



Web Services Atomic Transaction 1.1 (WS-AtomicTransaction)

Working Draft, November 23, 2005

Document Identifier:

wstx-wsat-1.1-spec-wd-01

Location:

<http://docs.oasis-open.org/wstx/wsat-1.1-spec-wd-01.doc>

Technical Committee:

OASIS WS-Tx TC

Chair(s):

Eric Newcomer, Iona
Ian Robinson, IBM

Editor(s):

Mark Little, Arjuna Technologies Ltd. <mark.little@arjuna.com>
Andrew Wilkinson, IBM <awilkinson@uk.ibm.com>

Abstract:

This specification provides the definition of the atomic transaction coordination type that is to be used with the extensible coordination framework described in the WS-Coordination specification. The specification defines three specific agreement coordination protocols for the atomic transaction coordination type: completion, volatile two-phase commit, and durable two-phase commit. Developers can use any or all of these protocols when building applications that require consistent agreement on the outcome of short-lived distributed activities that have the all-or-nothing property.

Status:

This document is published by the WS-Tx TX as a "working draft".

This document was last revised or approved by the WS-Tx TC on the above date. The level of approval is also listed above. Check the current location noted above for possible later revisions of this document. This document is updated periodically on no particular schedule.

Technical Committee members should send comments on this specification to the Technical Committee's email list. Others should send comments to the Technical Committee by using the "Send A Comment" button on the Technical Committee's web page at www.oasis-open.org/committees/ws-tx.

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 Technical Committee web page (www.oasis-open.org/committees/ws-tx/ipr.php).

The non-normative errata page for this specification is located at www.oasis-open.org/committees/ws-tx.

Notices

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

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

Copyright © OASIS Open 2005. *All Rights Reserved.*

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

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

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

Table of contents

1	Note on terminology.....	4
1.1	Composable Architecture	4
1.2	Namespace.....	4
1.2.1	Prefix Namespace.....	4
1.3	XSD and WSDL Files	4
1.4	AT Protocol Elements.....	5
2	Introduction.....	6
3	Atomic Transaction Context	7
4	Atomic Transaction Protocols.....	8
4.1	Preconditions	8
4.2	Completion Protocol.....	8
4.3	Two-Phase Commit Protocol.....	9
4.3.1	Volatile Two-Phase Commit Protocol	9
4.3.2	Durable Two-Phase Commit Protocol.....	10
4.3.3	2PC Diagram and Notifications.....	10
5	AT Policy Assertion.....	12
5.1	Assertion Model	12
5.2	Normative Outline	12
5.3	Assertion Attachment	13
5.4	Assertion Example	13
6	Transaction Faults	15
6.1	InconsistentInternalState.....	16
7	Security Model.....	17
8	Security Considerations	19
9	Use of WS-Addressing Headers.....	21
10	References	23
11	State Tables	25
	Appendix A. Acknowledgements	28
	Appendix B. Revision History	29

1 Note on terminology

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC2119 [KEYWORDS].

Namespace URIs of the general form `http://example.org` and `http://example.com` represents some application-dependent or context-dependent URI as defined in RFC 2396 [URI].

1.1 Composable Architecture

By using the SOAP [SOAP] and WSDL [WSDL] extensibility model, SOAP-based and WSDL-based specifications are designed to work together to define a rich Web services environment. As such, WS-AtomicTransaction by itself does not define all features required for a complete solution. WS-AtomicTransaction is a building block used with other specifications of Web services (e.g., WS-Coordination, WS-Security) and application-specific protocols that are able to accommodate a wide variety of coordination protocols related to the coordination actions of distributed applications.

1.2 Namespace

The XML namespace URI that MUST be used by implementations of this specification is:

```
http://schemas.xmlsoap.org/ws/2004/10/wsat
```

This is also used as the CoordinationContext type for atomic transactions.

1.2.1 Prefix Namespace

Prefix	Namespace
S	<code>http://www.w3.org/2003/05/soap-envelope</code>
wscor	<code>http://schemas.xmlsoap.org/ws/2004/10/wscor</code>
wsat	<code>http://schemas.xmlsoap.org/ws/2004/10/wsat</code>

If an action URI is used then the action URI MUST consist of the wsat namespace URI concatenated with the "/" character and the element name. For example:

```
http://schemas.xmlsoap.org/ws/2004/10/wsat/Commit
```

1.3 XSD and WSDL Files

The following links hold the XML schema and the WSDL declarations defined in this document.

`http://schemas.xmlsoap.org/ws/2004/10/wsat/wsat.xsd`

`http://schemas.xmlsoap.org/ws/2004/10/wsat/wsat.wsdl`

Soap bindings for the WSDL documents defined in this specification MUST use "document" for the *style* attribute.

28 **1.4 AT Protocol Elements**

29 The protocol elements define various extensibility points that allow other child or attribute content.
30 Additional children and/or attributes MAY be added at the indicated extension points but MUST NOT
31 contradict the semantics of the parent and/or owner, respectively. If a receiver does not recognize an
32 extension, the receiver SHOULD ignore the extension.

33

2 Introduction

34 The current set of Web service specifications [WSDL] [SOAP] defines protocols for Web service
35 interoperability. Web services increasingly tie together a number of participants forming large distributed
36 applications. The resulting activities may have complex structure and relationships.

37 The WS-Coordination specification defines an extensible framework for defining coordination types. This
38 specification provides the definition of an atomic transaction coordination type used to coordinate
39 activities having an "all or nothing" property. Atomic transactions commonly require a high level of trust
40 between participants and are short in duration. The Atomic Transaction specification defines protocols
41 that enable existing transaction processing systems to wrap their proprietary protocols and interoperate
42 across different hardware and software vendors.

