The Workflow Management Coalition Specification

Workflow Management Coalition Workflow Standard - Interoperability Wf-XML Binding

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

20 April 1999
Version 1.0α (editing changes applied 7/7/99 in Red)

Send comments to : WORKGROUP4@FTPLIST.AIIM.ORG
Rogowski@mail1.MONMOUTH.ARMY.MIL

Copyright © 1999 The Workflow Management Coalition

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior written permission of the Workflow Management Coalition except that reproduction, storage or transmission without permission is permitted if all copies of the publication (or portions thereof) produced thereby contain a notice that the Workflow Management Coalition and its members are the owners of the copyright therein.

This Specification has been authored by Workflow Management Coalition members.

Readers should note the status of this Alpha Version (section 1.4) and forecast changes (section 1.10).

1 Introduction

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

1.1 Purpose

It is the intention of this specification to describe a language that can be used to achieve the two basic types of interoperability defined in the abstract specification. Specifically, simple chained workflows and nested workflows. It will support these two types of interchange both synchronously and asynchronously. Furthermore, this specification will describe a language that is independent of any particular implementation mechanism, such as programming language, data transport mechanism, platform, hardware, etc.

1.2 Scope

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

1.3 Audience

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

1.4 Document Status

This document is a draft proposal submitted to the WfMC for approval. It may be changed in whole or in part at anytime as deemed necessary by the WfMC. Until given formal approval, this specification should not be utilized or referenced in any way except as a work-in-progress.

This version is available from the WfMC on a "by request" basis to facilitate industry feedback.

1.5 Documentation Conventions

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

- Throughout this document, there are issues raised that require further review or consideration. These points are indicated by a:

  ```
  ************************************************************
  **                          NOTE:                         **
  ************************************************************
  ```

  containing a brief description of the issue or question at hand. This document should be changed to reflect decisions made after review of these issues.

- In several of the examples, element names are replaced by the italicized words *interface-name* and *method-name*. These would be replaced by element names defined in this specification in actual messages.

1.6 Background

This specification has been generated as the result of vast amounts of work completed previously by groups such as the WfMC, OMG and many vendor organizations in an effort to define the functionality required to achieve interoperability among workflow systems. These efforts have resulted in the IF4 abstract specification [1], the jFlow specification [3] and the IF4 Internet e-mail MIME binding specification [4], as well as the Simple Workflow Access Protocol (SWAP) specification [5]. However, it has been determined that there is still a need to provide a light, generic, easy to implement specification in order to truly support the type of interoperability that has been the goal of these efforts.

1.7 Approach

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

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

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

In defining this specification, it is also highly desirable to leverage the considerable effort that has gone into the documents referenced earlier. In particular, the draft SWAP specification [5] has provided a unique opportunity to gain valuable real-world experience with an XML-based approach. Furthermore, various vendors have begun to prototype implementations based on the existing SWAP draft, providing grounds for input and feedback, and facilitating acceptance of this proposal as the specification matures. Therefore, every effort will be made to ensure maximum compatibility with these existing implementations so long as that compatibility does not negatively impact the stated goals of this specification. Should further review and experience warrant extensive changes to this spec, the spec may be revised and the vendors with existing implementations will have a choice of supporting either version.

While this specification is based on the core concepts and functionality described by the Interoperability Abstract specification [1], there does not exist a direct correlation between each specific operation and parameter defined in the Abstract and those defined in this specification. This is because this specification has been developed with the intent of describing a simple, easily implemented language that could quickly demonstrate viability in field implementations. However this is a living document, and as an experience base grows with fielded implementations, this spec should be revised and extended as appropriate to encompass greater levels of interoperability.

1.8 Data Types

Although there are obvious benefits to strong data type specifications in any language, XML unfortunately does not yet support many robust data types. For example, one may specify the content of an element as NUMBERS, NMCHARS, CDATA, etc. But there is no provision for types such as Int, Long, Date/Timestamp (not to mention arrays, structs, etc.). This is expected to change as XML-related initiatives

(such as the XML-Data or DCD proposals to the XML Schema working group) progress, but these efforts are not yet finalized.

For the purposes of this specification, the following data types will be enforced. These type definitions are extracted from the Open DataBase Connectivity (ODBC) [7] standard definitions for SQL data types.

- LONG VARCHAR - Variable length character data. Maximum length is data source-dependent.
- BIGINT - Exact numeric value with precision 19 (if signed) or 20 (if unsigned) and scale 0.
- FLOAT - Signed, approximate, numeric value with a mantissa precision 15.
- LONG VARBINARY - Variable length binary data. Maximum length is data source-dependent.
- TIMESTAMP - Date/time data. This data should be expressed in standard UTC format as specified by ISO 8601 [8].

1.9 Terminology

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

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

1.10 Changes Forecast

Following discussion at the WfMC meeting, 30/6/99, Dublin, it was agreed that the first full version of the document for product implementations would focus on the core objects and methods required to support interoperability.
These interfaces cover:
Process Definition, Process Instance, Observer, Activity Observer (where required). Some refinement and reclassification of the associated methods is expected as the specification moves to a beta version.

Other supporting interfaces will be subject to further discussion and do not form part of the initial core implementation set for interoperability. These are: Worklist, Work Item and Entrypoint.

In addition it should be noted that a number of implementation issues are to be addressed (section 3) and that further refinement of bindings to the underlying data transport environment (Appendix 3) are intended.

2 Technical Specification

2.1 Logical Resource Model

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

It is also understood that a resource may need to be aware of the context of a request in order to respond to it properly. For example, if a request originates from a source relevant to a particular activity within a set of activities the response may include information about the overall “parent” activity, whereas if the same request had originated from the parent, the response would not include this information. In order to support this concept we will borrow from Object-Oriented terminology, and group the interoperable functions defined herein into “interfaces” which identify their context. Within each interface, individual functions are termed “methods”. Each method may be passed a set of input parameters and return a set of results, which are dependent upon the interface in which it is operating. A resource implements an interface by supporting the methods defined to exist within that interface. Furthermore, a resource may implement more than one interface. The primary interfaces required for interoperability are named ProcessDefinition, ProcessInstance and Observer. Other interfaces are also defined, but are not explicitly required for basic interoperability. Conformance to the spec in this regard, as well as others, is discussed in the conformance section later in this document Each interface will be described as it is referenced later in this specification.

The primary interfaces serve the following roles. The ProcessDefinition interface is the most fundamental interface 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 interface represents the actual enactment of a given process definition and will have it’s 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. In doing so, the requestor may modify the properties of that definition to suit the particulars of this instance. 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 (possibly paused and resumed) and will eventually be completed or terminated.

The Observer interface provides a means by which a process instance may communicate its completion. In nested subprocesses, there must be a way for a requestor of a service enactment to determine or be informed when a subprocess completes. The Observer interface will provide this information by giving a process instance the resource identifier of the requestor. Any number of other resources may also register themselves as observers of a process instance and will also be informed when the instance completes. Furthermore, the Observer interface can be used to inform resources of other changes in the instance's status, or events other than completion.

2.2 Wf-XML Language Definition

Every Wf-XML message is an XML document instance, conforming to the XML 1.0 specification. Therefore, the first string that will appear in each message will be the XML declaration, which appears in the form of a PI as follows: '<?xml version="1.0"?>'. This implies that the default XML character encoding of UTF-8 will be used. Future versions of this specification may wish to address other encodings in order to support internationalization, and may consider use of the XML encoding declaration (such as <?xml version="1.0" encoding="ISO-8859-1"?>) . It should also be noted that, by default, XML languages are case-insensitive. This section will describe each element used within Wf-XML messages and its purpose, also providing examples. For a consolidated table of all defined elements and the Wf-XML DTD, see Appendix C.

The root element within a Wf-XML message is named "WF_XML" and so the skeleton of a message will appear as follows:

Example 1

```
<?xml version="1.0"?>
  <WF_XML>
  </WF_XML>
```

Each Wf-XML message may be one of two types, either a request or a response. Elements defined to indicate the type of message are named "request" and "response". In the case of a request, the request element will be the only direct child of the WF_XML element. For responses, both the request and response elements will appear for reasons discussed below. Therefore, the presence of a response element indicates that this message is a response, as opposed to a request. For example:

Example 2

```
<?xml version="1.0"?>
  <WF_XML>
    <request/>
  </WF_XML>
```

or

```
<?xml version="1.0"?>
  <WF_XML>
    <request/>
    <response/>
  </WF_XML>
```

2.2.1 Asynchronous Message Handling

Because the data transport mechanism of any given implementation is unknown to this specification, it is a stated goal of this document to support synchronous and asynchronous messaging. This

implies a need to coordinate request and response messages so that an implementation will be able to determine the request for which a response is being recieved. This will be accomplished by returning the contents of the original request, as well as any repsonse data, with each response message.

```
************************************************************
**                          NOTE:                         **
**     This seems to be the most efficient (simplest) way to handle this   **
**   as long as the potential size of the messages isn't a problem. If this  **
**    is indeed a concern, possibly only the requested interface, method  **
**    and sessionid would be sufficient. Other identifier-based schemes   **
**          could become quite elaborate and difficult to implement.      **
************************************************************
```

2.2.2 Security

In general, security considerations are out of the scope of this document due to the fact that they are largely dependent upon implementation-specific factors. This applies to user identification and authorization, as well as data and functional access control. While mechanisms such as SSL, PKI and LDAP may be suficient for some applications, they may be viewed as insufficient or overkill by others. Therefore the only provision made for security considerations in this specification will be the optional presence of a "session" identifier in Wf-XML messages. The element used for this purpose will be called "sessionid". It is assumed that verification of user credentials will be handled through implementation-specific protocols and that each implementation will generate a session id if necessary. For example:

Example 3

```
<?xml version="1.0"?>
  <WF_XML>
    <request>
      <sessionid>0x14F351C</sessionid>
    </request>
  </WF_XML>
```

2.2.3 Extensibility

The focus of this specification is on a minimal set of agreed upon functionality required for basic interoperability. However, it is acknowledged that there is a vast amount of functionality inherent in workflow systems that could impede interoperability if not properly handled. Therefore, it is necessary to provide a mechanism by which interoperating systems may exchange required implementation-specific data in a generalized fashion that will not impose restrictions on other systems not requiring that data. Therefore, no implementation should require this specific information in order to perform basic interoperable functionality.

