Wf-XML 2.0

XML Based Protocol for Run-Time Integration of Process Engines

Keith D. Swenson, Fujitsu Software Corporation.
Sameer Pradhan, Fujitsu Software Corporation.
Mike D. Gilger, Identitech.

Draft October 8, 2004

This document represents a proposal for a protocol standard. This version has not been approved for adoption. Future version may change substantially from this version.

Abstract

A standard protocol is needed to integrate process engines across the Internet or intranet and provide for their interaction. A process engine, a special type of asynchronous service (as defined by the Asynchronous Services Access Protocol (ASAP)), has a set of activities that represent steps in the execution of its service. Exposing those steps allows the service invoker to gain additional insight into the status of that service.

ASAP is a proposed way to provide the basic ability to control and monitor an asynchronous web service through the use of Simple Object Access Protocol (SOAP), and by transferring structured information encoded in XML. Controlling an asynchronous web service includes creating the service, setting up the service, starting the service, stopping the service, being informed of exceptions, being informed of the completion of the service and getting the results of the service. Monitoring the web service consists of checking on the current status of the service and getting a history of the execution of the service.

The external program that invokes a process need use only ASAP for the basic starting and monitoring. Wf-XML then builds upon and extends this interface for the special case of this service being a process engine. These extensions allow one to retrieve the list of activities that the process is waiting on. The activities can provide additional information about who is assigned to the activities, and possibly the remote sub-processes that have been invoked to fulfill the activities. Thus, Wf-XML takes the protocol from a simple exchange to start and monitor a process, to introspection of chains of related processes all invoked to satisfy the original goal.

This document will:

- Provide an executive overview.
- Specify the goals of Wf-XML as an extension of ASAP.
- Explain how resource (object) model works and how URIs are used to invoke methods of those resources.
- Specify preliminary details of the interface methods and parameters.

Table of contents

1.0 Executive Overview

1.1 Summary

ASAP offers a way to start an instance of an asynchronous web service (AWS), monitor it, control it, and be notified when it is complete. This service instance can perform just about anything for any purpose. The key aspect is that the service instance is something that one would like to start remotely, and it may take a long time to run to completion. Short-lived services could be invoked synchronously with SOAP and one would simply wait for completion, but because these process instances could last anywhere from a few minutes to a few months, they must be invoked asynchronously.

Wf-XML extends this in the special case that the asynchronous service is being invoked on a process engine. The Service Factory maps to a Process Definition; the Service Instance maps to a Process Instance. Process engines provide some additional capabilities for monitoring the process. First of all, because it is a process, and not simply an opaque service, there is a process diagram. This diagram can be retrieved for introspection. Second, since the process is composed of activities, one can ask the activities for their current values. An activity may itself represent an invocation of a yet another remote service, and the address of that service instance may be retrieved. Thirdly, the process definitions can be edited, removed, or added.

ASAP has established that a standard protocol is needed to connect asynchronous services, whether they are implemented as process engines or not. Wf-XML provides a way to merge process-oriented tools into the generic invocation framework. A process definition tool now has a standard way to retrieve or update a process definition. A process-monitoring tool can do the same for tracking process instances, and can follow links to sub-processes as well as sub-sub-processes.

1.2 Discussion of this Draft

Discussions on this draft are carried out on a forum list:

http://www.workflow-research.de/Forums/

Comments on the draft should be entered into the WG4 forum at that address. Please check the following web site for instructions on how to join the forum and for information on other documents from the same working group:

http://www.wfmc.org/

2 Goals

2.1 Problem Statement

ASAP allows one to start a remote service instance that is a process instance. We need a way to:

- Find the activities within that process that are currently being waited for.

- For each activity, find out who or what the activity is waiting for.
- Retrieve the process definition for the factory of an instance.
- Retrieve a list of process definitions (factories).
- Be able to add new definitions (new Service Factories) to the server.

2.2 Things to Achieve

In order to have a realizable agreement on useful capabilities in a short amount of time, it is important to be very clear about the goals of this effort.

- The Wf-XML extensions should adhere to the original guidelines of ASAP. The protocol should not reinvent anything unnecessarily. If a suitable standard exists, it should be used rather than re-implemented a different way. The protocol should be consistent with XML Protocol and SOAP.
- The protocol should be the minimal necessary to support a process engine.
- The protocol must be extensible. The first version will define a very minimal set of functionality. Yet a system must be able to extend the capability to fit the needs of a particular system, such that high level functionality can be communicated which gracefully degrades to interoperate with systems that do not handle those extensions.
- Like other Internet protocols, Wf-XML should not require or make any assumptions about the platform or the technology used to implement the generic asynchronous service.
- Terseness of expression is not a goal of this protocol. Ease of generation and parsability should be favored over compactness.

