Advancing E-Business Standards Since 1993

Asynchronous Service Access Protocol (ASAP) Version 1.0

Working Draft 2A, November 24, 2004

Document identifier:
wd-asap-spec-01

Location:
http://www.oasis-open.org/committees/documents.php?wg_abbrev=asap

Editors:
~~Jeffrey Ricker~~John Fuller, Individual, ~~<jricker@izarinc.com~~<jfuller@wernervas.com>~~>~~
Mayilraj Krishnan, Cisco Systems, <mkrishna@cisco.com>
Keith Swenson, Fujitsu Software, <KSwenson@us.fujitsu.com>
~~John Fuller~~Jeffrey Ricker, Individual, <~~<jfuller@wernervas.com>~~jricker@izarinc.com>

Committee Members:
~~Moshe Silverstein, Individual,<moses@silversteingroup.com>~~
Justin Brunt, TIBCO
Sameer Pradhan, Fujitsu, <sameerp@us.fujitsu.com>
Jeff Cohen, Individual

Abstract:
A standard protocol is needed to integrate asynchronous services across the Internet and provide for their interaction. The integration and interactions consist of control and monitoring of the services. *Control* means 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* means checking on the current status of the service and getting an execution history of the service. The protocol should be lightweight and easy to implement, so that a variety of devices and situations can be covered.

The Asynchronous Service Access Protocol (ASAP) is a proposed way to solve this problem through use of Simple Object Access Protocol (SOAP), and by transferring structured information encoded in XML. A new set of SOAP methods are defined, as well as the information to be supplied and the information returned in XML that accomplish the control and monitoring of generic asynchronous services.

This document will: provide an executive overview; specify the goals of ASAP; explain how the resource (object) model works; explain how uniform resource names (URI) are used to invoke methods of those resources; explain how to encode data to be sent or received; and specify preliminary details of the interface methods and parameters.

Status:
This document is updated periodically on no particular schedule. Send comments to the editor. Committee members should send comments on this specification to the asap@lists.oasis-open.org list. Others should subscribe to and send comments to the asap-comment@lists.oasis-open.org list. To subscribe, send an email message to asap-comment-request@lists.oasis-open.org with the word "subscribe" as the body of the message.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the ASAP TC web page (http://www.oasis-open.org/committees/asap/).

Table of Contents

1 Introduction

1.1 Summary

This protocol offers a way to start an instance of an asynchronous web service, 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 will take a long time to run to completion. Short-lived services would be invoked synchronously with Simple Object Access Protocol (SOAP) **[SOAP]** and one would simply wait for completion. Because certain service instances last anywhere from a few minutes to a few months, they must be invoked asynchronously.

How does it work? You must start with the URI of a service definition called a *factory*. A SOAP request to this URI will cause this service definition to generate a service instance, and return the URI of this new service instance that is used for all the rest of the requests. The service instance can be provided with data (any XML data structure) by another SOAP request. The current state of the service instance can be retrieved with another SOAP request. The service instance can be paused or resumed with another SOAP request. There is also a pair of requests that may be used to give input data to the service instance, and to ask for the current value of the output data.

What happens when it is done? The service instance runs asynchronously and takes whatever time it needs to complete. The originating program can, if it chooses, keep polling the state of the service instance in order to find out when it is complete. This will consume resources unnecessarily both on the originating side as well as the performing side. Instead, the originator may provide the service instance with an EPR for an observer. When the service instance is completed it will send a SOAP request to the EPR for each observer. This allows the originator to be put to sleep, freeing up operating system as well as network resources while waiting for the service instance to complete.

1.2 Not-so-technical executive summary

What does this mean in English? Most existing Internet protocols like HTTP are based on an unwritten assumption of instant gratification. If a client asks for any resource that takes longer than about a minute to generate, then the request times out, that is, it fails. We call anything on the Internet like HTML pages and GIF images a *resource*. Most resources such as web pages are static or require a very simple database query to create, so they easily meet the instant gratification requirement.

As we have applied Internet technology to more and more scenarios, this assumption of instant gratification has become more strained. A good example is wireless Internet. With wireless, the resource may take more than a minute to generate simply because of a poor connection.

A more telling example is electronic commerce. In commerce, it may not be a simple database query that generates a document but rather an entire corporate business process with a human approval involved. Very few corporate business processes especially those requiring management approval, take less than a minute to complete.

What needed in real world scenarios is ability to ask for a resource and for that resource to be able to respond, “The information isn’t ready yet. Where would you like me to send it when I’m done?” That is what ASAP considers as *start an instance of a generic asynchronous service and be notified when it is complete*. Someone asking for the resource should be able to pester, just like in the real world, with questions like, “Are you done yet? Where is that document I asked for?” That is what ASAP considers as *monitor*. Finally the requestor asking resource change mind in mid process, just like in the real world with statements like, “Change that to five widgets, not six.” That is what ASAP considers as *control*.

With such a protocol, business should be able to integrate not just applications but business processes, which is what electronic commerce is really all about. With such a protocol, business should also be able to integrate within and between enterprises much faster because of the ability to have manual processes look and act to everything else on the Internet as if it were actually automated.

Here is an example. An ASAP message is sent to a server requesting inventory levels of a certain part number. The server responds to the requestor “The information isn’t ready yet. Where would you like me to send it when I’m done?” The server then sends a message to Steve’s two-way pager in the warehouse asking him to type in the inventory level of the certain part number. After a coffee break, Steve duly types in the number. The server creates the proper message and responds to the requestor. To the outside world, an electronic message was sent and an electronic message was received. The result is automated inventory level tracking. Nobody need to know that Steve walked down the aisle and counted by hand.

1.3 Problem statement

Not all services are instantaneous. A standard protocol is needed to integrate asynchronous services (processes or work providers) across the Internet and provide for their interaction. The integration and interactions consist of control and monitoring of the service. *Control* means 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* means checking on the current status and getting execution history of the service.

The protocol should be lightweight and easy to implement, so that a variety of devices and situations can be covered.

1.4 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 protocol should not reinvent anything unnecessarily. If a suitable standard exists, it should be used rather than re-implement in a different way.
- The protocol should be consistent with XML Protocol and SOAP.
- This protocol should be easy to incorporate into other SOAP-based protocols that require asynchronous communication
- The protocol should be the minimal necessary to support a generic asynchronous service. This means being able to start, monitor, exchange data with, and control a generic asynchronous service on a different system.
- 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 requirement, 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, ASAP 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 generating, understanding and parsing should be favored over compactness.

Regarding human readability, the messages should be self-describing for the programmer, but they are not intended for direct display for the novice end user. This specification attempts to adhere to Eric S. Raymond’s ninth principle: “Smart data structures and dumb code works a lot better than the other way around,” or, paraphrased from Frederick P. Brooks, “Show me your [code] and conceal your [data structures], and I shall continue to be mystified. Show me your [data structures], and I won't usually need your [code]; it'll be obvious." **[RAYMOND]**

1.5 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:

- The goals of ASAP do not include a way to set up or to program the generic services in any way. Especially for the case where the service is a workflow service, ASAP does not provide a way to retrieve or submit process definitions. The service can be considered to be a "black box" which has been pre-configured to do a particular process. ASAP does not provide a way to discover what it is that the service is really doing, only that it does it (given some data to start with) and some time later completes (providing some result data back).
- ASAP will not include the ability to perform maintenance of the asynchronous web service such as installation or configuration.
- ASAP will not support statistics or diagnostics of collections of asynchronous web service. ASAP is designed for the control and monitoring of individual asynchronous web services.
- ASAP does not specify security. Rather, it relies on transport or session layer security. ASAP can adopt SOAP –specific security protocols once they are finalized.
- ASAP does not address service quality issues of transport such as guaranteed delivery, redundant delivery and non-repudiation. Rather, ASAP relies on the session layer, the transport layer, or other SOAP protocols to address these issues.

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.

1.6 Terminology

The key words *must*, *must not*, *required*, *shall*, *shall not*, *should*, *should not*, *recommended*, *may*, and *optional* in this document are to be interpreted as described in **[RFC2119]**.

Other specific terms are as follows.

Web Service: W3C Web Service Architecture Group **[W3C Arch]** defined Web Service as “A software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards”

Service: synonymous with web service.

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: 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.

`Factory`: 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.

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

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

`Results data`: The XML data created by the successful completion of the service.

1.7 Notation conventions

The following namespace prefixes are used throughout this document:

Prefix	Namespace URI	Definition
as	http://www.oasis-open.org/asap/1.0/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
wsa	http://schemas.xmlsoap.org/ws/2004/08/addressing	W3C WS Addressing 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.