Various places in this specification define placeholder elements for context-specific information. The names of these placehoilder elements are contextdata, resultdata, contextdatainfo and resultdatainfo. This information is to be exchanged as a list of name/value pairs, whereby any number of data items may be exchanged, each indicating its name and value. In the cases of contextdatainfo and resultdatainfo, 'value' is replaced by 'type' in order to indicate the data type of the field being described. These information pairs are represented by the elements "name", "value" and "type", which contain character data. Each pair in the list is contained within a wrapper element called "item". This item wrapper allows individual pairs of information to be handled discretely, as well as allowing multiple name/value pairs to be contained within a single item. Each occurrence of a placeholder element will contain one or more items, each item containing a name and value/type. For example:

Example 4

```
<?xml version="1.0"?>
  <WF_XML>
    <request>
          …
      <placeholder>
        <item>
          <name>xxx</name><value>yyy</value>
        </item>
        <item>
          <name>xxx</name><value>yyy</value>
        </item>
        <item>
          <name>xxx</name><value>yyy</value>
        </item>
          …
      </placeholder>
    </request>
  </WF_XML>
```

or

```
<?xml version="1.0"?>
  <WF_XML>
    <request>
          …
      <placeholder>
        <item>
          <name>xxx</name><type>yyy</ type >
        </item>
        <item>
          <name>xxx</name>< type >yyy</ type >
        </item>
        <item>
          <name>xxx</name>< type >yyy</ type >
        </item>
          …
      </placeholder>
    </request>
  </WF_XML>
```

```
**********************************************************
**                        NOTE:                         **
**  An alternate approach to this topic would be to allow an application  **
**   to use its own specific elements within context-specific data areas.  **
**  These would have to be agreed upon in an interoperability contract  **
**   on a case-by-case basis. XML does, of course, provide for this so  **
**                long as validity checking is not required.            **
**********************************************************
```

This mechanism provides for extensible interoperability within the constraints of the Wf-XML language. For further information regarding extensibility, see the section on Conformance later in this document.

2.2.4 Process Status

The WfMC has defined a standard set of valid process instance states. All process instances will be in one of the following states:

- open.notrunning.notstarted – A resource is in this state when it has not yet actively participated 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 temporarily suspended. At this point, no resources contained within it may be started.
- open.running – A resource is in this state when it is performing its part in the normal execution of a work process.
- closed.completed – A resource is in this state when it has finished its task in the overall work process. All resources contained within it are assumed to be complete at this point.
- closed.terminated – A resource is in this state when it has been abnormally ended before it completed its work process. At this point, all resources contained within it are assumed to be either completed or terminated.
- closed.aborted – A resource is in this state when it 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.

2.2.5 Error Handling

Should any exception occur during the execution of a Wf-XML method, that exception would have to be returned to the caller. Various types of exception can be anticipated, including warnings, recoverable errors and fatal errors, as well as application-specific error types. Therefore, an "exception" element has been defined to carry this information. This exception element will be returned with the results from all methods. If an error of any type has occurred, the exception information will contain the exception's type, message and any supporting information (such as filenames, error codes, etc.).

This information will be encoded in the elements "type", "msg" and "contextdata". 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. The 'type' and 'message' elements simply contain character data, whereas the contextdata element is used as it is throughout this specification (listing name/value pairs for each specific data item).

If no error has occurred, the exception's type element will contain the string "None", it's msg element will be empty and it will not contain a contextdata element. Finally, it may also be necessary to indicate nested exceptions (the fact that one exception has been caused by another, which was caused by another, etc). Therefore, exception elements may contain additional exception elements. For example:

Example 5

```
<?xml version="1.0"?>
  <WF_XML>
    <request>
      <interface-name>
        <method-name>
          …
        </method-name>
      </interface-name>
    </request>
    <response>
      <interface-name>
        <method-name>
          …
        </method-name>
      </interface-name>
      <exception>
```

```
        <type>Fatal</type>
        <msg>File not found</msg>
        <contextdata>
          <item>
            <name>file</name><value>C:\myfile</value>
          </item>
        </contextdata>
        <exception>
          <type>warning</type>
          <msg>The XML version given has been updated</msg>
        </exception>
      </exception>
    </response>
  </WF_XML>
```

2.2.6 Interface and Method Definitions

This section defines the individual interfaces and methods used to provide the functionality required by the abstract. See Appendix A for a concise mapping of methods to interfaces. The markup used in this section follows a consistent model whereby, within a Wf-XML request or response, the interface to be implemented will appear as an element within the message. That interface element will further contain an element whose name is the method to be executed. Input parameters or returned results will then be embedded within the method element. This model has the following structure:

Example 6

```
<?xml version="1.0"?>
  <WF_XML>
    <request>
      <sessionid>0x14F351C</sessionid>
      <interface-name>
        <method-name>
          <parameter/>
          <parameter/>
          …
        </method-name>
      </interface-name>
    </request>
  </WF_XML>
```

or

```
<?xml version="1.0"?>
  <WF_XML>
    <request>
      <sessionid>0x14F351C</sessionid>
      <interface-name>
        <method-name>
          <parameter/>
          <parameter/>
          …
        </method-name>
      </interface-name>
    </request>
    <response>
      <interface-name>
```

```
      <method-name>
        <result/>
        <result/>
        …
      </method-name>
    </interface-name>
    <exception>
      <type>None</type>
      <msg/>
    </exception>
  </response>
</WF_XML>
```

ProcessDefinition

This interface will be used to create process instances and find general information about process definitions. It contains the methods PropFind, CreateProcessInstance and ListInstances.

PropFind – used to retrieve the values of all properties defined for the given process definition resource. This set is defined to be the union of all the properties specified by all of the interfaces that the resource implements. Current values of all the properties of the resource are returned.

Parameters: ResourceID – Identifier of the resource whose properties are to be returned. When called on this interface, this is to be the identifier of a known process definition.

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

Example 7:

```
<?xml version="1.0"?>
<WF_XML>
  <request>
    <sessionid>0x14F351C</sessionid>
    <processdefinition>
      <propfind>
        <resourceid>xxx</resourceid>
        <resultdataattributes>
          <interfaces/>
          <key/>
          <validstates/>
          …
        </resultdataattributes>
      </propfind>
    </processdefinition>
  </request>
</WF_XML>
```

Results: Interfaces – The names of all the interfaces implemented on this resource. This result indicates the types of requests that can be made to this resource.

Name – A human readable identifier of the resource. This name is unique within the scope of its parent, and in some cases may be nothing more than a number.

Key – The globally unique identifier of the resource.

Subject – A short description of this process definition.

Description – A longer description of this process definition resource.

State – The current status of this resource.

ValidStates – A list of state values allowed by this resource.

ContextDataInfo – Metadata about the context-specific information required to create an instance of this process definition. This information will be encoded in data pairs containing the name and type of each item.

ResultDataInfo – Metadata about the context-specific information required to be returned by an instance of this process upon completion. This information will be encoded in data pairs containing the name and type of each item.

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

Example 8:

```
<?xml version="1.0"?>
<WF_XML>
<request>
    <sessionid>0x14F351C</sessionid>
    <processdefinition>
      <propfind>
        <resourceid>xxx</resourceid>
        <resultdataattributes>
            <interfaces/>
            <key/>
            <validstates/>
             …
        </resultdataattributes>
      </propfind>
    </processdefinition>
  </request>
  <response>
    <processdefinition>
      <propfind>
        <interfaces>xxx</interfaces>
        <name>xxx</name>
        <key>xxx</key>
        …
        <contextdatainfo>
          <item>
            <name>xxx</name><value>yyy</value>
          </item>
```

```
            …
          </contextdatainfo>
          …
        </propfind>
      </processdefinition>
      <exception>
      …
      </exception>
    </response>
  </WF_XML>
```

CreateProcessInstance – used to instantiate a known process definition. The instance will be created with context-specific properties set according to the input data, and started. The instance may optionally only be created and its properties set, at which point it could be started manually later.

