

Asynchronous Service Access Protocol (ASAP) Version 1.0

Working Draft G, June 4, 2004

Document identifier:

wd-asap-spec-01

Location:<http://www.oasis-open.org/committees/asap/docs/>**Editors:**

Jeffrey Ricker, Individual, <jricker@izarinc.com>
Mayilraj Krishnan, Cisco Systems, <mkrishna@cisco.com>
Keith Swenson, Fujitsu Software, <KSwenson@us.fujitsu.com>

Committee Members:

John Fuller, Individual, <jfuller@wernervas.com>
Moshe Silverstein, Individual, <moses@silversteingroup.com>
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

48	1	Introduction	4
49	1.1	Summary	4
50	1.2	Not-so-technical executive summary	4
51	1.3	Problem statement	5
52	1.4	Things to achieve	5
53	1.5	Things not part of the goals.....	6
54	1.6	Terminology.....	6
55	1.7	Notation conventions.....	7
56	1.8	Related documents	7
57	2	Resource model.....	8
58	2.1	Overview	8
59	2.2	Instance.....	9
60	2.3	Factory	9
61	2.4	Observer.....	9
62	2.5	URI	9
63	2.6	ContextData and ResultData	9
64	3	Protocol.....	10
65	3.1	SOAP	10
66	3.2	Request header.....	10
67	3.3	Response header	11
68	3.4	Body	12
69	4	Instance resource	13
70	4.1	Instance resource properties.....	13
71	4.2	GetProperties	15
72	4.3	SetProperties.....	16
73	4.4	Subscribe	17
74	4.5	Unsubscribe	17
75	4.6	ChangeState	18
76	5	Factory resource.....	20
77	5.1	Factory resource properties	20
78	5.2	GetProperties	21
79	5.3	CreateInstance	21
80	5.4	ListInstances	23
81	6	Observer resource	25
82	6.1	Observer resource properties	25
83	6.2	GetProperties	25
84	6.3	Completed	25
85	6.4	StateChanged	26
86	7	Data encoding.....	28
87	7.1	Context data and result data.....	28
88	7.2	Extensibility	28
89	7.3	State type	28

90	7.4 History type	29
91	7.5 Exceptions and error codes	30
92	7.6 Language	31
93	7.7 Security	31
94	8 References.....	33
95	8.1 Normative	33
96	Appendix A. Schema	34
97	Appendix B. Acknowledgments	41
98	Appendix C. Revision History	42
99	Appendix D. Notices	43
100		

101 1 Introduction

102 1.1 Summary

103 This protocol offers a way to start an instance of an asynchronous web service, monitor it, control
104 it, and be notified when it is complete. This service instance can perform just about anything for
105 any purpose. The key aspect is that the service instance is something that one would like to start
106 remotely, and it will take a long time to run to completion. Short-lived services would be invoked
107 synchronously with Simple Object Access Protocol (SOAP) **[SOAP]** and one would simply wait
108 for completion. Because certain service instances last anywhere from a few minutes to a few
109 months, they must be invoked asynchronously.

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

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

125 1.2 Not-so-technical executive summary

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

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

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

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

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

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

160 **1.3 Problem statement**

161 Not all services are instantaneous. A standard protocol is needed to integrate asynchronous
162 services (processes or work providers) across the Internet and provide for their interaction. The
163 integration and interactions consist of control and monitoring of the service. *Control* means
164 creating the service, setting up the service, starting the service, stopping the service, being
165 informed of exceptions, being informed of the completion of the service and getting the results of
166 the service. *Monitoring* means checking on the current status and getting execution history of the
167 service.

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

170 **1.4 Things to achieve**

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

- 173 • The protocol should not reinvent anything unnecessarily. If a suitable standard exists, it
174 should be used rather than re-implement in a different way.
- 175 • The protocol should be consistent with XML Protocol and SOAP.
- 176 • This protocol should be easy to incorporate into other SOAP-based protocols that require
177 asynchronous communication
- 178 • The protocol should be the minimal necessary to support a generic asynchronous service.
179 This means being able to start, monitor, exchange data with, and control a generic
180 asynchronous service on a different system.
- 181 • The protocol must be extensible. The first version will define a very minimal set of
182 functionality. Yet a system must be able to extend the capability to fit the needs of a
183 particular requirement, such that high level functionality can be communicated which
184 gracefully degrades to interoperate with systems that do not handle those extensions.
- 185 • Like other Internet protocols, ASAP should not require or make any assumptions about the
186 platform or the technology used to implement the generic asynchronous service.
- 187 • Terseness of expression is not a goal of this protocol. Ease of generating, understanding
188 and parsing should be favored over compactness.

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

195 1.5 Things not part of the goals

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

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

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

216 1.6 Terminology

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

219 Other specific terms are as follows.

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

226 Service: synonymous with web service.

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

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

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

238 Instance: This is the resource implemented by the web service that is actually performing the
239 requested work. These resources allow for the actual monitoring and controlling of the work.

240 Factory: This is the resource implemented by the service instance factory. Methods are provided
241 to start new service instances, to list or search for existing instances, and to provide definitional
242 information about the instances.

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

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

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

247 1.7 Notation conventions

248 The following namespace prefixes are used throughout this document:

Prefix	Namespace URI	Definition
as	http://www.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

249 *Table 1 Namespaces*

250 This specification uses an informal syntax we call *pseudo-XML* to describe the XML grammar of
251 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.	<code><p:tag name="nmtoken"/></code>
Paragraphs within tags are the description of the tag and should be thought of as commented out with <code><!-- --></code>	<code><p:tag> longer description of the purpose of the tag. </p:tag></code>
Characters are appended to elements and attributes as follows: "?" (0 or 1), "*" (0 or more), "+" (1 or more).	<code><p:tag>*</code>
Elements names ending in "." indicate that elements/attributes irrelevant to the context are being omitted or they are exactly as defined previously.	<code><p:tag.../></code>
Grammar in bold has not been introduced earlier in the document, or is of particular interest in an example.	<code><p:tag/></code>
"Extensible element" is a placeholder for elements from some "other" namespace (like ##other in XSD).	<code><!-- extensible element --></code>
The XML namespace prefixes (defined above) are used to indicate the namespace of the element being defined	
Examples starting with <code><?pseudo-xml?></code> contain enough information to conform to this specification; others examples are fragments and require additional information to be specified in order to conform.	<code><?pseudo-xml?></code>

252 *Table 2 Pseudo-XML documentation conventions*

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

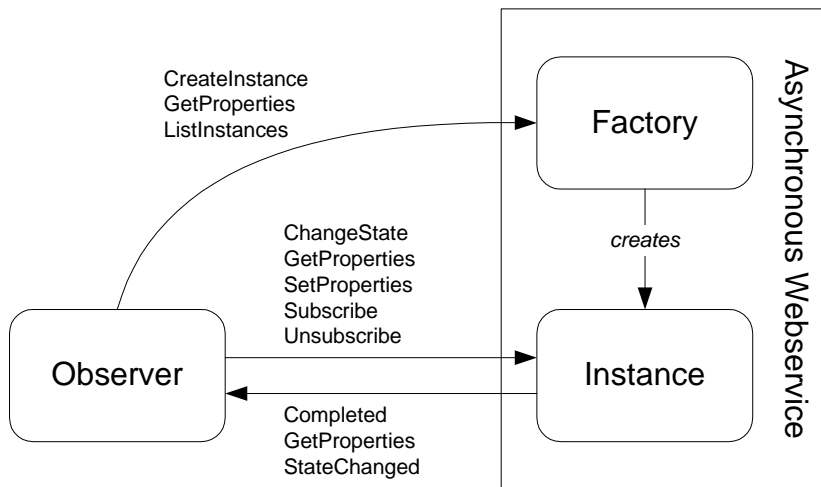
255 1.8 Related documents

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

257 **2 Resource model**

258 **2.1 Overview**

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

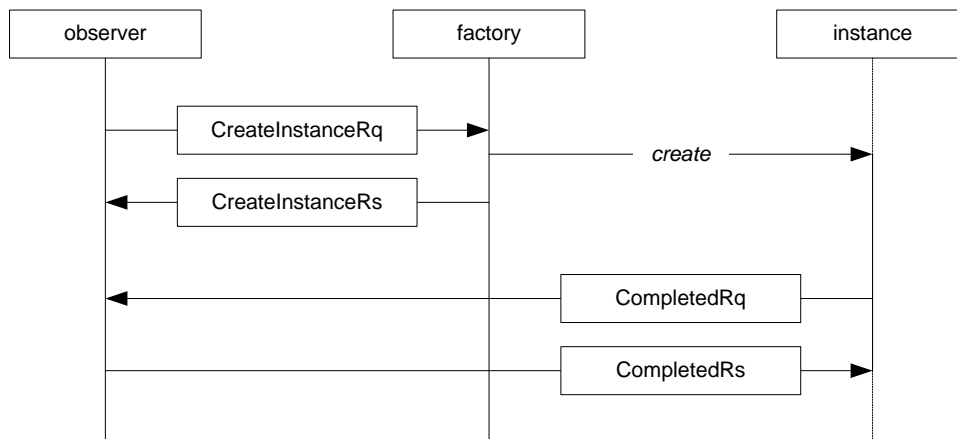


262
 263 *Figure 1 Resource types of an asynchronous web service and the methods they use*

