The Workflow Management Coalition Specification

Workflow Management Coalition Workflow Standard - Interoperability Wf-XML Binding

Document Number WFMC-TC-1023
Document Status – Final Draft

14-November-2001
Version 1.1

Send comments to: WORKGROUP4@FTPLIST.AIIM.ORG

Workflow Management Coalition (WfMC)

2436 North Federal Highway #374
Lighthouse Point, Florida 33064
USA
Tel: +1 954 782 3376
Fax: +1 954 782 6365
Email: wfmc@wfmc.org
Web: http://www.wfmc.org

First printing October 2001

Table of Contents

1 CHANGE HISTORY **5**
1.1 VERSION COMPATIBILITY 6
2 INTRODUCTION **6**
2.1 PURPOSE 6
2.2 SCOPE 6
2.3 AUDIENCE 6
2.4 BACKGROUND 6
2.5 DOCUMENT STATUS 7
2.6 DOCUMENTATION CONVENTIONS 7
3 TECHNICAL SPECIFICATION **8**
3.1 LOGICAL RESOURCE MODEL 8
3.2 LOGICAL INTERACTION MODEL 9
3.2.1 Synchronous Messaging 9
3.2.2 Asynchronous Messaging 10
3.2.3 Batch Messaging 10
3.3 SECURITY 12
3.4 WF-XML LANGUAGE DEFINITION 12
3.4.1 Wf-XML Namespace Definition 12
3.4.2 Data Types 13
3.4.3 Overall Message Structure 14
3.4.4 Message Transport Mechanism 16
3.4.5 Message Header Definition 19
3.4.6 Message Body Definition 21
3.4.7 Representation of Process Context and Result Data 23
3.4.8 Status 25
3.4.9 Error Handling 27
3.5 OPERATION DEFINITIONS 29
3.5.1 Control Operations 30
3.5.2 Process Definition Operations 32
3.5.3 Process Instance Operations 34
3.5.4 Observer Operations 37
4 RELATIONSHIP TO OTHER STANDARDS **41**
4.1 OMG WORKFLOW MANAGEMENT FACILITY STANDARD (JOINTFLOW) 41
5 IMPLEMENTATION ISSUES **42**
5.1 INTEROPERABILITY CONTRACT 42
6 CONFORMANCE **43**
6.1 CONFORMANCE PROFILES 43
6.1.1 Interoperability 43
6.1.2 Asynchronous 44
6.1.3 Batch 44
6.2 VERSION CONFORMANCE 44
6.3 OTHER CONSIDERATIONS 45
6.4 VALIDITY VS. WELL-FORMEDNESS 45
6.5 CONFORMANCE VS. EXTENSIBILITY 45
7 TRANSPORT LAYER BINDINGS **47**

7.1 HTTP 47

8 DOCUMENT TYPE DEFINITION (DTD) 49

APPENDIX A -- TERMINOLOGY 53

APPENDIX B - FUTURE DIRECTIONS (NON-NORMATIVE) 54

B.1 MESSAGING PROTOCOL 54
B.2 SPECIFICATION META-LANGUAGE 54
B.3 OPERATIONS 54
B.3.1 CONTROL OPERATIONS 55
B.3.1.1 WFQUERYINTERFACE 55
B.3.2 PROCESS DEFINITION OPERATIONS 55
B.3.2.1 ListInstances 55
B.3.3.1 SetData 55
B.3.3.2 Subscribe 55
B.3.3.4 GetHistory 55
B.4 ANCILLARY SUPPORTING MECHANISMS 55
B.4.1 INTEROPERABILITY CONTRACT 55
B.4.2 REFERENCE FRAMEWORK IMPLEMENTATION 56
B.4.3 CONFORMANCE TESTING HARNESS 56

APPENDIX C REFERENCES 57

1 Change History (Non-Normative)

A brief summary of changes made in this version follows:

- Enhanced support for asynchronous processing – A new message type (Acknowledgement) was added, as was specification of structures in the Transport section useful for correlation of messages.
- Support for batch processing added – This change allows multiple requests/responses to be delivered within a single message. Modifications to the content model of the root element (WfMessage) were required to effect this change.
- Enhanced support for Parallel-Synchronized processing – A new operation called Notify was added in the Observer group to allow for notification of arbitrary events.
- Context-specific data structures – A new recommended content model for context-specific data is specified that enhances interoperability through standardization.
- Errata corrections – Various changes to fix errors discovered in the 1.0 version of this specification.
- Miscellaneous – Editorial changes and minor technical clarifications throughout.

This list is a summary provided for convenience only and is by no means intended to be a comprehensive reference. The normative sections of this specification should be consulted for exact details of the changes made in this version.

2 Introduction (Normative)

This document represents a specification for a language based on the eXtensible Markup Language (XML) [1], designed to model the data transfer requirements set forth in the Workflow Management Coalition (WfMC)'s Interoperability Abstract specification [1]. This language will be used as the basis for concrete implementations of the functionality described in the Interoperability Abstract supporting the WfMC's Interface 4, as defined by the Workflow Reference Model [2].

2.1 Version Compatibility

This version (1.1) of the Wf-XML specification is fully backward compatible with its previous version (1.0). For the sake of clarity, the term "backward-compatible" is used here to mean that all changes made to the specification in this version have been additive, making it is a superset of version 1.0. For a more detailed explanation of conformance implications, see section 6 Conformance.

2.2 Purpose

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

- Support chained, nested and parallel-synchronized mo dels of interoperability
- Provide for both synchronous and asynchronous interactions
- Support individual and batch operations
- Remain implementation independent
- Define a light, easy-to-implement protocol

In order to achieve these goals, this specification will utilize a loosely coupled, message-based approach to facilitate rapid implementation using existing technologies. It will describe the syntax of these messages 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.

The XML language described herein, Wf-XML, can be used to implement the three models of interoperability defined in the Interoperability Abstract specification. Specifically, chained workflows, nested workflows and parallel-synchronized workflows are supported. Wf-XML supports these three types of interchanges both synchronously and asynchronously, and allows messages to be exchanged individually or in batch operations. Furthermore, this specification describes a language that is independent of any particular implementation mechanism, such as programming language, data transport mechanism, OS/hardware platform, etc. However, because HTTP is expected to be the most prevalent data transport mechanism used for interchanging Wf-XML messages, this specification provides a description of how Wf-XML messages are to be interchanged using this protocol.

2.3 Scope

The scope of this specification is equivalent to that defined by the Interoperability Abstract specification.

2.4 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.

2.5 Background

This specification is 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. Subsequently, the following documents comprise the basis of this specification:

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

Readers of this specification are encouraged to familiarize themselves with these documents in order to gain a more comprehensive understanding of the concepts that provide its foundation.

2.6 Document Status

This document is a publication of the Workflow Management Coalition (WfMC), representing version 1.1 of the Wf-XML specification. It may be obtained via the Internet from: http://www.aiim.org/wfmc/members/docs/Wf-XML-11.doc, or by E-mailing a request to: wfmc@wfmc.org.