Parameters: ResourceID – Identifier of the process definition resource that is to be used to create this instance.

Observer – Identifier of the resource that is to be the observer of the instance that is created by this method. This observer resource is then to be notified of state changes to the instance, most notably the completion of the instance.

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.

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

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

ContextData – Context-specific information required to create this process instance. This information will be encoded in data pairs containing the name and value of each item.

StartImmediately – An optional Boolean value (yes or no), indicating whether the newly created instance should be started immediately upon creation. The default is “yes”.

```
**************************************************************
**                          NOTE:                           **
** If the instance is not started immediately, it is assumed that PropPatch **
 **  will be used to change the status of the instance. Is this sufficient, or **
**          should a new method be created to support this operation?        **
**************************************************************
```

Example 9:

```
<?xml version="1.0"?>
<WF_XML>
  <request>
    <sessionid>0x14F351C</sessionid>
    <processdefinition>
```

```
        <createprocessinstance>
          <resourceid>xxx</resourceid>
          <observer>xxx</observer>
          <name>xxx</name>
          <subject>xxx</subject>
          <description>xxx</description>
          <contextdata>
            <item>
              <name>xxx</name><value>yyy</value>
            </item>
          </contextdata>
          <startimmediately>yes</startimmediately>
        </createprocessinstance>
      </processdefinition>
    </request>
</WF_XML>
```

Results: Key – the resource identifier of the newly created process instance.

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

Example 10:

```
<?xml version="1.0"?>
<WF_XML>
<request>
    <sessionid>0x14F351C</sessionid>
    <processdefinition>
      <createprocessinstance>
        <resourceid>xxx</resourceid>
        <observer>xxx</observer>
        <name>xxx</name>
        <subject>xxx</subject>
        <description>xxx</description>
        <contextdata>
          <item>
            <name>xxx</name><value>yyy</value>
          </item>
        </contextdata>
        <startimmediately>yes</startimmediately>
      </createprocessinstance>
    </processdefinition>
  </request>
  <response>
    <processdefinition>
      <createprocessinstance>
        <key>xxx</key>
      </createprocessinstance>
    </processdefinition>
    <exception>
    …
    </exception>
  </response>
</WF_XML>
```

ListInstances – 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.

Parameters: ResourceID – Identifier of the process definition resource whose instances are to be listed.

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

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

Example 11:

```
<?xml version="1.0"?>
<WF_XML>
  <request>
    <sessionid>0x14F351C</sessionid>
    <processdefinition>
      <listinstances>
        <resourceid>xxx</resourceid>
        <filter>xxx</filter>
        <filtertype>en_us</filtertype>
      </listinstances>
    </processdefinition>
  </request>
</WF_XML>
```

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

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

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

Example 12:

```
<?xml version="1.0"?>
<WF_XML>
<request>
    <sessionid>0x14F351C</sessionid>
    <processdefinition>
      <listinstances>
        <resourceid>xxx</resourceid>
        <filter>xxx</filter>
        <filtertype>en_us</filtertype>
      </listinstances>
    </processdefinition>
  </request>
  <response>
    <processdefinition>
      <listinstances>
        <instances>
          <instance>
            <key>xxx</key>
```

```
            <name>xxx</name>
            <priority>xxx</priority>
          </instance>
          …
        </instances>
      </listinstances>
    </processdefinition>
    <exception>
    …
    </exception>
  </response>
</WF_XML>
```

ProcessInstance

This interface is used to communicate with a particular instance of a process definition (or enactment of a service), acquiring information about the instance, controlling it and modifying it's properties. Since a given instance may continue to execute for any amount of time, methods may be called on an instance while it is executing. These methods may obtain status information, supply new input values (although how these are handled is dependent on the specific implementation and the task being performed), or obtain early results (although the results of a process instance are not final until the instance has been completed). This interface contains the methods PropFind, PropPatch, Terminate, Subscribe, Unsubscribe and GetHistory.

PropFind - used to retrieve the values of all properties defined for the given process instance resource. This set is defined to be the union of all the properties specified by all of the interfaces that the resource implements.

Parameters: ResourceID – Identifier of the resource whose properties are to be returned.

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

Example 13:

```
<?xml version="1.0"?>
<WF_XML>
  <request>
    <sessionid>0x14F351C</sessionid>
    <processinstance>
      <propfind>
        <resourceid>xxx</resourceid>
      </propfind>
    </processinstance>
  </request>
</WF_XML>
```

Results: Interfaces – The names of all the interfaces implemented on this resource. This result indicates the types of requests that can be made to this resource.

Name – A human readable identifier of the resource. This name is unique within the scope of its parent, and in some cases may be nothing more than a number.

Key – The globally unique identifier of the resource.

Subject – A short description of this process instance.

Description – A longer description of this process instance resource.

State – The current status of this resource.

ValidStates – A list of state values allowed by this resource.

Definition – the identifier of the process definition resource from which this instance was created.

Activities – A list of the activities defined in this process instance. Each returned activity will contain its:

- Key – the resource identifier of the activity
- Name – the name of the activity
- State – the current status of the activity
- ExpirationDate – the deadline for the activity
- Assignees – a list of the names of people who are assigned to the activity
- CreationDate – the date the activity was created
- HasExpired – a Boolean value (yes or no), indicating whether the activity is beyond a deadline

Observers – A list of resources that are registered observers of this process instance, if any exist. Each returned observer is identified by its:

- Key – the identifier of the resource

ResultData – Context-specific data that represents the current result values. This information will be encoded in data pairs containing the name and value of each item.

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

UserInterface – An optional identifier of a user interface resource for this process instance. Applications can use this identifier to allow a user to view or manipulate this instance.

Creator – The name of the user who created this process instance, if available.

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

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

Example 14:

```
<?xml version="1.0"?>
<WF_XML>
<request>
    <sessionid>0x14F351C</sessionid>
    <processinstance>
      <propfind>
        <resourceid>xxx</resourceid>
      </propfind>
    </processinstance>
  </request>
  <response>
    <processinstance>
      <propfind>
        <interfaces>
          <processinstance>
          <observer>
        </interfaces>
        <name>xxx</name>
        <key>xxx</key>
        <subject>xxx</subject>
        …
      </propfind>
    </processinstance>
    <exception>
    …
    </exception>
  </response>
</WF_XML>
```

PropPatch – used to set the values of any number of properties of the given process instance resource. This method 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 method to have any effect, but all parameters are optional. Current values of all the properties of the resource are returned.

Parameters: ResourceID – Identifier of the process instance resource whose properties are to be set.

Subject – A short description of this process instance.

Description – A longer description of this process instance resource.

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

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

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

Example 15:

```
<?xml version="1.0"?>
<WF_XML>
  <request>
    <sessionid>0x14F351C</sessionid>
    <processinstance>
      <proppatch>
        <resourceid>xxx</resourceid>
        <subject>xxx</subject>
        <description>xxx</description>
        <state>xxx</state>
        <priority>xxx</priority>
        <contextdata>
          <item>
            <name>xxx</name><value>xxx</value>
          </item>
        </contextdata>
      </proppatch>
    </processinstance>
  </request>
</WF_XML>
```

Results: Same as PropFind.

Example 16:

```
<?xml version="1.0"?>
<WF_XML>
<request>
    <sessionid>0x14F351C</sessionid>
    <processinstance>
      <proppatch>
        <resourceid>xxx</resourceid>
        <subject>xxx</subject>
        <description>xxx</description>
        <state>xxx</state>
        <priority>xxx</priority>
        <contextdata>
          <item>
            <name>xxx</name><value>xxx</value>
          </item>
        </contextdata>
      </proppatch>
    </processinstance>
  </request>
  <response>
    <processinstance>
      <proppatch>
        <interfaces>
          <processinstance>
          <observer>
```

```
        </interfaces>
        <name>xxx</name>
        <key>xxx</key>
        <subject>xxx</subject>
        …

      </proppatch>
    </processinstance>
    <exception>
    …
    </exception>
  </response>
</WF_XML>
```

Terminate – used to cancel the execution of a process when the initiator is no longer interested in the service being performed.

Parameters: ResourceID – Identifier of the process instance resource that is to be terminated.

Reason – An optional description of why this process instance is being terminated prematurely.

Example 17:

```
<?xml version="1.0"?>
<WF_XML>
  <request>
    <sessionid>0x14F351C</sessionid>
    <processinstance>
      <terminate>
        <resourceid>xxx</resourceid>
        <reason>xxx</reason>
      </terminate>
    </processinstance>
  </request>
</WF_XML>
```

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

Example 18:

```
<?xml version="1.0"?>
<WF_XML>
<request>
    <sessionid>0x14F351C</sessionid>
    <processinstance>
      <terminate>
        <resourceid>xxx</resourceid>
        <reason>xxx</reason>
      </terminate>
    </processinstance>
  </request>
  <response>
    <processinstance>
      <terminate/>
    </processinstance>
    <exception>
```

```
    …
    </exception>
  </response>
</WF_XML>
```

Subscribe – used to register a resource implementing the Observer interface (other than the creator of the instance) with a process instance, as a party interested in status changes and events that occur on that process instance. If this particular process instance resource does not support other observers, an exception will be returned to the caller.