43 To understand the protocol described in this specification, the following assumptions are made:

44 The reader is familiar with existing standards for two-phase commit protocols and with
45 commercially available implementations of such protocols. Therefore this section includes only
46 those details that are essential to understanding the protocols described.

47 The reader is familiar with the WS-Coordination [WSCOOR] specification that defines the
48 framework for the WS-AtomicTransaction coordination protocols.

49 The reader is familiar with WS-Addressing [WSADDR] and WS-Policy [WSPOLICY].

50 Atomic transactions have an all-or-nothing property. The actions taken prior to commit are only tentative
51 (i.e., not persistent and not visible to other activities). When an application finishes, it requests the
52 coordinator to determine the outcome for the transaction. The coordinator determines if there were any
53 processing failures by asking the participants to vote. If the participants all vote that they were able to
54 execute successfully, the coordinator commits all actions taken. If a participant votes that it needs to
55 abort or a participant does not respond at all, the coordinator aborts all actions taken. Commit makes the
56 tentative actions visible to other transactions. Abort makes the tentative actions appear as if the actions
57 never happened. Atomic transactions have proven to be extremely valuable for many applications. They
58 provide consistent failure and recovery semantics, so the applications no longer need to deal with the
59 mechanics of determining a mutually agreed outcome decision or to figure out how to recover from a
60 large number of possible inconsistent states.

61 Atomic Transaction defines protocols that govern the outcome of atomic transactions. It is expected that
62 existing transaction processing systems wrap their proprietary mechanisms and interoperate across
63 different vendor implementations.

64 3 Atomic Transaction Context

65 Atomic Transaction builds on WS-Coordination, which defines an activation and a registration service.
66 Example message flows and a complete description of creating and registering for coordinated activities
67 is found in the WS-Coordination specification [WSCOOR].

68 The Atomic Transaction coordination context must flow on all application messages involved with the
69 transaction.

70 Atomic Transaction adds the following semantics to the CreateCoordinationContext operation on the
71 activation service.

72 If the request includes the CurrentContext element, the target coordinator is interposed as a
73 subordinate to the coordinator stipulated inside the CurrentContext element.

74 If the request does not include a CurrentContext element, the target coordinator creates a new
75 transaction and acts as the root.

76 A coordination context may have an Expires attribute. This attribute specifies the earliest point in time at
77 which a transaction may be terminated solely due to its length of operation. From that point forward, the
78 transaction manager may elect to unilaterally roll back the transaction, so long as it has not transmitted a
79 Commit or a Prepared notification.

80 The Atomic Transaction protocol is identified by the following coordination type:

81 `http://schemas.xmlsoap.org/ws/2004/10/wsat`

82 4 Atomic Transaction Protocols

83 This specification defines the following protocols for atomic transactions.

84 **Completion:** The completion protocol initiates commitment processing. Based on each
85 protocol's registered participants, the coordinator begins with Volatile 2PC then proceeds through
86 Durable 2PC. The final result is signaled to the initiator.

87 **Two-Phase Commit (2PC):** The 2PC protocol coordinates registered participants to reach a
88 commit or abort decision, and ensures that all participants are informed of the final result. The
89 2PC protocol has two variants:

- 90 ○ **Volatile 2PC:** Participants managing volatile resources such as a cache should
91 register for this protocol.
- 92 ○ **Durable 2PC:** Participants managing durable resources such as a database should
93 register for this protocol.

94 A participant can register for more than one of these protocols by sending multiple Register messages.

95 4.1 Preconditions

96 The correct operation of the protocols requires that a number of preconditions **MUST** be established prior
97 to the processing:

- 98 1. The source **MUST** have knowledge of the destination's policies, if any, and the source **MUST** be
99 capable of formulating messages that adhere to this policy.
- 100 2. If a secure exchange of messages is required, then the source and destination **MUST** have a
101 security context.

102 4.2 Completion Protocol

103 The Completion protocol is used by an application to tell the coordinator to either try to commit or abort an
104 atomic transaction. After the transaction has completed, a status is returned to the application.

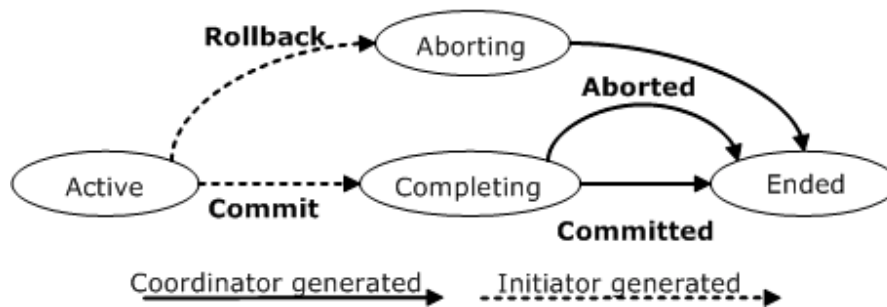
105 An initiator registers for this protocol using the following protocol identifier:

```
106 http://schemas.xmlsoap.org/ws/2004/10/wsat/Completion
```

107

108 The diagram below illustrates the protocol abstractly:

109



110
111

112 The coordinator accepts:

113 Commit

114 Upon receipt of this notification, the coordinator knows that the participant has completed
115 application processing and that it should attempt to commit the transaction.

116 Rollback

117 Upon receipt of this notification, the coordinator knows that the participant has terminated
118 application processing and that it should abort the transaction.

