody>
</WfMessage>
```

2.3.7 Message Body Definition

The message body provides operation specific data. For a request message, it contains an <*OperationName*.Request> element, which further contains request parameters; for a response message, it contains an <*OperationName*.Response> element containing result parameters returned by an operation. This naming convention allows for the appropriate specification of the content of the operation elements based on their context (within a Request or Response type message). For convenience, the list of valid operation elements is defined by two entities as shown below, one for Requests and one for Responses.

```
<!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 4:

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

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

Example 5:

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

2.3.8 Representation of Process Context and Result Data

Typically, a process is associated with some number of data items specific to that process. These data items may represent the properties of the Process Instance (Workflow Control or Workflow Relevant data), and/or any application related data associated with invoked applications during process enactment (Application data). (These terms are defined within the WfMC Glossary [9].) This collection of data items is called the context of the process in a Request message, and the result of the process in a Response message. When the process is enacted, those data items must be specified and accessible. For this

purpose, this specification provides a place to identify these data items in the form of elements named ContextData and ResultData. When a Process Definition is instantiated, the context of the resulting Process Instance is initialized with the contents of the ContextData element. When a Process Instance is completed, the resulting data is exchanged as the contents of the ResultData element.

In some cases, this data may take the form of parameters, each with a name and a value. For example:

Example 6:

```
<?xml version="1.0"?>
<WfMessage Version=”1.0”>
	<WfMessageTransport/>
	<WfMessageHeader>
		…
	</WfMessageHeader>
	<WfMessageBody>
		…
		<ContextData>
			<Name>User1</Name><Value>foo</Value>
			<Name>File1</Name><Value>bar</Value>
		</ContextData>
		…
	</WfMessageBody>
</WfMessage>
```

In other cases, it may appear as structured, tagged data. For example:

Example 7:

```
<?xml version="1.0"?>
<WfMessage Version=”1.0”>
	<WfMessageTransport/>
	<WfMessageHeader>
		…
	</WfMessageHeader>
	<WfMessageBody>
		…
		<ContextData>
			<Vehicle>
				<VehicleType>Car</VehicleType>
				<Specification>
					<Manufacturer>Mercedes</Manufacturer>
					<Model>450SL</Model>
				</Specification>
			</Vehicle>
		<Customer>John Doe</Customer>
		…
	</ContextData>
		…
	</WfMessageBody>
</WfMessage>
```

In still other cases, it may simply be flat character data, as in:

Example 8:


```
<?xml version="1.0"?>
<WfMessage Version=”1.0”>
       <WfMessageTransport/>
       <WfMessageHeader>
              …
       </WfMessageHeader>
       <WfMessageBody>
              …
              <ContextData>
                     http://www.xyz.com/process/contextfile
              </ContextData>
              …
       </WfMessageBody>
</WfMessage>
```

Because the nature and definition of this context data cannot be known (as it is particular to a given implementation), there is no way to define specific markup to identify it. Therefore, the content of the ContextData and ResultData elements must be specified on a case-by-case basis. As a placeholder for extensibility in this area, a default content model of “ANY” is defined for these elements. Appropriate specification of the content of these elements should be made in the interoperability contract between two enactment services, thereby extending this specification to meet their specific needs.

```
<!ELEMENT ContextData ANY>

<!ELEMENT ResultData ANY>
```

There are numerous ways this extensibility can be achieved, depending on the requirements of a given implementation:

- If full validity is required, the interoperating parties should agree upon the necessary changes to the content models of the ContextData and ResultData elements and provide an extended DTD in their interoperability contract, against which their Wf-XML instances can be validated. If this context and result data varies for each given process definition, separate DTDs may be required for each defined process in order to support full validation. These changes will not affect conformance to the specification, as they are anticipated extensions. For example, the following content models might be used to validate markup exchanged with an automobile manufacturer, as in example 7 above:

```
<!ELEMENT ContextData (Vehicle, Customer,…)>

<!ELEMENT Vehicle (VehicleType, Specification)>

<!ELEMENT Specification (Manufacturer, Model)>

<!ELEMENT Customer (…)>
```