1.8 Related documents

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

2 Resource model

2.1 Overview

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

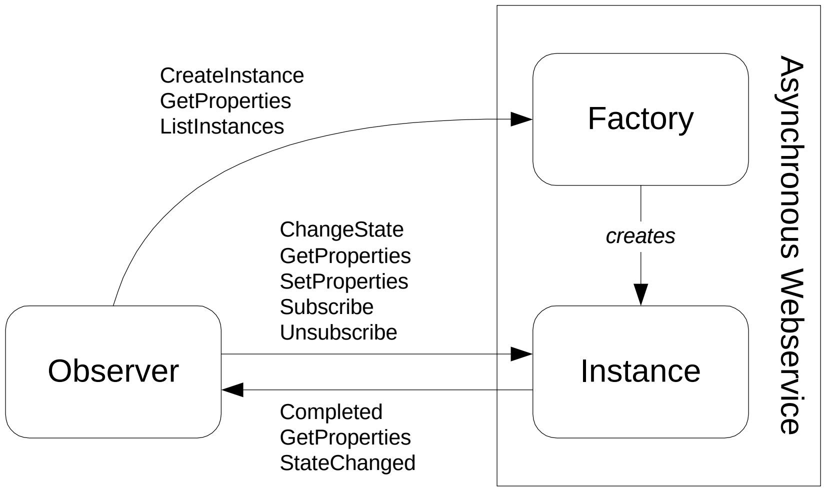

Figure 1 Resource types of an asynchronous web service and the methods they use

Typical use of this protocol would be as follows:

1. A Factory service receives a `CreateInstanceRq` message that contains `ContextData` and an EPR of an Observer
2. The Factory service creates an Instance service and subscribes the Observer to the Instance
3. The Factory responds to `CreateInstanceRq` message with a `CreateInstanceRs` message that contains an EPR for the Instance
4. The Instance service eventually completes its task and sends a `CompletedRq` message that contains the `ResultsData` to the Observer

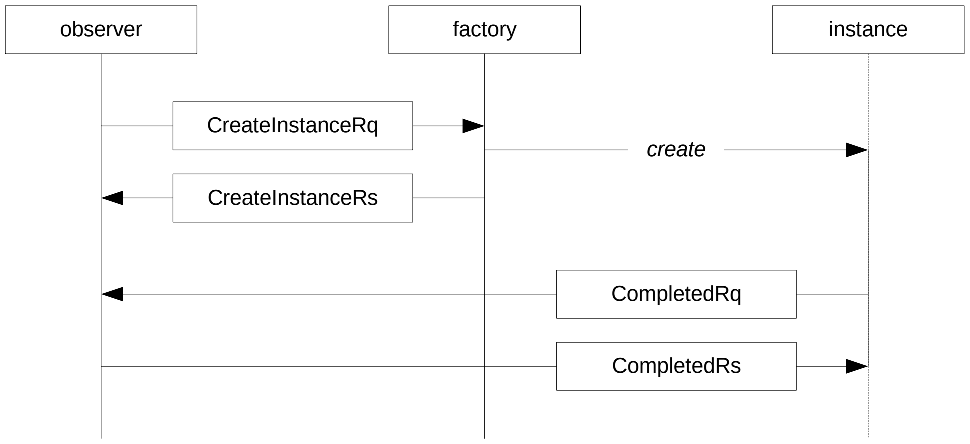

Figure 2 Typical use of ASAP

2.2 Instance

The Instance resource is the actual "performance of work". It embodies the context information that distinguishes one performance of one asynchronous service from another. Every time the asynchronous service is to be invoked, a new instance is created and given its own resource identifier. A service instance 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.

When a service is to be enacted, a requestor will reference a service factory's resource identifier and create an instance of that service. Since a new instance will be created for each enactment, the service factory may be invoked (or instantiated) any number of times simultaneously. However, each service instance will be unique and exist only once. Once created, a service instance may be started and will eventually be completed or terminated.

2.3 Factory

The Factory resource represents a "way of doing some work". It is the most fundamental resource 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 factory will provide a resource identifier. When a service is to be enacted, this resource identifier will be used to reference the desired asynchronous service to be executed. A service might be a set of tasks carried out by a group of individuals, or it might be set of machine instructions that make up an executable program, or it might be any combination of these. The important point to remember about a service factory is that while it embodies the knowledge of how work is performed, it does not actually do the work. The service instance does the work.

2.4 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 organization and business reasons. Observers subscribe to a service instance by providing an EPR. A service instance notifies all observers by sending SOAP messages to the observer EPRs.

2.5 URI

Each resource has an URI address, called the *key*. A given implementation has complete control over how it wishes to create the URI that identifies the resource. It should stick to a single method of producing these URI Keys, so that the names can serve as a unique identifier for the resource involved. The receiving program should treat it as an opaque value and not assume anything about the format of the URI. All instance keys must be unique.

URIs for resources are exchanged in WS-Addressing endpoint references, so that any additional information required for addressing the URI can be provided dynamically to protocol participants.

2.6 ContextData and ResultData

The heart of an asynchronous service is the `ContextData` and the `ResultData`. The `ContextData` and the `ResultData` are the unique part of a particular service; everything else is boilerplate. The `ContextData` is the query or the initial request to the service. The `ContextData` dictates, determines or implies what the service instance should create. The `ResultData` is what the service eventually creates for the observers.

3 Protocol

3.1 SOAP

Simple Object Access Protocol (SOAP) [8] is a protocol that defines a simple way for two programs to exchange information. The protocol consists of a client program that initiates a request to a server program. Any given program may be capable of being both a client and a server. Our use of these terms refers only to the role being performed by the program for a particular connection, rather than to the program's capabilities in general. The request involves the sending of a request message from the client to the server. The response involves the sending of a response message from the server back to the client. Both the request and response messages conform to the SOAP message format.

The root tag of an ASAP message is a SOAP envelope as defined by the SOAP standard.

The message must contain a SOAP header as per the SOAP standard for addressing and routing the message and must employ WS-Addressing. An ASAP message from a client must contain the Request element and a message from a server must contain a Response element.

3.2 Request header

A request header uses WS-Addressing message information header blocks.

~~/wsa:MessageID Allows for message correlation with wsa:RelatesTo in responses.~~

~~/wsa:ReplyTo If a response is required, provide a wsa:ReplyTo element.~~

~~/wsa:From The request MAY specify the endpoint reference for the resource that originated the request. This may be redundant with similar specifier in the transport layer.~~

~~/wsa:FaultTo If an endpoint should be sent faults, provide a wsa:FaultTo element.~~

/wsa:To The request MUST specify the key of the resource that the request is being made to. This may be redundant with similar specifier in the transport layer.

/wsa:ReplyTo If a response is required, provide a wsa:ReplyTo element. In most cases in the ASAP protocol, a response is required. If a client does not know , or does not have, an endpoint reference, then there is a special WS-Addressing defined “anonymous” address which can be used in some transports where there response can be delivered without knowing the endpoint address.

/wsa:MessageID Allows for message correlation with wsa:RelatesTo in responses.

/wsa:Action The request MUST specify a URI defining the semantics of the message. For ASAP exchange this must contain the IRI of the namespace combined with the QName of the tag inside the body tag. The WSDL files produced for ASAP exchanges must specify this value as the action value of that particular operation.

/wsa:From The use of this optional tag is discouraged in ASAP interchanges. The meaning is not clear, and it is not in any case useful since there is no guarantee that you can send a message to this endpoint, nor is there any definition of what endpoint can be used for.

/wsa:FaultTo For ASAP interchanges this MUST not be used. ASAP protocol is defined such that every request results in a response: either the result or a fault.

```
<?pseudo-xml?>
<env:Envelope xmlns:env="http://www.w3.org/2001/12/soap-envelope">
  <env:Header>
    <wsa:To> The URI of the receiver </wsa:To>
    <wsa:ReplyTo>?EPR for replies</wsa:ReplyTo>
    <wsa:MessageID>?xsd:anyURI</wsa:MessageID>
~~    <wsa:ReplyTo>?Optional EPR for replies</wsa:ReplyTo>~~
~~    <wsa:From>? The EPR of the sender </wsa:From>~~
~~    <wsa:FaultTo>?Optional EPR for faults</wsa:FaultTo>~~
~~    <wsa:To> The URI of the receiver </wsa:To>~~
    <wsa:Action>URI identifying the semantics of the message</wsa:Action>
  </env:Header>
  <env:Body>
    ...
  </env:Body>
</env:Envelope>
```

Example 1 Request header

3.3 Response header

WS-Addressing message information header blocks are used in responses.

`/wsa:From` The response MUST specify the endpoint reference of the resource that originated the response. This may be redundant with similar specifier in the transport layer. (Question: should this be ReplyTo instead?)

`/wsa:To` The response ~~MAY~~MUST specify the key of the resource that the response is being made to. This may be redundant with similar specifier in the transport layer. The original request may have come from an anonymous source, and the appropriate anonymous value should be used.

~~Note that the `wsa:To` is mandatory in a request and the `wsa:From` is mandatory in a response. The purpose is to enforce keys upon ASAP resources without placing an unnecessary burden on resources that are merely employing ASAP resources. For instance, a Java program that instantiates an asynchronous service instance may not know its own URL.~~

`/wsa:Action` The response MUST specify a URI defining the semantics of the message.