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

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

Example 19:

```
<?xml version="1.0"?>
<WF_XML>
  <request>
    <sessionid>0x14F351C</sessionid>
    <processinstance>
      <subscribe>
        <resourceid>xxx</resourceid>
        <observer>xxx</observer>
      </subscribe>
    </processinstance>
  </request>
</WF_XML>
```

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

Example 20:

```
<?xml version="1.0"?>
<WF_XML>
<request>
    <sessionid>0x14F351C</sessionid>
    <processinstance>
      <subscribe>
        <resourceid>xxx</resourceid>
        <observer>xxx</observer>
      </subscribe>
    </processinstance>
  </request>
  <response>
    <processinstance>
      <subscribe/>
    </processinstance>
    <exception>
    …
    </exception>
  </response>
</WF_XML>
```

Unsubscribe – used to remove a resource from the list of registered observers of a process instance. The calling resource will no longer receive event notifications after executing this method.

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

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

Example 21:

```
<?xml version="1.0"?>
<WF_XML>
  <request>
    <sessionid>0x14F351C</sessionid>
    <processinstance>
      <unsubscribe>
        <resourceid>xxx</resourceid>
        <observer>xxx</observer>
      </unsubscribe>
    </processinstance>
  </request>
</WF_XML>
```

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

Example 22:

```
<?xml version="1.0"?>
<WF_XML>
<request>
    <sessionid>0x14F351C</sessionid>
    <processinstance>
      <unsubscribe>
        <resourceid>xxx</resourceid>
        <observer>xxx</observer>
      </unsubscribe>
    </processinstance>
  </request>
  <response>
    <processinstance>
      <unsubscribe/>
    </processinstance>
    <exception>
    …
    </exception>
  </response>
</WF_XML>
```

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

Parameters: ResourceID – Identifier of the process instance resource whose history is to be retrieved.

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

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

Example 23:

```
<?xml version="1.0"?>
<WF_XML>
  <request>
    <sessionid>0x14F351C</sessionid>
    <processinstance>
      <gethistory>
        <resourceid>xxx</resourceid>
        <filter>xxx</filter>
        <filtertype>en_us</filtertype>
      </gethistory>
    </processinstance>
  </request>
</WF_XML>
```

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

```
************************************************************
**                        NOTE:                           **
** A list of event types is defined in Appendix A of the audit data speci- **
** fication (Interface 5), but no integer event codes are assigned to them. **
**       If these are necessary, they must either be located or assigned.    **
************************************************************
```

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

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

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

Example 24:

```
<?xml version="1.0"?>
<WF_XML>
<request>
    <sessionid>0x14F351C</sessionid>
    <processinstance>
      <gethistory>
        <resourceid>xxx</resourceid>
        <filter>xxx</filter>
        <filtertype>en_us</filtertype>
      </gethistory>
    </processinstance>
  </request>
  <response>
    <processinstance>
      <gethistory>
        <history>
          <event>
            <timestamp>xxx</timestamp>
            <eventcode>xxx</eventcode>
            <eventtype>xxx</eventtype>
            …
          </event>
          …
        </history>
      </gethistory>
    </processinstance>
    <exception>
    …
    </exception>
  </response>
</WF_XML>
```

Observer

The intent of this interface is to allow requesters of work or other resources to monitor the progress of a process instance and be notified upon it's completion. Methods are provided to allow a resource implementing this interface to obtain information about a process instance for which that resource is a registered observer, as well as to notify registered observers of events if the implementing resource is the performer of the work. Once an instance's registered observers have been notified of it's completion or termination, the resource responsible for that instance is not required to maintain it any longer. Therefore, while most events are not required to be notified, the "completed" and "terminated" events must always be

notified in order to indicate to observers that the results are final and this resource may shortly become inaccessible. This interface contains the methods PropFind, PropPatch, Complete, Terminated and Notify.

PropFind - used to retrieve the values of all properties defined for the given process instance resource. This set is defined to be the union of all the properties specified by all of the interfaces that the resource implements.

Parameters: ResourceID – Identifier of the process instance resource whose properties are to be returned.

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

Example 25:

```
<?xml version="1.0"?>
<WF_XML>
  <request>
    <sessionid>0x14F351C</sessionid>
    <observer>
      <propfind>
        <resourceid>xxx</resourceid>
        <resultdataattributes>
          <key/>
          <contextdata/>
        </resultdataattributes>
      </propfind>
    </observer>
  </request>
</WF_XML>
```

Results: Interfaces – The names of all the interfaces implemented on this resource. This result indicates the types of requests that can be made to this resource.

Key – The globally unique identifier of the resource.

ContextData – Context-specific data relevant to the process instance resource implementing this interface. This data is considered to be in the context of its containing subprocess. This information will be encoded in data pairs containing the name and value of each item.

Performer – a list of the resource identifiers of the process instances of this observer.

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

Example 26:

```
<?xml version="1.0"?>
<WF_XML>
<request>
    <sessionid>0x14F351C</sessionid>
    <observer>
      <propfind>
        <resourceid>xxx</resourceid>
```

```
            <resultdataattributes>
              <key/>
              <contextdata/>
            </resultdataattributes>
          </propfind>
        </observer>
      </request>
      <response>
        <observer>
          <propfind>
            <interfaces>
              <processinstance/>
              <observer/>
            </interfaces>
             <key>xxx</key>
             …
          </propfind>
        </observer>
        <exception>
        …
        </exception>
      </response>
    </WF_XML>
```

PropPatch - used to communicate the result values of a subprocess to its observing process instance resource before completion, in order to keep the values synchronized.

Parameters: ResourceID – Identifier of the process instance resource whose properties are to be set.

ResultData – context-specific data relevant to the observing resource's process instance. This information will be encoded in data pairs containing the name and value of each item.

Example 27:

```
<?xml version="1.0"?>
<WF_XML>
  <request>
    <sessionid>0x14F351C</sessionid>
    <observer>
      <proppatch>
        <resourceid>xxx</resourceid>
        <priority>xxx</priority>
        <resultdata>
          <item>
            <name>xxx</name><value>xxx</value>
          </item>
        </resultdata>
      </proppatch>
    </observer>
  </request>
</WF_XML>
```

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

Example 28:

```
<?xml version="1.0"?>
<WF_XML>
<request>
    <sessionid>0x14F351C</sessionid>
    <observer>
      <proppatch>
        <resourceid>xxx</resourceid>
        <priority>xxx</priority>
        <resultdata>
          <item>
            <name>xxx</name><value>xxx</value>
          </item>
        </resultdata>
      </proppatch>
    </observer>
  </request>
  <response>
    <observer>
      <proppatch/>
    </observer>
    <exception>
    …
    </exception>
  </response>
</WF_XML>
```

Complete – used to indicate to the observing resource that the identified process instance has been completed normally. This process instance resource may no longer exist after this point.

Parameters: ResourceID – Identifier of the process instance resource that has completed.

Exit – An optional indication of which exit point was reached.

ResultData – Context-specific data that represents the final values of all information relevant to this instance. This information will be encoded in data pairs containing the name and value of each item.

Example 29:

```
<?xml version="1.0"?>
<WF_XML>
  <request>
    <sessionid>0x14F351C</sessionid>
    <observer>
      <complete>
        <resourceid>xxx</resourceid>
        <exit>xxx</exit>
        <resultdata>
          <item>
            <name>xxx</name><value>xxx</value>
          </item>
        </resultdata>
      </complete>
    </observer>
```

```
  </request>
</WF_XML>
```

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

Example 30:

```
<?xml version="1.0"?>
<WF_XML>
<request>
    <sessionid>0x14F351C</sessionid>
    <observer>
      <complete>
        <resourceid>xxx</resourceid>
        <exit>xxx</exit>
        <resultdata>
          <item>
            <name>xxx</name><value>xxx</value>
          </item>
        </resultdata>
      </complete>
    </observer>
  </request>
  <response>
    <observer>
      <complete/>
    </observer>
    <exception>
    …
    </exception>
  </response>
</WF_XML>
```

Terminated – used to indicate to the observing resource that the identified process instance has been terminated before it reached its normal completion. This process instance resource may no longer exist after this point.

Parameters: ResourceID – Identifier of the process instance resource that has been terminated.

Reason – An optional description of why the instance was terminated prematurely.

Example 31:

```
<?xml version="1.0"?>
<WF_XML>
  <request>
    <sessionid>0x14F351C</sessionid>
    <observer>
      <terminated>
        <resourceid>xxx</resourceid>
        <reason>xxx</reason>
      </terminated>
    </observer>
  </request>
</WF_XML>
```

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

Example 32:

```
<?xml version="1.0"?>
<WF_XML>
<request>
    <sessionid>0x14F351C</sessionid>
    <observer>
      <terminated>
        <resourceid>xxx</resourceid>
        <reason>xxx</reason>
      </terminated>
    </observer>
  </request>
  <response>
    <observer>
      <terminated/>
    </observer>
    <exception>
    …
    </exception>
  </response>
</WF_XML>
```

Notify – used to send an event notification to a registered observer resource of a process instance.