- If well-formedness of the XML message is sufficient, interoperating parties can simply agree in their interoperability contract on the markup to be exchanged within the context elements, and leave the DTD declarations unchanged.
- Lastly, interoperating parties may exchange context information conforming to an external DTD or schema using namespace declarations to reference that external resource. This mechanism most easily allows for the utilization of industry standard markup within Wf-XML messages, and is the preferred approach. For example:

Example 9:


```
<?xml version="1.0"?>
<WfMessage Version=”1.0”>
       <WfMessageTransport/>
       <WfMessageHeader>
              …
       </WfMessageHeader>
       <WfMessageBody>
              …
          <ContextData>
               <auto:Vehicle
               xmlns:auto=”urn:autoindustry:schemas:vehicle.xsd”>
                    <VehicleType>Car</VehicleType>
                    <Specification>
                          <Manufacturer>Mercedes</Manufacturer>
                          <Model>450SL</Model>
                    </Specification>
               <auto:/Vehicle>
          </ContextData>
              …
       </WfMessageBody>
</WfMessage>
```

These mechanisms may also be combined to allow for workflow control, workflow relevant and application data to be exchanged simultaneously. However, regardless of which approach to extensibility is chosen, any markup contained within the ContextData and ResultData elements must not contain elements named ContextData and ResultData respectively, unless namespace qualified. Such usage would violate both validity and well-formedness of the XML instance.

2.3.9 Process Status

Another important aspect of a process is the state in which the process is in at a given point in time. In general, a process may be active or inactive to some degree for a number of reasons. The WfMC has defined a standard set of valid process instance states. These states are organized into several levels of granularity. While the higher level states defined below must be supported, an implementation may choose to omit the optional states or add additional states to those defined.

```
<!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 (EMPTY)>

<!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)

2.3.10 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 MainCode (#PCDATA)>

<!ELEMENT SubCode (#PCDATA)>

<!ELEMENT Type (#PCDATA)>

<!ELEMENT Subject (#PCDATA)>

<!ELEMENT Description (#PCDATA)>

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

This exception element will be returned as the contents of the <*OperationName*.Response> element, in lieu of the normal response data, from all operations in which an exception occurs. The exception information will contain the following elements:

MainCode - This is a three digit positive integer defined in the operation specification. It is operation-specific and gives some indication of what went wrong. Programs can use this code to calculate what to do when this exception occurs. 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 more specifically that the format of the key is wrong. This is where a vendor would specify errors that are specific to their processing. This element may be omitted if the MainCode is deemed sufficient. (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 10:


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

2.3.10.1 Exception Codes

The following is 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_ELEMENT_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

2.4 Operation Definitions

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

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

2.4.1 Process Definition Operations

This group of operations is used to create process instances. Currently it contains only the operation CreateProcessInstance.

2.4.1.1 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 ObserverKey (#PCDATA)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Subject (#PCDATA)>
<!ELEMENT Description (#PCDATA)>
<!ELEMENT ContextData ANY>
```

```
<!ELEMENT CreateProcessInstance.Request (ObserverKey?, Name?, Subject?, Description?,
ContextData)>

<!ATTLIST CreateProcessInstance.Request StartImmediately (true|false) #FIXED “true”>

<!ELEMENT ProcessInstanceKey (#PCDATA)>

<!ELEMENT CreateProcessInstance.Response (ProcessInstanceKey | Exception)>
```

Request Parameters:

StartImmediately - A Boolean value ("true" or "false"), indicating whether the newly created instance should be started immediately upon creation. The value of this parameter is currently always "true".

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. With the ObserverKey being set, the interoperability model of a nested sub-process is used, otherwise the model of a chained process is implied. (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).