`/wsa:RelatesTo` If the original request had a `MessageID`, then the response must carry one with that value in it. The requester can use this ID to correlate the response with the original request.

```
<?pseudo-xml?>
<env:Envelope xmlns:env="http://www.w3.org/2001/12/soap-envelope"
  xmlns:wsa= "http://schemas.xmlsoap.org/ws/2004/08/addressing"
>
  <env:Header>
    <wsa:MessageID>?xsd:anyURI<wsa:MessageID>
    <wsa:From>Endpoint reference of the sender</wsa:From>
    <wsa:To>? The URI of the receiver </wsa:To>
    <wsa:Action>Reply action</wsa:Action>
    <wsa:RelatesTo>?Message Id of the request for correlation<wsa:RelatesTo>
  </env:Header>
  <env:Body>
    ...
  </env:Body>
</env:Envelope>
```

Example 2 Response header

3.4 Compatibility with Earlier Implementations

Some implementations of ASAP were completed and made publicly available before the WS-Addressing specification was formulated. An implementation of ASAP MAY implement backward compatibility mechanism in order to interoperate in a consistent way with early implementations.

WS-Addressing introduced 4 new tags placed in the header of SOAP message which replaced earlier ASAP message header tags. These earlier tags will be referred to as "deprecated" tags and should not be used except in the context of backward compatibility. Because they are deprecated they are not included in the official schema structures, and will not be mentioned in other parts of this document.

When formulating messages to send, the deprecated tags should be included as redundant carriers of the information. Older implementations that do not know about the new tags will use the deprecated tags.

When receiving messages, the new tags should take precedence. If a new tag exists, its value should be used, and the presence of the corresponding deprecated tag should be ignored. But if the new tag is not presence, and the deprecated tag is present, then the value from the deprecated tag should be used as if it came from the new tag.

New Tag	Deprecated Tag	Comment
<wsa:To>	<as:RecieverKey>	Both are IRI addresses.
<wsa:ReplyTo>	<as:SenderKey>	The ReplyTo is an endpoint reference, while the SenderKey holds only the address part of the EPR.
<wsa:MessageId>	<as:RequestId>	Both of these denote unique ids for a particular exchange. RequestId was used in both request and reply, while MessageId use in the request, RelatesTo is used in the response.
<wsa:RelatesTo>	<as:RequestId>	

~~3.3~~3.5 Body

ASAP requires that there be one of the following elements within the body which represents the information needed for a specific operation:

	Factory	Instance	Observer
GetPropertiesRq	X	X	X
GetPropertiesRs	X	X	X
SetPropertiesRq		X	
SetPropertiesRs		X	
CompletedRq			X
CompletedRs			X
CreateInstanceRq	X		
CreateInstanceRs	X		
ListInstancesRq	X		
ListInstancesRs	X		
ChangeStateRq		X	
ChangeStateRs		X	
StateChangedRq			X
StateChangedRs			X
SubscribeRq		X	
SubscribeRs		X	
UnsubscribeRq		X	
UnsubscribeRs		X	
env:Fault	X	X	X

Table 3 The ASAP message body elements

These elements and their contents are described in detail in the sections on the specific operations.

4 Instance resource

All resources that represent the execution of a long-term asynchronous service must implement the Service Instance resource. The purpose of this resource type is to allow the work to proceed asynchronously from the caller. The Instance represents a unit of work, and a new instance of the Instance resource must be created for every time the work is to be performed.

The performing of the work may take anywhere from minutes to months, so there are a number of operations that may be called while the work is going on. While the work is proceeding, ASAP requests can be used to check on the state of the work. If the input data has changed in the meantime, new input values may be supplied to the Instance, though how it responds to new data is determined by details about the actual task it is performing. Early values of the result data may be requested, which may or may not be complete depending upon the details of the task being performed. The results are not final until the unit of work is completed. When the state of the Instance changes, it can send events to the Observer informing it of these changes. The only event that is absolutely required is the "completed" or "terminated" events that tell the requesting resource that the results are final and the Instance resource may be disappearing.

While a business process will implement Instance, it is important to note that there are also many other types of resources that will implement the Instance resource; it will also be implemented on any discrete task that needs to be performed asynchronously. Thus a wrapper for a legacy CICS transaction would implement the Instance resource so that that legacy application could be called and controlled by any program that speaks ASAP. A driver for an actual physical device, such as a numerical milling machine, would implement the Instance resource if that device were to be controlled by ASAP. Any program to be triggered by a process flow system that takes a long time to perform should implement the Instance resource, for example a program that automatically backs up all the hard drives for a computer. Since these resources represent discrete units of work (which have no subunits represented within the system) these resources will not need to have any activities.

4.1 Instance resource properties

Key: A URI that uniquely identifies this resource.

State: The current status of this resource. Please see more details on the status property later in section on Section 7.3 “State Type”. This property is not directly settable, but can be changed through the ChangeState command.

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

Subject: A short description of this process instance. This property can be set using SetProperties.

Description: A longer description of this process instance resource. This property can be set using SetProperties.

FactoryKey: EPR for the factory resource from which this instance was created.

Observers: A collection of endpoint references of registered observers of this process instance, if any exist.

ContextData: Context-specific data that represents the values that the service execution is expected to operate on.

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

History**:** Describes the sequence of events and time stamp of the process instance.

UserInterface: The address of a web based user interface for the process, should one exist. The remote asynchronous service may have a way to display this service instance to the user(s) who are involved in the local service. The URI in this EPR can be used in the local service to make a link to the remote service that can be navigated by a user to see directly the state of the remote process. An example of this might be a order process, which in turn schedules a shipment from a courier, and the courier provides a way to track the shipment, and so this EPR would allow the user to the purchase process to access the tracking display directly. This value is optional, and if not present then assumed that the remote service instance has no UI acceptable for the local users.

```
<?pseudo-xml?>
...
<as:Key> URI </as:Key>
<as:State>open.notrunning</as:State>
<as:Name> string </as:Name>
<as:Subject> string </as:Subject>
<as:Description> string </as:Description>
<as:FactoryKey> EPR </as:FactoryKey>
<as:Observers>
  <as:ObserverKey>* EPR </as:ObserverKey>
</as:Observers>
<as:ContextData>
  <-- extensible element -->
</as:ContextData>
<as:ResultData>
  <-- extensible element -->
</as:ResultData>
<as:History xlink:href=”url”/>
...
```

Example 3 Instance resource properties

```
<xsd:group name=”instancePropertiesGroup”>
  <xsd:sequence>
    <xsd:element name="Key" type="xsd:anyURI"/>
    <xsd:element name="State" type="stateType"/>
    <xsd:element name="Name" type="xsd:string"/>
    <xsd:element name="Subject" type="xsd:string"/>
    <xsd:element name="Description" type="xsd:string"/>
    <xsd:element name="FactoryKey" type="wsa:EndpointReferenceType"/>
    <xsd:element name="Observers">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name=”ObserverKey” type="wsa:EndpointReferenceType" maxOccurs="unbounded"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="ContextData">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:any namespace=”##any” processContents=”lax” minOccurs=”0” maxOccurs=”unbounded”/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="ResultData">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:any namespace=”##any” processContents=”lax” minOccurs=”0” maxOccurs=”unbounded”/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="History" type="historyType"/>
    <xsd:element name="UserInterface" type="wsa:EndpointReference" minOccurs="0"/>
  </xsd:sequence>
</xsd:group>

<xsd:complexType name="stateType">
  <xsd:simpleContent>
    <xsd:extension base="xsd:string">
      <xsd:attribute name="namespace" type="xsd:anyURI"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>

<xsd:element name=”Event”>
<xsd:complexType>
 <xsd:sequence>
  <xsd:element name="Time" type="xsd:dateTime"/>
  <xsd:element name="EventType">
    <xsd:simpleType>
    <xsd:restriction base="xsd:string">
    <xsd:enumeration value="InstanceCreated"/>
    <xsd:enumeration value="PropertiesSet"/>
    <xsd:enumeration value="StateChanged"/>
    <xsd:enumeration value="Subscribed"/>
    <xsd:enumeration value="Unsubscribed"/>
    <xsd:enumeration value="Error"/>
    </xsd:restriction>
    </xsd:simpleType>
  </xsd:element>
  <xsd:element name="SourceKey" type="wsa:EndpointReferenceType"/>
  <xsd:element name="Details" type="xsd:anyType"/>
  <xsd:element name="OldState" type="as:stateType"/>
  <xsd:element name="NewState" type="as:stateType"/>
 </xsd:sequence>
</xsd:complexType>
</xsd:element>
<xsd:complexType name=”historyType”>
 <xsd:sequence>
       <xsd:element ref=”Event” maxOccurs=”unbounded”/>
 </xsd:sequence>
</xsd:complexType>
```

Schema 1 Instance resource properties

4.2 GetProperties

This is a single method that returns all the values of all the properties of the resource.