2.3 Things not part of the goals

It is also good practice to clearly demark those things that are not to be covered by the first generation of this effort:

- Like ASAP, Wf-XML is not designed to handle the transfer of large amounts of process relevant data. The data associated with, and stored within, a service instance is not anticipated to be more than 64K Bytes, and in most cases will be quite a bit less than this. If larger amounts of data than this are needed then it is anticipated that this data would be placed into some form of external service and that the ASAP will only need to carry the URI to that data. For example, a document would be placed on a web server and given the URI would be retrieved through normal HTTP.
- Wf-XML does not specify security. Rather, it relies on transport or session layer security. ASAP can adopt SOAP – specific security protocols once they are finalized.
- Like ASAP, WfXML is attempting to define the reliability aspect of messaging for a web service interaction. Message reliability is considered to be part of the transport which ASAP and WfXML runs on top of. In the “WS” environment there exists a proposal for WS-Reliability which has been shown to be compatible with ASAP, and might be used, but any other such standard, if one is to be ratified, should be suitable.

These may be added in a later revision, but there is no requirement to support these from the first version, and so any discussion on these issues should not be part of ASAP working group meetings.

2.4 Audience

This document is intended for vendor organizations who wish to implement the Wf-XML protocol in e-commerce process flow engines, and also for consultants, VARs, or third party developers who wish to make a Wf-XML wrapper for an existing system service in order to integrate that service into a process flow or workflow environment.

2.5 Terminology

Web Service: a program or service accessible using XML and Internet technologies. Such a service might be found using UDDI, and might be described using WSDL. The typical interaction is: an originator makes a request is made to a web service, which performs the operation, and returns the result.

Asynchronous Web Service: A web service or set of web services designed around a mode of operation where a request is made to start an operation, and a later separate request is made to communicate the results of the operation. A number of requests may be made in between in order to control and monitor the asynchronous operation. The results of the operation may be delivered either by polling requests from the originator, or else by a notification request originated by the performer.

Method: An individual interoperable function is termed a “method”. Each method may be passed a set of request parameters and return a set of response parameters.

Resource types: Methods are divided into different groups to better identify their context. The primary groups of methods required for interoperability are named Instance, Factory, and Observer.

Instance Resource: This is the resource implemented by the Web Service that is actually performing the requested work. These resources allow for the actual monitoring and controlling of the work. Also called a Process Instance.

Factory Resource: This is the resource implemented by the service instance factory. Methods are provided to start new service instances, to list or search for existing instances, and to provide definitional information about the instances. Also called a process definition resource.

Observer Resource: This is a resource that a web service must implement in order to receive notification events from the service instance.

Process Definition Resource: synonymous with Factory Resource, but used when the factory is a process engine capable of describing the process it is enacting.

Process Instance: synonymous with Instance Resource.

Service Registry Resource: a resource that can add and remove process definitions (service factories) from the server. It is itself a service meta-factory, whose instances are factories.

Context Data: The XML data sent to initiate the service.

Result Data: The XML data created by the successful completion of the service.

2.6 Related Documents

An understanding of SOAP and how it works is assumed in order to understand this document.

Information on ASAP is available from the following address:

http://www.oasis-open.org/apps/org/workgroup/asap/index.php

as well as on the WfMC website.

2.7 Notation conventions

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119

The following namespace prefixes are used throughout this document:

Prefix	Namespace URI	Definition
wfx	http://www.wfmc.org/wfxml/2.0/	Wf-XML namespace
as	http://docs.oasis-open.org/asap/0.9/asap.xsd	ASAP namespace
env	http://schemas.xmlsoap.org/soap/envelope/	Envelope namespace from SOAP 1.1
enc	http://schemas.xmlsoap.org/soap/encoding/	Encoding namespace from SOAP 1.1
xsd	http://www.w3.org/2001/XMLSchema	XML Schema namespace

Table 1 Namespaces

This specification uses an informal syntax we call *pseudo-XML* to describe the XML grammar of an ASAP document. This syntax is similar to that employed by the WSDL 1.1 specification

Convention	Example
The syntax appears as an XML instance, but the values indicate the data types instead of values.	`<p:tag name="nmtoken"/>`

Paragraphs within tags are the description of the tag and should be thought of as commented out with <!-- -->	`<p:tag>`
	`  longer description of the`
	`  purpose of the tag.`
	`</p:tag>`

Characters are appended to elements and attributes as follows: "?" (0 or 1), "*" (0 or more), "+" (1 or more).	`<p:tag>*`
Elements names ending in "…" indicate that elements/attributes irrelevant to the context are being omitted or they are exactly as defined previously.	`<p:tag.../>`
Grammar in bold has not been introduced earlier in the document, or is of particular interest in an example.	`<p:tag/>`
"Extensible element" is a placeholder for elements from some "other" namespace (like ##other in XSD).	`<-- extensible element -->`
The XML namespace prefixes (defined above) are used to indicate the namespace of the element being defined	

Examples starting with <?pseudo-xml?> contain enough information to conform to this specification; others examples are fragments and require additional information to be specified in order to conform.	`<?pseudo-xml?>`

Table 2 Pseudo-XML documentation conventions

Formal syntax is available in supplementary XML Schema and WSDL specifications in the document.

3 Resource Model

3.1 Resources Type Overview

For the support of an asynchronous web service (AWS), five types of web services are defined to match the three roles of the interaction: Observer, ServiceRegistry, Factory, Instance, and Activity. A Web Service type is distinguished by the group of operations it supports, and so there are five groups of operations.