***Example 11*:**

```
<?xml version="1.0"?>
<WfMessage Version=”1.0”>
       <WfMessageHeader>
              <Request ResponseRequired=”Yes”/>
              <Key>http://www.XYZcompany.com/wfprocess/foo</Key>
       </WfMessageHeader>
       <WfMessageBody>
           <CreateProcessInstance.Request StartImmediately=”true”>
                <ObserverKey>http://www.ABCcompany.com/wfprocessor</Observ
              erKey>
                 <ContextData>
                        <Vehicle>
                               <VehicleType>Car</VehicleType>
                               <Specification>
                                      <Manufacturer>Mercedes</Manufacturer>
                                      <Model>450SL</Model>
                               </Specification>
                        </Vehicle>
                        <Customer>John Doe</Customer>
                 </ContextData>
              </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 12*:**

```
<?xml version="1.0"?>
<WfMessage Version="1.0">
	<WfMessageHeader>
		<Response/>
		<Key>http://www.XYZcompany.com/wfprocess/foo</Key>
	</WfMessageHeader>
	<WfMessageBody>
		<CreateProcessInstance.Response>
			<ProcessInstanceKey>http://www.ABCcompany.com/wfproces
			s/bar</ProcessInstanceKey>
		</CreateProcessInstance.Response>
	</WfMessageBody>
</WfMessage>
```

2.4.2 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.

2.4.2.1 GetProcessInstanceData

This operation is used to retrieve the values of 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|closed.abnormalCompleted.ab
orted)*">

<!ELEMENT ResultDataSet %ProcessInstanceData;>

<!ELEMENT Name (#PCDATA)>

<!ELEMENT Subject (#PCDATA)>

<!ELEMENT Description (#PCDATA)>

<!ELEMENT State %vstates;>

<!ELEMENT ValidStates %vstates;>
```

```
<!ELEMENT ObserverKey (#PCDATA)>
<!ELEMENT ProcessDefinitionKey (#PCDATA)>
<!ELEMENT Priority (#PCDATA)>
<!ELEMENT LastModified (#PCDATA)>
<!ELEMENT GetProcessInstanceData.Request (ResultDataSet?)>
<!ELEMENT GetProcessInstanceData.Response (%ProcessInstanceData; | Exception)>
```

Request Parameters:

ResultDataSet – 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 not specified, all results are returned. (optional)

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

***Example 13*:**

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

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

***Example 14*:**

```
<?xml version="1.0"?>
<WfMessage Version="1.0">
	<WfMessageHeader>
		<Request ResponseRequired="Yes"/>
		<Key>http://www.XYZcompany.com/wfprocess/foo</Key>
	</WfMessageHeader>
	<WfMessageBody>
		<GetProcessInstanceData.Request>
			<ResultDataSet>
				<Name/>
				<Priority/>
			</ResultDataSet>
		</GetProcessInstanceData.Request>
	</WfMessageBody>
</WfMessage>
```

Response Parameters:

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

Subject – A short description of this process instance. (Optional)

Description – A longer description of this process instance resource. (Optional)

State – The current status of this resource. (Optional)

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

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

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

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. (Optional)

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. (Optional)

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

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 of a response message for a GetProcessInstanceData operation:

***Example 15*:**

```
<?xml version="1.0"?>
<WfMessage Version="1.0">
	<WfMessageHeader>
		<Response/>
		<Key>http://www.XYZcompany.com/wfprocess/foo</Key>
	</WfMessageHeader>
	<WfMessageBody>
		<GetProcessInstanceData.Response>
			<Name>xxx</Name>
			<Subject>xxx</Subject>
			<ResultData>
				<Vehicle>
					<VehicleType>Car</VehicleType>
					<Specification>
						<Manufacturer>Mercedes</Manufacture
					r>
						<Model>450SL</Model>
					</Specification>
				</Vehicle>
				<Customer>John Doe</Customer>
			</ResultData>
		</GetProcessInstanceData.Response>
	</WfMessageBody>
</WfMessage>
```

2.4.2.2 ChangeProcessInstanceState

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

```
<!ELEMENT State %vstates;>

<!ELEMENT ChangeProcessInstanceState.Request (State)>

<!ELEMENT ChangeProcessInstanceState.Response (State | Exception)>
```

Request Parameters:

State – The new state to which the process instance should transition.