`GetPropertiesRq`: This is the main element present in the SOAP Body element.

```
<?pseudo-xml?>
<env:Envelope xmlns:env=http://www.w3.org/2001/12/soap-envelope”
  xmlns:wsa= “http://schemas.xmlsoap.org/ws/2004/08/addressing”>
  <env:Header>
    <wsa:...>
  </env:Header>
  <env:Body>
    <as:GetPropertiesRq/>
  </env:Body>
</env:Envelope>
```

Example 4 Instance resource GetProperties method request

```
<?pseudo-xml?>
<env:Envelope xmlns:env=http://www.w3.org/2001/12/soap-envelope
```

```
 xmlns:wsa= “http://schemas.xmlsoap.org/ws/2004/08/addressing”>
 <env:Header>
  <wsa:..>
 </env:Header>
 <env:Body>
  <as:GetPropertiesRs>
   <-- properties -->
  </as:GetPropertiesRs>
 </env:Body>
</env:Envelope>
```

Example 5 Instance resource GetProperties method response

```
<xsd:element name="GetPropertiesRq"/>
<xsd:element name="GetPropertiesRs" type=”instancePropertiesGroup”/>
```

Schema 2 Instance resource GetProperties method

4.3 SetProperties

All resources implement SetProperties and allow as parameters all of the settable properties. This method can be used to set at least the displayable name, the description, or the priority of a process flow 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 of 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://www.w3.org/2001/12/soap-envelope
 xmlns:wsa= “http://schemas.xmlsoap.org/ws/2004/08/addressing”>
 <env:Header>
  <wsa:...>
 </env:Header>
 <env:Body>
  <as:SetPropertiesRq>
   <as:Subject...>?
   <as:Description...>?
   <as:Priority...>?
   <as:Data>
    <-- extensible element -->
   </as:Data>
  </as:SetPropertiesRq>
 </env:Body>
</env:Envelope>

```

Example 6 Instance resource SetProperties method request

```
<?pseudo-xml?>
<env:Envelope xmlns:env=http://www.w3.org/2001/12/soap-envelope
 xmlns:wsa= “http://schemas.xmlsoap.org/ws/2004/08/addressing”>
 <env:Header>
  <wsa:...>
 </env:Header>
 <env:Body>
  <as:SetPropertiesRs...>
   Returns the same response as GetProperties
  </as:SetPropertiesRs>
 </env:Body>
```

```
</env:Envelope>
```

Example 7 Instance resource SetProperties method response

```
<xsd:element name="SetPropertiesRq">
<xsd:complexType>
  <xsd:sequence>
    <xsd:element name=”Subject” type=”xsd:string”/>
    <xsd:element name=”Description” type=”xsd:string”/>
    <xsd:element name=”Priority” type=”xsd:string”/>
    <xsd:element name=”Data”>
       <xsd:complexType>
       <xsd:sequence>
       <xsd:any namespace=”##any” processContents=”lax” minOccurs=”0” maxOccurs=”unbounded”/>
       </xsd:sequence>
       </xsd:complexType
       </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="SetPropertiesRs" type=”instancePropertiesGroup”/>
```

Schema 3 Instance resource SetProperties method

4.4 Subscribe

To allow scalability, Instances will notify Observers when important events occur. Observers must register their endpoint references with the Instance in order to be notified.

The subscribe method is a way for other implementations of the Observer Operation Group to register themselves to receive posts about changes in process instance state. Not all Instance resources will support this; those that do not support, will return an exception value that explains the error.

`ObserverKey`: Endpoint reference to a resource that both implements the Observer Operation Group and will receive the events

```
<?pseudo-xml?>
<env:Envelope xmlns:env=http://www.w3.org/2001/12/soap-envelope
  xmlns:wsa= “http://schemas.xmlsoap.org/ws/2004/08/addressing”>
<env:Header>
  <wsa:...>
  </env:Header>
<env:Body>
  <as:SubscribeRq>
    <as:ObserverKey> EPR </as:ObserverKey>
    </as:SubscribeRq>
  </env:Body>
</env:Envelope>
```

Example 6 Instance resource Subscribe method request

```
<?pseudo-xml?>
<env:Envelope xmlns:env=http://www.w3.org/2001/12/soap-envelope
  xmlns:wsa= “http://schemas.xmlsoap.org/ws/2004/08/addressing”>
<env:Header>
  <wsa:...>
  </env:Header>
<env:Body>
  <as:SubscribeRs/>
  </env:Body>
</env:Envelope>
```

Example 7 Instance resource Subscribe method response

```
<xsd:element name="SubscribeRq">
```

```
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="ObserverKey" type=”wsa:EndpointReference”/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="SubscribeRs"/>
```

Schema 4 Instance resource Subscribe method

4.5 Unsubscribe

This is the opposite of the subscribe method. Resource removed from being observers will no longer get events from this resource. The URI of the resource to be removed from the observers list must match exactly to an URI already in the list. If it does match, then that URI will be removed. If it does not match exactly, then there will be no change to the service instance.

```
<?pseudo-xml?>
<env:Envelope xmlns:env=http://www.w3.org/2001/12/soap-envelope
  xmlns:wsa= “http://schemas.xmlsoap.org/ws/2004/08/addressing”>
  <env:Header>
    <as:Request...>
  <wsa:..>
  </env:Header>
  <env:Body>
    <as:UnsubscribeRq>
      <as:ObserverKey> EPR </as:ObserverKey>
    </as:UnsubscribeRq>
  </env:Body>
</env:Envelope>
```

Example 8 Instance resource Unsubscribe method request

```
<?pseudo-xml?>
<env:Envelope xmlns:env=http://www.w3.org/2001/12/soap-envelope
  xmlns:wsa= “http://schemas.xmlsoap.org/ws/2004/08/addressing”>
  <env:Header>
    <wsa:...>
  </env:Header>
  <env:Body>
    <as:UnsubscribeRs/>
  </env:Body>
</env:Envelope>
```

Example 9 Instance resource Unsubscribe method response

```
<xsd:element name="UnsubscribeRq">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="ObserverKey" type=”wsa:EndpointReference”/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="UnsubscribeRs"/>
```

Schema 5 Instance resource Unsubscribe method

4.6 ChangeState

This method requests a change of state in the service. The instance service should send a StateChanged message to all observers.

```
<?pseudo-xml?>
<env:Envelope xmlns:env=http://www.w3.org/2001/12/soap-envelope
  xmlns:wsa= “http://schemas.xmlsoap.org/ws/2004/08/addressing”>
  <env:Header>
```

```
    <wsa:...>
  </env:Header>
  <env:Body>
    <as:ChangeStateRq>
      <as:State>the state requested</as:State>
    </as:ChangeStateRq>
  </env:Body>
</env:Envelope>
```

Example 10 Instance resource ChangeState method request

```
<?pseudo-xml?>
<env:Envelope xmlns:env=http://www.w3.org/2001/12/soap-envelope
  xmlns:wsa= “http://schemas.xmlsoap.org/ws/2004/08/addressing”>
  <env:Header>
    <wsa:…>
  </env:Header>
  <env:Body>
    <as:ChangeStateRs>
      <as:State>the state</as:State>
    </as:ChangeStateRs>
  </env:Body>
</env:Envelope>
```

Example 11 Instance resource ChangeState method response

```
<xsd:element name="ChangeStateRq">
 <xsd:complexType>
    <xsd:sequence>
      <xsd:element name=”State" type=”as:stateType”/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="ChangeStateRs">
       <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="State" type=”as:stateType”/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Schema 6 Instance resource ChangeState method

5 Factory resource

5.1 Factory resource properties

`Key`: A URI that uniquely identifies this resource. All resources must have a Key property.

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

`Subject`: A short description of this service. Note that the factory and the instance both have a subject. The subject of the factory should be general. The subject of an instance should be specific.

`Description`: A longer description of what the AWS will perform. . Note that the factory and the instance both have a subject. The subject of the factory should be general. The subject of an instance should be specific.

`ContextDataSchema`: An XML Schema representation of the context data that should be supplied when starting an instance of this process. This element contains ContextDataType and should not contain any other global element.

`ResultDataSchema`: an XML Schema representation of the data that will generate and return as a result of the execution of this process. This element contains ResultDataType and should not contain any other global element.

`Expiration`**:** The minimum amount of time the service instance will remain accessible as a resource after it has been completed for any reason. The requester must plan to pick up all data within this time span of service completion. Data might remain longer than this, but there is no guarantee. The value is expressed as an XML Schema duration data type. For instance, 120 days is expresses as “P120D”.