119 The initiator accepts:

120 Committed

121 Upon receipt of this notification, the initiator knows that the coordinator reached a decision to
122 commit.

123 Aborted

124 Upon receipt of this notification, the initiator knows that the coordinator reached a decision to
125 abort.

126 Conforming implementations must implement Completion.

127 4.3 Two-Phase Commit Protocol

128 The Two-Phase Commit (2PC) protocol is a Coordination protocol that defines how multiple participants
129 reach agreement on the outcome of an atomic transaction. The 2PC protocol has two variants: Durable
130 2PC and Volatile 2PC.

131 4.3.1 Volatile Two-Phase Commit Protocol

132 Upon receiving a Commit notification in the completion protocol, the root coordinator begins the prepare
133 phase of all participants registered for the Volatile 2PC protocol. All participants registered for this
134 protocol must respond before a Prepare is issued to a participant registered for Durable 2PC. Further
135 participants may register with the coordinator until the coordinator issues a Prepare to any durable
136 participant. A volatile recipient is not guaranteed to receive a notification of the transaction's outcome.

137 Participants register for this protocol using the following protocol identifier:

138

<http://schemas.xmlsoap.org/ws/2004/10/wsat/Volatile2PC>

139 **4.3.2 Durable Two-Phase Commit Protocol**

140 After receiving a Commit notification in the completion protocol and upon successfully completing the
141 prepare phase for Volatile 2PC participants, the root coordinator begins the Prepare phase for Durable
142 2PC participants. All participants registered for this protocol must respond Prepared or ReadOnly before
143 a Commit notification is issued to a participant registered for either protocol.

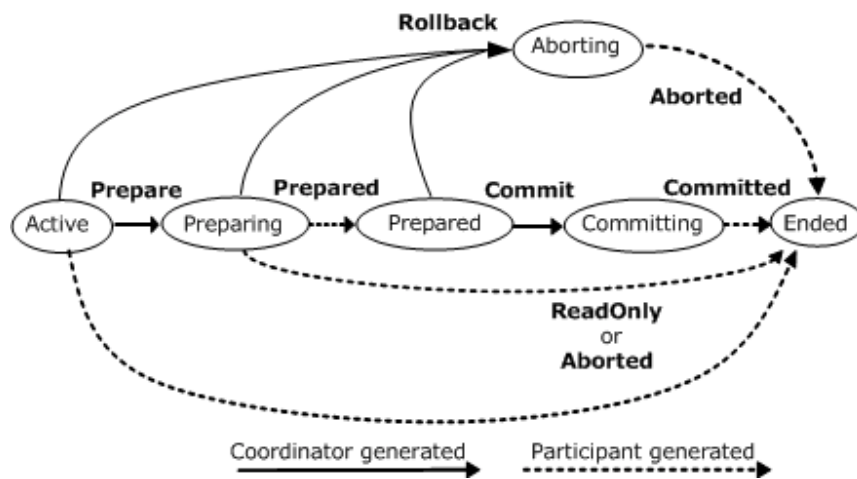
144 Participants register for this protocol using the following protocol identifier:

145 `http://schemas.xmlsoap.org/ws/2004/10/wsat/Durable2PC`

146 **4.3.3 2PC Diagram and Notifications**

147 The diagram below illustrates the protocol abstractly:

148



149

150 The participant accepts:

151 Prepare

152 Upon receipt of this notification, the participant knows to enter phase 1 and vote on the outcome
153 of the transaction. If the participant does not know of the transaction, it must vote to abort. If the
154 participant has already voted, it should resend the same vote.

155 Rollback

156 Upon receipt of this notification, the participant knows to abort, and forget, the transaction. This
157 notification can be sent in either phase 1 or phase 2. Once sent, the coordinator may forget all
158 knowledge of this transaction.

159 Commit

160 Upon receipt of this notification, the participant knows to commit the transaction. This notification
161 can only be sent after phase 1 and if the participant voted to commit. If the participant does not
162 know of the transaction, it must send a Committed notification to the coordinator.

163 The coordinator accepts:

164 Prepared

165 Upon receipt of this notification, the coordinator knows the participant is prepared and votes to
166 commit the transaction.

167 ReadOnly
168 Upon receipt of this notification, the coordinator knows the participant votes to commit the
169 transaction, and has forgotten the transaction. The participant does not wish to participate in
170 phase 2.

171 Aborted
172 Upon receipt of this notification, the coordinator knows the participant has aborted, and forgotten,
173 the transaction.

174 Committed
175 Upon receipt of this notification, the coordinator knows the participant has committed the
176 transaction. That participant may be safely forgotten.

177 Replay
178 Upon receipt of this notification, the coordinator may assume the participant has suffered a
179 recoverable failure. It should resend the last appropriate protocol notification.

180 Conforming implementations MUST implement the 2PC protocol.

181

5 AT Policy Assertion

182 WS-Policy Framework [WS-Policy] and WS-Policy Attachment [WS-PolicyAttachment] collectively define
183 a framework, model and grammar for expressing the capabilities, requirements, and general
184 characteristics of entities in an XML Web services-based system. To enable a web service to describe
185 transactional capabilities and requirements of a service and its operations, this specification defines a pair
186 of Atomic Transaction policy assertions that leverage the WS-Policy framework.

5.1 Assertion Model

188 The AT policy assertions are provided by a web service to qualify the transactional processing of
189 messages associated with the particular operation to which the assertions are scoped. The AT policy
190 assertions indicate:

- 191 1. whether a requester MAY, MUST or SHOULD NOT include an AtomicTransaction
192 CoordinationContext flowed with the message.
- 193 2. the capability of the target service to process the message under an atomic transaction
194 regardless of whether the requester supplies an AtomicTransaction CoordinationContext.