***Example 16*:**

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

Response Parameters:

State – The new state resulting from the operation.

Exceptions:

Exceptions which are supported for this operation:

WF_INVALID_KEY

WF_NO_AUTHORIZATION

WF_INVALID_STATE_TRANSITION

***Example 17*:**

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

2.4.3 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.

2.4.3.1 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. The ResponseRequired attribute will typically be set to false for this operation as it is normally only used as a notification to an observer that a state change event has occurred.

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

<!ELEMENT State %vstates;>

<!ELEMENT ResultData ANY>

<!ELEMENT LastModified (#PCDATA)>

<!ELEMENT ProcessInstanceStateChanged.Request (ProcessInstanceKey, State, ResultData?,
LastModified?)>

<!ELEMENT ProcessInstanceStateChanged.Response EMPTY>
```

Request Parameters:

ProcessInstanceKey – URI of the process instance resource that has 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 18:

```
<?xml version="1.0"?>
<WfMessage Version="1.0">
       <WfMessageHeader>
              <Request ResponseRequired="false"/>
              <Key>http://www.XYZcompany.com/wfprocess/foo</Key>
       </WfMessageHeader>
       <WfMessageBody>
              <ProcessInstanceStateChanged.Request>
                     <ProcessInstanceKey>http://www.XYZcompany.com/wfproces
                     s/foo
                     </ProcessInstanceKey>
                     <State>
                            <closed.completed/>
                     </State>
                     <ResultData>
                            <Vehicle>
                                   <VehicleType>Car</VehicleType>
                                   <Specification>
                                        <Manufacturer>Mercedes</Manufacture
                                   r>
                                        <Model>450SL</Model>
                                   </Specification>
                            </Vehicle>
                            <Customer>John Doe</Customer>
                     </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

If the ResponseRequired attribute is set to "true" in the ProcessInstanceStateChanged request a minimal response will be returned. This can be useful for trapping any errors that may occur during notification of the state change.

***Example 19*:**

```
<?xml version="1.0"?>
<WfMessage Version=”1.0”>
       <WfMessageHeader>
              <Response/>
              <Key>http://www.XYZcompany.com/wfprocess/foo</Key>
       </WfMessageHeader>
       <WfMessageBody>
              <ProcessInstanceStateChanged.Response/>
       </WfMessageBody>
</WfMessage>
```


3 Relationship to other Standards

3.1 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 were separated in the OMG Workflow Management Facility interfaces into a single operation, thereby improving performance by not requiring such fine-grained operations.

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 the 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 some interfaces that are not represented by Wf-XML at this point in time: WfActivity, WfResource, WfAssignment.

4 Implementation Issues

4.1 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 interoperability. 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:

- Data Requirements – application-specific data required to be transferred in order to utilize basic or extended functionality. This data will appear in the ContextData and ResultData elements. Any application-specific data type requirements should also be addressed here.

- 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.

- Transport Protocol Limitations – required header data, timeout values, buffer sizes, etc.

- Security Considerations – encryption methods, user verification, firewall configuration requirements, etc.

5 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.

5.1 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. 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.

5.2 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.

It is recommended that namespaces be used to facilitate the interchange of this application-specific data within Wf-XML messages. An implementation may utilize namespaces to differentiate process-related data from target application data, as well as from Wf-XML encoded protocol data. Proper namespace qualification of context-specific data will also shield it from changes to the Wf-XML protocol data as new versions are released.

This specification only requires well-formed data. Therefore, interoperating vendors may exchange any data they wish in the context-specific elements 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.

6 Transport Layer Bindings

Wf-XML messages for workflow interoperability can be communicated between interoperating workflow systems using many different transport mechanisms. This section provides a specification for a Hyper Text Transfer Protocol (HTTP) [12] binding , which is considered to be the most common transport mechanism utilized to communicate Wf-XML request and response messages between interoperating enactment services.

The support of this or any other specific transport layer binding is not required to be compliant with the specification. However, one of the specified transport layer bindings must be used to realistically effect interoperability, and this is the only specified binding to date.