Parameters: ResourceID – Identifier of the process instance resource on which some event has occurred.

EventObject – A list of information about the event:

```
************************************************************
**                          NOTE:                         **
**  A list of event types is defined in Appendix A of the audit data speci-  **
** fication (Interface 5), but no integer event codes are assigned to them.  **
**       If these are necessary, they must either be located or assigned.     **
************************************************************
```

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

- OldState – the former status of the resource, if this was a state changed event
- NewState – the current status of the resource, if this was a state changed event
- Transition – the name of the transition taken, if this was a state changed event
- ChangedData – a list of data items that were changed, if this was a data changed event. This information will be encoded in data pairs containing the name and value of each item.
- ChangedRole – the name of the role that changed, if this was a participant changed event
- Participants – a list of names of the participants of a role, if this was a participant changed event

Example 33:

```
<?xml version="1.0"?>
<WF_XML>
  <request>
    <sessionid>0x14F351C</sessionid>
    <observer>
      <notify>
        <resourceid>xxx</resourceid>
        <eventobject>
          <timestamp>xxx</timestamp>
          <eventcode>xxx</eventcode>
          <eventtype>xxx</eventtype>
          …
        </eventobject>
      </notify>
    </observer>
  </request>
</WF_XML>
```

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

Example 34:

```
<?xml version="1.0"?>
<WF_XML>
<request>
    <sessionid>0x14F351C</sessionid>
    <observer>
      <notify>
        <resourceid>xxx</resourceid>
        <eventobject>
          <timestamp>xxx</timestamp>
          <eventcode>xxx</eventcode>
          <eventtype>xxx</eventtype>
          …
        </eventobject>
      </notify>
    </observer>
  </request>
  <response>
    <observer>
```

```
      <notify/>
    </observer>
    <exception>
    …
    </exception>
  </response>
</WF_XML>
```

ActivityObserver

This interface is very similar to the Observer interface, with the primary difference being that it is relevant to a particular task, or "activity", within a process instance. Implementing the interface at this level provides a way to find the particular process instance that contains a given activity, which further provides a way of tracking all the activities contained by that process instance. Finally, if any of these activities are other (sub) process instances, they can also be discovered via this interface. This traversal of instance/activity relationships can support distributed, nested process management to any level. Because of this potential nesting, it is important to note that the ActivityObserver resource must supply/expect information going to/coming from a process instance to be in that instance's context. Therefore, data exchanged with that instance will be expressed with the set of fields relevant to that particular instance. This interface contains the methods PropFind, PropPatch and Complete.

PropFind - used to retrieve the values of all properties defined for the given activity. This set is defined to be the union of all the properties specified by all of the interfaces that the resource implements.

Parameters: ResourceID – Identifier of the activity whose properties are to be returned.

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

Example 35:

```
<?xml version="1.0"?>
<WF_XML>
  <request>
    <sessionid>0x14F351C</sessionid>
    <activityobserver>
      <propfind>
        <resourceid>xxx</resourceid>
      </propfind>
    </activityobserver>
  </request>
</WF_XML>
```

Results: Interfaces – The names of all the interfaces implemented on this resource. This result indicates the types of requests that can be made to this resource.

Name – A human readable identifier of the resource. This name is unique within the scope of its parent, and in some cases may be nothing more than a number.

Key – The globally unique identifier of the resource.

Subject – A short description of this process instance.

Description – A longer description of this process instance resource.

State – The current status of this resource.

ValidStates – A list of state values allowed by this resource.

ProcessInstance – the resource identifier of a subprocess instance, if this activity has a subprocess associated with it.

Container – the resource identifier of the process instance that contains this activity.

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

ContextData – Context-specific data relevant to this activity. This information will be encoded in data pairs containing the name and value of each item.

Assignees – a list of the names of people who are assigned to the activity

Conclusions – a list of conclusion values that may be used in the completed operation in order to indicate how the activity was completed.

UserInterface – the resource identifier of an interface to this activity that can be made accessible to a user by an application, if one exists.

ExpirationDate – an optional deadline for this activity.

HasExpired – a Boolean value (yes or no), indicating whether this value is currently beyond a deadline.

CreationDate – the date this activity was created.

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

Example 36:

```
<?xml version="1.0"?>
<WF_XML>
<request>
    <sessionid>0x14F351C</sessionid>
    <activityobserver>
      <propfind>
        <resourceid>xxx</resourceid>
      </propfind>
    </activityobserver>
  </request>
  <response>
```

```
    <activityobserver>
      <propfind>
        <interfaces>
          <processinstance/>
          <activityobserver/>
        </interfaces>
        <name>xxx</name>
        <key>xxx</key>
        …
      </propfind>
    </activityobserver>
    <exception>
    …
    </exception>
  </response>
</WF_XML>
```

PropPatch - used to set the values of any number of properties of the given activity.

Parameters: ResourceID – Identifier of the activity whose properties are to be set.

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

ResultData – Context-specific data that is the result of performing the activity. This can be a partial set of the data. This information will be encoded in data pairs containing the name and value of each item.

Example 37:

```
<?xml version="1.0"?>
<WF_XML>
  <request>
    <sessionid>0x14F351C</sessionid>
    <activityobserver>
      <proppatch>
        <resourceid>xxx</resourceid>
        <priority>xxx</priority>
        <resultdata>
          <item>
            <name>xxx</name><value>xxx</value>
          </item>
        </resultdata>
      </proppatch>
    </activityobserver>
  </request>
</WF_XML>
```

Results: Same as PropFind.

Example 38:

```
<?xml version="1.0"?>
<WF_XML>
<request>
    <sessionid>0x14F351C</sessionid>
```

```
    <activityobserver>
      <proppatch>
        <resourceid>xxx</resourceid>
        <priority>xxx</priority>
        <resultdata>
          <item>
            <name>xxx</name><value>xxx</value>
          </item>
        </resultdata>
      </proppatch>
    </activityobserver>
  </request>
  <response>
    <activityobserver>
      <proppatch>
        <interfaces>
          <processinstance/>
          <activityobserver/>
        </interfaces>
        <name>xxx</name>
        <key>xxx</key>
         …
      </proppatch>
    </activityobserver>
    <exception>
    …
    </exception>
  </response>
</WF_XML>
```

Complete – used to indicate that a work activity has been completed.

Parameters: ResourceID – Identifier of the activity that has been completed.

ResultData – Context-specific data that is the result of performing the activity. This can be a partial set of the data. This information will be encoded in data pairs containing the name and value of each item.

Option – An optional indication of the completion option that was chosen to complete the activity.

Example 39:

```
<?xml version="1.0"?>
<WF_XML>
  <request>
    <sessionid>0x14F351C</sessionid>
    <activityobserver>
      <complete>
        <resourceid>xxx</resourceid>
        <resultdata>
          <item>
            <name>xxx</name><value>xxx</value>
          </item>
        </resultdata>
        <option>xxx</option>
```

```
        </complete>
      </activityobserver>
    </request>
</WF_XML>
```

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

Example 40:

```
<?xml version="1.0"?>
<WF_XML>
<request>
    <sessionid>0x14F351C</sessionid>
    <activityobserver>
      <complete>
        <resourceid>xxx</resourceid>
        <resultdata>
          <item>
            <name>xxx</name><value>xxx</value>
          </item>
        </resultdata>
        <option>xxx</option>
      </complete>
    </activityobserver>
  </request>
  <response>
    <activityobserver>
      <complete/>
    </activityobserver>
    <exception>
    …
    </exception>
  </response>
</WF_XML>
```

WorkList

This interface allows the potentially numerous distributed processes being performed throughout an interoperating enterprise to be filtered based on their relevance to a particular (human) user. While the WorkList and WorkItem interfaces parallel the ProcessDefiniton and ProcessInstance interfaces in some respects, they differ in that WorkList and WorkItem represent only the information about work to be performed, rather than representing the actual resources that perform it. Implementing the WorkList interface, when a system assigns an activity to a user it would create a workitem for that user on the user's worklist server (identifiable via a user directory) that corresponds with the activity being assigned. This interface contains the same methods as the ProcessDefinition interface (PropFind, CreateProcessInstance and ListInstances).

PropFind - used to retrieve the values of all properties defined for the given process definition resource. This set is defined to be the union of all the properties specified by all of the interfaces that the resource implements.

Example 43:

```
<?xml version="1.0"?>
<WF_XML>
  <request>
    <sessionid>0x14F351C</sessionid>
```

```
    <worklist>
      <propfind>
        <resourceid>xxx</resourceid>
      </propfind>
    </worklist>
  </request>
</WF_XML>
```

CreateProcessInstance - used to instantiate a known process definition. The instance will be created with context-specific properties set according to the input data, and may optionally be started immediately upon creation. When invoked via this interface, a work item will also be added to the user's worklist server, which can be controlled through the WorkItem interface.

Example 44:

```
<?xml version="1.0"?>
<WF_XML>
  <request>
    <sessionid>0x14F351C</sessionid>
    <worklist>
      <createprocessinstance>
        <resourceid>xxx</resourceid>
        …
      </createprocessinstance>
    </worklist>
  </request>
</WF_XML>
```

ListInstances - used to retrieve a list of instances of the given process definition. Each instance is identified with its key, name and priority.