264 Typical use of this protocol would be as follows:

- 265 1. A Factory service receives a CreateInstanceRq message that contains ContextData
- 266 and the URI of an Observer
- 267 2. The Factory service creates an Instance service and subscribes the Observer to the Instance
- 268 3. The Factory responds to CreateInstanceRq message with a CreateInstanceRs
- 269 message that contains the URI of the Instance
- 270 4. The Instance service eventually completes its task and sends a CompletedRq message that
- 271 contains the ResultsData to the Observer

272



273
 274 *Figure 2 Typical use of ASAP*

275 **2.2 Instance**

276 The Instance resource is the actual "performance of work". It embodies the context information
277 that distinguishes one performance of one asynchronous service from another. Every time the
278 asynchronous service is to be invoked, a new instance is created and given its own resource
279 identifier. A service instance can be used only once: it is created, then it can be started, it can be
280 paused, resumed, terminated. If things go normally, it will eventually complete.

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

286 **2.3 Factory**

287 The Factory resource represents a "way of doing some work". It is the most fundamental
288 resource required for the interaction of generic services. It represents the description of a
289 service's most basic functions, and is the resource from which instances of a service will be
290 created. Since every service to be enacted must be uniquely identifiable by an interoperating
291 service or service requestor, the factory will provide a resource identifier. When a service is to be
292 enacted, this resource identifier will be used to reference the desired asynchronous service to be
293 executed. A service might be a set of tasks carried out by a group of individuals, or it might be
294 set of machine instructions that make up an executable program, or it might be any combination
295 of these. The important point to remember about a service factory is that while it embodies the
296 knowledge of how work is performed, it does not actually do the work. The service instance does
297 the work.

298 **2.4 Observer**

299 The Observer resource provides a means by which a service instance may communicate
300 information about events occurring during its execution, such as its completion or termination.
301 Third-party resources may have an interest in the status of a given service instance for various
302 organization and business reasons. Observers subscribe to a service instance by providing a
303 URI. A service instance notifies all observers by sending SOAP messages to the observer URI's.

304 **2.5 URI**

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

310 **2.6 ContextData and ResultData**

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

316

3 Protocol

317

3.1 SOAP

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

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

327 The message must contain a SOAP header as per the SOAP standard for addressing and routing
328 the message. An ASAP message will contain within the SOAP header either a *Request* element
329 or a *Response* element. A message from a client must contain the *Request* element and a
330 message from a server must contain a *Response* element.

3.2 Request header

332 The *Request* element contains the following elements.

333 *SenderKey*: The request *MAY* specify the URI or key of the resource that originated the request.
334 This may be redundant with similar specifier in the transport layer.

335 *ReceiverKey*: The request *MUST* specify the key of the resource that the request is being made
336 to. This may be redundant with similar specifier in the transport layer.

337 *ResponseRequired*: This optional tag may contain the following values: *Yes*, *No*, or *IfError*. If the
338 value specified is "Yes", a response must be returned for this request in all cases, and it must be
339 processed by the requesting resource. If the value specified is "No", a response may, but need
340 not be returned for this request, and if one is returned it may be ignored by the requesting
341 resource. If the value specified is "IfError", a response only needs to be returned for this request
342 in the case where an error has occurred processing it, and the requesting resource must process
343 the response. If this tag is not specified, the default value is assumed to be "Yes".

344 *RequestID*: The requester may optionally specify a unique ID for the request. If present, then
345 this ID must be returned to the requester in the *RequestID* tag of the response in order to
346 correlate that response with the original request. The value is assumed to be an opaque value.

347

348

349

350

351

352

353

354

355

356

357

358

359

360

361

362

```
<?pseudo-xml?>
<env:Envelope xmlns:env="http://www.w3.org/2001/12/soap-envelope">
  <env:Header>
    <as:Request xmlns:as="http://www.oasis-open.org/asap/0.9/asap.xsd">
      <as:SenderKey? The URI of the sender </as:SenderKey>
      <as:ReceiverKey> The URI of the receiver </as:ReceiverKey>
      <as:ResponseRequired>Yes|No|IfError</as:ResponseRequired>
      <as:RequestID?>
        Unique ID for message correlation by the requestor
      </as:RequestID>
    </as:Request>
  </env:Header>
  <env:Body>
    . . .
  </env:Body>
</env:Envelope>
```

363 *Example 1 Request header*

```
364 <xsd:element name="Request">
365   <xsd:complexType>
366     <xsd:sequence>
367       <xsd:element name="SenderKey" type="xsd:anyURI" minOccurs="0"/>
368       <xsd:element name="ReceiverKey" type="xsd:anyURI"/>
369       <xsd:element name="ResponseRequired" type="YesNoIfError" minOccurs="0">
370       <xsd:element name="RequestID" type="xsd:anyURI" minOccurs="0"/>
371     </xsd:sequence>
372   </xsd:complexType>
373 </xsd:element>
374 <xsd:simpleType name="YesNoIfError">
375   <xsd:restriction base="xsd:string">
376     <xsd:enumeration value="Yes"/>
377     <xsd:enumeration value="No"/>
378     <xsd:enumeration value="IfError"/>
379   </xsd:restriction>
380 </xsd:simpleType>
```

381 *Schema 1 Request header*

382 **3.3 Response header**

383 The presence of a Response element in the header indicates that this is an answer to a request.

384 *SenderKey*: The request **MUST** specify the URI or key of the resource that originated the
385 response. This may be redundant with similar specifier in the transport layer.

386 *ReceiverKey* The request **MAY** specify the key of the resource that the response is being made
387 to. This may be redundant with similar specifier in the transport layer.

388 Note that the *ReceiverKey* is mandatory in a request and the *SenderKey* is mandatory in a
389 response. The purpose is to enforce keys upon ASAP resources without placing an unnecessary
390 burden on resources that are merely employing ASAP resources. For instance, a Java program
391 that instantiates an AWS may not know its own URL.

392 *RequestID*. If the original request had a *RequestID* tag, then the response must carry one with
393 that value in it. The requester can use this ID to correlate the response with the original request.

```
394 <?pseudo-xml?>
395 <env:Envelope xmlns:env="http://www.w3.org/2001/12/soap-envelope">
396   <env:Header>
397     <as:Response xmlns:aws="http://www.ASAP.info/spec/1.0/">
398       <as:SenderKey> The URI of the sender </as:SenderKey>
399       <as:ReceiverKey?> The URI of the receiver </as:ReceiverKey>
400       <as:RequestID?>
401         Unique ID for message correlation by the requestor
402       </as:RequestID>
403     </as:Response>
404   </env:Header>
405   <env:Body>
406     ...
407   </env:Body>
408 </env:Envelope>
```

409 *Example 2 Response header*

```
410 <xsd:element name="Response">
411   <xsd:complexType>
412     <xsd:sequence>
413       <xsd:element ref="SenderKey" minOccurs="0"/>
414       <xsd:element ref="ReceiverKey"/>
415       <xsd:element ref="RequestID" minOccurs="0"/>
416     </xsd:sequence>
417   </xsd:complexType>
418 </xsd:element>
```

419 Schema 2 Response header

420 3.4 Body

421 ASAP requires that there be one of the following elements within the body which represents the
422 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

423 Table 3 The ASAP message body elements

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

426

4 Instance resource

427

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.

428
429
430

431

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.

432

433

434

435

436

437

438

439

440

441

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.

442

443

444

445

446

447

448

449

450

451

452

4.1 Instance resource properties

453

Key: A URI that uniquely identifies this resource.

454

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.

455

456

457

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

458

459

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

460

461

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

462

463

FactoryKey: URI of the factory resource from which this instance was created.

464

Observers: A collection of URI's of registered observers of this process instance, if any exist.

465

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

466

467

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.