2.7 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 character stream.
- In addition to the examples that appear throughout this document, extracts from the Document Type Definition (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)>
```

- Where a description of a generic construct is necessary, the generic construct will appear in *Italics*. For example, where operations are discussed in general without reference to any particular operation, the reference will appear as "*OperationName*". This string will be replaced by the name of a specific operation later in the document.

3 Technical Specification (Normative)

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.

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. An additional group named Control also exists to support certain optional functionality in this version of the specification. 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. A more detailed discussion of conformance to this specification is discussed in section 6: Conformance.

The Control group of operations serves to support the protocol level functions required to maintain interoperability among generic services. Currently, this group is used to enable the monitoring and control of batch messages only. However, it may also prove useful in future versions of this specification to support more dynamic forms of interoperability.

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 information about events occurring during its execution, such as 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. Furthermore, in parallel-synchronized processes (where each process may play the role of an observer) there must be a way for each process to be informed of events or changes in the other. Finally, third-party resources may have an interest in the status of a given process instance for various organizational reasons. The Observer group will provide this information by giving a process instance the resource identifier of the requestor, which will be the observer of that process instance. If other resources are to be notified of events occurring in the process instance, it is incumbent upon the observer to pass along information about events that it receives to those resources. Diagram 1 indicates the relationship between the primary groups of operations explained above:

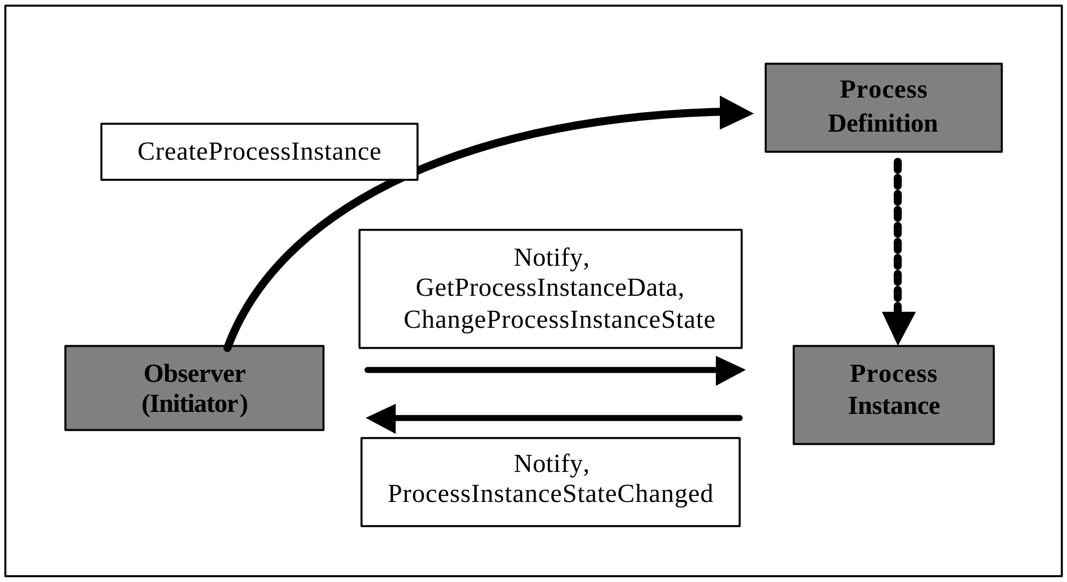
Process Definition
CreateProcessInstance
Notify, GetProcessInstanceData, ChangeProcessInstanceState
Observer (Initiator)
Process Instance
Notify, ProcessInstanceStateChanged

Diagram 1: Primary Operation Groups

3.2 Logical Interaction Model

In this specification, an “interaction” is considered to be the exchange between two generic services of protocol-related information. Wf-XML uses a “message” as the vehicle for providing interactions among generic services. Three types of interactions; called “Request”, “Acknowledgement” and “Response”, are used in messages exchanged between Wf-XML enabled services. A Request is used by a resource (A) to initiate an operation in a second resource (B), and/or to provide input to that resource.

An Acknowledgement is used in asynchronous implementations by a resource receiving a Wf-XML message to inform the sender that the message has been received. It should be noted that an Acknowledgement is used to acknowledge a message, as opposed to the interaction(s) contained in that message. In this case, the sender and receiver can be A or B depending on the message being acknowledged, which can contain an individual Request or Response, or a batch of interactions.

A Response is used by an enacting resource (B) to send the results of an operation to its requesting resource (A), providing output. Although the request and response interaction 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 (which also uses the names Request/Response), not every Wf-XML request requires a response.

3.2.1 Synchronous Messaging

In a synchronous exchange a resource (A) may wish to initiate a sub-process in a second resource (B) and suspend its normal processing until that sub-process completes, at that point becoming an observer of the sub-process. This lifecycle actually requires two separate synchronous exchanges. As shown in Diagram 2, the initiating resource (A) sends a request to the enacting resource (B), which sends back a response (to A) indicating that the process has been initiated. When the enacting resource (B) completes the process, it sends a request message to the initiating resource (A) to inform it of the completion. This message may require no response, as it is merely informational, but is referred to as a “Request” message nonetheless.

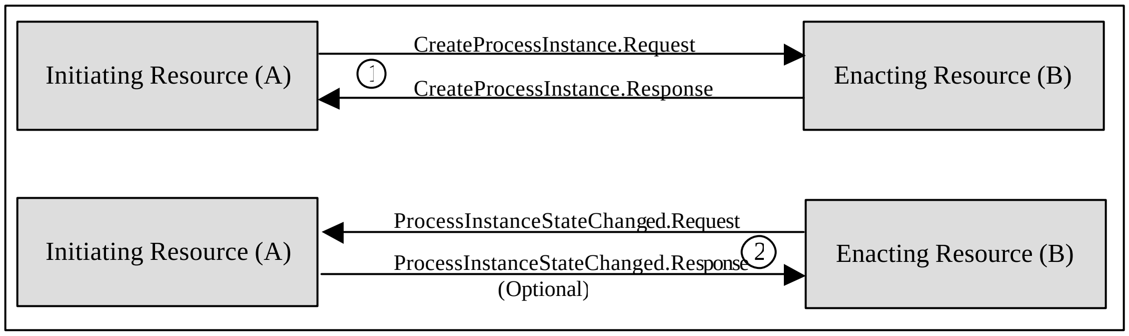
Initiating Resource (A)
CreateProcessInstance.Request
1
CreateProcessInstance.Response
Enacting Resource (B)
Initiating Resource (A)
ProcessInstanceStateChanged.Request
ProcessInstanceStateChanged.Response
(Optional)
2
Enacting Resource (B)

Diagram 2: Synchronous Message Exchange

3.2.2 Asynchronous Messaging

In an asynchronous exchange, as shown in Diagram 3, the initiating resource (A) sends a request to the enacting resource (B) to create a new process instance. The enacting resource (B) then sends an acknowledgement back to the initiator (A) informing it that the request has been received. This positive acknowledgement serves only to indicate that a message has been received and does not imply any additional semantics, such as the processing status of the operation. Exception or status information must be returned through subsequent protocol messages. Additional requirements for negative acknowledgements or other guaranteed messaging semantics should be handled at the application level.

At some later point in time, the enacting resource (B) sends a response to the initiating resource (A) indicating that the requested process instance has been created. The initiating resource (A) then sends an acknowledgement (to B) indicating that it received the response. Again, this acknowledgement serves only to indicate that the response message was received.

When the process being enacted by the enacting resource (B) subsequently completes, that resource (B) sends a request to the initiating resource (A) to inform it of the completion. The initiating resource (A) then sends an acknowledgement (to B) indicating that it received the request. In this case, the Request may require no response since it is only informational, and so the exchange ends here.

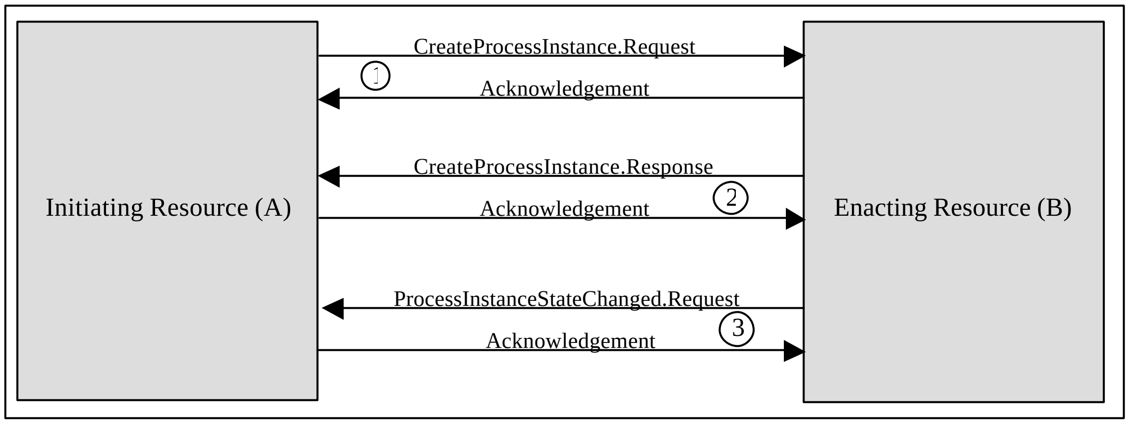
Initiating Resource (A)
CreateProcessInstance.Request
1
Acknowledgement
CreateProcessInstance.Response
2
Acknowledgement
ProcessInstanceStateChanged.Request
3
Acknowledgement
Enacting Resource (B)

Diagram 3: Asynchronous Message Exchange

3.2.3 Batch Messaging

In addition to the individual exchange of interactions described above, it is also desirable in some circumstances to exchange multiple Wf-XML interactions in a single message. This type of “batch” processing can be useful in high-volume transactional situations, such as EDI-style transactions. This specification describes a data format suitable for managing both individual and batch processing.

When processing batch Wf-XML messages, the interaction types “Request” and “Response” defined above apply to individual operations within the batch message, whereas the type “Acknowledgement” always applies to a message as a whole. This is an important distinction in the batch processing model, as an implementation may choose to combine Requests and Responses in a single batch message as appropriate for a given purpose. However, only a single Acknowledgement is required for an entire batch message, regardless of how many operations the batch message contains.

When exchanging a batch of interactions, the batch may contain Requests only, a combination of Requests and Responses or Responses only, as appropriate to the situation. While this batching of interactions may be convenient, an implementation may also choose to send Individual Responses to operations requested via a Batch message. This approach can prove useful for incremental progress tracking or partial result processing. The following diagrams illustrate a hypothetical batch messaging interchange utilizing these techniques. This scenario also utilizes asynchronous processing in order to illustrate the combined usage of these processing models.

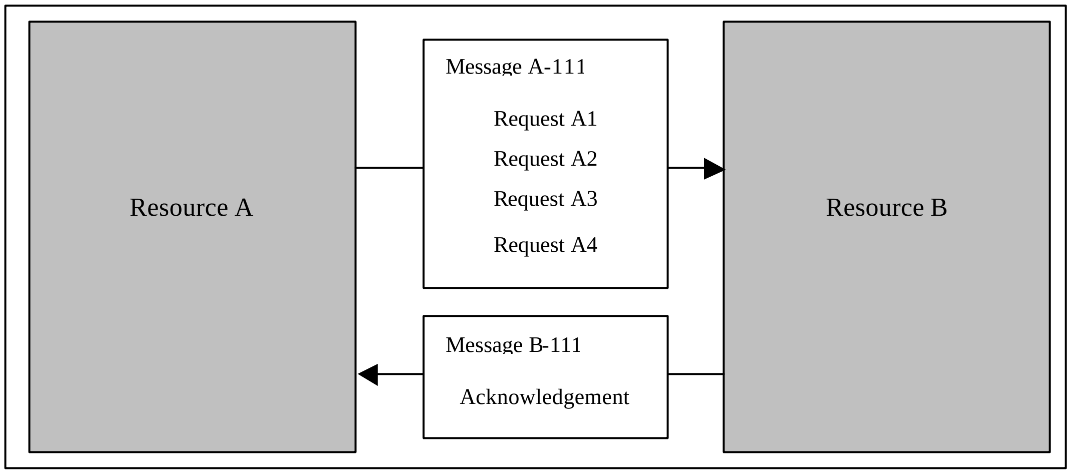

Diagram 4: Initial Batch Message

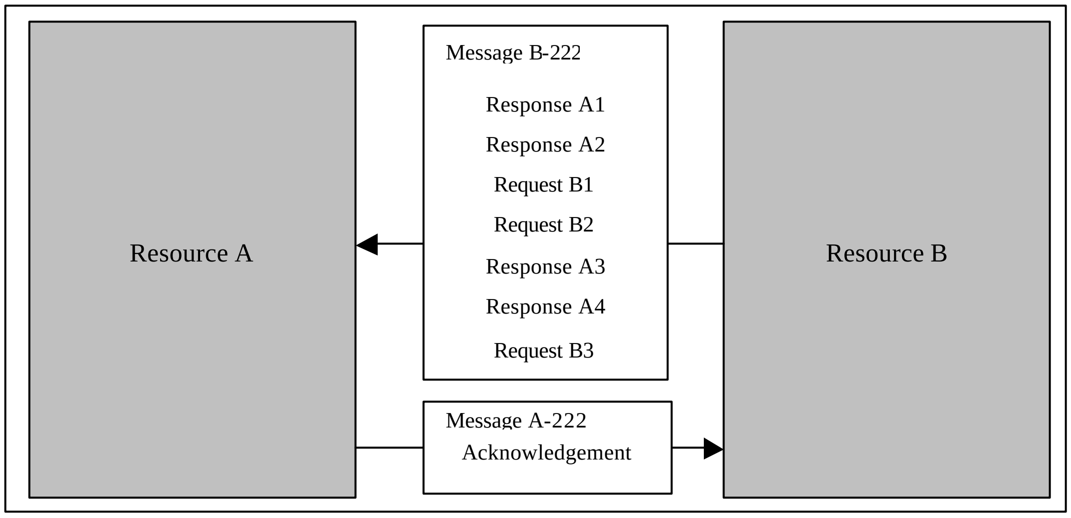

Diagram 5: Batch Message with Combined Interaction Types

Note that in these scenarios neither resource is explicitly labeled as an initiating or enacting resource, since they each serve both roles at some point in the execution of their various business processes.

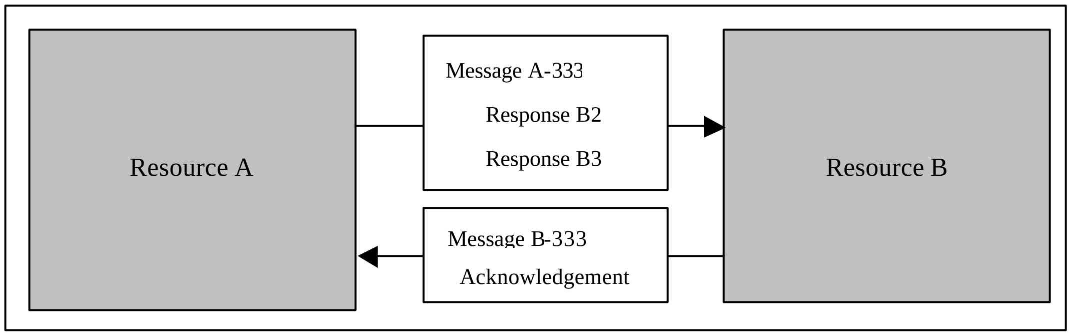

Diagram 6: Batch Message with Partial Result

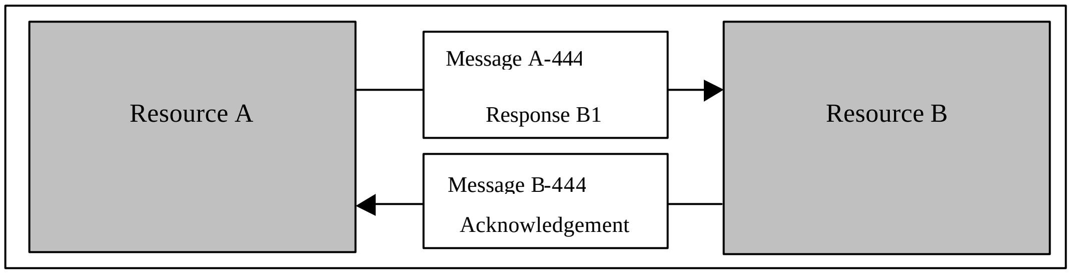

Diagram 7: Individual Response to Batch Requested Operation

While this scenario illustrates one possible message exchange using the processing models provided, there are clearly many other ways these messages and interaction types can be combined to accommodate different process management requirements.

3.3 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.

3.4 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.

3.4.1 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 the applications they invoke, and is likely to be marked up with languages 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 URI [14] will be used as the namespace identifier 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 identifier 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">.
```