Example 45:

```
<?xml version="1.0"?>
<WF_XML>
  <request>
    <sessionid>0x14F351C</sessionid>
    <worklist>
      <listinstances>
        <resourceid>xxx</resourceid>
      </listinstances>
    </worklist>
  </request>
</WF_XML>
```

WorkItem

The WorkItem interface differs somewhat substantially from ProcessInstance in that it's resource does not have any predefined action. It is merely a pointer to another resource that performs an activity, picking up the description of the task from the activity. This interface also provides a means of indicating the activity's status to the user. The user does not directly interact with the workitem, but rather with the activity via the workitem's pointer. Once the activity is completed, the workitem should be removed from the user's worklist. This interface contains the same methods as the ActivityObserver interface (PropFind, PropPatch and Complete).

PropFind - used to retrieve the values of all properties defined for the given process instance resource. This set is defined to be the union of all the properties specified by all of the interfaces that the resource implements.

Example 46:

```
<?xml version="1.0"?>
<WF_XML>
  <request>
    <sessionid>0x14F351C</sessionid>
    <workitem>
      <propfind>
        <resourceid>xxx</resourceid>
      </propfind>
    </workitem>
  </request>
</WF_XML>
```

PropPatch - used to set the values of any number of properties of the given activity.

Example 47:

```
<?xml version="1.0"?>
<WF_XML>
  <request>
    <sessionid>0x14F351C</sessionid>
    <workitem>
      <proppatch>
        <resourceid>xxx</resourceid>
        …
      </proppatch>
    </workitem>
  </request>
</WF_XML>
```

Complete -

Example 48:

```
<?xml version="1.0"?>
<WF_XML>
  <request>
    <sessionid>0x14F351C</sessionid>
    <workitem>
      <complete>
        <resourceid>xxx</resourceid>
        …
      </complete>
    </workitem>
  </request>
</WF_XML>
```

EntryPoint

This interface is used to create a process instance with a particular starting point within a process definition. If this interface is to be implemented, the process definition resource will have to provide a series of predefined entrypoints. Each entrypoint can potentially have it's own input requirements, access control, etc., which will have to be enforced by the implementation. This interface provides valuable functionality, but is optional, as process instances can be created directly through the ProcessDefinition

interface if no special entrypoint is required. This interface contains the methods PropFind and CreateProcessInstance.

PropFind - used to retrieve the values of all properties defined for the given process definition resource. This set is defined to be the union of all the properties specified by all of the interfaces that the resource implements.

Parameters: ResourceID – Identifier of the process definition resource whose properties are to be returned.

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

Example 49:

```
<?xml version="1.0"?>
<WF_XML>
  <request>
    <sessionid>0x14F351C</sessionid>
    <entrypoint>
      <propfind>
        <resourceid>xxx</resourceid>
      </propfind>
    </entrypoint>
  </request>
</WF_XML>
```

Results: Interfaces – The names of all the interfaces implemented on this resource. This result indicates the types of requests that can be made to this resource.

Name – A human readable identifier of the resource. This name is unique within the scope of its parent, and in some cases may be nothing more than a number.

Key – The globally unique identifier of the resource.

Subject – A short description of this process instance.

Description – A longer description of this process instance resource.

Conclusions – A list of conclusion values that can be used in the CreateProcessInstance method.

StartPointList – An optional list of predefined starting points within the identified process definition.

ContextDataInfo – Metadata about the context-specific information required to create an instance of this process definition. This information will be encoded in data pairs containing the name and type of each item.

ResultDataInfo – Metadata about the context-specific information required to be returned by an instance of this process upon completion. This information will be encoded in data pairs containing the name and type of each item.

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

Example 50:

```
<?xml version="1.0"?>
<WF_XML>
<request>
    <sessionid>0x14F351C</sessionid>
    <entrypoint>
      <propfind>
        <resourceid>xxx</resourceid>
      </propfind>
    </entrypoint>
  </request>
  <response>
    <entrypoint>
      <propfind>
        <interfaces>
          <processinstance/>
          <activityobserver/>
        </interfaces>
        <name>xxx</name>
        <key>xxx</key>
         …
      </propfind>
    </entrypoint>
    <exception>
    …
    </exception>
  </response>
</WF_XML>
```

CreateProcessInstance - used to instantiate a known process definition. The instance will be created with context-specific properties set according to the input data, and may optionally be started immediately upon creation. When invoked via this interface, a starting point may also be specified. A list of predefined starting points may be obtained through the PropFind method in this interface.

Parameters: ResourceID – Identifier of the process definition resource that is to be used to create this instance.

Observer – Identifier of the resource that is to be the observer of the instance that is created by this method. This observer resource is then to be notified of state changes to the instance, most notably the completion of the instance.

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.

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

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

StartPoint – The name of a predefined starting point within the identified process definition resource.

ContextData – Context-specific information required to create this process instance. This information will be encoded in data pairs containing the name and value of each item.

StartImmediately – An optional Boolean value (yes or no), indicating whether the newly created instance should be started immediately upon creation. The default is "yes".

Example 51:

```
<?xml version="1.0"?>
<WF_XML>
  <request>
    <sessionid>0x14F351C</sessionid>
    <entrypoint>
      <createprocessinstance>
        <resourceid>xxx</resourceid>
        <observer>xxx</observer>
        <name>xxx</name>
        <subject>xxx</subject>
        <description>xxx</description>
        <startpoint>xxx</startpoint>
        <contextdata>
          <item>
            <name>xxx</name><value>xx</value>
          </item>
        </contextdata>
        <startimmediately>yes</startimmediately>
      </createprocessinstance>
    </entrypoint>
  </request>
</WF_XML>
```

Results:

Key – the resource identifier of the newly created process instance.

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

Example 52:

```
<?xml version="1.0"?>
<WF_XML>
<request>
    <sessionid>0x14F351C</sessionid>
    <entrypoint>
      <createprocessinstance>
        <resourceid>xxx</resourceid>
        <observer>xxx</observer>
        <name>xxx</name>
        <subject>xxx</subject>
        <description>xxx</description>
        <startpoint>xxx</startpoint>
```

```
        <contextdata>
          <item>
            <name>xxx</name><value>xx</value>
          </item>
        </contextdata>
        <startimmediately>yes</startimmediately>
      </createprocessinstance>
    </entrypoint>
  </request>
  <response>
    <entrypoint>
      <createprocessinstance>
        <key>xxx</key>
      </createprocessinstance>
    </entrypoint>
    <exception>
    …
    </exception>
  </response>
</WF_XML>
```

3 Implementation Issues

3.1 Interoperability Contract

As this specification, and work on interoperability in general, is very preliminary, it is recognized that there will be many additional requirements that vendors will wish to fulfill and many features they will wish to provide that are not accounted for in this specification. Furthermore, it may be necessary for interoperating implementations to agree on fundamental issues not addressed herein, even to achieve basic interoperability. For these reasons, it is recommended that an interoperability contract be established among vendors participating in interoperable workflows. This contract will clearly define each vendor's expectations and requirements in all areas that may impede interoperabilty. A list of topics to be included in the interoperability contract is provided here as an example, but this list should by no means be considered complete. Each interoperating vendor must ensure that all factors impacting their implementation are addressed completely.

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

- Feature/Function Requirements – application-specific data required to be transferred in order to utilize basic or extended functionality.
- Filter Values – for use in those methods that allow result filtering. Specifics of the filtering mechanism and allowable values should be agreed upon.
- Data Types – any specific data types used by a vendor that differ from those described in this specification, or of which an interoperating vendor must be aware.
- Data Constraints – field lengths, allowable characters, character set encodings, overall message size, etc.
- Error Codes – application-specific error handling requirements such as codes, descriptions, required actions, etc.
- Protocol Limitations – required header data, timeout values, buffer sizes, etc.
- Security Considerations – encryption methods, user verification, firewall configuration requirements, etc.

4 Conformance

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

```
**********************************************************
**                        NOTE:                         **
**   It might also help to define "levels" of conformance based on how  **
**    many or which interfaces a system implements, just to add some    **
**                  definition to the procedure.                       **
**********************************************************
```

4.1 Validity vs. Well-Formedness

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

For this reason, this specification does not mandate validity of all document instances, rather it only requires that all Wf-XML messages be well-formed. An additional consideration in this respect is the fact that this specification is preliminary and remains a living document. It is very unlikely that all content models are structured adequately to support the degree of functionality and flexibility required at this point in the evolution of the spec. 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.

4.2 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 fucntionality 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 some level of conformance.

This can be achieved in two ways. First, as described above, this specification only requires well-formed data. Therefore, interoperating vendors may exchange any data they wish so long as that data meets the syntactic requirements of the XML specification. Although it would obviously be best from a functional perspective if the vendors were able to agree upon this data's markup, if they cannot the recipient of the unkown 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 provdied here.