6.1 HTTP

For HTTP, the communicating enactment services are considered HTTP servers (services 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 from the Wf-XML message header. Alternately, an implementation may chose to maintain the base URI for a resource internally and combine this with a relative URI in the message header to formulate an absolute URI. If an implementation wishes to utilize relative URIs in this way, further details should be agreed upon in the interoperability contract. In either case, this data is vendor-specific and must either be known beforehand (e.g., in case of a process definition key) orobtained as the result of a response parameter returned by a previous operation (e.g., "ProcessInstanceKey"). For the purposes of this binding, all final URIs will be resolvable 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 orresponse respectively.*
- *For authentication, the usual HTTP mechanisms are used. This includes usage of the respective HTTP header fields.*

7 Document Type Definition (DTD)

This section provides the WF_XML DTD for the purposes of implementation reference and optional data validation by an XML processor.

This DTD is designed with the intention of being simple and easy to implement, while supporting a robust and flexible structure.

```
<!-- Wf-XML DTD, Revision 1.0 Final - 11 April, 2000

If a DOCTYPE declaration is required to parse this set of declarations, the following line should be prepended to this file:

    <!DOCTYPE WfMessage PUBLIC "-//WfMC//DTD Wf-XML 1.0//EN"
"http://www.wfmc.org/standards/docs/Wf-XML-1.0.dtd" [

and the following line appended:

    ]>

-->


<!-- ~~~~~~~~~~~~~~~~~~~ Entity Declarations ~~~~~~~~~~~~~~~~~~~ -->


<!-- The ISOLangs entity provides the choices for the ResponseLang attribute of the Request element. These language codes are taken from the ISO 639:1988 standard, which can be used for further clarification of the names of each language and can be obtained from http://www.iso.ch/cate/d4766.html. Additional information is also available at: http://www.oasis-open.org/cover/iso639a.html. -->

<!ENTITY % ISOLangs
"(aa|ab|af|am|ar|as|ay|az|ba|be|bg|bh|bi|bn|bo|br|ca|co|cs|cy|da|de|dz|el|en|eo|es|et|eu|fa|fi|fj|fo|fr|fy|ga|gd|gl|gn|gu|ha|hi|hr|hu|hy|ia|ie|ik|in|is|it|iw|ja|ji|jw|ka|kk|kl|km|kn|ko|ks|ku|ky|la|ln|lo|lt|lv|mg|mi|mk|ml|mn|mo|mr|ms|mt|my|na|ne|nl|no|oc|om|or|pa|pl|ps|pt|qu|rm|rn|ro|ru|rw|sa|sd|sg|sh|si|sk|sl|sm|sn|so|sq|sr|ss|st|su|sv|sw|ta|te|tg|th|ti|tk|tl|tn|to|tr|ts|tt|tw|uk|ur|uz|vi|vo|wo|xh|yo|zh|zu)">


<!-- The following two entities are used to define the request and response elements for each operation. -->

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


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


<!-- The ProcessInstanceData entity defines the properties of a process instance that may be obtained using the GetPrcoessInstanceData operation. -->
```

```
<!ENTITY % ProcessInstanceData "(Name | Subject | Description | State | ValidStates | ObserverKey |
ResultData | ProcessDefinitionKey | Priority | LastModified)+">


<!-- This is the list of valid states defined by the WfMC for version 1.0 of Wf-XML. -->

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




<!-- ~~~~~~~~~~~~~~~~~~ Element Declarations ~~~~~~~~~~~~~~~~~~~~ -->


<!-- Root element -->

<!ELEMENT WfMessage (WfTransport?, WfMessageHeader, WfMessageBody)>

<!ATTLIST WfMessage Version CDATA #REQUIRED>


<!-- ~~~~~~~~~~ WfTransport ~~~~~~~~~~~~ -->

<!-- Used for transport-specific information, such as special security or asynchronous processing. --
>

<!ELEMENT WfTransport (CorrelationData?)>


<!ELEMENT CorrelationData (#PCDATA)>


<!-- ~~~~~~~~ WfMessageHeader ~~~~~~~~~~ -->

<!-- Information generally used in all messages, helpful for preprocessing. -->

<!ELEMENT WfMessageHeader ((Request | Response), Key)>


<!ELEMENT Request EMPTY>

<!ATTLIST Request ResponseRequired (Yes | No | IfError) #REQUIRED

                  ResponseLang %ISOLangs; #IMPLIED>


<!ELEMENT Response EMPTY>


<!-- The URI of the resource. -->

<!ELEMENT Key (#PCDATA)>


<!-- ~~~~~~~~~~ WfMessageBody ~~~~~~~~~~~~ -->

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