468

469

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

```
471 <?pseudo-xml?>
472 ...
473 <as:Key> URI </as:Key>
474 <as:State>open.notrunning</as:State>
475 <as:Name> string </as:Name>
476 <as:Subject> string </as:Subject>
477 <as:Description> string </as:Description>
478 <as:FactoryKey> URI </as:FactoryKey>
479 <as:Observers>
480   <as:ObserverKey>* URI </as:ObserverKey>
481 </as:Observers>
482 <as:ContextData>
483   <!-- extensible element -->
484 </as:ContextData>
485 <as:ResultData>
486   <!-- extensible element -->
487 </as:ResultData>
488 <as:History xlink:href="url"/>
489 ...
```

490 *Example 3 Instance resource properties*

```
491 <xsd:group name="instancePropertiesGroup">
492   <xsd:sequence>
493     <xsd:element name="Key" type="xsd:anyURI" />
494     <xsd:element name="State" type="stateType" />
495     <xsd:element name="Name" type="xsd:string" />
496     <xsd:element name="Subject" type="xsd:string" />
497     <xsd:element name="Description" type="xsd:string" />
498     <xsd:element name="FactoryKey" type="xsd:anyURI" />
499     <xsd:element name="Observers">
500       <xsd:complexType>
501         <xsd:sequence>
502           <xsd:element name="ObserverKey" type="xsd:anyURI"
503             maxOccurs="unbounded" />
504         </xsd:sequence>
505       </xsd:complexType>
506     </xsd:element>
507     <xsd:element name="ContextData">
508       <xsd:complexType>
509         <xsd:sequence>
510           <xsd:any namespace="##any" processContents="lax"
511             minOccurs="0" maxOccurs="unbounded" />
512         </xsd:sequence>
513       </xsd:complexType>
514     </xsd:element>
515     <xsd:element name="ResultData">
516       <xsd:complexType>
517         <xsd:sequence>
518           <xsd:any namespace="##any" processContents="lax"
519             minOccurs="0" maxOccurs="unbounded" />
520         </xsd:sequence>
521       </xsd:complexType>
522     </xsd:element>
523     <xsd:element name="History" type="historyType" />
524   </xsd:sequence>
525 </xsd:group>
526
527 <xsd:simpleType name="stateType">
528   <xsd:restriction base="xsd:string">
529     <xsd:enumeration value="open.notrunning" />
530     <xsd:enumeration value="open.notrunning.suspended" />
531     <xsd:enumeration value="open.running" />
532     <xsd:enumeration value="closed.completed" />
533     <xsd:enumeration value="closed.abnormalCompleted" />
534     <xsd:enumeration value="closed.abnormalCompleted.terminated" />
535     <xsd:enumeration value="closed.abnormalCompleted.aborted" />
536     <xsd:enumeration value="closed.abnormalCompleted.aborted" />
```

```

537 </xsd:restriction>
538 </xsd:simpleType>
539
540 <xsd:element name="Event">
541 <xsd:complexType>
542 <xsd:sequence>
543 <xsd:element name="Time" type="xsd:dateTime"/>
544 <xsd:element name="EventType">
545 <xsd:simpleType>
546 <xsd:restriction base="xsd:string">
547 <xsd:enumeration value="InstanceCreated"/>
548 <xsd:enumeration value="PropertiesSet"/>
549 <xsd:enumeration value="StateChanged"/>
550 <xsd:enumeration value="Subscribed"/>
551 <xsd:enumeration value="Unsubscribed"/>
552 <xsd:enumeration value="Error"/>
553 </xsd:restriction>
554 </xsd:simpleType>
555 </xsd:element>
556 <xsd:element name="SourceKey" type="xsd:anyURI"/>
557 <xsd:element name="Details" type="xsd:anyType"/>
558 <xsd:element name="OldState" type="as:stateType"/>
559 <xsd:element name="NewState" type="as:stateType"/>
560 </xsd:sequence>
561 </xsd:complexType>
562 </xsd:element>
563 <xsd:complexType name="historyType">
564 <xsd:sequence>
565 <xsd:element ref="Event" maxOccurs="unbounded"/>
566 </xsd:sequence>
567 </xsd:complexType>

```

568

569 *Schema 3 Instance resource properties*

570 **4.2 GetProperties**

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

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

```

573 <?pseudo-xml?>
574 <env:Envelope xmlns:env="http://www.w3.org/2001/12/soap-envelope">
575 <env:Header>
576 <as:Request...>
577 </env:Header>
578 <env:Body>
579 <as:GetPropertiesRq/>
580 </env:Body>
581 </env:Envelope>

```

582 *Example 4 Instance resource GetProperties method request*

```

583 <?pseudo-xml?>
584 <env:Envelope xmlns:env="http://www.w3.org/2001/12/soap-envelope">
585 <env:Header>
586 <as:Response...>
587 </env:Header>
588 <env:Body>
589 <as:GetPropertiesRs>
590 <!-- properties -->
591 </as:GetPropertiesRs>
592 </env:Body>
593 </env:Envelope>

```

594 *Example 5 Instance resource GetProperties method response*

```

595 <xsd:element name="GetPropertiesRq"/>
596 <xsd:element name="GetPropertiesRs" type="instancePropertiesGroup"/>

```

598 **4.3 SetProperties**

599 All resources implement SetProperties and allow as parameters all of the settable properties.
 600 This method can be used to set at least the displayable name, the description, or the priority of a
 601 process flow resource. This is an abstract interface, and the resources that implement this
 602 interface may have other properties that can be set in this manner. All of the parameters are
 603 optional, but to have any effect at least one of them must be present. This returns the complete
 604 info for the resource, just as the GetProperties method does, which will include any updated
 605 values.

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

```
610 <?pseudo-xml?>
611 <env:Envelope xmlns:env="http://www.w3.org/2001/12/soap-envelope">
612   <env:Header>
613     <as:Request...>
614   </env:Header>
615   <env:Body>
616     <as:SetPropertiesRq>
617       <as:Subject...?>
618       <as:Description...?>
619       <as:Priority...?>
620       <as>Data>
621         <!-- extensible element -->
622       </as>Data>
623     </as:SetPropertiesRq>
624   </env:Body>
625 </env:Envelope>
```

626 *Example 6 Instance resource SetProperties method request*

```
627 <?pseudo-xml?>
628 <env:Envelope xmlns:env="http://www.w3.org/2001/12/soap-envelope">
629   <env:Header>
630     <as:Response...>
631   </env:Header>
632   <env:Body>
633     <as:SetPropertiesRs...>
634       Returns the same response as GetProperties
635     </as:SetPropertiesRs>
636   </env:Body>
637 </env:Envelope>
```

638 *Example 7 Instance resource SetProperties method response*

```
639 <xsd:element name="SetPropertiesRq">
640   <xsd:complexType>
641     <xsd:sequence>
642       <xsd:element name="Subject" type="xsd:string"/>
643       <xsd:element name="Description" type="xsd:string"/>
644       <xsd:element name="Priority" type="xsd:string"/>
645       <xsd:element name="Data">
646         <xsd:complexType>
647           <xsd:sequence>
648             <xsd:any namespace="##any" processContents="lax"
649 minOccurs="0" maxOccurs="unbounded"/>
650           </xsd:sequence>
651         </xsd:complexType>
652       </xsd:element>
653     </xsd:sequence>
654   </xsd:complexType>
655 </xsd:element>
```



```
656 <xsd:element name="SetPropertiesRs" type="instancePropertiesGroup" />
```

657 *Schema 5 Instance resource SetProperties method*

658 4.4 Subscribe

659 To allow scalability, Instances will notify Observers when important events occur. Observers
660 must register their URI's with the Instance in order to be notified.