Secondly, there is the case where a vendor wishes to support markup defined in another DTD (possibly from HTML, an existing SGML DTD, or some other related standard). XML provides for this type of extension in the form of namespace declarations [11]. For example, if a Wf-XML message needed to include elements from the HTML 4.0 DTD it might define and use namespaces as follows:

```
<?xml version="1.0">
  <WF_XML xmlns:html="http://www.w3c.org/TR/REC-html40">
  …
    <html:a href="http://www.blahblah.com/blah">
    …
  </WF_XML>
```

```
**************************************************************
**                          NOTE:                           **
**  We should probably define a namespace for WF_XML, something like:  **
**              "http://www.wfmc.org/standards/docs/WF_XML.             **
**************************************************************
```

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

Appendix A - Mapping of Methods to Interfaces

This appendix is provided as a convenience to implementers of this specification. It provides information pertaining to the relationships among the various interfaces and methods defined for use in Wf-XML transactions. The following table indicates which methods are contained by each interface, and conversely which interfaces contain each method.

	Process Definition	Process Instance	Observer	Activity Observer	Work List	Work Item	Entrypoint
PropFind	X	X	X	X	X	X	X
PropPatch		X	X	X		X	
CreateProcess Instance	X				X		X
ListInstances	X				X		
Terminate		X					
Subscribe		X					
Unsubscribe		X					
GetHistory		X					
Complete			X	X		X	
Terminated			X				
Notify			X				

Table 1. Methods vs. Interfaces

Appendix B – EXCEPTIONS

This appendix provides a listing of exceptions and their types in a Wf-XML response message.

****
** NOTE: **
** Exceptions suggested below are based on current specification. Further analysis and discussions are necessary in this area. **
****

Exception Type	Exception Message
Fatal	Invalid XML Document
Fatal	Unknown Observer Method
Fatal	Invalid Resource ID
Fatal	Invalid Method
Fatal	User not authorized for the requested functions.
Fatal	Invalide State
Fatal	Process ID is missing
Fatal	Invalid Attribute Specified
Warning	The XML version given has been updated.
Warning	User Account will soon be expired
Warning	Invalid Attribute Specified
Non-Fatal	Response is missing one or more data elements
Non-Fatal	Inactive Process Instance Specified
Application	**Application specific errors**

Table 2. Exceptions

Appendix C - Consolidated Tag Table and DTD

This appendix provides a listing of all the XML elements defined for use in Wf-XML messages. In addition, the WF_XML DTD is provided for the purposes of implementation reference and optional data validation by an XML processor. The following lists indicate each tag name based on its purpose within a message. There are currently 87 elements defined for use in Wf-XML messages.

General message elements:

- WF_XML
- request
- response
- sessionid
- item
- name
- value
- type
- exception
- msg

Status elements:

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

Interface elements:

- processdefinition
- processinstance
- observer
- activityobserver
- worklist
- workitem
- entrypoint

Method elements:

- propfind
- proppatch
- createprocessinstance
- listinstances
- terminate
- subscribe
- unsubscribe
- gethistory
- complete
- terminated
- notify

Parameter elements:

- resourceid
- resultdataattributes
- subject
- description
- state
- priority
- startpoint
- contextdata
- startimmediately
- filter
- filtertype
- reason
- observer
- exit
- option
- resultdata
- eventobject

Result elements:

- interfaces
- key
- validstates
- definition
- activities
- activity
- observers
- expirationdate
- assignees
- conclusions
- creationdate
- hasexpired
- contextdatainfo
- resultdatainfo
- userinterface
- creator
- lastmodified
- history
- event
- timestamp
- eventcode
- eventtype
- responsible
- sourcekey
- sourcename
- containerkey
- oldstate
- newstate
- transition
- changeddata
- changedrole
- participants
- instances
- instance
- startpointlist
- performer

The DTD for Wf-XML messages is provided below. This DTD is designed with the intention of being simple and easy to implement, while supporting a robust and flexible structure. It employs no attribute definitions, elaborate entity references or special character encodings.

```
<!--
      This declaration set may be referenced as an external entity
using the following identifiers:

          ************************************************************
          **                        NOTE:                           **
          **          The following URL is yet to be determined.    **
          ************************************************************

            SYSTEM "http://www.wfmc.org/standards/docs/..."
            PUBLIC "-//WfMC//DTD Wf-XML 1.0//EN" SYSTEM
http://www.wfmc.org/standards/docs/...

Processors requiring a DOCTYPE wrapper should prepend the string
"<!DOCTYPE Wf-XML [" and append the string "]>" to this declaration
set.
-->

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

<!-- These are the currently defined Wf-XML Interfaces. -->
<!ENTITY % interfaces "processdefinition | processinstance | observer |
activityobserver | worklist | workitem | entrypoint">

<!ENTITY % results "interfaces | name | key | subject | description |
state | validstates | contextdatainfo | resultdatainfo | definition |
activities | observers | startpointlist | contextdata | resultdata |
priority | userinterface | creator | lastmodified | performer |
processinstance | containerkey | assignees | conclusions |
expirationdate | hasexpired | creationdate">

<!ENTITY % evntdata "timestamp, eventcode, eventtype, responsible,
sourcekey, sourcename, containerkey, oldstate?, newstate?, transition?,
changeddata?, changedrole?, participants?">

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

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

<!ELEMENT WF_XML (request | (request, response))>

<!ELEMENT request (sessionid?, (%interfaces;))>

<!ELEMENT response ((%interfaces;), exception)>

<!ELEMENT sessionid (CDATA)>

<!ELEMENT item (name, (value | type))+>

<!ELEMENT name (CDATA)>
```

```
<!ELEMENT value (CDATA)>

<!ELEMENT type (CDATA)>

<!ELEMENT exception (type, msg, contextdata*, exception?)>

<!ELEMENT msg (#PCDATA)>

<!ELEMENT open.notrunning.notstarted (EMPTY)>

<!ELEMENT open.notrunning.suspended (EMPTY)>

<!ELEMENT open.running (EMPTY)>

<!ELEMENT closed.completed (EMPTY)>

<!ELEMENT closed.terminated (EMPTY)>

<!ELEMENT closed.aborted (EMPTY)>

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

<!ELEMENT processdefinition (propfind | createprocessinstance |
listinstances)?>

<!ELEMENT processinstance (propfind | proppatch | terminate | subscribe
| unsubscribe | gethistory)?>

<!ELEMENT observer (propfind | proppatch | complete | terminated |
notify)?>

<!ELEMENT activityobserver (propfind | proppatch | complete)?>

<!ELEMENT worklist (propfind | createprocessinstance | listinstances)?>

<!ELEMENT workitem (propfind | proppatch | complete)?>

<!ELEMENT entrypoint (propfind | createprocessinstance)?>

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

<!--
   As these method elements will appear in both request and response
messages, their content models follow the form :
            ( (parameters) | (responses) )
   This model allows the method elements to contain allowable parameter
elements when the method's context is a request, and allowable response
elements when its context is a response. In cases where no responses
are modeled, the only response is an exception, which is contained
within every response.
-->

<!ELEMENT propfind ( (resourceid, resultdataattributes?) |
(%results;)*)>

<!ELEMENT proppatch ( (resourceid, (subject | description | state |
priority | contextdata | resultdata)+) | (%results;)*)>
```

```
<!ELEMENT createprocessinstance ( (resourceid, observer, name, subject,
description, startpoint?, contextdata, startimmediately?) | key )>

<!ELEMENT listinstances ( (resourceid?, filter?, filtertype?) |
instances* )>

<!ELEMENT terminate (resourceid, reason)>

<!ELEMENT subscribe (resourceid, observer)>

<!ELEMENT unsubscribe (resourceid, observer)>

<!ELEMENT gethistory ( (resourceid, filter?, filtertype?) | history )>

<!ELEMENT complete (resourceid, exit?, resultdata?, option?)>

<!ELEMENT terminated (resourceid, reason?)>

<!ELEMENT notify (resourceid, eventobject)>

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

<!ELEMENT resourceid (CDATA)>

<!ELEMENT resultdataattributes (%results;)+>

<!ELEMENT subject (CDATA)>

<!ELEMENT description (CDATA)>

<!ELEMENT state (%vstates;)>

<!ELEMENT priority (CDATA)>

<!ELEMENT startpoint (CDATA)>

<!ELEMENT contextdata (item+)>

<!ELEMENT startimmediately (CDATA)>

<!ELEMENT filter (CDATA)>

<!ELEMENT filtertype (CDATA)>

<!ELEMENT reason (CDATA)>

<!ELEMENT observer (CDATA)>

<!ELEMENT exit (CDATA)>

<!ELEMENT option (CDATA)>

<!ELEMENT resultdata (item+)>

<!ELEMENT eventobject (%evntdata;)>
```

```
<!-- ~~~~~~~~~~ Results ~~~~~~~~~~~~ -->

<!ELEMENT interfaces (%interfaces;)>

<!ELEMENT key (CDATA)>

<!ELEMENT validstates (%vstates;)+>

<!ELEMENT definition (CDATA)>

<!ELEMENT activities (activity*)>

<!ELEMENT activity (key, name, state, expirationdate, assignees,
creationdate, hasexpired)>

<!ELEMENT observers (key+)>

<!ELEMENT expirationdate (CDATA)>

<!ELEMENT assignees (name+)>

<!ELEMENT conclusions (CDATA)>

<!ELEMENT creationdate (CDATA)>

<!ELEMENT hasexpired (CDATA)>

<!ELEMENT contextdatainfo (item+)>

<!ELEMENT resultdatainfo (item+)>

<!ELEMENT userinterface (CDATA)>

<!ELEMENT creator (CDATA)>

<!ELEMENT lastmodified (CDATA)>

<!ELEMENT history (event+)>

<!ELEMENT event (%evntdata;)>

<!ELEMENT timestamp (CDATA)>

<!ELEMENT eventcode (CDATA)>

<!ELEMENT eventtype (CDATA)>

<!ELEMENT responsible (CDATA)>

<!ELEMENT sourcekey (CDATA)>

<!ELEMENT sourcename (CDATA)>

<!ELEMENT containerkey (CDATA)>

<!ELEMENT oldstate (%vstates;)>
```

```
<!ELEMENT newstate (%vstates;)>

<!ELEMENT transition (CDATA)>

<!ELEMENT changeddata (item+)>

<!ELEMENT changedrole (CDATA)>

<!ELEMENT participants (name+)>

<!ELEMENT instances (instance+)>

<!ELEMENT instance (key, name, priority)>

<!ELEMENT startpointlist (startpoint+)>

<!ELEMENT performer (key+)>
```