Three of these resources are described by ASAP: the observer, factory, and instance resources. The resources retain all the same methods and meanings from ASAP, but are extended with a couple of new methods.

For the purpose of discussion we describe the different role as if they came from different web services, but there is no requirement that each resource type be provided by a different web service; it is possible that a single web service may play more than one role in the interaction simply by implementing all the methods in more than one resource type.

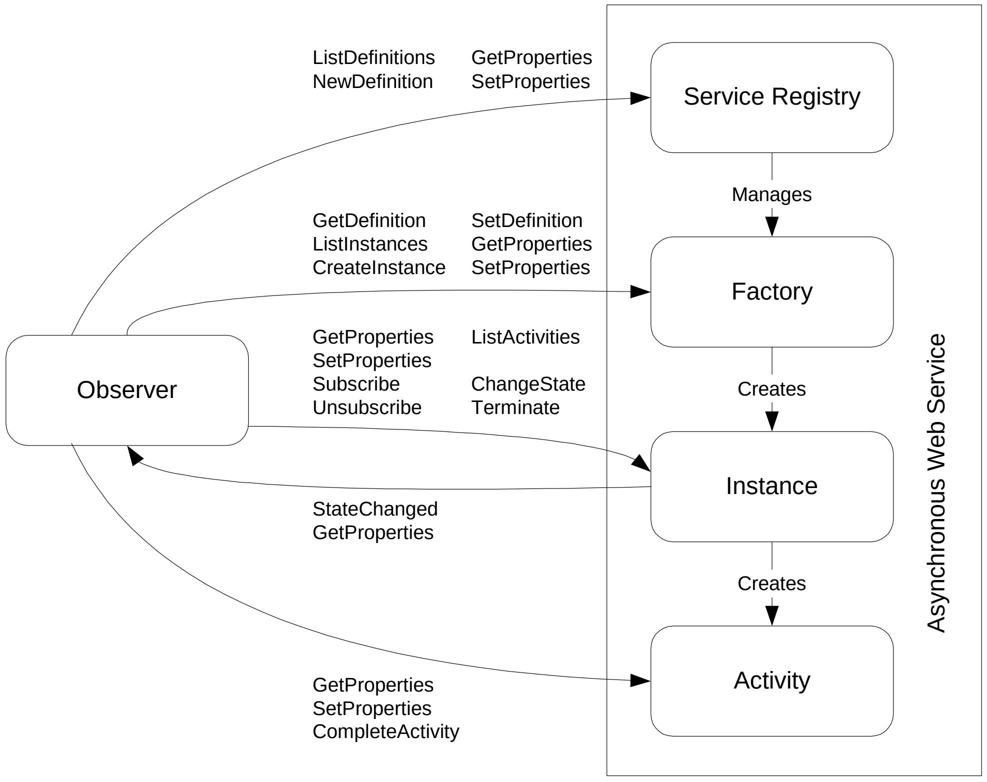

Figure 1 Resource types of a process engine web service and the methods they use

3.1.1 Observer

The Observer resource provides a means by which a service instance may communicate information about events occurring during its execution, such as its completion or termination. Third-party resources may have an interest in the status of a given service instance for various organizational reasons. The Observer group will provide this information by giving a service instance the resource identifier of the requestor, which will be the observer of that service instance.

3.1.2 Service Registry

This is a special purpose factory (or meta-factory) that can start new factories associated with new process definitions given to them. It can provide a list of factories, in the same way that a factory can provide a list of instances. This special purpose factory is needed to allow process definition tools to add new processes to the server.

3.1.3 Factory

The Factory resource represents a "way of doing some work". For a process engine, this is also known as a process definition resource. Once a process definition is created (that is a description of process to be performed), the process engine exposes this as a factory resource.

3.1.4 Instance

The Instance resource is the actual "performance of work"; it is the process instance. It embodies the context information that distinguishes one process instance from another. Some people call this a “case”. A process instance resource can be used only once: it is created, then it can be started, it can be paused, resumed, terminated. If things go normally, it will eventually complete.

3.1.5 Activity

The Activity resource is an extension of ASAP for Wf-XML. The process instance will at any point in time be waiting for what it considers to be an external action to be completed. The activity represents this wait-point within the process. The process may be waiting for a human to interact with it, or it may be waiting for the result of an automated step in the process. The activity presents information about what the process is waiting for, such as the assignee, and possibly detail about how long it has been waiting, and how long it is willing to wait. One way of invoking an external action is through the use of ASAP or Wf-XML. In this case, the activity is acting as an observer of that remote process. The activity can provide the URL of the remote process instance that it is waiting on.

3.2 URI

Like ASAP, each resource has an URI address, called the Key. A given implementation has complete control over how it wishes to create the URI that names the resource. It must stick to a single method of producing these URI Keys, so that the names can serve as a unique id for the resource involved. The receiving program should treat it as an opaque value and not assume anything about the format of the URI.

3.3 Parameters

Associated with a method will be any number of named values that form the parameters of the method. These will be represented in the body of the SOAP request message. Some parameters are optional and may be omitted. In general, extra parameters can be added for extensibility; implementations should ignore any parameters that it does not understand.