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

665 ObserverKey: URI to a resource that both implements the Observer Operation Group and will
666 receive the events

```
667 <?pseudo-xml?>  
668 <env:Envelope xmlns:env="http://www.w3.org/2001/12/soap-envelope">  
669 <env:Header>  
670 <as:Request...>  
671 </env:Header>  
672 <env:Body>  
673 <as:SubscribeRq>  
674 <as:ObserverKey> URI </as:ObserverKey>  
675 </as:SubscribeRq>  
676 </env:Body>  
677 </env:Envelope>
```

678 *Example 8 Instance resource Subscribe method request*

```
679 <?pseudo-xml?>  
680 <env:Envelope xmlns:env="http://www.w3.org/2001/12/soap-envelope">  
681 <env:Header>  
682 <as:Response...>  
683 </env:Header>  
684 <env:Body>  
685 <as:SubscribeRs/>  
686 </env:Body>  
687 </env:Envelope>
```

688 *Example 9 Instance resource Subscribe method response*

```
689 <xsd:element name="SubscribeRq">  
690 <xsd:complexType>  
691 <xsd:sequence>  
692 <xsd:element name="ObserverKey" type="xsd:anyURI" />  
693 </xsd:sequence>  
694 </xsd:complexType>  
695 </xsd:element>  
696 <xsd:element name="SubscribeRs" />
```

697 *Schema 6 Instance resource Subscribe method*

698 4.5 Unsubscribe

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

```
703 <?pseudo-xml?>  
704 <env:Envelope xmlns:env="http://www.w3.org/2001/12/soap-envelope">  
705 <env:Header>  
706 <as:Request...>
```

```

707
708 </env:Header>
709 <env:Body>
710 <as:UnsubscribeRq>
711 <as:ObserverKey> URI </as:ObserverKey>
712 </as:UnsubscribeRq>
713 </env:Body>
</env:Envelope>

```

714 *Example 10 Instance resource Unsubscribe method request*

```

715 <?pseudo-xml?>
716 <env:Envelope xmlns:env="http://www.w3.org/2001/12/soap-envelope">
717 <env:Header>
718 <as:Response...>
719 </env:Header>
720 <env:Body>
721 <as:UnsubscribeRs/>
722 </env:Body>
723 </env:Envelope>

```

724 *Example 11 Instance resource Unsubscribe method response*

```

725 <xsd:element name="UnsubscribeRq">
726 <xsd:complexType>
727 <xsd:sequence>
728 <xsd:element name="ObserverKey" type="xsd:anyURI"/>
729 </xsd:sequence>
730 </xsd:complexType>
731 </xsd:element>
732 <xsd:element name="UnsubscribeRs"/>

```

733 *Schema 7 Instance resource Unsubscribe method*

734 4.6 ChangeState

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

```

737 <?pseudo-xml?>
738 <env:Envelope xmlns:env="http://www.w3.org/2001/12/soap-envelope">
739 <env:Header>
740 <as:Request...>
741 </env:Header>
742 <env:Body>
743 <as:ChangeStateRq>
744 <as:State>the state requested</as:State>
745 </as:ChangeStateRq>
746 </env:Body>
747 </env:Envelope>

```

748 *Example 12 Instance resource ChangeState method request*

```

749 <?pseudo-xml?>
750 <env:Envelope xmlns:env="http://www.w3.org/2001/12/soap-envelope">
751 <env:Header>
752 <as:Response...>
753 </env:Header>
754 <env:Body>
755 <as:ChangeStateRs>
756 <as:State>the state</as:State>
757 </as:ChangeStateRs>
758 </env:Body>
759 </env:Envelope>

```

760 *Example 13 Instance resource ChangeState method response*

```

761 <xsd:element name="ChangeStateRq">
762 <xsd:complexType>
763 <xsd:sequence>

```

```
764     <xsd:element name="State" type="as:stateType" />
765   </xsd:sequence>
766 </xsd:complexType>
767 </xsd:element>
768 <xsd:element name="ChangeStateRs">
769   <xsd:complexType>
770     <xsd:sequence>
771       <xsd:element name="State" type="as:stateType" />
772     </xsd:sequence>
773   </xsd:complexType>
774 </xsd:element>
```

775 *Schema 8 Instance resource ChangeState method*

776

5 Factory resource

777

5.1 Factory resource properties

778

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

779

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

780

781

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.

782

783

784

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.

785

786

787

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.

788

789

790

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.

791

792

793

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 expressed as "P120D".

794

795

796

797

798

```
<?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>
...
```

799

800

801

802

803

804

805

806

807

808

809

810

811

812

813

814

815

Example 14 Factory resource properties

816

```
<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:sequence>
</xsd:group>
```

817

818

819

820

821

822

823

824

825

826

```

827 <xsd:complexType name="schemaType">
828 <xsd:any namespace="##other" />
829 <xsd:attribute name="href" type="xsd:anyURI" />
830 </xsd:complexType>
831 <xsd:complexType name="ContextDataType">
832 <xsd:sequence>
833 <xsd:any namespace="##other" />
834 </xsd:sequence>
835 </xsd:complexType>
836 <xsd:complexType name="ResultDataType">
837 <xsd:sequence>
838 <xsd:any namespace="##other" />
839 </xsd:sequence>
840 </xsd:complexType>

```

841 *Schema 9 Factory resource properties*

842 5.2 GetProperties

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

```

846 <?pseudo-xml?>
847 <env:Envelope xmlns:env="http://www.w3.org/2001/12/soap-envelope">
848 <env:Header>
849 <as:Request...>
850 </env:Header>
851 <env:Body>
852 <as:GetPropertiesRq/>
853 </env:Body>
854 </env:Envelope>

```

855 *Example 15 Factory resource GetProperties method request*

```

856 <?pseudo-xml?>
857 <env:Envelope xmlns:env="http://www.w3.org/2001/12/soap-envelope">
858 <env:Header>
859 <as:Response...>
860 </env:Header>
861 <env:Body>
862 <as:GetPropertiesRs>
863 <!-- properties -->
864 </as:GetPropertiesRs>
865 </env:Body>
866 </env:Envelope>

```

867 *Example 16 Factory resource GetProperties method response*

```

868 <xsd:element name="GetPropertiesRq" />
869 <xsd:element name="GetPropertiesRs" type="factoryPropertiesGroup" />

```

870 *Schema 10 Factory resource GetProperties method*

871

872 5.3 CreateInstance

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

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

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

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

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

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

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

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

```
891 <?pseudo-xml?>
892 <env:Envelope xmlns:env="http://www.w3.org/2001/12/soap-envelope">
893   <env:Header>
894     <as:Request...>
895   </env:Header>
896   <env:Body>
897     <as:CreateInstanceRq>
898       <as:StartImmediately>Yes|No</as:StartImmediately>
899       <as:ObserverKey>? URI </as:ObserverKey>
900       <as:Name>? string </as:Name>
901       <as:Subject>? string </as:Subject>
902       <as:Description>? string </as:Description>
903       <as:ContextData>
904         <!-- extensible element -->
905       </as:ContextData>
906     </as:CreateInstanceRq>
907   </env:Body>
908 </env:Envelope>
```

909 *Example 17 Factory resource CreateInstance method request*

```
910 <?pseudo-xml?>
911 <env:Envelope xmlns:env="http://www.w3.org/2001/12/soap-envelope">
912   <env:Header>
913     <as:Response>
914       <as:Key>URI of the process definition receiving this request</as:Key>
915     </as:Response>
916   </env:Header>
917   <env:Body>
918     <as:CreateInstanceRs>
919       <as:InstanceKey> URI </as:InstanceKey>
920     </as:CreateInstanceRs>
921   </env:Body>
922 </env:Envelope>
```

923 *Example 18 Factory resource CreateInstance method request*

```
924 <xsd:element name="CreateInstanceRq">
925   <xsd:complexType>
926     <xsd:sequence>
927       <xsd:element name="StartImmediately" type="xsd:boolean"/>
928       <xsd:element name="ObserverKey" type="xsd:anyURI" minOccurs="0"/>
929       <xsd:element name="Name" type="xsd:string" minOccurs="0"/>
930       <xsd:element name="Subject" type="xsd:string" minOccurs="0"/>
931       <xsd:element name="Description" type="xsd:string" minOccurs="0"/>
932       <xsd:element name="ContextData">
933         <xsd:complexType>
934           <xsd:sequence>
935             <xsd:any namespace="##any" processContents="lax"
936               minOccurs="0" maxOccurs="unbounded"/>
```

```

937         </xsd:sequence>
938     </xsd:complexType>
939 </xsd:element>
940 </xsd:sequence>
941 </xsd:complexType>
942 </xsd:element>
943 <xsd:element name="CreateInstanceRs">
944     <xsd:element name="InstanceKey" type="xsd:anyURI" />
945 </xsd:element>

```

946 *Schema 11 Factory resource CreateInstance method*

947 **5.4 ListInstances**

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

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

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

```

952 <?pseudo-xml?>
953 <env:Envelope xmlns:env="http://www.w3.org/2001/12/soap-envelope">
954     <env:Header>
955         <as:Request...>
956     </env:Header>
957     <env:Body>
958         <as>ListInstancesRq>
959             <as:Filter filterType="nmtoken"?>
960                 string
961             </as:Filter>
962         </as>ListInstancesRq>
963     </env:Body>
964 </env:Envelope>