Appendix D - Example Bindings

D.1 HTTP

Introduction

This section is extracted from detailed requirements for interacting with the JCALS/Wf-XML interface from the perspective of an outside or remote workflow engine wishing to start, monitor, and receive status of JCALS process definitions and instances. It describes the structure of variables passed in URLs, the XML file layout, and the handling of login and password information into JCALS/Wf-XML on a non-interactive interface. The JCALS/Wf-XML engine currently uses HTTP as a communication protocol.

Format of JCALS/Wf-XML URLs

The JCALS Wf-XML engine uses Microsoft's Internet Service Application Interface or Netscape's Service Application Interface (ISAPI/NSAPI) and the Component Object Module (COM) interface on a Windows NT server platform. As such, all URLs directed to JCALS/Wf-XML must include the module and method names of these objects in the URL. This allows the JCALS web server to initialize the correct COM object for Wf-XML messages.

Here is an example of a URL including the address of a JCALS web server and the required COM directive:

http://www.jcalswebserver.com/jpwss/cgi-bin/jpcweb.dll?Scraper&ComId=JPCSS.SWAP^MethodName=JSWAP

- http://www.jcalswebserver.com/ defines the HTTP address of the JCALS web server.
- jpwss/cgi-bin/jpcweb.dll represents the path and name of the ISAPI interface (jpcweb.dll)
- ?Scraper is the name of the method used for Wf-XML (the leading '?' is required)
- &ComId=JPCSS.SWAP is the name of the Wf-XML COM object (the leading '&' is required)
- ^MethodName=JSWAP is the COM method name used for external Wf-XML interaction.

The actual text of the Wf-XML file is passed as part of the URL. This is done by adding the URL variable 'WF_XML=' followed by the text of the XML files as a string. For example:

WF_XML= <?xml version="1.0" ?><WF_XML><INTERFACE>PROCESSINSTANCE</INTERFACE><METHOD>TERMINATE</METHOD><REASON>Instance no longer required</REASON></WF_XML>

The Wf-XML HTTP interface uses several name/value variables that are passed with the URL. These variables are used directly by the JCALS/Wf-XML engine and are application specific, i.e., not part of the generic Wf-XML protocol. These are outlined in the table below.

Variable	***Value***	***Comment***
ComID	JPCSS.SWAP	Always set to this value
MethodName	JSWAP	Always set to this value
UserOid	0123456789123456	The JCALS UserObject Id (always 16 chars)
SiteName	LocalDataServer	The name of the JCALS data server descriptor at your site (for non-interactive logins)
UserName	JCALS user name	Used for non-interactive logins
Password	JCALS password	Used for non-interactive logins
AccessCode	JCALS database password	Used for non-interactive logins
INSTANCE_ID	01234567890123456	The Object ID of either a ProcessDefinition or a Process Instance
WF_XML	<?xml version="1.0" …	The actual XML of the Wf-XML request
WF_URL	http://www.swapplace.com/proc_inst.xml	The address of a Wf-XML page which can be retrieved using HTTP

For non-interactive, server-to-server access, the **SiteName**, **UserName**, **Password**, and **AccessCode** are passed with the initial Wf-XML URL request (Sitename is be optional). This information

is required to assure strong identification and authentication to enable interaction with the secure JCALS system. This validation starts a JCALS session for the given user. JCALS open sessions will terminate after one hour of inactivity.

NOTE: It is assumed that Secure Socket Layer (SSL) will be utilized to transmit these values, and for all HTTP communications dealing with JCALS/Wf-XML.

After the initial Wf-XML request is made and a JCALS session is started, subsequent requests need only include the **UserOid** to link the user back to the appropriate session. The **UserOid** can be obtained from the <KEY> tag returned with the XML response of several Wf-XML methods (e.g. LISTINSTANCES, CREATEPROCESSINSTANCE). Calls to most Wf-XML methods require the **INSTANCE_ID** variable specified with the URL or embedded in the XML file. The value specified with **INSTANCE_ID** is used to notify the Wf-XML module of a particular Process Definition or Process Instance associated with a request. The **INSTANCE_ID** for a *PROCESSDEFINITION* can be obtained through from the <KEY> value returned with a LISTINSTANCES request. The **INSTANCE_ID** for a *PROCESSINSTANCE* is returned with the <KEY> tag following a CREATEPROCESSINSTANCE request.

Wf-XML XML files may be passed with the URL using the **WF_XML** variable. This would normally be the last variable attached to the URL (see above).

Example

PROCESSDEFINITION
METHOD: CREATEPROCESSINSTANCE

URL

The URL for a CREATEPROCESSINSTANCE method is the URL returned in the <KEY> tag of LISTINSTANCES.

Example: http://www.swapserver.com/jpwss/cgi-bin/jpcweb.dll?Scraper&ComId=JPCSS.SWAP^MethodName=JSWAP^UserOid=000C000200003F4E^INSTANCE_ID=000700020000612E

XML Request

```
<?xml version="1.0" standalone="yes"?>
<?Template ID: INSTANCE_ID=0006000200004FB1 ?>

<WF_XML>
        <INTERFACE>
                PROCESSDEFINITION
        </INTERFACE>
        <METHOD>
                CREATEPROCESSINSTANCE
        </METHOD>
        <OBSERVER>
                http://www.someobserver.com/
        </OBSERVER>
        <EMAIL>
                rheim@jcals.csc.com
        </EMAIL>
        <NAME>
                test_template
        </NAME>
        <SUBJECT>
                TEST_JOB46
        </SUBJECT>
        <DESCRIPTION>
                A Wf-XML CREATEPROCESSINSTANCE CREATED JOB
```

```
        </DESCRIPTION>
        <CONTEXTDATA>
                <TEMPLATEORGKEY>
                        0
                </TEMPLATEORGKEY>
                <WFOLDERNAME>
                        WORKFOLDER_46
                </WFOLDERNAME>
                <WFOLDERKEY>
                        0
                </WFOLDERKEY>
                <ALLOWOTHEROBSERVERS>
                        TRUE
                </ALLOWOTHEROBSERVERS>
                <WFOLDERFILE1>
                        http://www.server.com/swap/stage/TICA806.wmf
                </WFOLDERFILE1>
                <WFOLDERFILE2>
                        http://www.server.com/swap/stage/http06.doc
                </WFOLDERFILE2>

                <INCLUDEXML>
                        YES
                </INCLUDEXML>
        </CONTEXTDATA>
        <STARTIMMEDIATELY>
                YES
        </STARTIMMEDIATELY>
</WF_XML>
```

Notice the addition of two files to be included in the JCALS Workfolder that is created with this request. The tag <WFOLDERFILEX> is used to give the HTTP address of each file to be included in the workfolder. The X represents a number that is incremented for each file to be included, as in the example. The current implementation supports up to 20 files.
The INCLUDEXML tag tells the JCALS Wf-XML module to save the XML request file in the workfolder.

XML Response

```
<?xml version="1.0" standalone="yes" ?>
<WF_XML>
        <KEY>
                http://www.swapserver.com/jpwss/cgi-
        bin/jpcweb.dll?Scraper&ComId=JPCSS.SWAP^MethodName=JSWAP^UserOid=000C0
        00200003E8F^INSTANCE_ID=0007000200008B02
        </KEY>
        <EXCEPTION>
                NONE
        </EXCEPTION>
</WF_XML>
```

The <KEY> value returned is the URL of the created process instance. The INSTANCE_ID is passed with the URL of any PROCESSINSTANCE or OBSERVER method to identify the ProcessInstance (or JCALS job).

Appendix E - 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, 29 Nov. 1994. http://www.wfmc.org/standards/docs/rmv1-16.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, 2.0, June 1996. http://www.wfmc.org/standards/docs/glossary.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