`ServiceGroupReference`: Optional EPR for a service group relating created instances.

```
<?pseudo-xml?>
...
<as:Key> URI </as:Key>
<as:Name> xsd:string </as:Name>
<as:Subject> xsd:string </as:Subject>
<as:Description> xsd:string </as:Description>
<as:ContextDataSchema>
   <xsd:schema xmlns:xsd=”http://www.w3.org/2001/XMLSchema”>
   <-- factory specific items of the context data schema -->
  </xsd:schema>
<as:ResultDataSchema>
  <xsd:schema xmlns:xsd=”http://www.w3.org/2001/XMLSchema”>
   <-- factory specific items of the result data schema -->
  </xsd:schema>
</as:ResultDataSchema>
<as:Expiration> xsd:duration </as:Expiration>
...
```

Example 12 Factory resource properties

```
<xsd:group name=”factoryPropertiesGroup”>
  <xsd:sequence>
    <xsd:element name="Key" type=”xsd:anyURI”/>
    <xsd:element name="Name" type=”xsd:string”/>
    <xsd:element name="Subject" type=”xsd:string”/>
    <xsd:element name="Description" type=”xsd:string”/>
    <xsd:element name="ContextDataSchema" type=”ContextDataType”/>
    <xsd:element name="ResultDataSchema" type=”ResultDataType”/>
  <xsd:element name="Expiration" type="xsd:duration"/>
  <xsd:element name=”ServiceGroupReference” type = ”wsa:EndpointReference”>
 </xsd:sequence>
</xsd:group>
<xsd:complexType name=”schemaType”>
 <xsd:any namespace=”##other”/>
 <xsd:attribute name=”href” type=”xsd:anyURI”/>
</xsd:complexType>
<xsd:complexType name=”ContextDataType”>
      <xsd:sequence>
         <xsd:any namespace=”##other”/>
      </xsd:sequence>
</xsd:complexType>
<xsd:complexType name=”ResultDataType”>
      <xsd:sequence>
         <xsd:any namespace=”##other”/>
      </xsd:sequence>
</xsd:complexType>
```

Schema 7 Factory resource properties

5.2 GetProperties

The Factory resource `GetProperties` method request is exactly the same as the Instance resource `GetProperties` request. The response returns the properties particular to the factory resource.

```
<?pseudo-xml?>
<env:Envelope xmlns:env=http://www.w3.org/2001/12/soap-envelope
 xmlns:wsa= “http://schemas.xmlsoap.org/ws/2004/08/addressing”>
 <env:Header>
  <wsa:...>
 </env:Header>
 <env:Body>
  <as:GetPropertiesRq/>
 </env:Body>
</env:Envelope>
```

Example 13 Factory resource GetProperties method request

```
<?pseudo-xml?>
<env:Envelope xmlns:env=http://www.w3.org/2001/12/soap-envelope
 xmlns:wsa= “http://schemas.xmlsoap.org/ws/2004/08/addressing”>
 <env:Header>
  <wsa:...>
 </env:Header>
 <env:Body>
  <as:GetPropertiesRs>
   <-- properties -->
  </as:GetPropertiesRs>
 </env:Body>
</env:Envelope>
```

Example 14 Factory resource GetProperties method response

```
<xsd:element name="GetPropertiesRq"/>
<xsd:element name="GetPropertiesRs" type=”factoryPropertiesGroup”/>
```

Schema 10 Factory resource GetProperties method

5.3 CreateInstance

Given a process definition resource, this method is how instances of that process are created. There are two modes: create the process, with data, and start it immediately; or just create it and put the data on it and start it manually.

`StartImmediately` element holds a Boolean value to say whether the process instances that is created should be immediately started, or whether it should be put into an initial state for later starting by use of the “start” operation. If this tag is missing, the default value is “Yes”.

`ObserverKey`: holds the endpoint reference that will receive events from the created process instance. This observer resource (if it is specified) is to be notified of events impacting the execution of this process instance such as state changes, and most notably the completion of the instance.

`Name`: A human readable name of the new instance. There is no commitment that this name be used in any way other than to return this value as the name. There are no implied uniqueness constraints.

`Subject`: A short description of the purpose of the new instance.

`Description`: A longer description of the purpose of the newly created instance.

`ContextData`: Context-specific data required to create this service instance. Must conform to the schema specified by the ContextDataSchema.

`InstanceKey`: The endpoint reference of the new Instance resource that has been created. This is NOT the same as the key for the factory that is in the Response header.

```
<?pseudo-xml?>
<env:Envelope xmlns:env=http://www.w3.org/2001/12/soap-envelope
  xmlns:wsa= “http://schemas.xmlsoap.org/ws/2004/08/addressing”>
  <env:Header>
    <wsa:...>
  </env:Header>
  <env:Body>
    <as:CreateInstanceRq>
      <as:StartImmediately>Yes|No</as:StartImmediately>
      <as:ObserverKey>? EPR </as:ObserverKey>
      <as:Name>? string </as:Name>
      <as:Subject>? string </as:Subject>
      <as:Description>? string </as:Description>
      <as:ContextData>
        <-- extensible element -->
      </as:ContextData>
    </as:CreateInstanceRq>
  </env:Body>
</env:Envelope>
```

Example 15 Factory resource CreateInstance method request

```
<?pseudo-xml?>
<env:Envelope xmlns:env=http://www.w3.org/2001/12/soap-envelope
  xmlns:wsa= http://schemas.xmlsoap.org/ws/2004/08/addressing
>
  <env:Header>
       <wsa:To>URI of the process definition receiving this request</wsa:To>
  </env:Header>
  <env:Body>
    <as:CreateInstanceRs>
      <as:InstanceKey> EPR </as:InstanceKey>
    </as:CreateInstanceRs>
  </env:Body>
</env:Envelope>
```

Example 16 Factory resource CreateInstance method request

```
<xsd:element name="CreateInstanceRq">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="StartImmediately" type="xsd:boolean"/>
      <xsd:element name="ObserverKey" type=”wsa:EndpointReference” minOccurs="0"/>
      <xsd:element name="Name” type=”xsd:string” minOccurs="0"/>
      <xsd:element name="Subject" type=”xsd:string” minOccurs="0"/>
      <xsd:element name="Description" type=”xsd:string” minOccurs="0"/>
      <xsd:element name="ContextData">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:any namespace=”##any” processContents=”lax” minOccurs=”0” maxOccurs=”unbounded”/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="CreateInstanceRs">
    <xsd:element name="InstanceKey"  type="wsa:EndpointReference"/>
</xsd:element>
```

Schema 11 Factory resource CreateInstance method

5.4 ListInstances

This method returns a collection of process instances, each instance described by a few important process instance properties.

`Filter`: Specifies what kinds of process instance resource you are interested in.

`FilterType`: indicates what language the filter is expressed in.

```
<?pseudo-xml?>
<env:Envelope xmlns:env=http://www.w3.org/2001/12/soap-envelope
  xmlns:wsa= “http://schemas.xmlsoap.org/ws/2004/08/addressing”>
  <env:Header>
    <wsa:...>
  </env:Header>
  <env:Body>
    <as:ListInstancesRq>
      <as:Filter filterType="nmtoken">?
        string
      </as:Filter>
    </as:ListInstancesRq>
  </env:Body>
</env:Envelope>
```

Example 17 Factory resource ListInstances method request

```
<?pseudo-xml?>
<env:Envelope xmlns:env=http://www.w3.org/2001/12/soap-envelope
  xmlns:wsa= “http://schemas.xmlsoap.org/ws/2004/08/addressing”>
  <env:Header>
    <as:Response...>
  </env:Header>
  <env:Body>
    <as:ListInstancesRs>
      <as:Instance>*
        <as:InstanceKey> EPR </as:InstanceKey>
        <as:Name...>?
        <as:Subject...>?
        <as:Priority...>?
      </as:Instance>
    </as:ListInstancesRs>
  </env:Body>
</env:Envelope>
```

Example 18 Factory resource ListInstances method response

```
<xsd:element name="ListInstancesRq">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Filter" type=”FilterType”>
      </xsd:element>
    </xsd:sequence>
</xsd:complexType>
</xsd:element>
<xsd:complexType name=”FilterType”>
 <xsd:simpleContent>
       <xsd:extension base=”xsd:string”>
       <xsd:attribute name=”filterType” type=”xsd:NMTOKEN”/>
   </xsd:extension>
 </xsd:simpleContent>
</xsd:complexType>

<xsd:element name="ListInstancesRs">
<xsd:complexType>
 <xsd:sequence>
<xsd:element ref=”Instance” maxOccurs=”unbounded” minOccurs=”0”/>
 </xsd:sequence>
</xsd:complexType>
</xsd:element>

<xsd:element name=”Instance”>
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="InstanceKey" type=”xsd:anyURI”/>
      <xsd:element name="Name” type=”xsd:string” minOccurs="0"/>
      <xsd:element name="Subject” type=”xsd:string” minOccurs="0"/>
      <xsd:element ref="Priority" type=”xsd:int” minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Schema 12 Factory resource ListInstances method

6 Observer resource

6.1 Observer resource properties

The Observer resource can receive events about the state changes of a service instance. An observer is expected to have a Key.

`Key`: a URI that uniquely identifies the resource. All resources must have a Key property.