```

965 *Example 19 Factory resource ListInstances method request*

```

966 <?pseudo-xml?>
967 <env:Envelope xmlns:env="http://www.w3.org/2001/12/soap-envelope">
968     <env:Header>
969         <as:Response...>
970     </env:Header>
971     <env:Body>
972         <as>ListInstancesRs>
973             <as:Instance>*
974                 <as:InstanceKey> URI </as:InstanceKey>
975                 <as>Name...?>
976                 <as:Subject...?>
977                 <as:Priority...?>
978             </as:Instance>
979         </as>ListInstancesRs>
980     </env:Body>
981 </env:Envelope>

```

982 *Example 20 Factory resource ListInstances method response*

```

983 <xsd:element name="ListInstancesRq">
984     <xsd:complexType>
985         <xsd:sequence>
986             <xsd:element name="Filter" type="FilterType">
987                 </xsd:element>
988         </xsd:sequence>
989     </xsd:complexType>
990 </xsd:element>
991 <xsd:complexType name="FilterType">
992     <xsd:simpleContent>
993         <xsd:extension base="xsd:string">
994             <xsd:attribute name="filterType" type="xsd:NMTOKEN" />
995         </xsd:extension>

```

```
996 </xsd:simpleContent>
997 </xsd:complexType>
998
999 <xsd:element name="ListInstancesRs">
1000 <xsd:complexType>
1001 <xsd:sequence>
1002 <xsd:element ref="Instance" maxOccurs="unbounded" minOccurs="0"/>
1003 </xsd:sequence>
1004 </xsd:complexType>
1005 </xsd:element>
1006
1007 <xsd:element name="Instance">
1008 <xsd:complexType>
1009 <xsd:sequence>
1010 <xsd:element name="InstanceKey" type="xsd:anyURI"/>
1011 <xsd:element name="Name" type="xsd:string" minOccurs="0"/>
1012 <xsd:element name="Subject" type="xsd:string" minOccurs="0"/>
1013 <xsd:element ref="Priority" type="xsd:int" minOccurs="0"/>
1014 </xsd:sequence>
1015 </xsd:complexType>
1016 </xsd:element>
```

1017 *Schema 12 Factory resource ListInstances method*

1018 6 Observer resource

1019 6.1 Observer resource properties

1020 The Observer resource can receive events about the state changes of a service instance. An
1021 observer is expected to have a Key.

1022 Key: a URI that uniquely identifies this resource. All resources must have a Key property.

```
1023 <xsd:element name="Key" type="xsd:anyURI" />
```

1024 *Schema 13 Observer resource properties*

1025 6.2 GetProperties

1026 This method is the same as it was with Instance and Factory resources.

```
1027 <xsd:element name="GetPropertiesRq" />  
1028 <xsd:element name="GetPropertiesRs" type="observerPropertiesGroup" />
```

1029 *Schema 14 Observer resource GetProperties method*

1030

1031 6.3 Completed

1032 The Completed method indicates that the Instance has completed the work. This is the 'normal'
1033 completion.

1034 This function signals to the observer resource that the started process is completed its task, and
1035 will no longer be processing. There is no guarantee that the resource will persist after this point
1036 in time.

1037 InstanceKey: The URI of a process that is performing this work

1038 ResultData: Context-specific data that represents the current values resulting from process
1039 execution. This information will be encoded as described in the section Process Context and
1040 Result Data above. If result data are not yet available, the ResultData element is returned empty.

```
1041 <?pseudo-xml?>  
1042 <env:Envelope xmlns:env="http://www.w3.org/2001/12/soap-envelope">  
1043   <env:Header>  
1044     <as:Request...>  
1045   </env:Header>  
1046   <env:Body>  
1047     <as:CompletedRq>  
1048       <as:InstanceKey> URI </as:Instance>  
1049       <as:ResultData>  
1050         <!-- extensible element -->  
1051       </as:ResultData>  
1052     </as:CompletedRq>  
1053   </env:Body>  
1054 </env:Envelope>
```

1055 *Example 21 Observer resource Completed method request*

```
1056 <?pseudo-xml?>  
1057 <env:Envelope xmlns:env="http://www.w3.org/2001/12/soap-envelope">  
1058   <env:Header>  
1059     <as:Response...>  
1060   </env:Header>
```

1061
1062
1063
1064

```
<env:Body>  
  <as:CompletedRs />  
</env:Body>  
</env:Envelope>
```

1065 *Example 22 Observer resource Completed method response*

1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081

```
<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" />
```

1082 *Schema 15 Observer resource Completed method*

1083 6.4 StateChanged

1084 Observers receive a StateChanged message from the Instance when the state of the Instance
1085 changes. The response to a notify event is not necessary. Typically, the header request tag will
1086 specify that no response is necessary.

1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098

```
<?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>
```

1099 *Example 23 Observer resource StateChanged method request*

1100
1101
1102
1103
1104
1105
1106
1107
1108

```
<?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>
```

1109 *Example 24 Observer resource StateChanged method response*

1110
1111
1112
1113
1114
1115
1116
1117
1118

```
<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" />
```

1119 *Schema 16 Observer resource StateChanged method*

1120 7 Data encoding

1121 7.1 Context data and result data

1122 The heart of an asynchronous service is the `ContextData` and the `ResultData`. The
1123 `ContextData` and the `ResultData` are the unique part of a particular service; everything else
1124 is boilerplate. The `ContextData` is the query or the initial request to the service. The
1125 `ContextData` dictates, determines or implies what the service instance should create. The
1126 `ResultData` is what the service eventually creates for the observers.

1127 The service factory should provide a schema for the `ContextData` element and `ResultData`
1128 element. The schema may be XML Schema or Relax NG. ASAP follows the SOAP and XML
1129 Schema data type encoding specifications.

1130 7.2 Extensibility

1131 Actual implementations of these resources may extend the set of properties returned. This
1132 document defines the required minimum set, as well as an optional set. Every implementation
1133 MUST return the required properties. The implementation may optionally return additional
1134 properties. Those additional properties should be elements of a namespace that is not ASAP.
1135 Use of extended properties must be carefully considered because this may limit the ability to
1136 interoperate with other systems. In general no system should be coded so as to require an
1137 extended attribute. Instead it should be able to function is the extended properties are missing.
1138 Future versions of this specification will cover the adoption of new properties to be considered
1139 part of the specification.

1140 7.3 State type

1141 The overall status of the asynchronous web service is defined by a state property value. This is a
1142 string value composed of words separated by periods. The status value must start with one of
1143 the seven defined values below, but the value can be extended by adding words on the end of
1144 the status separated by periods. The extension must be a refinement of one of the seven states
1145 defined here, such that it is not necessary to understand the extension. The intention is that these
1146 extensions may be proposed for future inclusion in the standard. The seven defined base states
1147 are:

1148 `open.notrunning`: A resource is in this state when it has been instantiated, but is not currently
1149 participating in the enactment of a work process.

1150 `open.notrunning.suspended`: A resource is in this state when it has initiated its participation in
1151 the enactment of a work process, but has been suspended. At this point, no resources contained
1152 within it may be started.

1153 `open.running`: A resource is in this state when it is performing its part in the normal execution of
1154 a work process.

1155 `closed.completed`: A resource is in this state when it has finished its task in the overall work
1156 process. All resources contained within it are assumed complete at this point.

1157 `closed.abnormalCompleted`: A resource is in this state when it has completed abnormally. At
1158 this point, the results for the completed tasks are returned.

1159 `closed.abnormalCompleted.terminated`: A resource is in this state when it has been terminated
1160 by the requesting resource before it completed its work process. At this point, all resources
1161 contained within it are assumed to be completed or terminated.

1162 closed.abnormalCompleted.aborted: A resource is in this state when the execution of its
1163 process has been abnormally ended before it completed its work process. At this point, no
1164 assumptions are made about the state of the resources contained within it.

```
1165 <xsd:simpleType name="stateType">  
1166 <xsd:restriction base="xsd:string">  
1167 <xsd:enumeration value="open.notrunning" />  
1168 <xsd:enumeration value="open.notrunning.suspended" />  
1169 <xsd:enumeration value="open.running" />  
1170 <xsd:enumeration value="closed.completed" />  
1171 <xsd:enumeration value="closed.abnormalCompleted" />  
1172 <xsd:enumeration value="closed.abnormalCompleted.terminated" />  
1173 <xsd:enumeration value="closed.abnormalCompleted.aborted" />  
1174 <xsd:enumeration value="closed.abnormalCompleted.aborted" />  
1175 </xsd:restriction>  
1176 </xsd:simpleType>
```

1178

1179 *Schema 17 stateType*

1180 These state values come from the Workflow Management Coalition standards.

1181 7.4 History type

1182 The history is optional. It contains a list of events. An event is a state change that can occur in the
1183 asynchronous service that is externally identifiable. Notifications can be sent to an observer in
1184 order to inform it of the particular event.