195 The AT policy assertions are semantically independent of one another, and may be used together or in
196 isolation.

5.2 Normative Outline

198 The normative outlines for the AT policy assertions are:

```
199 <wsat:ATAssertion [wsp:Optional="true"]? ... >  
200 ...  
201 </wsat:ATAssertion>
```

202 The following describes additional, normative constraints on the outline listed above:

203 /wsat:ATAssertion

204 A policy assertion that specifies that an atomic transaction MUST be flowed inside a requester's
205 message. From the perspective of the requester, the target service that processes the transaction MUST
206 behave as if it had participated in the transaction. The transaction MUST be represented as a SOAP
207 header in CoordinationContext format, as defined in WS-Coordination [WS-Coordination].

208 /wsat:ATAssertion/@wsp:Optional="true"

209 Per WS-Policy [WS-Policy], this is compact notation for two policy alternatives, one with and one without
210 the assertion. Presence of both policy alternatives indicates that the behavior indicated by the assertion is
211 optional, such that an atomic transaction MAY be flowed inside a requester's message. The absence of
212 the assertion is interpreted to mean that a transaction SHOULD NOT be flowed inside a requester's
213 message.

```
214 <wsat:ATAlwaysCapability ... />
```

215 The following describes additional, normative constraints on the outline listed above:

216 /wsat:ATAlwaysCapability

217 A policy assertion that specifies a capability of the target service indicating that a requester's message
218 will be processed transactionally regardless of whether the requester supplies an AtomicTransaction
219 CoordinationContext. If an AtomicTransaction context is provided by the requester, it will be used.
220 Otherwise the processing of the message will be within a transaction implicitly started and ended by the
221 target service's environment as part of the processing of that message.

222 5.3 Assertion Attachment

223 Because the AT policy assertions indicate atomic transaction behavior for a single operation, the
224 assertions have Operation Policy Subject [WS-PolicyAttachment].

225 WS-PolicyAttachment defines two WSDL [WSDL 1.1] policy attachment points with Operation Policy
226 Subject:

227 wsdl:portType/wsdl:operation – A policy expression containing the AT policy assertion MUST
228 NOT be attached to a wsdl:portType; the AT policy assertions specify a concrete behavior
229 whereas the wsdl:portType is an abstract construct.

230 wsdl:binding/wsdl:operation – A policy expression containing the AT policy assertions SHOULD
231 be attached to a wsdl:binding.

232 5.4 Assertion Example

233 An example use of the AT policy assertion follows:

```
234 (01) <wsdl:definitions
235 (02)     targetNamespace="bank.example.com"
236 (03)     xmlns:tns="bank.example.com"
237 (04)     xmlns:wSDL="http://schemas.xmlsoap.org/wSDL/"
238 (05)     xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
239 (06)     xmlns:wsat="http://schemas.xmlsoap.org/ws/2004/10/wsat"
240 (07)     xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
241 wssecurity-utility-1.0.xsd" >
242 (08)
243 (09)     <wsp:Policy wsu:Id="TransactedPolicy1" >
244 (10)         <wsat:ATAssertion wsp:optional="true" />
245 (11)         <!-- omitted assertions -->
246 (12)     </wsp:Policy>
247 (13)     <wsp:Policy wsu:Id="TransactedPolicy2" >
248 (14)         <wsat:ATAlwaysCapability />
249 (15)         <!-- omitted assertions -->
250 (16)     </wsp:Policy>
251 (17)     <!-- omitted elements -->
252 (18)     <wsdl:binding name="BankBinding" type="tns:BankPortType" >
253 (19)     <!-- omitted elements -->
```

```

254 (20) <wsdl:operation name="QueryBalance" >
255 (21) <wsp:PolicyReference URI="#TransactedPolicy2"
256 wsdl:required="true" />
257 (22) <!-- omitted elements -->
258 (23) </wsdl:operation>
259 (24) <wsdl:operation name="TransferFunds" >
260 (25) <wsp:PolicyReference URI="#TransactedPolicy1"
261 wsdl:required="true" />
262 (26) <!-- omitted elements -->
263 (27) </wsdl:operation>
264 (28) </wsdl:binding>
265 (29) </wsdl:definitions>

```

266

267 Lines (9-12) are a policy expression that includes an AT policy assertion (Line 10) to indicate that an
268 atomic transaction in WS-Coordination [[WS-Coordination](#)] format MAY be used.

269 Lines (13-16) are a policy expression that includes an AT policy assertion (Line 14) to indicate that a
270 capability of the target service is that it will process messages in a transaction regardless of whether any
271 AtomicTransaction CoordinationContext is sent by the requester.

272 Lines (20-23) are a WSDL [[WSDL 1.1](#)] binding. Line (21) indicates that the policy in Lines (13-16) applies
273 to this binding, specifically indicating that QueryBalance messages are processed in an atomic
274 transaction regardless of whether a requester provides an AtomicTransaction CoordinationContext.

275 Lines (24-27) are a WSDL [[WSDL 1.1](#)] binding. Line (25) indicates that the policy in Lines (9-12) applies
276 to this binding, specifically indicating that an atomic transaction MAY flow inside messages.

277 6 Transaction Faults

278 WS-AtomicTransaction faults MUST include as the [action] property the following fault action URI:

279 `http://schemas.xmlsoap.org/ws/2004/10/wsat/fault`

280 The faults defined in this section are generated if the condition stated in the preamble is met. Faults are
281 targeted at a destination endpoint according to the fault handling rules defined in [WSADDR].

282 The definitions of faults in this section use the following properties:

283 [Code] The fault code.

284 [Subcode] The fault subcode.

285 [Reason] The English language reason element.

286 [Detail] The detail element. If absent, no detail element is defined for the fault.