3.4 Returned Values

The result of request is a block of data encoded in XML in the body of the SOAP message. Each method defines the values that it will return. Some values are marked as optional and are not required, the rest are required to be part of the return set. An implementation of a method may return more than the required set of named value, including new values unique to that implementation, as long as ignoring those values has no effect on the usefulness of the required values. Clients of such implementation should be coded to properly handle responses that do not include those extended values.

4 Protocol

4.1 ASAP

Please refer to the ASAP document for details on how the basic ASAP messages are build on top of SOAP structures. ASAP requires that there be one of the following elements within the body that represents the information needed for a specific operation:

- GetPropertiesRq
- GetPropertiesRs
- SetPropertiesRq
- SetPropertiesRs
- CreateInstanceRq
- CreateInstanceRs
- ListInstancesRq
- ListInstancesRs
- ChangeStateRq
- ChangeStateRs
- StateChangedRq
- StateChangedRs
- NotifyRq
- NotifyRs
- SubscribeRq
- SubscribeRs
- UnsubscribeRq
- UnsubscribeRs
- env:Fault

These tags and their contents are described in detail in the sections on the specific operations. Wf-XML extends these for new operations, which require one of the following elements within the SOAP body:

- GetDefinitionRq

- GetDefinitionRs
- NewDefinitionRq
- NewDefinitionRs
- SetDefinitionRq
- SetDefinitionRs
- ListDefinitionsRq
- ListDefinitionsRs
- ListActivitiesRq
- ListActivitiesRs
- CompleteActivityRq
- CompleteActivityRs
- GetHistoryRq
- GetHistoryRs

4.2 Request & Response

SenderKey, ReceiverKey, ResponseRequired, RequestID: are defined by ASAP.

5 Resource Methods

This section covers the methods associated with each of the resources: Observer, ServiceRegistry, Factory, Instance, and Activity as per Figure 1.

5.1 Observer Resource

There are no differences in the Observer from the ASAP.

5.2 ServiceRegistry

This represents a registry of the business processes available with this service engine. Operations that can be performed include getting a list of the process definitions that already exist in the system, add new definitions, and get and set the properties of the service registry.

5.2.1 Service Registry Properties

- Key
- Name
- Description
- Version
- Status (Note, not the status as defined in Wf-XML 1.1, this is a usage status for the process definition itself, so a different name may need to be created)

Available Methods:

- ListDefinitions
- NewDefinition
- GetProperties
- SetProperties

5.2.2 ListDefinitions

Returns information on each of the currently available processes. The response returns the latest version of the process definitions URIs that are enabled for instance creation.

Process Definitions can be very large, so the function of this command is to return the URI to each process definition so that more information can be obtained by doing a GetDefinition request to the Factory resource. Also returned in the response are the Process Name, Description, and Current version for each process definition.

Note that if a specific process definition name is provided, then the ListDefinition command will return all versions of the specified process definition. Otherwise, only the current versions of all available processes are returned.

Note that only currently enabled process definitions are returned in this request. If disabled or un-available process definitions are desired (i.e. process definitions that might be in work or resources that they require are off line), then the status property should be set to disabled, and all disabled process definitions will be returned.

```
<?pseudo-xml?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Header>
    <as:Request...>
  </env:Header>
  <env:Body>
    <wfx:ListDefinitionsRq>
      <as:Name>Name</as:Name>?
      <wfx:Status>Status</wfx:Status>?
    </wfx:ListDefinitionsRq>
  </env:Body>
</env:Envelope>
```

Example 1: Service Registry resource ListDefinitions method request

```
<?pseudo-xml?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Header>
    <as:Response...>
  </env:Header>
  <env:Body>
    <wfx:ListDefinitionsRs>
      <wfx:DefinitionInfo>*
        <wfx:DefinitionKey> URI </as: DefinitionKey>
        <as:Name...>
        <as:Description...>
        <wfx:Version...>
        <wfx:Status...>
      </wfx:DefinitionInfo>
    </wfx:ListDefinitionsRs>
  </env:Body>
</env:Envelope>
```

Example 2: Service Registry resource ListDefinitions method response

```
<xsd:element name="ListDefinitionsRq">
</xsd:element>
<xsd:element name="ListDefinitionsRs">
  <xsd:complexType>
```

```
    <xsd:sequence>
      <xsd:element ref="DefinitionKey"/>
      <xsd:element ref="Name” minOccurs="0"/>
      <xsd:element ref="Description” minOccurs="0"/>
      <xsd:element ref="Version” minOccurs="0"/>
      <xsd:element ref="Status” minOccurs="0"/>
      </xsd:sequence>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Schema 1: Service Registry resource ListDefinitions method

5.2.3 NewDefinition

Creates a new process definition in a service registry. This new definitions then becomes available for creating instances. The process definition can be described in a variety of different process languages, and the request specifies the language. All process engines must support XPDL representation to satisfy interoperability requirement, but they may additionally support other process formats.

Note that the process names are unique for a specific factory, and if a NewDefinition method is called with an existing name, then the method shall return a pre-defined error.