1185 Time: the date/time of the event that occurred

1186 EventType: One of an enumerated set of values to specify event types: InstanceCreated,
1187 PropertiesSet, StateChanged, Subscribed, Unsubscribed, Error. The event types correspond to
1188 the message types that the resource can receive.

1189 SourceKey: The URI of the resource that triggered this event, usually an observer resource but
1190 perhaps the instance resource itself.

1191 Details: A catchall element for containing any data appropriate.

1192 OldState: The state of the instance resource before this event occurred.

1193 NewState: The state of the instance resource before this event occurred.

```
1194 <xsd:element name="Event">  
1195 <xsd:complexType>  
1196 <xsd:sequence>  
1197 <xsd:element name="Time" type="xsd:dateTime" />  
1198 <xsd:element name="EventType" />  
1199 <xsd:simpleType>  
1200 <xsd:restriction base="xsd:string">  
1201 <xsd:enumeration value="InstanceCreated" />  
1202 <xsd:enumeration value="PropertiesSet" />  
1203 <xsd:enumeration value="StateChanged" />  
1204 <xsd:enumeration value="Subscribed" />  
1205 <xsd:enumeration value="Unsubscribed" />  
1206 <xsd:enumeration value="Error" />  
1207 </xsd:restriction>  
1208 </xsd:simpleType>  
1209 </xsd:element>  
1210 <xsd:element name="SourceKey" type="xsd:anyURI" />  
1211 <xsd:element name="Details" type="xsd:anyType" />  
1212 <xsd:element name="OldState" type="as:stateType" />  
1213 <xsd:element name="NewState" type="as:stateType" />  
1214 </xsd:sequence>  
1215 </xsd:complexType>
```

1216
1217
1218
1219
1220
1221

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

1222 *Schema 17 complexType*

1223 **7.5 Exceptions and error codes**

1224 All messages have the option of returning an exception. Exceptions are handled in the manner
1225 specified by SOAP 1.2. The header information should be the same, but in the body of the
1226 response, instead of having an ASAP element such as GetPropertiesRs or CreateInstanceRs,
1227 there will be the SOAP exception element env:Fault.

1228 Multi server transactions: ASAP does not include any way for multiple servers to participate in
1229 the same transactions. It will be up to individual systems to determine what happen if a ASAP
1230 request fails; In some cases it should be ignored, in some cases it should cause that transaction
1231 to fail, and in some cases the operation should be queued to repeat until it succeeds.

1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247

```
<?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>
```

1248 *Example 25 Exception*

1249 These error codes are chosen to be specific with the error codes defined by the Workflow
1250 Management Coalition Wf-MXL 1.1 specification.

1251	Header-specific	100 Series
1252	These exceptions deal with missing or invalid parameters in the header.	
1253	ASAP_PARSING_ERROR	101
1254	ASAP_ELEMENT_MISSING	102
1255	ASAP_INVALID_VERSION	103
1256	ASAP_INVALID_RESPONSE_REQUIRED_VALUE	104
1257	ASAP_INVALID_KEY	105
1258	ASAP_INVALID_OPERATION_SPECIFICATION	106
1259	ASAP_INVALID_REQUEST_ID	107
1260		
1261	Data	200 Series
1262	These exceptions deal with incorrect context or result data	
1263	ASAP_INVALID_CONTEXT_DATA	201
1264	ASAP_INVALID_RESULT_DATA	202
1265	ASAP_INVALID_RESULT_DATA_SET	203
1266		
1267	Authorization	300 Series
1268	A user may not be authorized to carry out this operation on a particular resource, e.g., may	
1269	not create a process instance for that process definition.	

1270	ASAP_NO_AUTHORIZATION	301
1271		
1272	Operation	400 Series
1273	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.	
1274		
1275	ASAP_OPERATION_FAILED	401
1276		
1277		
1278	Resource Access	500 Series
1279	A valid Key has been used, however this operation cannot currently be invoked on the specified resource.	
1280		
1281	ASAP_NO_ACCESS_TO_RESOURCE	501
1282	ASAP_INVALID_FACTORY	502
1283	ASAP_MISSING_INSTANCE_KEY	503
1284	ASAP_INVALID_INSTANCE_KEY	504
1285		
1286	Operation-specific	600 Series
1287	These are the more operation specific exceptions. Typically, they are only used in a few operations, possibly a single one.	
1288		
1289	ASAP_INVALID_STATE_TRANSITION	601
1290	ASAP_INVALID_OBSERVER_FOR_RESOURCE	602
1291	ASAP_MISSING_NOTIFICATION_NAME	603
1292	ASAP_INVALID_NOTIFICATION_NAME	604
1293	ASAP_HISTORY_NOT_AVAILABLE	605
1294		

1295 7.6 Language

1296 ASAP messages should indicate their preferred language using the xml:lang attribute either in the
1297 SOAP Envelope element (the root element) or in the ASAP Request or Response element.

1298 7.7 Security

1299 HTTP provides for both authenticated as well as anonymous requests. Because of the nature of
1300 process flow in controlling access to resources, many operations will not be allowed unless
1301 accompanied by a valid and authenticated user ID. There are two primary means that this will be
1302 provided: HTTP authorization header or transport level encryption such as SSL.

1303 The first and most common method of authentication over HTTP is through the use of the
1304 Authorization header. This header carries a user name and a password that can be used to
1305 validate against a user directory. If the request is attempted but the authentication of the user
1306 fails, or the Authorization header field is not present, then the standard HTTP error "401
1307 Unauthorized" is the response. Within this, there are two authentication schemes:

- 1308 • Basic involves carrying the name and password in the authorization field and is not
1309 considered secure.
- 1310 • A digest authentication for HTTP is specified in IETF RFC-2069
1311 [<http://ietf.org/rfc/rfc2069.html>], which offers a way to securely authenticate without sending
1312 the password in the clear.

1313 Second, encryption at the transport level, such as SSL, can provide certificate based
1314 authentication of the user making the request. This is much more secure than the previous
1315 option, and should be used when high security is warranted.

1316 Because the lower protocol levels are providing the user ID, ASAP does not specify how to send
1317 the client user ID. The authenticated user ID can be assumed to be present in the server at the
1318 time of handling the request.

1319 Note that since most ASAP interactions are between programs that we would normally consider
1320 to be servers (i.e. process flow engine to process flow engine) the conclusion can be made that
1321 all such process flow engines will need a user id and associated values (e.g. password or
1322 certificate) necessary to authenticate themselves to other servers. Servers must be configured
1323 with the appropriate safeguards to assure that these associated values are protected from view.
1324 Under no circumstances should a set of process flow engines be configured to make anonymous
1325 ASAP requests that update information since the only way to be sure that the request is coming
1326 from a trustable source is through the authentication.

1327 With the authentication requirements above, of either HTTP authorization header field or SSL
1328 secure transport, ASAP should be able to protect and safeguard sensitive data while allowing
1329 interoperability to and from any part of the Internet.

1330

8 References

1331

8.1 Normative

1332

[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.