287 For SOAP 1.2, the [Code] property MUST be either "Sender" or "Receiver". These properties are
288 serialized into text XML as follows:

289

SOAP Version	Sender	Receiver
SOAP 1.2	S:Sender	S:Receiver

290

291 The properties above bind to a SOAP 1.2 fault as follows:

```
292 <S:Envelope>  
293   <S:Header>  
294     <wsa:Action>  
295       http://schemas.xmlsoap.org/ws/2004/10/wsat/fault  
296     </wsa:Action>  
297     <!-- Headers elided for clarity. -->  
298   </S:Header>  
299   <S:Body>  
300     <S:Fault>  
301       <S:Code>  
302         <S:Value>[Code]</S:Value>  
303         <S:Subcode>  
304           <S:Value>[Subcode]</S:Value>  
305         </S:Subcode>  
306       </S:Code>  
307       <S:Reason>  
308         <S:Text xml:lang="en">[Reason]</S:Text>  
309       </S:Reason>  
310       <S:Detail>  
311         [Detail]  
312         ...  
313       </S:Detail>  
314     </S:Fault>  
315   </S:Body>  
316 </S:Envelope>
```

317 The properties bind to a SOAP 1.1 fault as follows:

318 `<S11:Envelope>`

```
319 <S11:Body>
320 <S11:Fault>
321 <faultcode>[Subcode]</faultcode>
322 <faultstring xml:lang="en">[Reason]</faultstring>
323 </S11:Fault>
324 </S11:Body>
325 </S11:Envelope>
```

326 6.1 InconsistentInternalState

327 This fault is sent by a participant to indicate that it cannot fulfill its obligations. This indicates a global
328 consistency failure and is an unrecoverable condition.

329 Properties:

330 **[Code]** Sender

331 **[Subcode]** wsat:InconsistentInternalState

332 **[Reason]** A global consistency failure has occurred. This is an unrecoverable condition.

333 **[Detail]** unspecified

334

7 Security Model

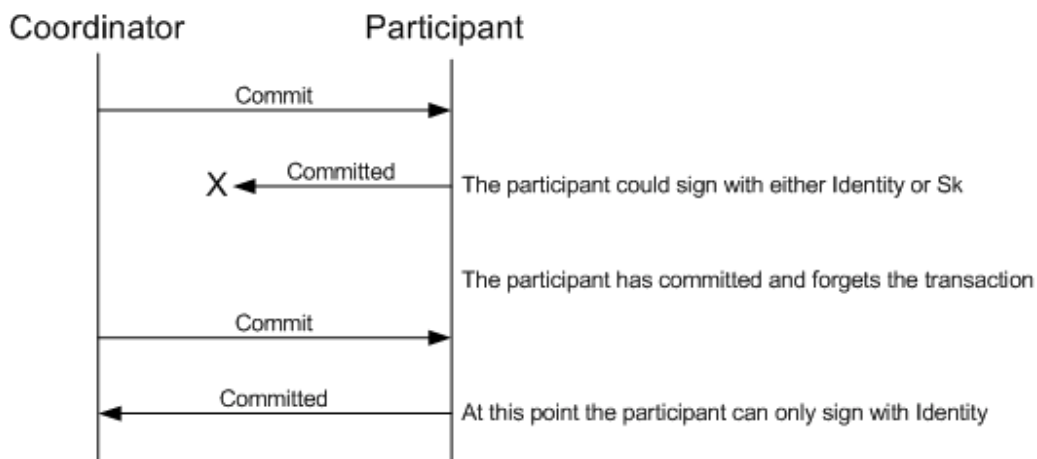
335 The security model for atomic transactions builds on the model defined in WS-Coordination [WSCOOD].
336 That is, services have policies specifying their requirements and requestors provide claims (either implicit
337 or explicit) and the requisite proof of those claims. Coordination context creation establishes a base
338 secret which can be delegated by the creator as appropriate.

339 Because atomic transactions represent a specific use case rather than the general nature of coordination
340 contexts, additional aspects of the security model can be specified.

341 All access to atomic transaction protocol instances is on the basis of identity. The nature of transactions,
342 specifically the uncertainty of systems means that the security context established to register for the
343 protocol instance may not be available for the entire duration of the protocol.

344 Consider for example the scenarios where a participant has committed its part of the transaction, but for
345 some reason the coordinator never receives acknowledgement of the commit. The result is that when
346 communication is re-established in the future, the coordinator will attempt to confirm the commit status of
347 the participant, but the participant, having committed the transaction and forgotten all information
348 associated with it, no longer has access to the special keys associated with the token.

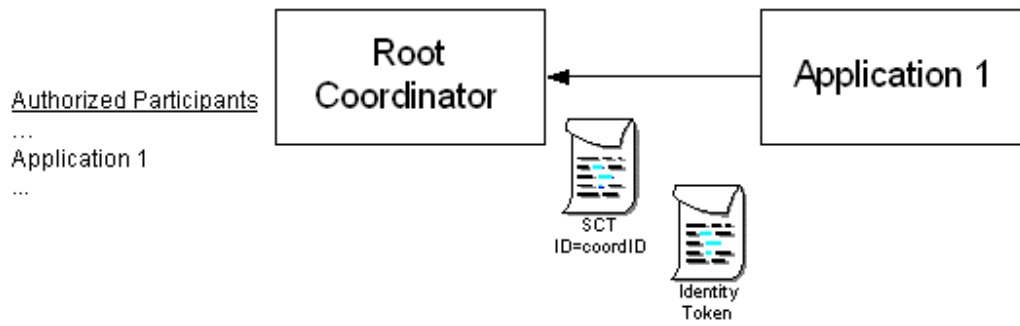
349 The participant can only prove its identity to the coordinator when it indicates that the specified
350 transaction is not in its log and assumed committed. This is illustrated in the figure below:



351

352 There are, of course, techniques to mitigate this situation but such options will not always be successful.
353 Consequently, when dealing with atomic transactions, it is critical that identity claims always be proven to
354 ensure that correct access control is maintained by coordinators.