```
<?pseudo-xml?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Header>
    <as:Request...>
  </env:Header>
  <env:Body>
    <wfx: NewDefinitionRq>
      <wfx:ProcessLanguage> XPDL </wfx:ProcessLanguage>
      <wfx:Definition>
        <xpdl...>
      </wfx:Definition>
    </wfx: NewDefinitionRq>
  </env:Body>
</env:Envelope>
```

Example 3: Factory resource NewDefinition request

```
<?pseudo-xml?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Header>
    <as:Response...>
  </env:Header>
  <env:Body>
    <wfx: NewDefinitionRs>
      <xpdl...>
    </wfx: NewDefinitionRs>
  </env:Body>
</env:Envelope>
```

Example 4: Factory resource NewDefinition response

```
<xsd:element name="NewDefinitionRq">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref=”ProcessLanguage”/>
      <xsd:element name="Definition">
        <<any>>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="NewDefinitionRs">
  <<any>>
</xsd:element name="NewDefinitionRs">
```

Schema 2: Factory resource NewDefinition method

5.3 Factory Resource

The Factory resource provides all the functions as specified by ASAP:

5.3.1 Factory Resource Properties

Key, PortType, Name, Subject, Description, ValidStates, ContextDataSchema, ResultDataSchema, and Expiration are defined by ASAP.

The Factory resource retains the following operations from ASAP:

- GetProperties
- CreateInstance
- ListInstances
- SetProperties

New operations added by Wf-XML:

- GetDefinition
- SetDefinition

5.3.2 GetDefinition

Since a factory represents a process definition, and because process definitions are potentially lengthy things that we would not want to mix in with the other properties of a factory, there is a special method to retrieve the process definition as an xml structure. It can be retrieved in a variety of different process languages, and the request specifies the language. All process engines must support XPDL representation to satisfy interoperability requirement, but they may additionally support other process formats.

```
<?pseudo-xml?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Header>
    <as:Request...>
  </env:Header>
  <env:Body>
    <wfx:GetDefinitionRq>
      <wfx:ProcessLanguage> XPDL </wfx:ProcessLanguage>
    </wfx:GetDefinitionRq>
  </env:Body>
</env:Envelope>
```

Example 5: Factory resource GetDefinition request

```
<?pseudo-xml?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Header>
    <as:Response...>
  </env:Header>
  <env:Body>
    <wfx:GetDefinitionRs>
      <xpdl...>
    </wfx:GetDefinitionRs>
  </env:Body>
</env:Envelope>
```

Example 6: Factory resource GetDefinition response

```
<xsd:element name="GetDefinitionRq">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref=”ProcessLanguage”/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="GetDefinitionRs">
  <<any>>
</xsd:element name="GetDefinitionRs">
```

Schema 3: Factory resource GetDefinition method

5.3.3 SetDefinition

Just as there is a special method to retrieve the process definition as an xml structure, there is also a special method to set (or update) a process definition. It can be set in a variety of different process languages, and the request specifies the language. All process engines must support XPDL representation to satisfy interoperability requirement, but they may additionally support other process formats.

The Factory should support versioning of the process definitions. As the SetDefinition method is called, the Factory will update the process definition, and if successful, it will increment the version of the process, which will be returned within the response.

```
<?pseudo-xml?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Header>
    <as:Request...>
  </env:Header>
  <env:Body>
    <wfx:SetDefinitionRq>
      <wfx:ProcessLanguage> XPDL </wfx:ProcessLanguage>
      <wfx:Definition>
        <xpdl...>
      </wfx:Definition>
    </wfx:SetDefinitionRq>
  </env:Body>
</env:Envelope>
```

Example 7: Factory resource SetDefinition request

```
<?pseudo-xml?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Header>
```

```
    <as:Response...>
  </env:Header>
  <env:Body>
    <wfx:SetDefinitionRs>
      <xpdl...>
    </wfx:SetDefinitionRs>
  </env:Body>
</env:Envelope>
```

Example 8: Factory resource SetDefinition response

```
<xsd:element name="SetDefinitionRq">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref=”ProcessLanguage”/>
      <xsd:element name="Definition">
        <<any>>
     </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="SetDefinitionRs">
  <<any>>
</xsd:element name="SetDefinitionRs">
```

Schema 4: Factory resource SetDefinition method

5.4 Instance Resource

The Instance resource provides all the functions as specified by ASAP:

- GetProperties
- SetProperties
- Terminate
- Subscribe
- Unsubscribe

For a process engine, the service instance is synonymous with a process instance, and some additional methods allow access to the process information. Specifically:

- ListActivities

5.4.1 Instance Resource Properties

Key, PortTypes, State, Name, Subject, Description, ValidStates, FactoryKey, Observers, ContextData, ResultData, Priority, LastModified, History retain the definitions from ASAP.

5.4.2 ListActivities

Returns information on each of the currently active activities. This does not return a list of all possible activities as defined by the process definition. The only activities returned here are the ones that are currently enabled, and waiting for a response. This corresponds to things that have been assigned to people and which have not been completed. It also corresponds to sub-processsub-processes that have been invoked using a nested sub-process pattern, and it is waiting for the sub-process to complete.

Activities are resources themselves, so the most important function of this command is to return the URI values for each activity. After that, more information can be obtained by doing a GetProperties request to the activity resource. But making an additional SOAP request to get common information such as the name and description of the activity is a needless overhead, so some additional information is included in this response.

The response makes use of the name and description elements that are defined by ASAP, but introduces new elements for ActivityURI and Assignee (which is a symbolic name of the assigned user or agent). There can be multiple assignees for a single activity.

