



Web Services Business Activity (WS-BusinessActivity) Version 1.1

OASIS Standard incorporating Approved Errata

12 July 2007

Specification URIs:

This Version:

<http://docs.oasis-open.org/ws-tx/wstx-wsba-1.1-spec-errata-os/wstx-wsba-1.1-spec-errata-os.html>
<http://docs.oasis-open.org/ws-tx/wstx-wsba-1.1-spec-errata-os.doc>
<http://docs.oasis-open.org/ws-tx/wstx-wsba-1.1-spec-errata-os.pdf>

Previous Version:

<http://docs.oasis-open.org/ws-tx/wstx-wsba-1.1-spec-os/wstx-wsba-1.1-spec-os.html>
<http://docs.oasis-open.org/ws-tx/wstx-wsba-1.1-spec-os.doc>
<http://docs.oasis-open.org/ws-tx/wstx-wsba-1.1-spec-os.pdf>

Latest Approved Version:

<http://docs.oasis-open.org/ws-tx/wstx-wsba-1.1-spec/wstx-wsba-1.1-spec.html>
<http://docs.oasis-open.org/ws-tx/wstx-wsba-1.1-spec.doc>
<http://docs.oasis-open.org/ws-tx/wstx-wsba-1.1-spec.pdf>

Technical Committee:

OASIS Web Services Transaction (WS-TX) TC

Chair(s):

Eric Newcomer, Iona
Ian Robinson, IBM

Editor(s):

Tom Freund, IBM <tjfreund@us.ibm.com>
Mark Little, JBoss Inc. <mark.little@jboss.com>

Declared XML Namespaces:

<http://docs.oasis-open.org/ws-tx/wsba/2006/06>

Abstract:

The WS-BusinessActivity specification provides the definition of two Business Activity coordination types: AtomicOutcome or MixedOutcome, that are to be used with the extensible coordination framework described in the WS-Coordination specification. This specification also defines two specific Business Activity agreement coordination protocols for the Business Activity coordination types: BusinessAgreementWithParticipantCompletion, and BusinessAgreementWithCoordinatorCompletion. Developers can use these protocols when building applications that require consistent agreement on the outcome of long-running distributed activities.

Status:

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 "Latest Approved Version" location noted above for possible later revisions of this document.

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

Copyright © OASIS Open 2007. All Rights Reserved.

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full Policy may be found at the OASIS website.

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 section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, 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 OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification.

OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this specification by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification. OASIS may include such claims on its website, but disclaims any obligation to do so.

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' procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on 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 implementers or users of this OASIS Committee Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

Table of Contents

1	Introduction.....	5
1.1	Model	5
1.2	Composable Architecture	6
1.3	Terminology	6
1.4	Namespace.....	7
1.4.1	Prefix Namespace	7
1.5	XSD and WSDL Files	7
1.6	Protocol Elements.....	7
1.7	Normative References	7
2	Business Activity Context	9
3	Coordination Types and Protocols	10
3.1	Preconditions	10
3.2	BusinessAgreementWithParticipantCompletion Protocol.....	10
3.3	BusinessAgreementWithCoordinatorCompletion Protocol.....	13
4	Policy Assertions	16
4.1	Assertion Models	16
4.2	Normative Outlines	16
4.3	Assertion Attachment.....	17
4.4	Assertion Example.....	17
5	Security Considerations	19
6	Use of WS-Addressing Headers	21
A.	Acknowledgements	22
B.	State Tables for the Agreement Protocols	23
B.1	Participant view of BusinessAgreementWithParticipantCompletion	24
B.2	Coordinator view of BusinessAgreementWithParticipantCompletion	26
B.3	Participant view of BusinessAgreementWithCoordinatorCompletion	28
B.4	Coordinator view of BusinessAgreementWithCoordinatorCompletion	30

1 Introduction

The current set of Web service specifications **[WSDL]** **[SOAP 1.1]** **[SOAP 1.2]** define protocols for Web service interoperability. Web services increasingly tie together a number of participants forming large distributed applications. The resulting activities may have complex structure and relationships.

The WS-Coordination **[WSCOOOR]** specification defines an extensible framework for defining coordination types.

This specification provides the definition of two Business Activity coordination types used to coordinate activities that apply business logic to handle exceptions that occur during the execution of activities of a business process. Actions are applied immediately and are permanent. Compensating actions may be invoked in the event of an error. WS-BusinessActivity defines protocols that enable existing business process and work flow systems to wrap their proprietary mechanisms and interoperate across trust boundaries and different vendor implementations.

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

- The reader is familiar with the WS-Coordination **[WSCOOOR]** specification which defines the framework for the Business Activity coordination protocols.
- The reader is familiar with WS-Addressing **[WSADDR]** and WS-Policy **[WSPOLICY]**.

Business activities have the following characteristics:

- A business activity may consume many resources over a long duration.
- There may be a significant number of atomic transactions involved.
- Individual tasks within a business activity can be seen prior to the completion of the business activity, their results may have an impact outside of the computer system.
- Responding to a request may take a very long time. Human approval, assembly, manufacturing, or delivery may have to take place before a response can be sent.
- In the case where a business exception requires an activity to be logically undone, abort is typically not sufficient. Exception handling mechanisms may require business logic, for example in the form of a compensation task, to reverse the effects of a previously completed task.
- Participants in a business activity may be in different domains of trust where all trust relationships are established explicitly.