355 There is still value in coordination context-specific tokens because they offer a bootstrap mechanism so
356 that all participants need not be pre-authorized. As well, it provides additional security because only
357 those instances of an identity with access to the token will be able to securely interact with the coordinator
358 (limiting privileges strategy). This is illustrated in the figure below:



359

360 The "list" of authorized participants ensures that application messages having a coordination context are
 361 properly authorized since altering the coordination context ID will not provide additional access unless (1)
 362 the bootstrap key is provided, or (2) the requestor is on the authorized participant "list" of identities.

363

8 Security Considerations

364 It is strongly RECOMMENDED that the communication between services be secured using the
365 mechanisms described in WS-Security [WSSec]. In order to properly secure messages, the body and all
366 relevant headers need to be included in the signature. Specifically, the
367 <wscoor:CoordinationContext> header needs to be signed with the body and other key message
368 headers in order to "bind" the two together.

369 In the event that a participant communicates frequently with a coordinator, it is RECOMMENDED that a
370 security context be established using the mechanisms described in WS-Trust [WSTrust] and WS-
371 SecureConversation [WSSecConv] allowing for potentially more efficient means of authentication.

372 It is common for communication with coordinators to exchange multiple messages. As a result, the usage
373 profile is such that it is susceptible to key attacks. For this reason it is strongly RECOMMENDED that the
374 keys be changed frequently. This "re-keying" can be effected a number of ways. The following list
375 outlines four common techniques:

376 Attaching a nonce to each message and using it in a derived key function with the shared secret

377 Using a derived key sequence and switch "generations"

378 Closing and re-establishing a security context (not possible for delegated keys)

379 Exchanging new secrets between the parties (not possible for delegated keys)

380 It should be noted that the mechanisms listed above are independent of the SCT and secret returned
381 when the coordination context is created. That is, the keys used to secure the channel may be
382 independent of the key used to prove the right to register with the activity.

383 The security context MAY be re-established using the mechanisms described in WS-Trust [WSTrust] and
384 WS-SecureConversation [WSSecConv]. Similarly, secrets can be exchanged using the mechanisms
385 described in WS-Trust. Note, however, that the current shared secret SHOULD NOT be used to encrypt
386 the new shared secret. Derived keys, the preferred solution from this list, can be specified using the
387 mechanisms described in WS-SecureConversation.

388 The following list summarizes common classes of attacks that apply to this protocol and identifies the
389 mechanism to prevent/mitigate the attacks:

390 **Message alteration** – Alteration is prevented by including signatures of the message information
391 using WS-Security [WSSec].

392 **Message disclosure** – Confidentiality is preserved by encrypting sensitive data using WS-
393 Security.

394 **Key integrity** – Key integrity is maintained by using the strongest algorithms possible (by
395 comparing secured policies – see WS-Policy [WSPOLICY] and WS-SecurityPolicy
396 [WSSecPolicy]).

397 **Authentication** – Authentication is established using the mechanisms described in WS-Security
398 and WS-Trust [WSTrust]. Each message is authenticated using the mechanisms described in
399 WS-Security [WSSec].

400 **Accountability** – Accountability is a function of the type of and string of the key and algorithms
401 being used. In many cases, a strong symmetric key provides sufficient accountability. However,
402 in some environments, strong PKI signatures are required.

403 **Availability** – Many services are subject to a variety of availability attacks. Replay is a common
404 attack and it is RECOMMENDED that this be addressed as described in the next bullet. Other
405 attacks, such as network-level denial of service attacks are harder to avoid and are outside the
406 scope of this specification. That said, care should be taken to ensure that minimal processing be
407 performed prior to any authenticating sequences.

408 **Replay** – Messages may be replayed for a variety of reasons. To detect and eliminate this
409 attack, mechanisms should be used to identify replayed messages such as the timestamp/nonce
410 outlined in WS-Security [WSSEC]. Alternatively, and optionally, other technologies, such as
411 sequencing, can also be used to prevent replay of application messages.

412 9 Use of WS-Addressing Headers

413 The messages defined in WS-Coordination and WS-AtomicTransaction can be classified into four types:

414 Request messages: **CreateCoordinationContext** and **Register**.

415 Reply messages: **CreateCoordinationContextResponse** and **RegisterResponse**.

416 Notification messages: **Commit**, **Rollback**, **Committed**, **Aborted**, **Prepare**,
417 **Prepared**, **ReadOnly** and **Replay**.

418 Fault messages

419 Request and reply messages follow the standard "Request Reply" pattern as defined in WS-Addressing.
420 Notification messages follow the standard "one way" pattern as defined in WS-Addressing. There are
421 two types of notification messages:

422 A notification message is a terminal message when it indicates the end of a
423 coordinator/participant relationship. **Committed**, **Aborted** and **ReadOnly** are
424 terminal messages.

425 A notification message is not a terminal message when it does not indicate the end
426 of a coordinator/participant relationship. **Commit**, **Rollback**, **Prepare**, **Prepared**
427 and **Replay** are not terminal messages.

428 The following statements define addressing interoperability requirements for the respective WS-
429 Coordination and WS-AtomicTransaction message types:

430 Request messages

431 MUST include a wsa:MessageID header.

432 MUST include a wsa:ReplyTo header.

433 Reply messages

434 MUST include a wsa:RelatesTo header, specifying the MessageID from the
435 corresponding Request message.

436 Non-terminal notification messages

437 MUST include a wsa:ReplyTo header

438 Terminal notification messages

439 SHOULD NOT include a wsa:ReplyTo header

440 Fault messages

441 MUST include a wsa:RelatesTo header, specifying the MessageID from the Request or
442 Notification message that generated the fault condition.