```
<?pseudo-xml?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Header>
    <as:Request...>
  </env:Header>
  <env:Body>
    <wfx:ListActivitiesRq/>
  </env:Body>
</env:Envelope>
```

Example 9: Instance resource ListActivities method request

```
<?pseudo-xml?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Header>
    <as:Response...>
  </env:Header>
  <env:Body>
    <wfx:ListActivitiesRs>
      <wfx:ActivityInfo>*
        <wfx:ActivityKey> URI </as:ActivityKey>
        <as:Name...>
        <as:Description...>
        <wfx:Assignee...>*
      </wfx:ActivityInfo>
    </wfx:ListActivitiesRs>
  </env:Body>
</env:Envelope>
```

Example 10: Instance resource ListActivities method response

```
<xsd:element name="ListInstancesRq">
</xsd:element>
<xsd:element name="ListInstancesRs">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="ActivityKey"/>
      <xsd:element ref="Name” minOccurs="0"/>
      <xsd:element ref="Description” minOccurs="0"/>
      <xsd:element ref="Assignee” minOccurs="0" maxOccurs=”unbounded”/>
      </xsd:sequence>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Schema 5: Instance resource ListActivities method

5.5 Activity Resource

Represents a place whether the process is currently waiting for external input. There can be multiple of these if the process supports multiple branches executing in parallel.

Key – A URI that uniquely identifies this resource.

State – The current status of this activity. Will indicate whether the activity is waiting on a human or on a sub-process.

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

Description – A longer description of this activity resource. This property can be set using SetProperties.

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

InstanceKey – URI of the instance resource that this activity is a part of.

RemoteInstance – the URI of a remote service instance that this activity is waiting on.

StartedDate – the date that this activity was started, the point in time that this process instance started waiting for this external input.

DueDate – the date that this activity is expected to be completed by. “Expectation” is a loose term here. This will not be calculated using the average time that the activity take, but instead the maximum acceptable time, after which the activity is considered late, and special actions may be taken to resolve the issue.

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

5.5.1 GetProperties

Similar to the GetProperties method of the instance resource, this is a single method that returns all the values of all the properties of the activity resource.

```
<?pseudo-xml?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Header>
    <as:Request...>
  </env:Header>
  <env:Body>
    <as:GetPropertiesRq/>
  </env:Body>
</env:Envelope>
```

Example 11: Activity resource GetProperties method request

```
<?pseudo-xml?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Header>
    <as:Response..>
  </env:Header>
  <env:Body>
    <as:GetPropertiesRs>
      <-- properties -->
    </as:GetPropertiesRs>
  </env:Body>
</env:Envelope>
```

Example 12: Activity resource GetProperties method response

```
<xsd:element name="GetPropertiesRq"/>
<xsd:element name="GetPropertiesRs" type=”activityProperties”/>
<xsd:complexType name=”activityProperties”>
  <xsd:element ref=”Key”/>
  <xsd:element ref=”State”/>
  <xsd:element ref=”Name”/>
  <xsd:element ref=”Description”/>
  <xsd:element ref=”ValidStates”/>
  <xsd:element ref=”InstanceKey”/>
  <xsd:element ref=”RemoteInstance”/>
  <xsd:element ref=”StartedDate”/>
  <xsd:element ref=”DueDate”/>
</xsd:complexType>
```

Schema 6: Instance resource GetProperties method

5.5.2 SetProperties

This method is similar to the SetProperties method of the instance resource. The SetProperties method allows as parameters all of the settable properties of the Activity resource. This method can be used to set at least the displayable name, the description, or the state of an activity resource. This is an abstract interface, and the resources that implement this interface may have other properties that can be set in this manner. All of the parameters are optional, but to have any effect at least one of them must be present. This returns the complete info for the resource, just as the GetProperties method does, which will include any updated values.

`Data`: A collection elements that represent the context of this Instance. The elements are from the schema defined by this resource. The context is considered to be the union of the previous context and these values, which means that a partial set of values can be used to update just those elements in the partial set having no effect on elements not present in the call.

```
<?pseudo-xml?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Header>
    <as:Request...>
  </env:Header>
  <env:Body>
    <as:SetPropertiesRq>
      <as:Name...>
      <as:Description...>
      <as:State...>
      <as:Data>
        <-- extensible element -->
      </as:Data>
    </as:SetPropertiesRq>
  </env:Body>
</env:Envelope>
```

Example 13: Instance resource SetProperties method request

```
<?pseudo-xml?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Header>
    <as:Response...>
  </env:Header>
  <env:Body>
```

```
    <as:SetPropertiesRs...>
      Returns the same response as GetProperties
    </as:SetPropertiesRs>
  </env:Body>
</env:Envelope>
```

Example 14: Instance resource SetProperties method response

```

<xsd:element name="SetPropertiesRq">
  <xsd:complexType>
    <xsd:element ref=”Name”/>
    <xsd:element ref=”Description”/>
    <xsd:element ref=”State”/>
    <xsd:element name=”Data” type=”xsd:anyType”>
  </xsd:complexType>
</xsd:element>
<xsd:element name="SetPropertiesRs" type=”activityProperties”/>
```

Schema 7: Instance resource SetProperties method

5.5.3 CompleteActivity

This operation tells the activity that it is complete. This could be used by the user who has been assigned to the activity to tell the system that the work is complete. If this activity is waiting for a sub-process to complete, then this method should not be used, and instead the observer resource “Completed” operation should be used to indicate that the sub-process is complete, and the process engine will find and complete the activity that is waiting on the sub-process.