Note however, that if explicit namespace prefixing is used on an element, all children of that element belonging to the Wf-XML namespace must also be prefixed. Although the namespace prefix "wf" used above is recommended, the Wf-XML namespace identifier may be bound to any prefix to avoid prefix collision with qualified element names outside the scope of this specification.

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 due to the complexity involved in validating multiple-namespace documents against a DTD, no support is provided for this functionality in the Wf-XML DTD. Therefore, Wf-XML documents requiring multiple namespaces are only required to be well formed.

3.4.2 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:

- Boolean – value may be either "True" or "False"
- Integer – a numeric value containing no decimal precision component. Data fields of this type may be further constrained where used in this specification.
- String – value may contain a sequence of characters of arbitrary length
- Date – value may contain a date/time specification as described in section 3.4.2.1
- URI – value may contain a string conforming to the Universal Resource Identifier (URI) specification [14]. It should be noted that this is not required to be an absolute URI. In certain circumstances, it may only be necessary to provide a local identifier resolvable by the service processing a message. Furthermore, an implementation may wish to maintain base URIs internally, thereby only requiring a relative URI within the Wf-XML message. In these cases, semantics and mechanisms for processing these relative URIs should be agreed upon in the interoperability contract.
- UUID - value may contain a string conforming to the UUID specification [15]

Where no specific data type is indicated for a value, the type will default to String. Within Wf-XML messages, context-specific data conforming to other specifications may be exchanged as described in section 3.4.7, "Representation of Process Context and Result Data". This data is not subject to the data type constraints of this specification and should be validated based on the specification to which it conforms.

3.4.2.1 Date and Time Values

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 Date/Time 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.

3.4.3 Overall Message Structure

The following DTD segment defines the top-level (or "root") element of a Wf-XML message:

```
<!ELEMENT WfMessage ((WfTransport, (WfMessageHeader, WfMessageBody) *) | (WfMessageHeader,
WfMessageBody))>

<!ATTLIST WfMessage Version CDATA #FIXED "1.1"
                    xml:space (default | preserve) #IMPLIED
                    xml:lang NMTOKEN #IMPLIED>
```

This root element is named "WfMessage", and it carries a required attribute named "Version", as well as the reserved XML attributes xml:space and xml:lang. These constructs have the following semantic constraints and meaning:

Version – The value of this attribute indicates the particular version of this specification with which this message conforms. It may be used by an implementation to determine whether this message can be processed. If the service receiving this message cannot support the version of the specification to which it conforms it must return a response containing appropriate exception information, as described in section 3.4.9.

xml:space – This attribute is used to indicate whether whitespace within this element is to be ignored or preserved, as specified by the XML 1.0 recommendation [6].

xml:lang – This attribute is used to indicate the natural language used within this element, as specified by the XML 1.0 recommendation [6].

Within the root WfMessage element, each Wf-XML message contains the following structure:

- An optional section for transport-specific information named “WfTransport”. If necessary, this section will be used to convey information relevant to a particular implementation’s transport protocol. For the purposes of asynchronous processing, this section will be used to convey acknowledgement information. For the purposes of batch processing, this section will be used to indicate that the special processing is required. Therefore, whenever asynchronous and/or batch processing is being performed this section of the message must be present.
- Zero or more message headers named “WfMessageHeader”. The message header will contain information relevant to routing and preprocessing of the message. The message header must not be present in an acknowledgement message, in which all acknowledgement information will be conveyed in the transport section. A single message header must be present to perform an individual operation. Multiple message headers may be present if this message is to be processed as a batch. In this case, each message header must be accompanied by a message body.
- Zero or more message bodies named “WfMessageBody”. The operation specific information is placed in the message body. The message body must not be present in an acknowledgement message, in which all acknowledgement information will be conveyed in the transport section. A single message body must be present to perform an individual operation. Multiple message bodies may be present if this message is to be processed as a batch. In this case, each message body must be accompanied by a message header.

Therefore, the skeleton of a Wf-XML message will appear as follows in an individual operation (with the optional transport section included):

Example 1:
```
<?xml version=”1.0”?>
<WfMessage xmlns=”http://www.wfmc.org/standards/docs/Wf-XML” Version=”1.1”>
   <WfTransport>
          …
   </WfTransport>
   <WfMessageHeader>
          …
   </WfMessageHeader>
   <WfMessageBody>
          …
   </WfMessageBody>
</WfMessage>
```

and as follows in a batch operation:

Example 2:
```
<?xml version=”1.0”?>
<WfMessage xmlns=”http://www.wfmc.org/standards/docs/Wf-XML” Version=”1.1”>
   <WfTransport>
          …
   </WfTransport>
   <WfMessageHeader>
          …
   </WfMessageHeader>
   <WfMessageBody>
          …
   </WfMessageBody>
   <WfMessageHeader>
          …
   </WfMessageHeader>
   <WfMessageBody>
          …
   </WfMessageBody>
          …
</WfMessage>
```

Lastly, the message skeleton would appear as follows in an acknowledgement message (used during asynchronous processing), with additional details of the acknowledgement information specified in section 3.4.4:

Example 3:
```
<?xml version=”1.0”?>
<WfMessage xmlns=”http://www.wfmc.org/standards/docs/Wf-XML” Version=”1.1”>
   <WfTransport>
         …
   </WfTransport>
</WfMessage>
```

3.4.4 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 integrity, reliable messaging (retransmission, duplication detection), session management, etc. However, it will often be necessary to provide information to support these capabilities in a Wf-XML message. It is for this reason that the WfTransport section is provided. This section of a Wf-XML message is optional and contains markup constructs designed to facilitate the implementation of asynchronous and batch transport mechanisms. Details appearing in this section regarding the requirements of any particular transport should be specified by the binding protocol for that transport.

The following DTD extract illustrates the predefined structure of the WfTransport section:

```
<!ELEMENT WfTransport (Dialog?, CorrelationData?, Exception?)>
<!ELEMENT Dialog ((Acknowledgement, Key) | (ReplyToKey, Key?) | Key)?>
<!ATTLIST Dialog Type (synch | asynch) ”synch”
                 Mode (individual | batch) ”individual”
                 MessageID CDATA #IMPLIED>
<!ELEMENT Acknowledgement EMPTY>
<!ATTLIST Acknowledgement ReceivedAt CDATA #REQUIRED>
<!ELEMENT Key (#PCDATA)>
<!ELEMENT ReplyToKey (#PCDATA)>
<!ELEMENT CorrelationData (#PCDATA)>
```

These elements and attributes can be combined in a number of different ways to support a variety of messaging models (individual/synchronous, individual/asynchronous, batch/synchronous or batch/asynchronous), provided they adhere to the semantic constraints specified herein. Each of them, as well as the entire WfTransport element, is optional in order to allow specific transport bindings and implementations to use them as necessary. If the entire WfTransport element is omitted, the default messaging model is individual/synchronous. These structures have the following semantic constraints and meanings:

Dialog – This element contains information relevant to the kind of dialog being established between interoperating services, such as whether responses are to be handled synchronously or asynchronously and whether this message contains a single or multiple interactions. Specific characteristics of the processing required to support this message are described by the elements and attributes below. If this element is omitted the messaging model defaults to individual/synchronous. (optional)

Type – This attribute is used to indicate whether the message should be handled synchronously or asynchronously. If this attribute’s value is “synch”, this message must be handled synchronously. In this case, no further communication will occur until the requested processing is completed. Upon completion of the requested processing, a response must be returned to the requesting service immediately.

If this attribute's value is "asynch", this message must be handled asynchronously. In this case, the receiving service must return an acknowledgement to the initiating service upon receipt of a request. Upon completion of the requested processing, a response may be returned to the initiating service. If a response is returned, the initiating service must send an acknowledgement to the enacting service upon receipt of the response.

If this attribute is omitted, the message type defaults to "synch". (optional)

Mode –The value of this attribute indicates the kind of processing required for this message to the resource receiving it. If the value of this attribute is "individual", the message should be processed as a single interaction. If this attribute's value is "batch", additional information required for batch processing will be specified elsewhere in this message. If this attribute is omitted, the message's mode defaults to "individual". (optional)

MessageID –This attribute must be present when asynchronous or batch processing is being performed. It contains a unique identifier used to correlate an acknowledgement of a message, and/or to identify a batch message in control operations. In an acknowledgement, this attribute's value must correspond with the value of the same attribute in the message to which the acknowledgement relates. The value of this attribute must be of type UUID. (optional)

Acknowledgment – The presence of this element indicates that this is an acknowledgement, used in asynchronous processing. Therefore, this element must not be present if the value of the Type attribute on the Dialog element is set to "synch". If this element is present the message must not contain a message header or message body. Furthermore, an Acknowledgement must only acknowledge a single message and must be processed individually. Therefore, the Mode attribute on the Dialog element must be set to "individual" when this element is present, as an Acknowledgement message must not require batch processing information. Receipt of this message indicates that the corresponding message, identified by the value of the MessageID attribute described above, has been received. (optional)

ReceivedAt – The value of this attribute indicates the time at which the acknowledged message was received by the recipient. The value of this attribute must be of type Date.