443
444 Notification messages are addressed by both coordinators and participants using the Endpoint
445 References initially obtained during the Register-RegisterResponse exchange. If a wsa:ReplyTo header
446 is present in a notification message it MAY be used by the recipient, for example in cases where a
447 Coordinator or Participant has forgotten a transaction that is completed and needs to respond to a resent

448 protocol message. Permanent loss of connectivity between a coordinator and a participant in an in-doubt
449 state can result in data corruption.

450 If a wsa:FaultTo header is present on a message that generates a fault condition, then it MUST be used
451 by the recipient as the destination for any fault. Otherwise, fault messages MAY be addressed by both
452 coordinators and participants using the Endpoint References initially obtained during the Register-
453 RegisterResponse exchange.

454 All messages are delivered using connections initiated by the sender. Endpoint References MUST
455 contain physical addresses and MUST NOT use well-known "anonymous" endpoint defined in WS-
456 Addressing.

457

458 **10References**

459 **[KEYWORDS]**

460 S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119, Harvard
461 University, March 1997

462 **[SOAP]**

463 W3C Note, "SOAP: Simple Object Access Protocol 1.1," 08 May 2000

464 **[URI]**

465 T. Berners-Lee, R. Fielding, L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax,"
466 RFC 2396, MIT/LCS, U.C. Irvine, Xerox Corporation, August 1998

467 **[XML-ns]**

468 W3C Recommendation, "Namespaces in XML," 14 January 1999

469 **[XML-Schema1]**

470 W3C Recommendation, "XML Schema Part 1: Structures," 2 May 2001

471 **[XML-Schema2]**

472 W3C Recommendation, "XML Schema Part 2: Datatypes," 2 May 2001

473 **[WSCOOR]**

474 Web Services Coordination (WS-Coordination), Arjuna Technologies Ltd., BEA Systems, Hitachi
475 Ltd., IBM, IONA Technologies and Microsoft, August 2005

476 **[WSADDR]**

477 Web Services Addressing (WS-Addressing), Microsoft, IBM, Sun, BEA Systems, SAP, Sun,
478 August 2004

479 **[WSPOLICY]**

480 Web Services Policy Framework (WS-Policy), VeriSign, Microsoft, Sonic Software, IBM, BEA
481 Systems, SAP, September 2004

482 **[WSPOLICYATTACH]**

483 Web Services Policy Attachment (WS-PolicyAttachment), VeriSign, Microsoft, Sonic Software,
484 IBM, BEA Systems, SAP, September 2004

485 **[WSDL]**

486 Web Services Description Language (WSDL) 1.1

487 "<http://www.w3.org/TR/2001/NOTE-wsdl-20010315>"

488 **[WSec]**

489 OASIS Standard 200401, March 2004, "Web Services Security: SOAP Message Security 1.0
490 (WS-Security 2004)"

491 **[WSecPolicy]**

492 Web Services Security Policy Language (WS-SecurityPolicy), Microsoft, VeriSign, IBM, RSA
493 Security, July 2005

494 **[WSecConv]**

495 Web Services Secure Conversation Language (WS-SecureConversation), OpenNetwork, Layer7,
496 Netegrity, Microsoft, Reactivity, IBM, VeriSign, BEA Systems, Oblix, RSA Security, Ping Identity,
497 Westbridge, Computer Associates, February 2005

498 **[WSTrust]**

499 Web Services Trust Language (WS-Trust), OpenNetwork, Layer7, Netegrity, Microsoft,
500 Reactivity, VeriSign, IBM, BEA Systems, Oblix, RSA Security, Ping Identity, Westbridge,
501 Computer Associates, February 2005.

502

11 State Tables

503 The following state tables specify the behavior of coordinators and participants when presented with
504 protocol messages or internal events. These tables present the view of a coordinator or participant with
505 respect to a single partner. A coordinator with multiple participants can be understood as a collection of
506 independent coordinator state machines.

507 Each cell in the tables uses the following convention:

508

Legend
<i>action to take</i> next state

509

510 Each state supports a number of possible events. Expected events are processed by taking the
511 prescribed action and transitioning to the next state. Unexpected protocol messages will result in a fault
512 message, with a standard fault code such as Invalid State or Inconsistent Internal State. Events that may
513 not occur in a given state are labelled as N/A.