`Option`: The name of the path to be used after completing the activity. This parameter is optional.

```
<?pseudo-xml?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Header>
    <as:Request...>
  </env:Header>
  <env:Body>
    <as:CompleteActivityRq>
      <as:Option> option </as:Option>
    </as:CompleteActivityRq>
  </env:Body>
</env:Envelope>
```

Example 15: Activity resource CompleteActivity method request

```
<?pseudo-xml?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Header>
    <as:Response...>
  </env:Header>
  <env:Body>
    <as: CompleteActivityRs/>
  </env:Body>
</env:Envelope>
```

Example 16: Activity resource CompleteActivity method response

```
<xsd:element name=" CompleteActivityRq">
  <xsd:complexType>
```

```
    <xsd:sequence>
      <xsd:element ref=”Option”/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name=" CompleteActivityRs"/>
```

Schema 8: Activity resource CompleteActivity method

6 Data Encoding

6.1 Size Limits and Data Resources

The HTTP protocol places no limit on the size of the result information for a particular request, but in practice if the data gets to be too large, there requests will tend to get very slow. ASAP has a built in assumption that all core data is returned with most requests to the process. As a practical limit, process instance should be designed to have a maximum of 64K of data total, and an average process instance handling only 5 to 10K bytes.

If the process instance needs to handle more data than this, then the larger pieces of data should be stored as separate documents on a HTTP server and passed as URI reference. In this case the tag that would normally carry the information is empty, and has an “xlink” attribute specifying the URI to the data.

If data is passed to a process instance as a reference, there is an implicit commitment that the data resource will remain accessible until the process instance is completed. If the process instance returns data as a reference, that data resource must remain accessible for until the process is completed, and then for the amount of time specified by the CleanupInterval property on the process definition.

6.2 Data Types

Like ASAP, Wf-XML requires that this list of datatypes MUST be supported by a client are:

- xsd:boolean
- xsd:integer
- xsd:string
- xsd:dateTime
- xsd:anyURI

6.3 Extensibility

Actual implementations of these resources may extend the set of properties returned. This document defines the required minimum set, as well as an optional set. Every implementation MUST return the required properties. The implementation may optionally return additional properties. Use of extended properties must be carefully considered because this may limit the ability to interoperate with other systems. In general no system should be coded so as to require an extended attribute. Instead it should be able to function is the extended properties are missing. Future versions of this specification will cover the adoption of new properties to be considered part of the specification.

6.4 PortTypes Property

WfXML add two new port types to the three that ASAP had:

```
<xsd:simpleType name="PortTypes">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="Instance"/>
        <xsd:enumeration value="Factory"/>
        <xsd:enumeration value="Observer"/>
        <xsd:enumeration value="Activity"/>
        <xsd:enumeration value="ServiceRegistry"/>
    </xsd:restriction>
</xsd:simpleType>
```

Schema 1 PortTypes

6.5 Exceptions and Error Codes

All messages have the option of returning an exception. Exceptions are handled in the manner specified by SOAP 1.2. The header information should be the same, but in the body of the response, instead of having an ASAP element such as GetPropertiesRs or CreateInstanceRs, there will be the SOAP exception element env:Fault.

Multi server transactions: ASAP does not include any way for multiple servers to participate in the same transactions. It will be up to individual systems to determine what happen if a ASAP request fails; In some cases it should be ignored, in some cases it should cause that transaction to fail, and in some cases the operation should be queued to repeat until it succeeds.

```
<?pseudo-xml?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Header>
    <as:Response...>
  </env:Header>
  <env:Body>
    <env:Fault>
      <faultcode>env:Sender</faultcode>
      <faultstring>Header specific error</faultstring>
      <detail>
        <as:ErrorCode>104</as:ErrorCode>
        <as:ErrorMessage>Invalid key</as:ErrorMessage>
      </detail>
    </env:Fault>
  </env:Body>
</env:Envelope>
```

Example 1 Exception

These error codes are chosen to be specific with the error codes defined by the Workflow Management Coalition Wf-MXL 1.1 specification. [Fit this is with SOAP Fault structure, improve the names since Fault uses string names.]