```
<xsd:element name="Key" type=”xsd:AnyURI”/>
```

Schema 13 Observer resource properties

6.2 GetProperties

This method is the same as it was with Instance and Factory resources.

```
<xsd:element name="GetPropertiesRq"/>
<xsd:element name="GetPropertiesRs" type=”observerPropertiesGroup”/>
```

Schema 14 Observer resource GetProperties method

6.3 Completed

The Completed method indicates that the Instance has completed the work. This is the 'normal' completion.

This function signals to the observer resource that the started process is completed its task, and will no longer be processing. There is no guarantee that the resource will persist after this point in time.

`InstanceKey`: The URI of a process that is performing this work. The process is addressable by the wsa:From EPR.

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

```
<?pseudo-xml?>
<env:Envelope xmlns:env="http://www.w3.org/2001/12/soap-envelope">
  <env:Header>
    <as:Request...>
  </env:Header>
  <env:Body>
    <as:CompletedRq>
      <as:InstanceKey> URI </as:Instance>
      <as:ResultData>
        <-- extensible element -->
      </as:ResultData>
    </as:CompletedRq>
  </env:Body>
</env:Envelope>
```

Example 19 Observer resource Completed method request

```
<?pseudo-xml?>
<env:Envelope xmlns:env="http://www.w3.org/2001/12/soap-envelope">
  <env:Header>
    <as:Response...>
  </env:Header>
  <env:Body>
    <as:CompletedRs/>
  </env:Body>
</env:Envelope>
```

Example 20 Observer resource Completed method response

```
<xsd:element name="CompletedRq">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name=”InstanceKey” type=”xsd:anyURI”/>
      <xsd:element name=”ResultData”>
        <xsd:complexType>
          <xsd:sequence>
            <xsd:any namespace=”##any” processContents=”lax” minOccurs=”0” maxOccurs=”unbounded”/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="CompletedRs"/>
```

Schema 15 Observer resource Completed method

6.4 StateChanged

Observers receive a StateChanged message from the Instance when the state of the Instance changes. The response to a notify event is not necessary. Typically, the header request tag will specify that no response is necessary.

```
<?pseudo-xml?>
<env:Envelope xmlns:env="http://www.w3.org/2001/12/soap-envelope">
  <env:Header>
    <as:Request...>
  </env:Header>
  <env:Body>
    <as:StateChanged>
      <as:State> …
      <as:PreviousState> …
    </as:StateChanged>
  </env:Body>
</env:Envelope>
```

Example 21 Observer resource StateChanged method request

```
<?pseudo-xml?>
<env:Envelope xmlns:env="http://www.w3.org/2001/12/soap-envelope">
  <env:Header>
    <as:Response...>
  </env:Header>
  <env:Body>
    <as:StateChangedRs/>
  </env:Body>
</env:Envelope>
```

Example 22 Observer resource StateChanged method response

```
<xsd:element name="StateChangedRq">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name=”State" type=”as:stateType”/>
      <xsd:element name=”PreviousState" type=”as:stateType”/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

```
<xsd:element name="StateChangedRs"/>
```

Schema 16 Observer resource StateChanged method

7 Data encoding

7.1 Context data and result data

The heart of an asynchronous service is the `ContextData` and the `ResultData`. The `ContextData` and the `ResultData` are the unique part of a particular service; everything else is boilerplate. The `ContextData` is the query or the initial request to the service. The `ContextData` dictates, determines or implies what the service instance should create. The `ResultData` is what the service eventually creates for the observers.

The service factory should provide a schema for the `ContextData` element and `ResultData` element. The schema may be XML Schema or Relax NG. ASAP follows the SOAP and XML Schema data type encoding specifications.

7.2 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. Those additional properties should be elements of a namespace that is not ASAP. 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.

7.3 State type

The overall status of the asynchronous web service is defined by a state property value. This is a string value composed of words separated by periods. The status value must start with one of the seven defined values below, but the value can be extended by adding words on the end of the status separated by periods. The extension must be a refinement of one of the seven states defined here, such that it is not necessary to understand the extension. The intention is that these extensions may be proposed for future inclusion in the standard. The seven defined base states are:

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

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

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

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

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

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

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

```
<xsd:complexType name="stateType">
  <xsd:simpleContent>
    <xsd:extension base="xsd:string">
      <xsd:attribute name="namespace" type="xsd:anyURI"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
```

Schema 17 stateType

These state values come from the Workflow Management Coalition standards.

7.4 History type

The history is optional. It contains a list of events. An event is a state change that can occur in the asynchronous service that is externally identifiable. Notifications can be sent to an observer in order to inform it of the particular event.

Time: the date/time of the event that occurred

EventType: One of an enumerated set of values to specify event types: InstanceCreated, PropertiesSet, StateChanged, Subscribed, Unsubscribed, Error. The event types correspond to the message types that the resource can receive.

SourceKey: The EPR of the resource that triggered this event, usually an observer resource but perhaps the instance resource itself.

Details: A catchall element for containing any data appropriate.

OldState: The state of the instance resource before this event occurred.

NewState: The state of the instance resource before this event occurred.

```
<xsd:element name=”Event”>
 <xsd:complexType>
 <xsd:sequence>
<xsd:element name="Time" type="xsd:dateTime"/>
<xsd:element name="EventType">
  <xsd:simpleType>
<xsd:restriction base="xsd:string">
 <xsd:enumeration value="InstanceCreated"/>
   <xsd:enumeration value="PropertiesSet"/>
   <xsd:enumeration value="StateChanged"/>
   <xsd:enumeration value="Subscribed"/>
    <xsd:enumeration value="Unsubscribed"/>
    <xsd:enumeration value="Error"/>
</xsd:restriction>
</xsd:simpleType>
</xsd:element>
<xsd:element name="SourceKey" type="wsa:EndpointReference"/>
<xsd:element name="Details" type="xsd:anyType"/>
<xsd:element name="OldState" type="as:stateType"/>
<xsd:element name="NewState" type="as:stateType"/>
 </xsd:sequence>
 </xsd:complexType>
 </xsd:element>
<xsd:complexType name=”historyType”>
 <xsd:sequence>
       <xsd:element ref=”Event” maxOccurs=”unbounded”/>
 </xsd:sequence>
</xsd:complexType>
```

Schema 17 complexType

7.5 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://www.w3.org/2001/12/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 23 Exception

These error codes are chosen to be specific with the error codes defined by the Workflow Management Coalition Wf-MXL 1.1 specification.

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

7.6 Language

ASAP messages should indicate their preferred language using the xml:lang attribute either in the SOAP Envelope element (the root element) or in the ASAP Request or Response element.

7.7 Security

HTTP provides for both authenticated as well as anonymous requests. Because of the nature of process flow in controlling access to resources, many operations will not be allowed unless accompanied by a valid and authenticated user ID. There are two primary means that this will be provided: HTTP authorization header or transport level encryption such as SSL.

The first and most common method of authentication over HTTP is through the use of the Authorization header. This header carries a user name and a password that can be used to validate against a user directory. If the request is attempted but the authentication of the user fails, or the Authorization header field is not present, then the standard HTTP error "401 Unauthorized" is the response. Within this, there are two authentication schemes:

- Basic involves carrying the name and password in the authorization field and is not considered secure.
- A digest authentication for HTTP is specified in IETF RFC-2069 [http://ietf.org/rfc/rfc2069.html], which offers a way to securely authenticate without sending the password in the clear.

Second, encryption at the transport level, such as SSL, can provide certificate based authentication of the user making the request. This is much more secure than the previous option, and should be used when high security is warranted.

Because the lower protocol levels are providing the user ID, ASAP does not specify how to send the client user ID. The authenticated user ID can be assumed to be present in the server at the time of handling the request.

Note that since most ASAP interactions are between programs that we would normally consider to be servers (i.e. process flow engine to process flow engine) the conclusion can be made that all such process flow engines will need a user id and associated values (e.g. password or certificate) necessary to authenticate themselves to other servers. Servers must be configured with the appropriate safeguards to assure that these associated values are protected from view. Under no circumstances should a set of process flow engines be configured to make anonymous ASAP requests that update information since the only way to be sure that the request is coming from a trustable source is through the authentication.

With the authentication requirements above, of either HTTP authorization header field or SSL secure transport, ASAP should be able to protect and safeguard sensitive data while allowing interoperability to and from any part of the Internet.