Key – This element is used within the transport section of a message when batch and/or asynchronous processing is being performed. It supplements the Key element in the message header (described below) in two ways:

- When batch processing is being performed the message contains multiple headers, making it impossible to use the Key element in the header to route the message. Therefore, this element provides the identifier of the resource to which a batch message is to be sent.
- When asynchronous processing is being performed there is no way to include a Key in an Acknowledgement message, since it contains no header. Therefore, this element provides the identifier of the resource to which an Acknowledgement is to be sent.

The contents of this element must be of type URI. (optional)

ReplyToKey – This element must be present in messages containing Requests or Responses when asynchronous processing is being performed. It contains the identifier of the resource to which an acknowledgement or response to this message should be sent. This element should not be present in messages containing an Acknowledgement. The contents of this element must be of type URI. (optional)

CorrelationData – This element contains implementation-specific data and or structures necessary to correlate message traffic between interoperating resources. Since this data will be particular to the interaction between two interoperating resources, the specific details of its structure and format must be agreed upon in the interoperability contract. This element is maintained in this version of the specification for backward compatibility. (optional)

Exception – the Exception element that appears here is used to describe any errors that may have been encountered relative to the transport section. The content of this element is described in section

3.4.9 Error Handling. When this element is used in the transport section of a message, the exception code 800 “WF_OTHER” should be used and extended as necessary to convey the specifics of the error.

The following examples illustrate some of the possible ways the structures in the WfTransport section may be used to support various processing models. Example 4 illustrates the transport section for a synchronous exchange of an individual message.

Example 4:

```
<?xml version=”1.0”?>
<WfMessage xmlns=”http://www.wfmc.org/standards/docs/Wf-XML” Version=”1.1”>
   <WfTransport>
      <Dialog Type=”synch” Mode=”individual”/>
   </WfTransport>
         …
</WfMessage>
```

It should also be noted that given this processing model, any or all of the Type attribute, Mode attribute, Dialog element or WfTransport element could have been omitted without impacting the semantics of the message processing, as this is the default behavior. Example 5 illustrates the transport section for an asynchronous exchange of an individual request or response.

Example 5:

```
<?xml version=”1.0”?>
<WfMessage xmlns=”http://www.wfmc.org/standards/docs/Wf-XML” Version=”1.1”>
   <WfTransport>
      <Dialog Type=”asynch” Mode=”individual” MessageID=”5a98d32e-7854-c751-
      5491-7d4a0c4e7102”>
         <ReplyToKey>http://www.myco.com/purchasing/orders</ReplyToKey>
      </Dialog>
   </WfTransport>
         …
</WfMessage>
```

Example 6 illustrates the transport section for an asynchronous exchange of a batch message.

Example 6:

```
<?xml version=”1.0”?>
<WfMessage xmlns=”http://www.wfmc.org/standards/docs/Wf-XML” Version=”1.1”>
   <WfTransport>
      <Dialog Type=”asynch” Mode=”batch” MessageID=”5a98d32e-7854-c751-5491-
      8f55e8a210ba”>
         <ReplyToKey>http://www.myco.com/purchasing/orders</ReplyToKey>
         <Key>http://www.exampleco.com/processes</Key>
      </Dialog>
   </WfTransport>
         …
</WfMessage>
```

Example 7 illustrates an acknowledgement of the batch message in example 6. Note that the value of the Mode attribute on the Dialog element has been set to “individual” although this is an acknowledgement of a batch message. This is because the Mode attribute indicates the processing mode required to handle this message, not a message it might reference. Also, note that the message being acknowledged is correlated via the MessageID.

Example 7:

```
<?xml version=”1.0”?>
<WfMessage xmlns=”http://www.wfmc.org/standards/docs/Wf-XML” Version=”1.1”>
   <WfTransport>
      <Dialog Type=”asynch” Mode=”individual” MessageID=”5a98d32e-7854-c751-
      5491-8f55e8a210ba”>
         <Acknowledgement ReceivedAt=”2001-09-24T16:31:05Z”>
         <Key>http://www.myco.com/purchasing/orders</Key>
      </Dialog>
   </WfTransport>
</WfMessage>
```

3.4.5 Message Header Definition

The message header contains information that is generically useful to all interactions, such as resource identifiers, operation names, etc. Separation of this information from the message body enables pre-processing of Wf-XML 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|h
a|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|n
a|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|t
o|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

                  RequestID CDATA #IMPLIED>

<!ELEMENT Response EMPTY>

<!ATTLIST Response RequestID CDATA #IMPLIED>
```

These structures have the following semantic constraints and meanings:

Request – The presence of this element indicates that this interaction is a request. If this element is used it must carry the ResponseRequired attribute described below. It may also optionally specify the ResponseLang attribute. (optional)

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 may, but need not 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. (optional)

RequestID – This attribute of the Request element is used in asynchronous and batch processing to uniquely identify a request (potentially within a batch of interactions) so that it can later be correlated with its response. The value of this attribute must be of type UUID. (optional)

Response – The presence of this element indicates that this interaction is a response. Receipt of this interaction indicates that processing of the corresponding request has been completed. (optional)

RequestID – This attribute of the Response element is used in asynchronous and batch processing to correlate a response with its request. The value of this attribute must correspond to the value of the RequestID attribute on a Request previously sent by the resource receiving this response. The value of this attribute must be of type UUID. (optional)

Key – This element contains the identifier of the resource that is the target of this request, or the source of this response. In the case of batch message processing, message routing is indicated by the Key element in the transport section, and the content of this Key must be used by the Wf-XML processor to identify the particular resource to which this operation applies. The content of this element must be of type URI.

For example, an asynchronous individual request message would appear as follows:

Example 8:

```
<?xml version=”1.0”?>
<WfMessage xmlns=”http://www.wfmc.org/standards/docs/Wf-XML” Version=”1.1”>
   <WfTransport>
      <Dialog Type=”asynch” MessageID=”4308d23b-e78c-2390-6271-743891d60a52”>
         <ReplyToKey>http://www.myco.com/purchasing/orders/4089259</ReplyToKey>
      </Dialog>
   </WfTransport>
   <WfMessageHeader>
      <Request ResponseRequired=”Yes” RequestID=”4308d23b-675d-3b47-0931-
      768c4a0528b3”/>
      <Key>http://www.exampleco.com/processes/86947325</Key>
   </WfMessageHeader>
   <WfMessageBody>
               …
   </WfMessageBody>
</WfMessage>
```

An acknowledgement of this request would appear as in example 9.

Example 9:

```
<?xml version=”1.0”?>
<WfMessage xmlns=”http://www.wfmc.org/standards/docs/Wf-XML” Version=”1.1”>
   <WfTransport>
      <Dialog Type=”asynch” MessageID=”4308d23b-e78c-2390-6271-743891d60a52”>
         <Acknowledgement ReceivedAt=”2001-09-24T16:48:30Z”/>
         <Key>http://www.myco.com/purchasing/orders/4089259</Key>
      </Dialog>
   </WfTransport>
</WfMessage>
```

A response to this request would appear as in example 10.

Example 10:

```
<?xml version=”1.0”?>
<WfMessage xmlns=”http://www.wfmc.org/standards/docs/Wf-XML” Version=”1.1”>
   <WfTransport>
      <Dialog Type=”asynch” MessageID=”4308d23b-430b-29e0-a867-32087b581afe”>
         <ReplyToKey>http://www.exampleco.com/processes/86947325</ReplyToKey>
      </Dialog>
   </WfTransport>
   <WfMessageHeader>
      <Response RequestID=”4308d23b-675d-3b47-0931-768c4a0528b3”/>
      <Key>http://www.myco.com/purchasing/orders/4089259</Key>
   </WfMessageHeader>
   <WfMessageBody>
         …
   </WfMessageBody>
</WfMessage>
```

Given that this example illustrates an asynchronous scenario, this response would be acknowledged as follows:

Example 11:

```
<?xml version=”1.0”?>
<WfMessage xmlns=”http://www.wfmc.org/standards/docs/Wf-XML” Version=”1.1”>
   <WfTransport>
      <Dialog Type=”asynch” MessageID=”4308d23b-430b-29e0-a867-32087b581afe”>
         <Acknowledgement ReceivedAt=”2001-09-24T16:54:14Z”/>
         <Key>http://www.exampleco.com/processes/86947325</Key>
      </Dialog>
   </WfTransport>
</WfMessage>
```

3.4.6 Message Body Definition

The message body provides operation specific data. For a request, it contains an <*OperationName*.Request> element, which further contains request parameters; for a response, it contains an <*OperationName*.Response> element, containing result parameters returned by the 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). Therefore, the content of the WfMessageBody element is defined as follows:

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

The entities referenced here contain element declarations for specific operations as described above, and so will be specified in the Operations section of this document.

This model would have the following structure for an asynchronous batch of request interactions. Note that multiple types of interactions (requests and responses) may be combined within a batch message. The examples shown here provide only one of the possible scenarios in order to illustrate the cycle of asynchronous processing.

Example 12:

```
<?xml version="1.0"?>
<WfMessage xmlns="http://www.wfmc.org/standards/docs/Wf-XML" Version="1.1">
   <WfTransport>
      <Dialog Type="asynch" Mode="batch" MessageID="e85327bc-dc9e-2878-4b52-
      947852107643">
         <ReplyToKey>http://www.myco.com/purchasing/orders</ReplyToKey>
         <Key>http://www.exampleco.com/processes</Key>
      </Dialog>
   </WfTransport>
   <WfMessageHeader>
      <Request ResponseRequired="Yes" RequestID="d4253789-ce42-9165-3bed-
      825731d8d941"/>
      <Key>http://www.exampleco.com/processes/86947325</Key>
   </WfMessageHeader>
   <WfMessageBody>
      <OperationName.Request>
            …
      </OperationName.Request>
   </WfMessageBody>
   <WfMessageHeader>
      <Request ResponseRequired="Yes" RequestID="54879aac-ffe3-8d92-cd74-
      7983547bac21"/>
      <Key>http://www.exampleco.com/processes/79843209</Key>
   </WfMessageHeader>
   <WfMessageBody>
      <OperationName.Request>
            …
      </OperationName.Request>
   </WfMessageBody>
   <WfMessageHeader>
      <Request ResponseRequired="Yes" RequestID="25b76322-8ac6-e509-baca-
      172483dabcf3"/>
      <Key>http://www.exampleco.com/processes/30817842</Key>
   </WfMessageHeader>
   <WfMessageBody>
      <OperationName.Request>
            …
      </OperationName.Request>
   </WfMessageBody>
</WfMessage>
```

An acknowledgement of this message might appear as in example 13.

Example 13:

```
<?xml version="1.0"?>
<WfMessage xmlns="http://www.wfmc.org/standards/docs/Wf-XML" Version="1.1">
   <WfTransport>
      <Dialog Type="asynch" Mode="individual" MessageID="e85327bc-dc9e-2878-
      4b52-947852107643">
         <Acknowledgement ReceivedAt="2001-08-23T17:12:42Z"/>
         <Key>http://www.myco.com/purchasing/orders</Key>
      </Dialog>
   </WfTransport>
</WfMessage>
```

A response to this request might appear as follows:

Example 14:

```
<?xml version="1.0"?>
<WfMessage xmlns="http://www.wfmc.org/standards/docs/Wf-XML" Version="1.1">
   <WfTransport>
      <Dialog Type="asynch" Mode="batch" MessageID="832ba946-8754-4237-e8f0-
      924678a8e031">
         <ReplyToKey>http://www.exampleco.com/processes/order-
         handler.jsp</ReplyToKey>
         <Key>http://www.myco.com/purchasing/orders</Key>
      </Dialog>
   </WfTransport>
   <WfMessageHeader>
      <Response RequestID="d4253789-ce42-9165-3bed-825731d8d941"/>
      <Key>http://www.exampleco.com/processes/86947325</Key>
   </WfMessageHeader>
   <WfMessageBody>
      <OperationName.Response>
            …
      </OperationName.Response>
   </WfMessageBody>
   <WfMessageHeader>
      <Response RequestID="54879aac-ffe3-8d92-cd74-7983547bac21"/>
      <Key>http://www.exampleco.com/processes/79843209</Key>
   </WfMessageHeader>
   <WfMessageBody>
      <OperationName.Response>
            …
      </OperationName.Response>
   </WfMessageBody>
   <WfMessageHeader>
      <Response RequestID="25b76322-8ac6-e509-baca-172483dabcf3"/>
      <Key>http://www.exampleco.com/processes/30817842</Key>
   </WfMessageHeader>
   <WfMessageBody>
      <OperationName.Response>
            …
      </OperationName.Response>
   </WfMessageBody>
</WfMessage>
```

Finally, an acknowledgement of this response might appear as follows:

Example 15:

```
<?xml version="1.0"?>
<WfMessage xmlns="http://www.wfmc.org/standards/docs/Wf-XML" Version="1.1">
   <WfTransport>
      <Dialog Type="asynch" Mode="batch" MessageID="832ba946-8754-4237-e8f0-
      924678a8e031">
         <Acknowledgement ReceivedAt="2001-09-24T17:32:10Z"/>
         <Key>http://www.exampleco.com/processes/order-handler.jsp</Key>
      </Dialog>
      </WfTransport>
</WfMessage>
```

3.4.7 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 when the process is being instantiated, and the "result" of the process when the process has been completed/terminated. 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.

The data within these elements may take many forms, depending on the type of data being exchanged and the particular requirements of an implementation. The structure of this data may also vary from process definition to process definition. Therefore, it is recommended that a detailed description of context and result data exchange requirements be agreed upon in an interoperability contract between interoperating services.

Because the nature and definition of this context data cannot be known (as it is particular to a given process definition), it can be difficult 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. The reserved attributes xml:space and xml:lang are provided here, in addition to the root element, in order to allow these characteristics to be overridden for context-specific data. 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>
<!ATTLIST ContextData xml:space (default | preserve) #IMPLIED
                      xml:lang NMTOKEN #IMPLIED>
<!ELEMENT ResultData ANY>
<!ATTLIST ResultData xml:space (default | preserve) #IMPLIED
                     xml:lang NMTOKEN #IMPLIED>
```

While the flexibility provided by this content model is essential, a more structured modeling of this data would allow for enhanced interoperability by providing a means by which a Wf-XML processor could distinguish the fields within a context-specific data section. These fields could then be dispatched for separate processing appropriately, as determined by the implementation. In order to foster greater levels of interoperability, this version of the Wf-XML specification provides the following markup to be used to specify parameters in the ContextData and ResultData elements.

```
<!ELEMENT Parameter (Name, Value+)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Value (#PCDATA)>
```

These elements have the following semantic constraints and meanings:

Parameter – This element provides a bounding mechanism used to indicate that its contents constitute a single parameter to the process on which an operation is being performed. Two properties of this parameter, Name and Value, are provided as described below. However, further syntactic and semantic constraints of this parameter are to be determined by the agreements set forth in the interoperability contract. This element may appear zero or more times within the ContextData and/or ResultData elements.

Name – This element simply provides an identifier for the parameter. The syntactic and semantic constraints applicable to this element are to be determined by the agreements set forth in the interoperability contract, and any additional markup required here should be defined in that contract.

Value – The content of this element constitutes the value assigned to the parameter. Multiple values may be specified by including multiple "Value" elements within the parameter. The syntactic and semantic constraints applicable to this element are to be determined by the agreements set forth in the interoperability contract, and any additional markup required here should be defined in that contract.

The following example illustrates how this markup may be used to exchange context-specific data, assuming appropriate agreements have been made in the interoperability contract regarding the content of the Value element in the "VehDesc" parameter.

Example 16:

```
<?xml version="1.0"?>
<WfMessage xmlns="http://www.wfmc.org/standards/docs/Wf-XML" Version="1.1">
   <WfMessageHeader>
      …
   </WfMessageHeader>
   <WfMessageBody>
      …
      <ContextData>
         <Parameter>
            <Name>POID</Name>
            <Value>3878547</Value>
         </Parameter>
         <Parameter>
            <Name>OrderConf</Name>
            <Value>http://www.exampleco.com/orders/3878547</Value>
         </Parameter>
         <Parameter>
            <Name>VehDesc</Name>
            <Value>
               <Vehicle>
                  <VehicleType>Car</VehicleType>
                  <Specification>
                     <Manufacturer>Audi</Manufacturer>
                     <Model>A4</Model>
                  </Specification>
               </Vehicle>
            </Value>
         </Parameter>
      </ContextData>
            …
   </WfMessageBody>
</WfMessage>
```

For compatibility reasons it cannot be mandated that this markup replaces the existing v1.0 content models. Therefore, although legacy (v1.0) content must be accepted by a version 1.1 compliant implementation, such an implementation should generate this Parameter markup as the content of the Context Data and ResultData elements in all cases, as this is the preferred way to achieve interoperability in this version of the specification. Using this markup allows an implementation to rely on receiving a parsable structure that can be delegated to an appropriate routine for further processing. For this reason, the ANY content model extensibility provisions from Wf-XML v1.0 are hereby deprecated.

3.4.8 Status

Another important aspect of a process is the state that 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 here must be supported, an implementation may choose to omit the optional states or add additional states to those defined.

```
<!ENTITY % states "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>
```

These structures have the following semantic constraints and meanings:

open.notrunning – A resource is in this state when it has been instantiated, but is not currently participating in the enactment of a work process.

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 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 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 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)

These states are also used when performing batch message processing (by operations described in section 3.5.1 Control Operations) to indicate the overall status of a batch message. This status should not be confused with the status of processes initiated by these messages. Rather, it is the state of the message processing itself that is indicated. This state reflects the progress made on the processing of each individual operation that comprises the batch message, so that (for example) the batch is not considered complete until each individual operation is complete. Therefore, in the context of batch message processing, these structures have the following semantic constraints and meanings:

open.notrunning – A message is in this state when it has been received, but is not currently being processed. For example, the message may be in some type of internal queue awaiting processing.

open.notrunning.suspended – A message is in this state when it has been received and its processing has been initiated, but it is not currently being processed. This state may be the result of an explicit state change request or normal internal processing delays. (optional)

open.running – A message is in this state when it is currently being processed. Some of the operations in the batch may have been completed at this point, others may be in-progress and still others may not yet have been initiated.

closed.completed – A message is in this state when it has been completely processed. This means that all operations within the batch have been processed successfully. Responses to any requested operations in the batch will be (or have been) sent to the requestor.

closed.abnormalCompleted – A message is in this state when its processing has been completed abnormally. This means that one or more operations in the batch were not processed successfully, although some operations in the batch may have been completed successfully. Responses to any requested operations in the batch will be (or have been) sent to the requestor.

closed.abnormalCompleted.terminated – A message is in this state when its processing has been cancelled by the initiator. Some of the operations in the batch may have been completed at this point. Responses to any requested operations in the batch that have been completed will be (or have been) sent to the requestor. Operations that have not been completed when the message is terminated are ignored. (optional) closed.abnormalCompleted.aborted – A message is in this state when its processing has been abnormally ended as the result of an internal processing error. Some of the operations in the batch may have been completed at this point. Responses to any requested operations in the batch that have been completed will be (or have been) sent to the requestor. Operations that have not been completed when the message is terminated are ignored. (optional)

3.4.9 Error Handling

Should any exception occur during the execution of a Wf-XML operation, information regarding that exception must be returned to the caller. Various types of exceptions can be anticipated, including temporary and fatal error types. Therefore, an element named "Exception" 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 as the contents of the <*OperationName*.Response> element, in lieu of the normal response data, from all operations in which an exception occurs. These structures have the following semantic constraints and meanings:

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 that occurred. It can either be "F" for fatal error or "T" for temporary error.

Subject - This is a one-line text description of the exception.

Description - A several-line text description of the exception, 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 17:

```
<?xml version=”1.0”?>
<WfMessage xmlns=”http://www.wfmc.org/standards/docs/Wf-XML” Version=”1.1”>
   <WfMessageHeader>
      <Response/>
      <Key>http://www.exampleco.com/processes/86947325</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>
```

3.4.9.1 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 exception codes are used in the operations’ specifications.

WfMessageHeader	**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
WF_INVALID_REQUEST_ID	106
Data	**200 Series**
These exceptions deal with incorrect context or result data	
WF_INVALID_CONTEXT_DATA	201
WF_INVALID_RESULT_DATA	202
WF_INVALID_RESULT_DATA_SET	203
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
Internal	**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 currently be invoked on the specified resource.

WF_NO_ACCESS_TO_RESOURCE	500
WF_INVALID_PROCESS_DEFINITION	502
WF_MISSING_PROCESS_INSTANCE_KEY	503
WF_INVA LID_PROCESS_INSTANCE_KEY	504

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
WF_MISSING_ NOTIFICATION_NAME	602
WF_INVALID_NOTIFICATION_NAME	603

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

The relevance of these exceptions to various operations is specified in the operation definitions given in section 3.5. Those definitions also reference "General" exceptions relating to all operations. The following codes are included in these general exceptions: 100 Series (100 - 106), 300, 400, 500, 800. All other exceptions are relevant to specific operations as defined in section 3.5.

3.5 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 Control, ProcessDefinition, ProcessInstance and Observer groups, as well as each of the operations in detail. In order to focus more clearly on the syntax of the operations, the examples in this section will assume synchronous individual processing. However, these operations can of course be processed asynchronously and/or in batch messages as described earlier.