Header-specific	**100 Series**
These exceptions deal with missing or invalid parameters in the header.	
ASAP_PARSING_ERROR	101
ASAP_ELEMENT_MISSING	102
ASAP_INVALID_VERSION	103
ASAP_INVALID_RESPONSE_REQUIRED_VALUE	104

ASAP_INVALID_KEY	105
ASAP_INVALID_OPERATION_SPECIFICATION	106
ASAP_INVALID_REQUEST_ID	107

Data	**200 Series**

These exceptions deal with incorrect context or result data

ASAP_INVALID_CONTEXT_DATA	201
ASAP_INVALID_RESULT_DATA	202
ASAP_INVALID_RESULT_DATA_SET	203

Authorization	**300 Series**

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

ASAP_NO_AUTHORIZATION	301

Operation	**400 Series**

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

ASAP_OPERATION_FAILED	401

Resource Access	**500 Series**

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

ASAP_NO_ACCESS_TO_RESOURCE	501
ASAP_INVALID_FACTORY	502
ASAP_MISSING_INSTANCE_KEY	503
ASAP_INVALID_INSTANCE_KEY	504

Operation-specific	**600 Series**

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

ASAP_INVALID_STATE_TRANSITION	601
ASAP_INVALID_OBSERVER_FOR_RESOURCE	602
ASAP_MISSING_NOTIFICATION_NAME	603
ASAP_INVALID_NOTIFICATION_NAME	604
ASAP_HISTORY_NOT_AVAILABLE	605

7 References

The following documents are relevant to this specification, and may be referenced in the text:

[1] Workflow Terminology (English), The Workflow Management Coalition, WFMC-TC-1011, version 2.0, June-1996, in PDF Format: [http://www.aiim.org/wfmc/standards/docs/glossary.pdf]. The terms used in this document are consistent with those found in this glossary.

[2] "Interoperability / Internet e-mail MIME Binding", The Workflow Management Coalition, WFMC-TC-1018, 1.1, July 1998 describes interoperation between workflow services using SMTP/MIME mail: [http://www.aiim.org/wfmc/standards/docs/if4-a.pdf.]

[3] "Reference Model", The Workflow Management Coalition, WFMC-TC-1003, 29-Nov-94, 1.1 describes the major components of a workflow system: http://www.aiim.org/wfmc/standards/docs/rmv1-16.pdf

[4] "HTTP - Hypertext Transfer Protocol" the latest information about HTTP can be found at [http://www.w3.org/Protocols/]

[5] "Hypertext Transfer Protocol -- HTTP/1.1", RFC-2068. the current proposed standard: http://www.w3.org/Protocols/rfc2068/rfc2068

[6] Crocker, D., "Standard for the Format of ARPA Internet Text Messages", STD 11 [http://globecom.net/ietf/std/std11.html], RFC 822 [http://globecom.net/ietf/rfc/rfc822.html] , UDEL, August 1982:

[7] "An Extension to HTTP: Digest Access Authentication" RFC-2069 January1997: http://www.w3.org/Protocols/rfc2069/rfc2069.txt

[8] "Simple Object Access Protocol (SOAP) Version 1.2 Part 1: Messaging Framework" W3C Working Draft 17 December 2001. http://www.w3.org/TR/soap12-part1/

[9] "Universal Description, Discovery and Integration (UDDI) Version 2.0 Programmer's API Specification, http://www.uddi.org/pubs/ProgrammersAPI-V2.00-Open-20010608.pdf

[10] "Web Services Description Language (WSDL) 1.1" W3C Note 15 March 2001. http://www.w3.org/TR/wsdl

[11] "Web Services Flow Language (WSFL)" May 2001, http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf

[12] Universally Unique Identifier (UUID)

[13]" XML Schema Part 0: Primer", W3C 2 May 2001, http://www.w3.org/TR/xmlschema-1/

[14] "XML Schema Part 1: Structures", W3C 2 May 2001, http://www.w3.org/TR/xmlschema-1/

[15] Eric Steven Raymond, "The Cathedral and the Bazaar", 24 Aug 2001. http://www.tuxedo.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/x188.html

8 Version History and Notes

2002.01.07 - Initial Draft

2002.02.03 - Another pass, include details, error code, XML Schema

2002.02.11 – Revision of terminology and reorganization of sections

2002.05.15 – Update from reviewer comments

2003.02.13 – Comments from San Diego meeting

2003.09.21 – Rewrite to accommodate changes from ASAP

2003.10.02 – Inclusion of commands for activities and for creating new factories (process definitions) This version distributed for working group review.

9 Acknowledgements

A number of people have participated in the development of this document and the related ideas that come largely from earlier work:

Jeffrey Ricker, Izar Inc.
Mike Marin, FileNET
Edwin Kodhabakchien, Collaxa Inc.
Dave Hollingsworth, ICL/Fujitsu
Marc-Thomas Schmidt, IBM
Greg Bolcer, Endeavors Technology, Inc
Dan Matheson, CoCreate
George Buzsaki and Surrendra Reddy, Oracle Corp.
Larry Masinter, Xerox PARC
Martin Adder
Mark Fisher, Thomson
David Jakopac and David Hurst, Lisle Technology Partners
Kevin Mitchell
Ian Prittie
Members of the Workflow Management Coalition

And many others...

10 Author Contact Information

Keith D Swenson, Fujitsu Software Corp, kswenson@us.fujitsu.com
Sameer Pradhan, Fujitsu Software Corp, sameerp@us.fujitsu.com

11 Open Issues