1333

1334

[SOAP]

Simple Object Access Protocol

1335

[W3C Arch]

Web Services Architecture Working Group,

1336

<http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/>

1337

[XSD]

XML Schema Part 1 & Part 2

1338

<http://www.w3.org/TR/xmlschema-1/> **and**

1339

<http://www.w3.org/TR/xmlschema-2/>

1340

[RAYMOND]

The Art of Unix Programming by Eric S. Raymond, Addison Wesley
Publishers

1341

1342

1343

1344

Appendix A. Schema

```
1345 <?xml version="1.0"?>
1346 <xsd:schema
1347   targetNamespace="http://www.oasisopen.org/asap/0.9/asap.xsd"
1348   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
1349   xmlns:as="http://www.oasis-open.org/asap/0.9/asap.xsd",
1350   elementFormDefault="qualified">
1351
1352 <xsd:annotation>
1353   <xsd:documentation xml:lang="en">
1354     Asynchronous Service Access Protocol
1355
1356     Jeffrey Ricker
1357     DRAFT
1358     Original 2002.02.18
1359     Revised 2003.11.04
1360
1361     Revised 2004.02.26, John Fuller
1362     Edited to match up with wd-asap-spec-01d.doc
1363
1364     Revised 2004.04.14, Sameer Pradhan
1365     Edited to allow optional elements to be absent.
1366
1367     Revised 2004.06.04 Mayilraj Krishnan
1368     Edited for Context and Result Data and Indent the schema
1369
1370   </xsd:documentation>
1371 </xsd:annotation>
1372
1373 <!-- ===== simple property elements ===== -->
1374
1375 <xsd:complexType name="schemaType">
1376   <xsd:sequence>
1377     <xsd:any namespace="##other"/>
1378   </xsd:sequence>
1379   <xsd:attribute name="href" type="xsd:anyURI"/>
1380 </xsd:complexType>
1381
1382 <xsd:complexType name="ContextDataType">
1383   <xsd:sequence>
1384     <xsd:any namespace="##other"/>
1385   </xsd:sequence>
1386 </xsd:complexType>
1387
1388 <xsd:complexType name="ResultDataType">
1389   <xsd:sequence>
1390     <xsd:any namespace="##other"/>
1391   </xsd:sequence>
1392 </xsd:complexType>
1393
1394 <xsd:simpleType name="stateType">
1395   <xsd:restriction base="xsd:string">
1396     <xsd:enumeration value="open.notrunning"/>
```

```

1397     <xsd:enumeration value="open.notrunning.suspended" />
1398     <xsd:enumeration value="open.running" />
1399     <xsd:enumeration value="closed.completed" />
1400     <xsd:enumeration value="closed.abnormalCompleted" />
1401     <xsd:enumeration value="closed.abnormalCompleted.terminated" />
1402     <xsd:enumeration value="closed.abnormalCompleted.aborted" />
1403   </xsd:restriction>
1404 </xsd:simpleType>
1405
1406 <xsd:element name="Event">
1407   <xsd:complexType>
1408     <xsd:sequence>
1409       <xsd:element name="Time" type="xsd:dateTime" />
1410       <xsd:element name="EventType">
1411         <xsd:simpleType>
1412           <xsd:restriction base="xsd:string">
1413             <xsd:enumeration value="InstanceCreated" />
1414             <xsd:enumeration value="PropertiesSet" />
1415             <xsd:enumeration value="StateChanged" />
1416             <xsd:enumeration value="Subscribed" />
1417             <xsd:enumeration value="Unsubscribed" />
1418             <xsd:enumeration value="Error" />
1419           </xsd:restriction>
1420         </xsd:simpleType>
1421       </xsd:element>
1422       <xsd:element name="SourceKey" type="xsd:anyURI" />
1423       <xsd:element name="Details" type="xsd:anyType" />
1424       <xsd:element name="OldState" type="as:stateType" />
1425       <xsd:element name="NewState" type="as:stateType" />
1426     </xsd:sequence>
1427   </xsd:complexType>
1428 </xsd:element>
1429
1430 <xsd:complexType name="historyType">
1431   <xsd:sequence>
1432     <xsd:element ref="as:Event" maxOccurs="unbounded" />
1433   </xsd:sequence>
1434 </xsd:complexType>
1435
1436 <xsd:simpleType name="YesNoIfError">
1437   <xsd:restriction base="xsd:string">
1438     <xsd:enumeration value="Yes" />
1439     <xsd:enumeration value="No" />
1440     <xsd:enumeration value="IfError" />
1441   </xsd:restriction>
1442 </xsd:simpleType>
1443
1444 <!-- ===== headers ===== -->
1445
1446 <xsd:element name="Request">
1447   <xsd:complexType>
1448     <xsd:sequence>
1449       <xsd:element name="SenderKey" type="xsd:anyURI"
1450         minOccurs="0" />
1451       <xsd:element name="ReceiverKey" type="xsd:anyURI" />
1452

```

```

1453         <xsd:element name="ResponseRequired" type="as:YesNoIfError"
1454             minOccurs="0"/>
1455         <xsd:element name="RequestID" type="xsd:anyURI"
1456             minOccurs="0"/>
1457     </xsd:sequence>
1458 </xsd:complexType>
1459 </xsd:element>
1460
1461 <xsd:element name="Response">
1462     <xsd:complexType>
1463         <xsd:sequence>
1464             <xsd:element name="SenderKey" type="xsd:anyURI"/>
1465             <xsd:element name="ReceiverKey" type="xsd:anyURI"
1466                 minOccurs="0"/>
1467             <xsd:element name="RequestID" type="xsd:anyURI"
1468                 minOccurs="0"/>
1469         </xsd:sequence>
1470     </xsd:complexType>
1471 </xsd:element>
1472
1473 <!-- ===== properties ===== -->
1474
1475 <xsd:group name="instancePropertiesGroup">
1476     <xsd:sequence>
1477         <xsd:element name="Key" type="xsd:anyURI"/>
1478         <xsd:element name="State" type="as:stateType"/>
1479         <xsd:element name="Name" type="xsd:string"/>
1480         <xsd:element name="Subject" type="xsd:string"/>
1481         <xsd:element name="Description" type="xsd:string"/>
1482         <xsd:element name="FactoryKey" type="xsd:anyURI"/>
1483         <xsd:element name="Observers">
1484             <xsd:complexType>
1485                 <xsd:sequence>
1486                     <xsd:element name="ObserverKey" type="xsd:anyURI"
1487                         maxOccurs="unbounded" minOccurs="0"/>
1488                 </xsd:sequence>
1489             </xsd:complexType>
1490         </xsd:element>
1491         <xsd:element name="ContextData">
1492             <xsd:complexType>
1493                 <xsd:sequence>
1494                     <xsd:any namespace="##any" processContents="lax"
1495                         minOccurs="0" maxOccurs="unbounded"/>
1496                 </xsd:sequence>
1497             </xsd:complexType>
1498         </xsd:element>
1499         <xsd:element name="ResultData">
1500             <xsd:complexType>
1501                 <xsd:sequence>
1502                     <xsd:any namespace="##any" processContents="lax"
1503                         minOccurs="0" maxOccurs="unbounded"/>
1504                 </xsd:sequence>
1505             </xsd:complexType>
1506         </xsd:element>
1507         <xsd:element name="History" type="as:historyType"/>
1508     </xsd:sequence>
1509 </xsd:group>

```

```

1510
1511 <xsd:group name="factoryPropertiesGroup">
1512   <xsd:sequence>
1513     <xsd:element name="Key" type="xsd:anyURI"/>
1514     <xsd:element name="Name" type="xsd:string"/>
1515     <xsd:element name="Subject" type="xsd:string"/>
1516     <xsd:element name="Description" type="xsd:string"/>
1517     <xsd:element name="ContextDataSchema" type="as:ContextDataType"/>
1518     <xsd:element name="ResultDataSchema" type="as:ResultDataType"/>
1519     <xsd:element name="Expiration" type="xsd:duration"/>
1520   </xsd:sequence>
1521 </xsd:group>
1522
1523 <xsd:group name="observerPropertiesGroup">
1524   <xsd:sequence>
1525     <xsd:element name="Key" type="xsd:anyURI"/>
1526   </xsd:sequence>
1527 </xsd:group>
1528
1529 <!-- ===== messages ===== -->
1530
1531 <xsd:element name="GetPropertiesRq"/>
1532
1533 <xsd:element name="GetPropertiesRs">
1534   <xsd:complexType>
1535     <xsd:choice>
1536       <xsd:group ref="as:instancePropertiesGroup"/>
1537       <xsd:group ref="as:factoryPropertiesGroup"/>
1538       <xsd:group ref="as:observerPropertiesGroup"/>
1539     </xsd:choice>
1540   </xsd:complexType>
1541 </xsd:element>
1542
1543 <xsd:element name="SetPropertiesRq">
1544   <xsd:complexType>
1545     <xsd:sequence>
1546       <xsd:element name="Subject" type="xsd:string"/>
1547       <xsd:element name="Description" type="xsd:string"/>
1548       <xsd:element name="Priority" type="xsd:int"/>
1549       <xsd:element name="Data">
1550         <xsd:complexType>
1551           <xsd:sequence>
1552             <xsd:any namespace="##any" processContents="lax"
1553               minOccurs="0" maxOccurs="unbounded"/>
1554           </xsd:sequence>
1555         </xsd:complexType>
1556       </xsd:element>
1557     </xsd:sequence>
1558   </xsd:complexType>
1559 </xsd:element>
1560
1561 <xsd:element name="SetPropertiesRs">
1562   <xsd:complexType>
1563     <xsd:choice>
1564       <xsd:group ref="as:instancePropertiesGroup"/>
1565       <xsd:group ref="as:factoryPropertiesGroup"/>
1566       <xsd:group ref="as:observerPropertiesGroup"/>