8 References

8.1 Normative

[RFC2119]	S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, http://www.ietf.org/rfc/rfc2119.txt, IETF RFC 2119, March 1997.
[SOAP]	Simple Object Access Protocol
[W3C Arch]	Web Services Architecture Working Group, http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/
[XSD]	XML Schema Part 1 & Part 2 http://www.w3.org/TR/xmlschema-1/ **and** http://www.w3.org/TR/xmlschema-2/
[WS-Addressing]	Don Box, et al., *Web services Addressing (WS-Addressing)*, W3C Member Submission, August 2004, http://www.w3.org/Submission/2004/SUBM-ws-addressing-20040810/
[RAYMOND]	The Art of Unix Programming by Eric S. Raymond, Addision Wesley Publishers

Appendix A. Schema

```
<?xml version="1.0"?>
<xsd:schema
 targetNamespace= "http://docs.oasis-open.org/asap/1.0/asap.xsd"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:wsa= "http://schemas.xmlsoap.org/ws/2004/08/addressing"
 xmlns:as="http://docs.oasis-open.org/asap/1.0/asap.xsd"
>

<xsd:import namespace= "http://schemas.xmlsoap.org/ws/2004/08/addressing"
   schemaLocation="http://schemas.xmlsoap.org/ws/2004/08/addressing"/>

<!-- ===== simple property elements ===== -->

<xsd:complexType name="schemaType">
   <xsd:sequence>
    <xsd:any namespace="##other"/>
   </xsd:sequence>
   <xsd:attribute name="href" type="xsd:anyURI"/>
</xsd:complexType>

<xsd:complexType name="ContextDataType">
   <xsd:sequence>
    <xsd:any namespace="##other"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="ResultDataType">
   <xsd:sequence>
    <xsd:any namespace="##other"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="stateType">
  <xsd:simpleContent>
    <xsd:extension base="xsd:string">
    <xsd:attribute name="namespace" type="xsd:anyURI"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>

<xsd:element name="Event">
   <xsd:complexType>
    <xsd:sequence>
       <xsd:element name="Time" type="xsd:dateTime"/>
       <xsd:element name="EventType">
        <xsd:simpleType>
           <xsd:restriction base="xsd:string">
            <xsd:enumeration value="InstanceCreated"/>
            <xsd:enumeration value="PropertiesSet"/>
            <xsd:enumeration value="StateChanged"/>
            <xsd:enumeration value="Subscribed"/>
            <xsd:enumeration value="Unsubscribed"/>
            <xsd:enumeration value="Error"/>
           </xsd:restriction>
        </xsd:simpleType>
       </xsd:element>
       <xsd:element name="SourceKey" type="wsa:EndpointReferenceType"/>
       <xsd:element name="Details" type="xsd:anyType"/>
       <xsd:element name="OldState" type="as:stateType"/>
       <xsd:element name="NewState" type="as:stateType"/>
    </xsd:sequence>
   </xsd:complexType>
</xsd:element>

<xsd:complexType name="historyType">
   <xsd:sequence>
    <xsd:element ref="as:Event" maxOccurs="unbounded">
    </xsd:element>
   </xsd:sequence>
</xsd:complexType>

<xsd:simpleType name="YesNoIfError">
   <xsd:restriction base="xsd:string">
    <xsd:enumeration value="Yes"/>
    <xsd:enumeration value="No"/>
    <xsd:enumeration value="IfError"/>
  </xsd:restriction>
</xsd:simpleType>

<!-- ===== properties ===== -->

<xsd:group name="resourcePropertiesGroup">
  <xsd:sequence>
  <xsd:element name="Key" type="xsd:anyURI"/>
  </xsd:sequence>
</xsd:group>

<xsd:group name="descriptivePropertiesGroup">
  <xsd:sequence>
  <xsd:element name="Name" type="xsd:string"/>
  <xsd:element name="Subject" type="xsd:string"/>
  <xsd:element name="Description" type="xsd:string"/>
  </xsd:sequence>
</xsd:group>


<xsd:group name="instancePropertiesGroup">
  <xsd:sequence>
  <xsd:group ref="as:resourcePropertiesGroup"/>
  <xsd:group ref="as:descriptivePropertiesGroup"/>
  <xsd:element name="State" type="as:stateType"/>
  <xsd:element name="FactoryKey" type="wsa:EndpointReferenceType"/>
  <xsd:element name="Observers">
  <xsd:complexType>
    <xsd:sequence>
     <xsd:element      name="ObserverKey"      type="wsa:EndpointReferenceType"
maxOccurs="unbounded" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
  </xsd:element>
  <xsd:element name="ContextData">
    <xsd:complexType>
     <xsd:sequence>
     <xsd:any namespace="##any" processContents="lax"
     minOccurs="0" maxOccurs="unbounded"/>
     </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="ResultData">
    <xsd:complexType>
     <xsd:sequence>
```

```
       <xsd:any namespace="##any" processContents="lax"
        minOccurs="0" maxOccurs="unbounded"/>
     </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
    <xsd:element name="History" type="as:historyType"/>
    <xsd:element name="UserInterface" type="wsa:EndpointReferenceType"
     minOccurs="0"/>
   </xsd:sequence>
</xsd:group>

<xsd:group name="factoryPropertiesGroup">
  <xsd:sequence>
  <xsd:group ref="as:resourcePropertiesGroup"/>
  <xsd:group ref="as:descriptivePropertiesGroup"/>
  <xsd:element name="ContextDataSchema" type="as:ContextDataType"/>
  <xsd:element name="ResultDataSchema" type="as:ResultDataType"/>
  <xsd:element name="Expiration" type="xsd:duration"/>
  <xsd:element name="ServiceGroupReference" type="wsa:EndpointReferenceType"/>
  </xsd:sequence>
</xsd:group>

<xsd:group name="observerPropertiesGroup">
  <xsd:sequence>
    <xsd:group ref="as:resourcePropertiesGroup"/>
  </xsd:sequence>
</xsd:group>

<!-- ====== messages ===== -->

<xsd:element name="GetPropertiesRq"/>

<xsd:element name="GetPropertiesRs">
   <xsd:complexType>
    <xsd:choice>
       <xsd:group ref="as:instancePropertiesGroup"/>
       <xsd:group ref="as:factoryPropertiesGroup"/>
       <xsd:group ref="as:observerPropertiesGroup"/>
    </xsd:choice>
   </xsd:complexType>
</xsd:element>

<xsd:element name="SetPropertiesRq">
   <xsd:complexType>
    <xsd:sequence>
       <xsd:element name="Subject" type= "xsd:string"/>
       <xsd:element name="Description" type="xsd:string"/>
       <xsd:element name="Priority" type="xsd:int"/>
       <xsd:element name="Data">
        <xsd:complexType>
           <xsd:sequence>
            <xsd:any namespace="##any" processContents="lax"
             minOccurs="0" maxOccurs="unbounded"/>
           </xsd:sequence>
        </xsd:complexType>
       </xsd:element>
    </xsd:sequence>
   </xsd:complexType>
</xsd:element>

<xsd:element name="SetPropertiesRs">
  <xsd:complexType>
     <xsd:choice>
```

```
        <xsd:group ref="as:instancePropertiesGroup"/>
        <xsd:group ref="as:factoryPropertiesGroup"/>
        <xsd:group ref="as:observerPropertiesGroup"/>
      </xsd:choice>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="SubscribeRq">
     <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="ObserverKey" type="wsa:EndpointReferenceType"/>
      </xsd:sequence>
     </xsd:complexType>
  </xsd:element>

  <xsd:element name="SubscribeRs"/>

  <xsd:element name="UnsubscribeRq">
     <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="ObserverKey" type="wsa:EndpointReferenceType"/>
      </xsd:sequence>
     </xsd:complexType>
  </xsd:element>

  <xsd:element name="UnsubscribeRs"/>

  <xsd:element name="CreateInstanceRq">
     <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="StartImmediately" type="xsd:boolean"/>
         <xsd:element name="ObserverKey" type="wsa:EndpointReferenceType"
          minOccurs="0"/>
         <xsd:element name="Name" type="xsd:string" minOccurs="0"/>
         <xsd:element name="Subject" type="xsd:string" minOccurs="0"/>
         <xsd:element name="Description" type="xsd:string"
          minOccurs="0"/>
         <xsd:element name="ContextData">
          <xsd:complexType>
             <xsd:sequence>
              <xsd:any namespace="##any" processContents="lax"
               minOccurs="0" maxOccurs="unbounded"/>
             </xsd:sequence>
          </xsd:complexType>
         </xsd:element>
      </xsd:sequence>
     </xsd:complexType>
  </xsd:element>

  <xsd:element name="CreateInstanceRs">
     <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="InstanceKey" type="wsa:EndpointReferenceType"/>
      </xsd:sequence>
     </xsd:complexType>
  </xsd:element>

  <xsd:complexType name="FilterType">
     <xsd:simpleContent>
      <xsd:extension base="xsd:string">
         <xsd:attribute name="filterType" type="xsd:NMTOKEN"/>
      </xsd:extension>
     </xsd:simpleContent>
```

```
</xsd:complexType>

<xsd:element name="ListInstancesRq">
   <xsd:complexType>
    <xsd:sequence>
       <xsd:element name="Filter" type="as:FilterType">
       </xsd:element>
    </xsd:sequence>
   </xsd:complexType>
</xsd:element>

<xsd:element name="Instance">
   <xsd:complexType>
    <xsd:sequence>
       <xsd:element name="InstanceKey" type="wsa:EndpointReferenceType"/>
       <xsd:element name="Name" type="xsd:string" minOccurs="0"/>
       <xsd:element name="Subject" type= "xsd:string" minOccurs="0"/>
       <xsd:element name="Priority" type="xsd:int" minOccurs="0"/>
    </xsd:sequence>
   </xsd:complexType>
</xsd:element>

<xsd:element name="ListInstancesRs">
   <xsd:complexType>
    <xsd:sequence>
       <xsd:element ref="as:Instance" maxOccurs="unbounded"
        minOccurs="0"/>
    </xsd:sequence>
   </xsd:complexType>
</xsd:element>

<xsd:element name="CompletedRq">
   <xsd:complexType>
    <xsd:sequence>
       <xsd:element name="InstanceKey" type="xsd:anyURI"/>
       <xsd:element name="ResultData">
        <xsd:complexType>
           <xsd:sequence>
            <xsd:any namespace="##any" processContents="lax"
             minOccurs="0" maxOccurs="unbounded"/>
           </xsd:sequence>
        </xsd:complexType>
       </xsd:element>
    </xsd:sequence>
   </xsd:complexType>
</xsd:element>

<xsd:element name="CompletedRs"/>

<xsd:element name="ChangeStateRq">
   <xsd:complexType>
    <xsd:sequence>
       <xsd:element name="State" type="as:stateType"/>
    </xsd:sequence>
   </xsd:complexType>
</xsd:element>

<xsd:element name="ChangeStateRs">
  <xsd:complexType>
    <xsd:sequence>
    <xsd:element name="State" type="as:stateType"/>
    </xsd:sequence>
  </xsd:complexType>
```

```
</xsd:element>

<xsd:element name="StateChangedRq">
  <xsd:complexType>
    <xsd:sequence>
    <xsd:element name="State" type="as:stateType"/>
    <xsd:element name="PreviousState" type="as:stateType"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="StateChangedRs"/>

</xsd:schema>
```