	Control	**Process Definition**	**Process Instance**	**Observer**
GetBatchMessageState	X			
ChangeBatchMessageState	X			
CreateProcessInstance		X		
GetProcessInstanceData			X	
ChangeProcessInstanceState			X	
ProcessInstanceStateChanged				X
Notify				X

Table 1: Wf-XML Operations

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 "(GetBatchMessageState.Request | ChangeBatchMessageState.Request |
CreateProcessInstance.Request | GetProcessInstanceData.Request |  ChangeProcessInstanceState.Request |
ProcessInstanceStateChanged.Request | Notify.Request)">

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

3.5.1 Control Operations

This group of operations is used to affect administrative interactions among interoperating services. Such interactions are typically unrelated to specific processes. This group currently contains the operations GetBatchMessageState and ChangeBatchMessageState.

3.5.1.1 GetBatchMessageState

This operation is used to retrieve the status of a batch message previously sent to a given resource. The state of the batch is described using the status elements provided in section 3.4.8 Status.

```
<!ELEMENT GetBatchMessageState.Request (MessageID)>

<!ELEMENT MessageID (#PCDATA)>

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

<!ELEMENT State (%states;)?>
```

These structures have the following semantic constraints and meanings:

Request Parameters:

MessageID – The unique identifier of the batch message whose status is to be retrieved. The data contained within this element must be of type UUID and must match the MessageID of a batch message previously sent to the resource receiving this request.

***Example 18*:**

```
<?xml version="1.0"?>
<WfMessage xmlns=”http://www.wfmc.org/standards/docs/Wf-XML” Version=”1.1”>
    <WfMessageHeader>
        <Request ResponseRequired=”Yes”/>
        <Key>http://www.exampleco.com/processes</Key>
    </WfMessageHeader>
    <WfMessageBody>
        <GetBatchMessageState.Request>
            <MessageID>e85327bc-dc9e-2878-4b52-947852107643</MessageID>
        </GetBatchMessageState.Request>
    </WfMessageBody>
</WfMessage>
```

Response Parameters:

State – The current state of the batch message, as described in section 3.4.8 Status.

Exceptions:

There are no exceptions specific to this operation. All general exceptions apply, as defined in section 3.4.9.1.

***Example 19*:**

```
<?xml version="1.0"?>
<WfMessage xmlns=”http://www.wfmc.org/standards/docs/Wf-XML” Version=”1.1”>
   <WfMessageHeader>
      <Response/>
      <Key>http://www.exampleco.com/processes</Key>
   </WfMessageHeader>
   <WfMessageBody>
      <GetBatchMessageState.Response>
         <State>
            <open.running/>
         </State>
      </GetBatchMessageState.Response>
   </WfMessageBody>
</WfMessage>
```

3.5.1.2 ChangeBatchMessageState

This operation is used to change the status of a batch message previously sent to a given resource. The new state must be specified by one of the status elements provided in section 3.4.8 Status.

```
<!ELEMENT ChangeBatchMessageState.Request (MessageID, State)>
<!ELEMENT MessageID (#PCDATA)>
<!ELEMENT State (%states;)?>
<!ELEMENT ChangeBatchMessageState.Response (State | Exception)>
```

These structures have the following semantic constraints and meanings:

Request Parameters:

MessageID – The unique identifier of the batch message whose status is to be changed. The data contained within this element must be of type UUID and must match the MessageID of a batch message previously sent to the resource receiving this request.

State – The new state to which the identified batch message is to be changed.

***Example 20*:**

```
<?xml version="1.0"?>
<WfMessage xmlns=”http://www.wfmc.org/standards/docs/Wf-XML” Version=”1.1”>
   <WfMessageHeader>
      <Request ResponseRequired=”Yes”/>
      <Key>http://www.exampleco.com/processes</Key>
   </WfMessageHeader>
   <WfMessageBody>
      <ChangeBatchMessageState.Request>
         <MessageID>e85327bc-dc9e-2878-4b52-947852107643</MessageID>
         <State>
            <closed.abnormalCompleted.terminated/>
         </State>
      </ChangeBatchMessageState.Request>
   </WfMessageBody>
</WfMessage>
```

Response Parameters:

State – The new state to which the identified batch message has been changed. If the request was processed successfully, the contents of this element should match the state requested.

Exceptions:

In addition to all general exceptions defined in section 3.4.9.1, the following exceptions are specifically supported for this operation:

WF_INVALID_STATE_TRANSITION

***Example 21*:**

```
<?xml version="1.0"?>
<WfMessage xmlns=”http://www.wfmc.org/standards/docs/Wf-XML” Version=”1.1”>
   <WfMessageHeader>
      <Response/>
      <Key>http://www.exampleco.com/processes</Key>
   </WfMessageHeader>
   <WfMessageBody>
      <ChangeBatchMessageState.Response>
         <State>
            <closed.abnormalCompleted.terminated/>
         </State>
      </ChangeBatchMessageState.Response>
   </WfMessageBody>
</WfMessage>
```

3.5.2 Process Definition Operations

This group of operations is used to perform actions on process definitions, such as creating process instances based on those definitions. The set of process definitions supported by a given enactment service must be predefined. Currently this group contains only the operation CreateProcessInstance.

3.5.2.1 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 automatically started.

```
<!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 CreateProcessInstance.Response ((ProcessInstanceKey, Name?) | Exception)>
<!ELEMENT ProcessInstanceKey (#PCDATA)>
```

These structures have the following semantic constraints and meanings:

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. The resource specified must be the service requesting the operation. This observer resource (if it is specified) is to be notified of events impacting the execution of this process instance such as state changes, and most notably the completion of the instance. With the ObserverKey being set, the interoperability model of a nested or parallel-synchronized sub-process is implied, 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. If the requested name is not used, the assigned name may be returned with the CreateProcessInstance.Response message to inform the initiator of the new name. (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 upon in the interoperability contract (see section on Process Context and Result Data above).

***Example 22*:**

```
<?xml version="1.0"?>
<WfMessage xmlns=”http://www.wfmc.org/standards/docs/Wf-XML” Version=”1.1”>
    <WfMessageHeader>
        <Request ResponseRequired=”Yes”/>
        <Key>http://www.exampleco.com/processes/86947325</Key>
    </WfMessageHeader>
    <WfMessageBody>
        <CreateProcessInstance.Request StartImmediately=”true”>
          <ObserverKey>http://www.myco.com/purchasing/orders/4089259</ObserverKey>
          <ContextData>
                <Parameter>
                    <Name>VehDesc</Name>
                    <Value>
                        <Vehicle>
                            <VehicleType>Car</VehicleType>
                            <Specification>
                                <Manufacturer>Audi</Manufacturer>
                                <Model>A4</Model>
                            </Specification>
                        </Vehicle>
                    </Value>
                </Parameter>
                <Parameter>
                    <Name>Customer</Name>
                    <Value>John Doe</Value>
                </Parameter>
          </ContextData>
        </CreateProcessInstance.Request>
    </WfMessageBody>
</WfMessage>
```

Response Parameters:

ProcessInstanceKey – URI of the newly created process instance.

Name – The name actually assigned to the newly created process instance by the enacting resource. (optional)

Exceptions:

In addition to all general exceptions defined in section 3.4.9.1, the following exceptions are supported specifically for this operation:

WF_MISSING_PROCESS_INSTANCE_KEY

WF_INVALID_PROCESS_INSTANCE_KEY

WF_INVALID_PROCESS_DEFINITION

WF_INVALID_OBSERVER_FOR_RESOURCE

***Example 23*:**

```
<?xml version="1.0"?>
<WfMessage xmlns="http://www.wfmc.org/standards/docs/Wf-XML" Version="1.1">
   <WfMessageHeader>
      <Response/>
      <Key>http://www.exampleco.com/processes/86947325</Key>
   </WfMessageHeader>
   <WfMessageBody>
      <CreateProcessInstance.Response>
         <ProcessInstanceKey>http://www.exampleco.com/orders/86947325-32914
         </ProcessInstanceKey>
      </CreateProcessInstance.Response>
   </WfMessageBody>
</WfMessage>
```

3.5.3 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.

3.5.3.1 GetProcessInstanceData

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

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

<!ENTITY % states "open.notrunning | open.notrunning.suspended | open.running | closed.completed |
closed.abnormalCompleted.terminated | closed.abnormalCompleted.aborted">

<!ELEMENT GetProcessInstanceData.Request (ResultDataSet?)>

<!ELEMENT ResultDataSet (%ProcessInstanceData;)+>

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

<!ELEMENT Name (#PCDATA)>

<!ELEMENT Subject (#PCDATA)>

<!ELEMENT Description (#PCDATA)>

<!ELEMENT State (%states;)?>

<!ELEMENT ValidStates (%states;)*>

<!ELEMENT ObserverKey (#PCDATA)>

<!ELEMENT ProcessDefinitionKey (#PCDATA)>

<!ELEMENT Priority (#PCDATA)>

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

These structures have the following semantic constraints and meanings:

Request Parameters:

ResultDataSet –This parameter contains a set of properties to be returned, where this set can be all of the properties or a subset of them. Note that the desired properties are specified by including their respective elements within this element. When included here, each property element should be empty. Any content of these contained elements should be ignored by the service receiving this message. If this element is not present, all process instance properties are returned. (optional)

The following example requests all properties of a particular ProcessInstance:

***Example 24*:**

```
<?xml version="1.0"?>
<WfMessage xmlns=”http://www.wfmc.org/standards/docs/Wf-XML” Version=”1.1”>
   <WfMessageHeader>
      <Request ResponseRequired=”Yes”/>
      <Key>http://www.exampleco.com/orders/86947325-32914</Key>
   </WfMessageHeader>
   <WfMessageBody>
      <GetProcessInstanceData.Request/>
   </WfMessageBody>
</WfMessage>
```

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

***Example 25*:**

```
<?xml version="1.0"?>
<WfMessage xmlns=”http://www.wfmc.org/standards/docs/Wf-XML” Version=”1.1”>
   <WfMessageHeader>
      <Request ResponseRequired=”Yes”/>
      <Key>http://www.exampleco.com/orders/86947325-32914</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 (as specified in the Interoperability Contract) that represents the current values resulting from process execution. 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:

In addition to all general exceptions defined in section 3.4.9.1, the following exceptions are supported specifically for this operation:

WF_INVALID_RESULT_DATA

WF_INVALID_RESULT_DATA_SET

WF_INVALID_OBSERVER_FOR_RESOURCE

The following is an example of a response for a GetProcessInstanceData operation:

***Example 26*:**

```
<?xml version="1.0"?>
<WfMessage xmlns=”http://www.wfmc.org/standards/docs/Wf-XML” Version=”1.1”>
   <WfMessageHeader>
      <Response/>
      <Key>http://www.exampleco.com/orders/86947325-32914</Key>
   </WfMessageHeader>
   <WfMessageBody>
      <GetProcessInstanceData.Response>
         <Name>Order32914</Name>
         <Priority>3</Priority>
      </GetProcessInstanceData.Response>
   </WfMessageBody>
</WfMessage>
```

3.5.3.2 ChangeProcessInstanceState

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

```
<!ELEMENT ChangeProcessInstanceState.Request (State)>
<!ELEMENT ChangeProcessInstanceState.Response (State | Exception)>
<!ELEMENT State (%states;)?>
```

These structures have the following semantic constraints and meanings:

Request Parameters:

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

***Example 27*:**

```
<?xml version="1.0"?>
<WfMessage xmlns=”http://www.wfmc.org/standards/docs/Wf-XML” Version=”1.1”>
   <WfMessageHeader>
      <Request ResponseRequired=”Yes”/>
      <Key>http://www.exampleco.com/orders/86947325-32914</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:

In addition to all general exceptions defined in section 3.4.9.1, the following exceptions are supported specifically for this operation:

WF_INVALID_STATE_TRANSITION

WF_INVALID_OBSERVER_FOR_RESOURCE

***Example 28*:**
```
<?xml version="1.0"?>
<WfMessage xmlns=”http://www.wfmc.org/standards/docs/Wf-XML” Version=”1.1”>
   <WfMessageHeader>
      <Response/>
      <Key>http://www.exampleco.com/orders/86947325-32914</Key>
   </WfMessageHeader>
   <WfMessageBody>
      <ChangeProcessInstanceState.Response>
         <State>
            <open.notrunning.suspended/>
         </State>
      </ChangeProcessInstanceState.Response>
   </WfMessageBody>
</WfMessage>
```

3.5.4 Observer Operations

This group of operations allows the requester of work (the Observer of a process instance) to be notified of events and status changes impacting the execution of a process instance. This group contains the operations ProcessInstanceStateChanged and Notify.