```
<!ELEMENT CreateProcessInstance.Request (ObserverKey?, Name?, Subject?, Description?,
ContextData)>
<!ATTLIST CreateProcessInstance.Request StartImmediately (true|false) #FIXED "true">

<!ELEMENT ObserverKey (#PCDATA)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Subject (#PCDATA)>
<!ELEMENT Description (#PCDATA)>

<!ELEMENT ContextData ANY>

<!ELEMENT GetProcessInstanceData.Request (ResultDataSet?)>
<!ELEMENT ResultDataSet %ProcessInstanceData;>

<!ELEMENT State %vstates;>
<!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 (State)>

<!ELEMENT ProcessInstanceStateChanged.Request (ProcessInstanceKey, State, ResultData?,
LastModified?)>

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



```
<!ELEMENT CreateProcessInstance.Response (ProcessInstanceKey | Exception)>

<!ELEMENT GetProcessInstanceData.Response (%ProcessInstanceData; | Exception)>

<!ELEMENT ChangeProcessInstanceState.Response (State | Exception)>

<!ELEMENT ProcessInstanceStateChanged.Response (Exception?)>

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


Appendix A -- Terminology

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

- DTD – Document Type Definition, a set of markup declarations that provide a grammar for a class of documents.
- Element – A component of an XML document consisting of markup and the text contained within that 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 - A component of an XML document used to associate named properties with an element.
- Entity – A unit of storage in which the contents of the storage unit are associated with a name.
- Document Instance – An instance of a document type (or class of documents).

Appendix B - Additional Operation Definitions (Non-Normative)

The operations contained within this appendix are not within the scope of the initial specification but are reserved for future use. The following table summarizes the operations covered in this section.

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

In addition to these operations, several suggestions have been made to enhance the capability of Wf-XML to support reliable, asynchronous messaging over HTTP for the purposes of enterprise EDI transaction management.. These changes will also be considered in a future revision of this specification.

B.1 General Operations

B.1.1 WfQueryInterface

This operation is used to query an implementation for various generic capabilities. In particular, it can be used to determine the capabilities of an implementation to support various security requirements, conformance to this specification or XML processing features.

B.2 Process Definition Operations

B.2.1 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.

B.3 Process Instance Operations

B.3.1 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.

B.3.2 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.

B.3.3 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.

B.3.4 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. However, if there is, it will be returned by this method.

B.4 Observer Operations

B.4.1 Notify

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

B.5 Other

B.5.1 Asynchronous and Batch Messaging over HTTP

This processing addresses various changes to provide enhancements to the HTTP binding. These changes would support more reliable asynchronous messaging over HTTP, as well as allow for batch processing of requests and responses in support of enterprise EDI transaction processing.

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

[12] "Hypertext Transfer Protocol -- HTTP/1.1", R. Fielding et al., Request for Comments 2616, June 1998. http://www.ietf.org/rfc/rfc2616.txt

[13] "Code for the Representation of Names of Languages", International Standards Organization, ISO 639:1988, 1, 30 November 1992. http://www.iso.ch/cate/d4766.html

[14] "Uniform Resource Identifiers (URI): Generic Syntax" , T Berners-Lee, R. Fielding, L. Masinter, Request for Comments 2396, August 1998. http://www.ietf.org/rfc/rfc2396.txt