WSDL Elements

```
<?xml version="1.0" encoding="utf-8"?>
<wsd:definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
           xmlns:wsd="http://schemas.xmlsoap.org/wsdl/"
           xmlns:xsd="http://www.w3.org/2001/XMLSchema"
           xmlns:asdl="http://docs.oasis-open.org/asap/1.0/asap.wsdl"
           xmlns:asap="http://docs.oasis-open.org/asap/1.0/asap.xsd"
           xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
           targetNamespace="http://docs.oasis-open.org/asap/1.0/asap.wsdl"
>

<wsd:types>
<xsd:schema elementFormDefault= "qualified"
            targetNamespace= "http://docs.oasis-open.org/asap/1.0/asap.wsdl">
   <xsd:import namespace="http://docs.oasis-open.org/asap/1.0/asap.xsd"
               schemaLocation="http://docs.oasis-open.org/asap/1.0/asap.xsd"/>
</xsd:schema>
</wsd:types>

<!-- Messages =========== -->

<wsd:message name="completedRequest">
  <wsd:part name="body" element="asap:CompletedRq"/>
</wsd:message>

<wsd:message name="completedResponse">
  <wsd:part name="body" element="asap:CompletedRs"/>
</wsd:message>

<wsd:message name="createInstanceRequest">
  <wsd:part name="body" element="asap:CreateInstanceRq"/>
</wsd:message>

<wsd:message name="createInstanceResponse">
  <wsd:part name="body" element="asap:CreateInstanceRs"/>
</wsd:message>

<wsd:message name="getPropertiesRequest">
  <wsd:part name="body" element="asap:GetPropertiesRq"/>
</wsd:message>

<wsd:message name="getPropertiesResponse">
  <wsd:part name="body" element="asap:GetPropertiesRs"/>
</wsd:message>

<wsd:message name="listInstancesRequest">
  <wsd:part name="body" element="asap:ListInstancesRq"/>
</wsd:message>

<wsd:message name="listInstancesResponse">
  <wsd:part name="body" element="asap:ListInstancesRs"/>
</wsd:message>

<wsd:message name="setPropertiesRequest">
  <wsd:part name="body" element="asap:SetPropertiesRq"/>
</wsd:message>

<wsd:message name="setPropertiesResponse">
```

```
  <wsd:part name="body" element="asap:SetPropertiesRs"/>
</wsd:message>

<wsd:message name="subscribeRequest">
  <wsd:part name="body" element="asap:SubscribeRq"/>
</wsd:message>

<wsd:message name="subscribeResponse">
  <wsd:part name="body" element="asap:SubscribeRs"/>
</wsd:message>

<wsd:message name="unsubscribeRequest">
  <wsd:part name="body" element="asap:UnsubscribeRq"/>
</wsd:message>

<wsd:message name="unsubscribeResponse">
  <wsd:part name="body" element="asap:UnsubscribeRs"/>
</wsd:message>

<wsd:message name="stateChangedRequest">
  <wsd:part name="body" element="asap:StateChangedRq"/>
</wsd:message>

<wsd:message name="changeStateRequest">
  <wsd:part name="body" element="asap:ChangeStateRq"/>
</wsd:message>

<wsd:message name="stateChangedResponse">
  <wsd:part name="body" element="asap:StateChangedRs"/>
</wsd:message>

<wsd:message name="changeStateResponse">
  <wsd:part name="body" element="asap:ChangeStateRs"/>
</wsd:message>

<!-- Port types ============== -->

<wsd:portType name="InstancePortType">
  <wsd:operation name="GetProperties">
    <wsd:input message="asdl:getPropertiesRequest"/>
    <wsd:output message="asdl:getPropertiesResponse"/>
  </wsd:operation>
  <wsd:operation name="SetProperties">
    <wsd:input message="asdl:setPropertiesRequest"/>
    <wsd:output message="asdl:setPropertiesResponse"/>
  </wsd:operation>
  <wsd:operation name="Subscribe">
    <wsd:input message="asdl:subscribeRequest"/>
    <wsd:output message="asdl:subscribeResponse"/>
  </wsd:operation>
  <wsd:operation name="Unsubscribe">
    <wsd:input message="asdl:unsubscribeRequest"/>
    <wsd:output message="asdl:unsubscribeResponse"/>
  </wsd:operation>
  <wsd:operation name="ChangeState">
    <wsd:input message="asdl:changeStateRequest"/>
    <wsd:output message="asdl:changeStateResponse"/>
  </wsd:operation>
</wsd:portType>

<wsd:portType name="FactoryPortType">
  <wsd:operation name="GetProperties">
    <wsd:input message="asdl:getPropertiesRequest"/>
    <wsd:output message="asdl:getPropertiesResponse"/>
  </wsd:operation>
  <wsd:operation name="CreateInstance">
    <wsd:input message="asdl:createInstanceRequest"/>
    <wsd:output message="asdl:createInstanceResponse"/>
  </wsd:operation>
  <wsd:operation name="ListInstances">
    <wsd:input message="asdl:listInstancesRequest"/>
    <wsd:output message="asdl:listInstancesResponse"/>
  </wsd:operation>
</wsd:portType>

<wsd:portType name="ObserverPortType">
  <wsd:operation name="GetProperties">
    <wsd:input message="asdl:getPropertiesRequest"/>
    <wsd:output message="asdl:getPropertiesResponse"/>
  </wsd:operation>
  <wsd:operation name="Completed">
    <wsd:input message="asdl:completedRequest"/>
    <wsd:output message="asdl:completedResponse"/>
  </wsd:operation>
  <wsd:operation name="StateChanged">
    <wsd:input message="asdl:stateChangedRequest"/>
    <wsd:output message="asdl:stateChangedResponse"/>
  </wsd:operation>
</wsd:portType>

</wsd:definitions>
```

Appendix B. Acknowledgments

The following individuals were members of the committee during the development of this specification:

- Jeffrey Ricker
- Keith Swenson, Fujitsu
- Moshe Silverstein, iWay Software
- John Fuller, for EasyASAP
- Jeff Cohen, for .Net ASAP

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

- 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
- Paul Lyman, United Technologies
- Ian Prittie
- Members of the Workflow Management Coalition
- And many others....

Appendix C. Revision History

Rev	Date	By Whom	What
wd-01d	2003-09-09	Jeffrey Ricker	Draft for first meeting
wd-01e	2004-04-22	Mayilraj Krishnan	Draft for Publishing
Wd-01 f	2004-06-01	Mayilraj Krishnan	Schema and Minor changes
Wd-02a	2004-11-24	John Fuller	Added UserInterface instance property Corrected typos, formatting Introduced use of WS Addressing

Appendix D. Notices