```

```

1567     </xsd:choice>
1568 </xsd:complexType>
1569 </xsd:element>
1570
1571 <xsd:element name="SubscribeRq">
1572   <xsd:complexType>
1573     <xsd:sequence>
1574       <xsd:element name="ObserverKey" type="xsd:anyURI" />
1575     </xsd:sequence>
1576   </xsd:complexType>
1577 </xsd:element>
1578
1579 <xsd:element name="SubscribeRs" />
1580
1581 <xsd:element name="UnsubscribeRq">
1582   <xsd:complexType>
1583     <xsd:sequence>
1584       <xsd:element name="ObserverKey" type="xsd:anyURI" />
1585     </xsd:sequence>
1586   </xsd:complexType>
1587 </xsd:element>
1588
1589 <xsd:element name="UnsubscribeRs" />
1590
1591 <xsd:element name="CreateInstanceRq">
1592   <xsd:complexType>
1593     <xsd:sequence>
1594       <xsd:element name="StartImmediately" type="xsd:boolean" />
1595       <xsd:element name="ObserverKey" type="xsd:anyURI"
1596         minOccurs="0" />
1597       <xsd:element name="Name" type="xsd:string" minOccurs="0" />
1598       <xsd:element name="Subject" type="xsd:string" minOccurs="0" />
1599       <xsd:element name="Description" type="xsd:string"
1600         minOccurs="0" />
1601       <xsd:element name="ContextData">
1602         <xsd:complexType>
1603           <xsd:sequence>
1604             <xsd:any namespace="##any" processContents="lax"
1605               minOccurs="0" maxOccurs="unbounded" />
1606           </xsd:sequence>
1607         </xsd:complexType>
1608       </xsd:element>
1609     </xsd:sequence>
1610   </xsd:complexType>
1611 </xsd:element>
1612
1613 <xsd:element name="CreateInstanceRs">
1614   <xsd:complexType>
1615     <xsd:sequence>
1616       <xsd:element name="InstanceKey" type="xsd:anyURI" />
1617     </xsd:sequence>
1618   </xsd:complexType>
1619 </xsd:element>
1620
1621 <xsd:complexType name="FilterType">
1622   <xsd:simpleContent>
1623     <xsd:extension base="xsd:string">

```

```

1624         <xsd:attribute name="filterType" type="xsd:NMTOKEN"/>
1625     </xsd:extension>
1626 </xsd:simpleContent>
1627 </xsd:complexType>
1628
1629 <xsd:element name="ListInstancesRq">
1630     <xsd:complexType>
1631         <xsd:sequence>
1632             <xsd:element name="Filter" type="as:FilterType">
1633                 </xsd:element>
1634             </xsd:sequence>
1635         </xsd:complexType>
1636 </xsd:element>
1637
1638 <xsd:element name="Instance">
1639     <xsd:complexType>
1640         <xsd:sequence>
1641             <xsd:element name="InstanceKey" type="xsd:anyURI"/>
1642             <xsd:element name="Name" type="xsd:string" minOccurs="0"/>
1643             <xsd:element name="Subject" type="xsd:string" minOccurs="0"/>
1644             <xsd:element name="Priority" type="xsd:int" minOccurs="0"/>
1645         </xsd:sequence>
1646     </xsd:complexType>
1647 </xsd:element>
1648
1649 <xsd:element name="ListInstancesRs">
1650     <xsd:complexType>
1651         <xsd:sequence>
1652             <xsd:element ref="as:Instance" maxOccurs="unbounded"
1653                 minOccurs="0"/>
1654         </xsd:sequence>
1655     </xsd:complexType>
1656 </xsd:element>
1657
1658 <xsd:element name="CompletedRq">
1659     <xsd:complexType>
1660         <xsd:sequence>
1661             <xsd:element name="InstanceKey" type="xsd:anyURI"/>
1662             <xsd:element name="ResultData">
1663                 <xsd:complexType>
1664                     <xsd:sequence>
1665                         <xsd:any namespace="##any" processContents="lax"
1666                             minOccurs="0" maxOccurs="unbounded"/>
1667                     </xsd:sequence>
1668                 </xsd:complexType>
1669             </xsd:element>
1670         </xsd:sequence>
1671     </xsd:complexType>
1672 </xsd:element>
1673
1674 <xsd:element name="CompletedRs"/>
1675
1676 <xsd:element name="ChangeStateRq">
1677     <xsd:complexType>
1678         <xsd:sequence>
1679             <xsd:element name="State" type="as:stateType"/>
1680         </xsd:sequence>

```

```
1681     </xsd:complexType>
1682 </xsd:element>
1683
1684 <xsd:element name="ChangeStateRs">
1685     <xsd:complexType>
1686         <xsd:sequence>
1687             <xsd:element name="State" type="as:stateType"/>
1688         </xsd:sequence>
1689     </xsd:complexType>
1690 </xsd:element>
1691
1692 <xsd:element name="StateChangedRq">
1693     <xsd:complexType>
1694         <xsd:sequence>
1695             <xsd:element name="State" type="as:stateType"/>
1696             <xsd:element name="PreviousState" type="as:stateType"/>
1697         </xsd:sequence>
1698     </xsd:complexType>
1699 </xsd:element>
1700
1701 <xsd:element name="StateChangedRs"/>
1702
1703 </xsd:schema>
1704
1705
```


1706

Appendix B. Acknowledgments

1707 The following individuals were members of the committee during the development of this
1708 specification:

- 1709 • Jeffrey Ricker
- 1710 • Keith Swenson, Fujitsu
- 1711 • Moshe Silverstein, iWay Software
- 1712 • John Fuller, for EasyASAP
- 1713 • Jeff Cohen, for .Net ASAP

1714 A number of people have participated in the development of this document and the related ideas
1715 that come largely from earlier work:

- 1716 • Mike Marin, FileNET
- 1717 • Edwin Kodhabakchien, Collaxa Inc.
- 1718 • Dave Hollingsworth, ICL/Fujitsu
- 1719 • Marc-Thomas Schmidt, IBM
- 1720 • Greg Bolcer, Endeavors Technology, Inc
- 1721 • Dan Matheson, CoCreate
- 1722 • George Buzsaki and Surrendra Reddy, Oracle Corp.
- 1723 • Larry Masinter, Xerox PARC
- 1724 • Martin Adder
- 1725 • Mark Fisher, Thomson
- 1726 • David Jakopac and David Hurst, Lisle Technology Partners
- 1727 • Kevin Mitchell
- 1728 • Paul Lyman, United Technologies
- 1729 • Ian Prittie
- 1730 • Members of the Workflow Management Coalition
- 1731 • And many others....

1732

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

1733

1734

Appendix D. Notices

1735 OASIS takes no position regarding the validity or scope of any intellectual property or other rights
1736 that might be claimed to pertain to the implementation or use of the technology described in this
1737 document or the extent to which any license under such rights might or might not be available;
1738 neither does it represent that it has made any effort to identify any such rights. Information on
1739 OASIS's procedures with respect to rights in OASIS specifications can be found at the OASIS
1740 website. Copies of claims of rights made available for publication and any assurances of licenses
1741 to be made available, or the result of an attempt made to obtain a general license or permission
1742 for the use of such proprietary rights by implementors or users of this specification, can be
1743 obtained from the OASIS Executive Director.

1744 OASIS invites any interested party to bring to its attention any copyrights, patents or patent
1745 applications, or other proprietary rights which may cover technology that may be required to
1746 implement this specification. Please address the information to the OASIS Executive Director.

1747 **Copyright © OASIS Open 2003. All Rights Reserved.**

1748 This document and translations of it may be copied and furnished to others, and derivative works
1749 that comment on or otherwise explain it or assist in its implementation may be prepared, copied,
1750 published and distributed, in whole or in part, without restriction of any kind, provided that the
1751 above copyright notice and this paragraph are included on all such copies and derivative works.
1752 However, this document itself does not be modified in any way, such as by removing the
1753 copyright notice or references to OASIS, except as needed for the purpose of developing OASIS
1754 specifications, in which case the procedures for copyrights defined in the OASIS Intellectual
1755 Property Rights document must be followed, or as required to translate it into languages other
1756 than English.

1757 The limited permissions granted above are perpetual and will not be revoked by OASIS or its
1758 successors or assigns.

1759 This document and the information contained herein is provided on an "AS IS" basis and OASIS
1760 DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO
1761 ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE
1762 ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A
1763 PARTICULAR PURPOSE.