Atomic Transaction 2PC protocol (Coordinator View)							
Inbound Events	States						
	None	Active	Preparing	Prepared	PreparedSuccess	Committing	Aborting
Register	<i>Invalid State</i> None	<i>Send RegisterResponse</i> Active	<i>Durable: Invalid State</i> Aborting <i>Volatile: Send RegisterResponse</i> Active	N/A	<i>Invalid State</i> PreparedSuccess	<i>Invalid State</i> Committing	<i>Invalid State</i> Aborting
Prepared	<i>Durable: Send Rollback</i> <i>Volatile: Invalid State</i> None	<i>Invalid State</i> Aborting	<i>Record Vote</i> Preparing	N/A	<i>Ignore</i> PreparedSuccess	<i>Resend Commit</i> Committing	<i>Resend Rollback, and forget</i> Aborting
ReadOnly	<i>Ignore</i> None	<i>Forget</i> Active	<i>Forget</i> Preparing	N/A	<i>Invalid State</i> PreparedSuccess	<i>Invalid State</i> Committing	<i>Forget</i> Aborting
Aborted	<i>Ignore</i> None	<i>Forget</i> Aborting	<i>Forget</i> Aborting	N/A	<i>Invalid State</i> PreparedSuccess	<i>Invalid State</i> Committing	<i>Forget</i> Aborting
Committed	<i>Ignore</i> None	<i>Invalid State</i> Aborting	<i>Invalid State</i> Aborting	N/A	<i>Invalid State</i> PreparedSuccess	<i>Forget</i> Committing	<i>Invalid State</i> Aborting
Replay	<i>Durable: Send Rollback</i> <i>Volatile: Invalid State</i> None	<i>Send Rollback</i> Aborting	<i>Send Rollback</i> Aborting	N/A	<i>Ignore</i> PreparedSuccess	<i>Send Commit</i> Committing	<i>Send Rollback</i> Aborting
Internal Events							
User Commit	<i>Return Aborted</i> None	<i>Send Prepare</i> Preparing	<i>Ignore</i> Preparing	N/A	<i>Ignore</i> PreparedSuccess	<i>Return Committed</i> Committing	<i>Return Aborted</i> Aborting
User Rollback	<i>Return Aborted</i> None	<i>Send Rollback</i> Aborting	<i>Send Rollback</i> Aborting	N/A	<i>Invalid State</i> PreparedSuccess	<i>Invalid State</i> Committing	<i>Return Aborted</i> Aborting
Expires Times out	N/A	<i>Send Rollback</i> Aborting	<i>Send Rollback</i> Aborting	N/A	<i>Ignore</i> PreparedSuccess	<i>Ignore</i> Committing	<i>Ignore</i> Aborting
Comms Times out	N/A	N/A	<i>Resend Prepare</i> Preparing	N/A	N/A	<i>Resend Commit</i> Committing	N/A
Commit Decision	N/A	N/A	<i>Record Outcome</i> PreparedSuccess	N/A	N/A	N/A	N/A
Write Done	N/A	N/A	N/A	N/A	<i>Send Commit</i> Committing	N/A	N/A
Write Failed	N/A	N/A	N/A	N/A	<i>Send Rollback</i> Aborting	N/A	N/A
All Forgotten	N/A	Active	None	N/A	N/A	None	None

514

515 Notes:

516 1. Transitions with a "N/A" as their action are inexpressible. A TM should view these
517 transitions as serious internal consistency issues, and probably fatal.

518 2. Internal events are those that are created either within a TM itself, or on its local
519 system.

520 “Forget” implies that the subordinate’s participation is removed from the coordinator (if
 521 necessary), and otherwise the message is ignored

Atomic Transaction 2PC protocol (Participant View)							
Inbound Events	States						
	None	Active	Preparing	Prepared	PreparedSuccess	Committing	Aborting
Register Response	<i>Register Subordinate</i> Active	<i>Invalid State</i> Active	<i>Invalid State</i> Aborting	<i>Invalid State</i> Prepared	<i>Invalid State</i> PreparedSuccess	<i>Invalid State</i> Committing	<i>Invalid State</i> Aborting
Prepare	<i>Send Aborted</i> None	<i>Gather Vote</i> <i>Dedslon</i> Preparing	<i>Ignore</i> Preparing	<i>Ignore</i> Prepared	<i>Resend Prepared</i> PreparedSuccess	<i>Ignore</i> Committing	<i>Resend Aborted, and forget</i> Aborting
Commit	<i>Send Committed</i> None	<i>Invalid State</i> Aborting	<i>Invalid State</i> Aborting	<i>Invalid State</i> Aborting	<i>Initiate commit decision</i> Committing	<i>Ignore</i> Committing	<i>InconsistentInternalState</i> Aborting
Rollback	<i>Send Aborted</i> None	<i>Initiate Rollback,</i> <i>Send Aborted, and Forget</i> Aborting	<i>Initiate Rollback,</i> <i>Send Aborted, and Forget</i> Aborting	<i>Initiate Rollback,</i> <i>Send Aborted, and Forget</i> Aborting	<i>Initiate Rollback,</i> <i>Send Aborted, and Forget</i> Aborting	<i>InconsistentInternalState</i> Committing	<i>Send Aborted, and Forget</i> Aborting
Internal Events							
Expires Times out	N/A	<i>Send Aborted</i> Aborting	<i>Send Aborted</i> Aborting	<i>Ignore</i> Prepared	<i>Ignore</i> PrepareSuccess	<i>Ignore</i> Committing	<i>Ignore</i> Aborting
Comms Times out	N/A	N/A	N/A	N/A	<i>Resend Prepared</i> PreparedSuccess	N/A	N/A
Commit Decision	N/A	N/A	<i>Record Commit</i> Prepared	N/A	N/A	<i>Send Committed and Forget</i> Committing	N/A
Rollback Decision	N/A	N/A	<i>Send Aborted</i> Aborting	N/A	N/A	N/A	N/A
Write Done	N/A	N/A	N/A	<i>Send Prepared</i> PreparedSuccess	N/A	N/A	N/A
Write Failed	N/A	N/A	N/A	<i>Initiate Rollback,</i> <i>Send Aborted, and Forget</i> Aborting	N/A	N/A	N/A
All Forgotten	None	N/A	<i>Send ReadOnly</i> None	N/A	N/A	None	None

522

523 Notes:

- 524 1. Transitions with a “N/A” as their action are inexpressible. A TM should view these
 525 transitions as serious internal consistency issues, and probably fatal.
- 526 2. Internal events are those that are created either within a TM itself, or on its local
 527 system.

528 **Appendix A. Acknowledgements**

529 The following individuals have participated in the creation of this specification and are
530 gratefully acknowledged:

531 **Participants:**

532 [Participant Name, Affiliation | Individual Member]

533 [Participant Name, Affiliation | Individual Member]

534

Appendix B. Revision History

Revision	yy-mm-dd	Editor	Changes Made
01	05-11-22	Mark Little	Initial Working Draft
01	05-11-23	Andrew Wilkinson	Initial Working Draft

535