3.5.4.1 ProcessInstanceStateChanged

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

<!ELEMENT ProcessInstanceKey (#PCDATA)>

<!ELEMENT State (%states;)?>

<!ELEMENT ResultData ANY>

<!ELEMENT LastModified (#PCDATA)>

<!ELEMENT ProcessInstanceStateChanged.Response (Exception?)>
```

These structures have the following semantic constraints and meanings:

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 29:

```
<?xml version="1.0"?>
<WfMessage xmlns="http://www.wfmc.org/standards/docs/Wf-XML" Version="1.1">
   <WfMessageHeader>
      <Request ResponseRequired="No"/>
      <Key>http://www.myco.com/purchasing/orders/4089259</Key>
   </WfMessageHeader>
   <WfMessageBody>
      <ProcessInstanceStateChanged.Request>
         <ProcessInstanceKey>http://www.exampleco.com/orders/86947325-32914
         </ProcessInstanceKey>
         <State>
            <closed.completed/>
         </State>
         <ResultData>
            <Parameter>
               <Name>VehDesc</Name>
               <Value>
                  <Vehicle>
                     <VehicleType>Car</VehicleType>
                     <Specification>
                        <Manufacturer>Audi</Manufacturer>
                        <Model>A4</Model>
                     </Specification>
                  </Vehicle>
               </Value>
            </Parameter>
            <Parameter>
               <Name>Customer</Name>
               <Value>John Doe</Value>
            </Parameter>
         </ResultData>
      </ProcessInstanceStateChanged.Request>
   </WfMessageBody>
</WfMessage>
```

Response Parameters:

None

Exceptions:

In addition to all general exceptions defined in section 3.4.9.1, the following exceptions are supported specifically for this operation:

WF_INVALID_RESULT_DATA

WF_MISSING_PROCESS_INSTANCE_KEY

WF_INVALID_PROCESS_INSTANCE_KEY

WF_INVALID_STATE_TRANSITION

WF_INVALID_OBSERVER_FOR_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 30*:**

```
<?xml version="1.0"?>
<WfMessage xmlns=”http://www.wfmc.org/standards/docs/Wf-XML” Version=”1.1”>
   <WfMessageHeader>
      <Response/>
      <Key>http://www.myco.com/purchasing/orders/4089259</Key>
   </WfMessageHeader>
   <WfMessageBody>
      <ProcessInstanceStateChanged.Response/>
   </WfMessageBody>
</WfMessage>
```

3.5.4.2 Notify

This operation provides the means by which two process instances may communicate and synchronize while they are running. It is used to notify an observer (which is very likely another process instance) of the occurrence of an event in a process instance that is relevant to the further operation of the observer. This operation should not be used to inform a resource of state changes within a process instance, instead the ProcessInstanceStateChanged operation should be used for that purpose. This operation is to be used to notify a resource of application-specific events. The nature of these events and details regarding them must therefore be agreed upon in the interoperability contract in order to make use of this operation.

Typically, these notifications will deal with changes in application data that can impact the interoperation of the processes. Therefore, this operation may indicate (within ContextData) information regarding the affected data items in addition to information regarding the event itself. The Notify operation elements are structured as follows:

```
<!ELEMENT Notify.Request (ProcessInstanceKey, NotificationName, ContextData)>
<!ELEMENT ProcessInstanceKey (#PCDATA)>
<!ELEMENT NotificationName (#PCDATA)>
<!ELEMENT ContextData ANY>
<!ELEMENT Notify.Response (Exception?)>
```

These structures have the following semantic constraints and meanings:

Request Parameters:

ProcessInstanceKey – Key of the process instance that invokes the operation.

NotificationName – Name of the message for notification. The contents of this element are subject to agreements made in the interoperability contract, as events are specific to particular application and/or process instance requirements.

ContextData – Context-specific data that represents application data to be delivered to the observer. This information will be encoded as described in the section on Process Context and Result Data.

Example 31:
```
<?xml version=”1.0”?>
<WfMessage xmlns=”http://www.wfmc.org/standards/docs/Wf-XML” Version=”1.1”>
   <WfMessageHeader>
      <Request ResponseRequired=”No”/>
      <Key>http://www.myco.com/purchasing/orders/4089259</Key>
   </WfMessageHeader>
   <WfMessageBody>
      <Notify.Request>
         <ProcessInstanceKey>http://www.exampleco.com/orders/86947325-32914
         </ProcessInstanceKey>
         <NotificationName>OrderChanged</NotificationName>
         <ContextData>
            <Parameter>
               <Name>VehDesc</Name>
               <Value>
                  <Vehicle>
                     <VehicleType>Car</VehicleType>
                     <Specification>
                        <Manufacturer>Audi</Manufacturer>
                        <Model>A4</Model>
                     </Specification>
                  </Vehicle>
               </Value>
            </Parameter>
         </ContextData>
      </Notify.Request>
   </WfMessageBody>
</WfMessage>
```

Response Parameters:

None.

Exceptions:

In addition to all general exceptions defined in section 3.4.9.1, the following exceptions are supported specifically for this operation:

WF_INVALID_CONTEXT_DATA

WF_MISSING_PROCESS_INSTANCE_KEY

WF_INVALID_PROCESS_INSTANCE_KEY

WF_INVALID_OBSERVER_FOR_RESOURCE

WF_MISSING_ NOTIFICATION_NAME

WF_INVALID_ NOTIFICATION_NAME

Example 32:
```
<?xml version="1.0"?>
<WfMessage xmlns=”http://www.wfmc.org/standards/docs/Wf-XML” Version=”1.1”>
   <WfMessageHeader>
      <Response/>
      <Key>http://www.myco.com/purchasing/orders/4089259</Key>
   </WfMessageHeader>
   <WfMessageBody>
      <Notify.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 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 Observer 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.

5 Implementation Issues

5.1 Interoperability Contract

It is recognized that there may be additional requirements outside the scope of the 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 specific data transfer requirements should also be addressed here.

- Data Constraints – application-specific data type requirements, 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 Specifics – required protocol header data, timeout values, buffer sizes, asynchronous or batch processing requirements, etc.

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

- Key/ID Requirements – Details regarding management of (relative) keys, format of implementation-specific identifiers (of objects), etc.

- Process Synchronization – Specifics regarding events of which a process must be notified in order to synchronize

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. In order to assist in determining conformance, this section also introduces several categories for features of the specification, and names for those features.

There are four high-level categories for specification features:

- Interoperability Models
- Message Processing Types
- Protocol Constructs
- Operations

The features within these categories are defined as follows. There are three Interoperability Models currently defined by the WfMC Workflow Reference Model: Chained, Nested and Parallel-Synchronized.

The Message Processing Types currently available as of this version are: Synchronous, Asynchronous, Individual and Batch.

The following Protocol Constructs are available as of this version: Header/Body, Transport/Header/Body, Transport Only and Transport & Multiple Header/Body.

The complete list of operations currently specified is: GetBatchMessageState, ChangeBatchMessageState, CreateProcessInstance, GetProcessInstanceData, ChangeProcessInstanceState, ProcessInstanceStateChanged and Notify.

These categories and features are organized into several conformance profiles, as described below. Products claiming conformance to this specification must provide a clear conformance statement indicating the following information:

1. The conformance profile(s) supported.
2. The transport mechanism(s) supported. For this version of the specification, HTTP is the only transport mechanism supported.

For example, a vendor can claim to implement this specification, and declare their implementation to be "conformant to the interoperability and asynchronous profiles over http".

6.1 Conformance Profiles

A vendor can claim conformance to one or more of the following profiles: Interoperability, Asynchronous, or Batch. Every conformant implementation must implement the Interoperability profile. The Asynchronous and Batch profiles are optional.

6.1.1 Interoperability (Mandatory)

Every implementation of this specification must implement the Interoperability profile, which inherently supports the Chained and Nested Interoperability Models. This profile includes support for the basic Message Processing Types: Synchronous and Individual. It also includes support for the basic Protocol Construct: Header/Body. The Protocol Construct Transport/Header/Body may also optionally be supported in this profile. Finally, in conforming to this profile an implementation must support the following Operations, as defined in this specification:

- CreateProcessInstance
- GetProcessInstanceData
- ChangeProcessInstanceState

- ProcessInstanceStateChanged
- Notify

6.1.2 Asynchronous (Optional)

An implementation of the optional asynchronous profile must support the Message Processing Type: Asynchronous. It must also support for the Protocol Constructs: Transport/Header/Body and Transport Only.

6.1.3 Batch (Optional)

An implementation of the optional batch profile must support the Message Processing Type: Batch. It must also support the Protocol Construct: Transport & Multiple Header/Body. Lastly, this profile includes support for the following operations, as defined in this specification:

- GetBatchMessageState
- ChangeBatchMessageState

6.2 Version Conformance

As stated in section 1.1 Version Compatibility, this version of the Wf-XML specification is backward compatible with version 1.0. In this section, we describe the conformance impact of changes made to this version of the specification.

This specification defines no requirement for a Wf-XML v1.0 processor to be upgraded to support version 1.1 messages. It also defines no requirement that a Wf-XML v1.1 processor support version 1.0 messages. However if a processor is to support both versions of this specification, it must ensure that any message it sends or receives is conformant with the version of the specification indicated by the value of the “Version” attribute on the message’s WfMessage element.

The following table is provided to assist in determining which features are available in each version of the specification. This is not a conformance profile matrix, but is intended to segregate version 1.0 capabilities from those only available in version 1.1. As stated above, the asynchronous and batch processing features available in version 1.1 are not required to be supported in order for a processor to be conformant with this version of the specification.

	Version 1.0	**Version 1.1**
Interoperability Models		
Chained	X	X
Nested	X	X
Parallel-Synchronized		X*
Message Processing Types		
Synchronous	X	X
Asynchronous		X
Individual	X	X
Batch		X
Protocol Constructs		
Header/Body	X	X
Transport/Header/Body	X	X

Transport Only (Acknowledgement)		X
Transport & Multiple Header/Body (Batch)		X
Operations		
GetBatchMessageState		X
ChangeBatchMessageState		X
CreateProcessInstance	X	X
GetProcessInstanceData	X	X
ChangeProcessInstanceState	X	X
ProcessInstanceStateChanged	X	X
Notify		X

Table 2: Feature Availability

* The Parallel-Synchronized Interoperability Model is partially supported in this version of the specification via the Notify operation. Complete support will be provided in future versions.

6.3 Other Considerations

The critical factor in determining conformance lies in a vendor’s ability to implement the functionality described by the specification according to the conformance profiles. However, other XML-related factors described here may also impact an implementation.

6.4 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 Document Type Definition (DTD) or XML Schema Description (XSD) file. The XML specification imposes no requirement on a document instance to be valid, only well-formed. In the case where well-formed data is to be processed, the burden of validating syntactic or semantic constraints over and above those specified by the XML specification lies entirely with the processing application.

For this reason, this specification does not mandate validity of all document instances, rather it only requires that all Wf-XML messages are well-formed and compliant with the semantic constraints imposed herein. It is therefore the responsibility of an application implementing this specification to ensure that these constraints are not violated.

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.

6.5 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.

7 Transport Layer Bindings

Wf-XML messages for workflow interoperability can be communicated between interoperating workflow systems using many different transport mechanisms/protocols. As these protocols support a fundamental requirement of message-based interoperability, their behavior and the actions they specify must not be compromised in their usage by this specification. Furthermore, the behaviors and actions specified by Wf-XML must be supported by these underlying protocols, while not having any dependencies on any particular protocol. Therefore, this section will define the relationships and interactions between Wf-XML and its underlying transport mechanism.

This section provides a specification for a Hypertext Transfer Protocol (HTTP) [12] binding. This is considered the most common transport mechanism utilized to communicate Wf-XML messages between interoperating enactment services. The support of this or any other particular transport layer binding is not required for an implementation to be compliant with this 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.

7.1 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 [14] 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. This key can be found in one of two places within the message. The primary location is the "Key" element within the WfMessageHeader element. However if the message is an Acknowledgement (used in asynchronous processing) or a Batch message (containing multiple headers), then the "Key" element within the Dialog element of the WfTransport section serves as the identifier of the resource to which a POST method is directed.

As an alternative to absolute addressing, 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) or obtained 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://…".

In synchronous processing, the Wf-XML Request message is the one MIME body part for input and the Wf-XML Response message is the one MIME body part for output. Furthermore, these messages must be included with their respective HTTP requests/responses. That is to say, that a Wf-XML Request message must be sent with an HTTP request and a Wf-XML Response message must be sent with its HTTP response.

In asynchronous processing, the Wf-XML message (consisting of Requests and/or Responses) is the one MIME body part for input and the Wf-XML Acknowledgement message is the one MIME body part for output.

All Wf-XML MIME body parts must use the MIME content type "Content-type: text/xml" in the HTTP-method header, and the Content-length must be set according to the length of the Wf-XML Request or Response message respectively.

All Wf-XML message processing is subject to the successful execution of HTTP method processing. Therefore, all HTTP status codes must be interpreted independently of this specification, and all Wf-XML processing assumes successful completion of HTTP procedures prior to execution. For authentication, the usual HTTP mechanisms should be used. This includes usage of the respective HTTP header fields.

8 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 1.1 - 24 October, 2001                                          -->

<!--                                                                                      -->

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

<!--   '<!DOCTYPE WfMessage ['                                                            -->

<!-- and the following line appended:                                                     -->

<!--   ']>'                                                                               -->

<!--                                                                                      -->

<!-- If a PUBLIC identifier is used to reference this DTD from a document instance, the following  -->
<!-- identifier should be used:                                                           -->

<!--   'PUBLIC "-//WfMC//DTD Wf-XML 1.1//EN"                                              -->

<!--              http://www.wfmc.org/standards/docs/Wf-XML-1.1.dtd                       -->

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


<!-- ~~~~~~~~~~~~~~~~~~~ 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|h
a|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|n
a|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|t
o|tr|ts|tt|tw|uk|ur|uz|vi|vo|wo|xh|yo|zh|zu)">


<!-- The xml:space attribute may be used to indicate that white space should be preserved. -->

<!ENTITY % space "xml:space (default | preserve) #IMPLIED">

<!-- The xml:lang attribute may be used to indicate the natural language used in an element. -->

<!ENTITY % lang "xml:lang NMTOKEN #IMPLIED">


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

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

```
<!ENTITY % OperationResponse "GetBatchMessageState.Response | ChangeBatchMessageState.Response |
CreateProcessInstance.Response | GetProcessInstanceData.Response | ChangeProcessInstanceState.Response |
ProcessInstanceStateChanged.Response | Notify.Response">


<!-- The ProcessInstanceData entity defines the properties of a process instance that may be obtained using the
GetProcessInstanceData 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 1.1 of Wf-XML. -->

<!ENTITY % states "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) *) | (WfMessageHeader,
WfMessageBody))>

<!ATTLIST WfMessage Version CDATA #FIXED “1.1”

                    %space;

                    %lang;>


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

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

<!ELEMENT WfTransport (Dialog?, CorrelationData?, Exception?)>

<!ELEMENT Dialog ((Acknowledgement, Key) | (ReplyToKey, Key?) | Key)?>

<!ATTLIST Dialog Type (synch | asynch) ”synch”

                Mode (individual | batch) ”individual”

                MessageID CDATA #IMPLIED>

<!ELEMENT Acknowledgement EMPTY>

<!ATTLIST Acknowledgement ReceivedAt CDATA #REQUIRED>

<!ELEMENT Key (#PCDATA)>

<!ELEMENT ReplyToKey (#PCDATA)>

<!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
```

```
                        RequestID CDATA #IMPLIED>
<!ELEMENT Response EMPTY>
<!ATTLIST Response RequestID CDATA #IMPLIED>

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

<!ELEMENT GetBatchMessageState.Request (MessageID)>
<!ELEMENT MessageID (#PCDATA)>

<!ELEMENT ChangeBatchMessageState.Request (MessageID, State)>
<!ELEMENT State (%states;)?>

<!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>
<!ATTLIST ContextData %space;
                            %lang;>

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

<!ELEMENT ValidStates (%states;)*>
<!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>
<!ATTLIST ResultData %space;
                           %lang;>
```

```
<!ELEMENT ProcessDefinitionKey (#PCDATA)>
<!ELEMENT Priority (#PCDATA)>
<!ELEMENT LastModified (#PCDATA)>


<!ELEMENT ChangeProcessInstanceState.Request (State)>


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


<!ELEMENT Notify.Request (ProcessInstanceKey, NotificationName, ContextData)>
<!ELEMENT NotificationName (#PCDATA)>


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


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


<!ELEMENT CreateProcessInstance.Response ((ProcessInstanceKey, Name?) | Exception)>


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


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


<!ELEMENT ProcessInstanceStateChanged.Response (Exception?)>


<!ELEMENT Notify.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 [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 - Future Directions (Non-Normative)

This appendix describes several features of the Wf-XML specification for which enhancements are being considered in future versions. It is provided as a convenience for developers, as it may provide some insight into possible future directions for an implementation. None of the changes discussed here should be relied upon in any way, as they are merely preliminary suggestions and are highly subject to change.

B.1 Messaging Protocol

This specification can be considered to consist of two major components: a messaging protocol and core interoperability functions. The messaging protocol is essential in that it comprises the overall message structure, an exception handling mechanism, an identification and addressing mechanism, and transport layer bindings. The WfMC created the messaging protocol used by Wf-XML from scratch, as there was no known suitable protocol available at the time of its creation.

However, there has been much attention focused on this area by industry and standards bodies as the use of XML-based messaging became increasingly prevalent. Recently, this attention has resulted in the development of protocols such as XML-RPC, Blocks (BXXP), SOAP, XP and ebXML TR&P. The coalition is now considering some of these protocols as alternatives to the native Wf-XML protocol. This consideration may result in the replacement of the Wf-XML messaging protocol by one of these emerging standards, or independence from the messaging protocol layer if no single standard can be selected.

B.2 Specification Meta-Language

This specification currently uses the Document Type Definition (DTD) syntax to define the Wf-XML vocabulary and grammar. However, it is recognized that due to the diverse usage of XML there are now numerous alternative schema definition languages available. These include the W3C XML Schema Definition language (XSDL), Schematron, RELAX, TREX and others.

It is highly likely that the next version of this specification will leverage the W3C XML Schema Definition language's enhanced capabilities for enhanced semantic and data type validation. There is also some potential for updating the specification in a schema-neutral fashion, allowing for the creation of "schema language bindings" to accommodate multiple preferences. However, this flexibility will need to be balanced against the possible impact to interoperability.

B.3 Operations

The operations contained within this section are reserved for future use. High-level descriptions of these operations are provided here, although details of their functionality have yet to be determined. The following table summarizes the operations covered in this section.

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

Table 3: Additional Operations

B.3.1 Control Operations

B.3.1.1 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.3.2 Process Definition Operations

B.3.2.1 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.3.3 Process Instance Operations

B.3.3.1 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.3.3.2 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.3.3.3 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.3.3.4 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 Ancillary Supporting Mechanisms

This section describes developments being considered that may or may not be specified within this document. Nevertheless, these developments would lend to interoperability and/or assist developers in implementing this specification and validating their implementations.

B.4.1 Interoperability Contract

While certain recommendations are made within this specification pertaining to the interoperability contract between workflow enactment services, there is currently no well-defined syntax or structure for such a contract. As the creation and usage of web services continues to propagate throughout various industries, advances are being made in the area of dynamic interoperability. One such advance exists in the form of the trading partner agreement Markup Language (tpaML) specification, which is targeted for use by the ebXML initiative. This specification may prove useful as a basis for Wf-XML interoperability contracts.

B.4.2 Reference Framework Implementation

A very helpful tool for any implementers of a new specification is a reference implementation framework on which development can be based. The coalition would like to make such a reference framework available, pending availability of resources. There is also consideration being given to developing such a framework in an open source environment.

B.4.3 Conformance Testing Harness

Conformance testing of its specifications has long been a goal of the WfMC, and this is true in the case of the Wf-XML specification. Steps have been taken to facilitate this testing by the coalition both within and outside of this specification, and further work will be done in future versions of this document to enhance its testability. There is also some potential to develop a test harness or certification mechanism of some sort in conjunction with the framework development discussed above.

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

[15] International Organization for Standardization, ISO/IEC 11578 (1996). "Information technology - Open Systems Interconnection - Remote Procedure Call (RPC)". See also: http://www.opengroup.org/onlinepubs/009629399/apdxa.htm#tagcjh_20