The Business Activity protocols defined in this specification have the following design points:

- All state transitions are reliably recorded, including application state and coordination metadata.
- All non-terminal notifications are acknowledged in the protocol to ensure a consistent view of state between the coordinator and participant. A coordinator or participant may solicit the status of its partner or retry sending notifications in order to achieve this.
- Each notification is defined as an individual message. Transport level request/response retry and time out are not sufficient mechanisms to achieve end-to-end agreement coordination for long-running activities.

1.1 Model

Business Activity coordination protocols provide the following flexibility:

- A business application may be partitioned into business activity scopes. A business activity scope is a business task consisting of a general-purpose computation carried out as a bounded set of operations on a collection of Web services that require a mutually agreed outcome. There may be any number of hierarchical nesting levels. Nested scopes:

- 43 – Allow a business application to select which child tasks are included in the overall outcome
44 processing. For example, a business application might solicit an estimate from a number of
45 suppliers and choose a quote or bid based on lowest-cost.
- 46 – Allow a business application to catch an exception thrown by a child task, apply an exception
47 handler, and continue processing even if something goes wrong. When a child completes its
48 work, it may be associated with a compensation that is registered with the parent activity.
- 49 • A participant task within a business activity may specify that it is leaving a business activity. This
50 provides the ability to exit a business activity and allows business programs to delegate
51 processing to other scopes. The participant list is dynamic and a participant may exit the protocol
52 at any time without waiting for the outcome of the protocol.
- 53 • The Business Activity coordination protocols allow a participant task within a business activity to
54 specify its outcome directly without waiting for solicitation. Such a feature is generally useful when
- 55 • A task fails so that the notification can be used by a business activity exception handler to modify
56 the goals and drive processing in a timely manner.
- 57 • The Business Activity coordination protocols allow participants in a coordinated business activity
58 to perform "tentative" operations as a normal part of the activity. The result of such "tentative"
59 operations may become visible before the activity is complete and may require business logic to
60 run in the event that the operation needs to be compensated. Such a feature is critical when the
61 joint work of a business activity requires many operations performed by independent services
62 over a long period of time.

63 1.2 Composable Architecture

64 By using the XML [**XML**], SOAP [**SOAP 1.1**] [**SOAP 1.2**] and WSDL [**WSDL**] extensibility model, SOAP-
65 based and WSDL-based specifications are designed to work together to define a rich Web services
66 environment. As such, WS-BusinessActivity by itself does not define all features required for a complete
67 solution. WS-BusinessActivity is a building block used with other specifications of Web services (e.g.,
68 WS-Coordination [**WSCOOR**], WS-Security [**WSSec**]) and application-specific protocols that are able to
69 accommodate a wide variety of coordination protocols related to the coordination actions of distributed
70 applications.

71 1.3 Terminology

72 The uppercase key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD",
73 "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as
74 described in RFC2119 [**RFC2119**].

75 This specification uses an informal syntax to describe the XML grammar of the XML fragments below:

- 76 • The syntax appears as an XML instance, but the values indicate the data types instead of values.
- 77 • Element names ending in "..." (such as <element.../> or <element...>) indicate that
78 elements/attributes irrelevant to the context are being omitted.
- 79 • Attributed names ending in "..." (such as name=...) indicate that the values are specified below.
- 80 • Grammar in bold has not been introduced earlier in the document, or is of particular interest in an
81 example.
- 82 • <!-- description --> is a placeholder for elements from some "other" namespace (like ##other in
83 XSD).
- 84 • Characters are appended to elements, attributes, and <!-- descriptions --> as follows: "?" (0 or 1),
85 "*" (0 or more), "+" (1 or more). The characters "[" and "]" are used to indicate that contained
86 items are to be treated as a group with respect to the "?", "*", or "+" characters.
- 87 • The XML namespace prefixes (defined below) are used to indicate the namespace of the element
88 being defined.

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

1.4 Namespace

The XML namespace **[XML-ns]** URI that **MUST** be used by implementations of this specification is:

```
http://docs.oasis-open.org/ws-tx/wsba/2006/06
```

1.4.1 Prefix Namespace

The following namespaces are used in this document:

Prefix	Namespace
wscoor	http://docs.oasis-open.org/ws-tx/wscoor/2006/06
wsba	http://docs.oasis-open.org/ws-tx/wsba/2006/06

1.5 XSD and WSDL Files

Dereferencing the XML namespace defined in [section 1.4](#) will produce the Resource Directory Description Language (RDDL) **[RDDL]** document that describes this namespace, including the XML schema **[XML-Schema1]** **[XML-Schema2]** and WSDL **[WSDL]** declarations associated with this specification.

SOAP bindings for the WSDL **[WSDL]**, referenced in the RDDL **[RDDL]** document, **MUST** use "document" for the *style* attribute.

1.6 Protocol Elements

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

1.7 Normative References

- [RDDL]** Jonathan Borden, Tim Bray, eds. "Resource Directory Description Language (RDDL) 2.0", <http://www.openhealth.org/RDDL/20040118/rddl-20040118.html>, January 2004.
- [RFC2119]** S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels", <http://www.ietf.org/rfc/rfc2119.txt>, IETF RFC 2119, March 1997.
- [SOAP 1.1]** W3C Note, "SOAP: Simple Object Access Protocol 1.1," <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>, 08 May 2000.
- [SOAP 1.2]** W3C Recommendation, "SOAP Version 1.2 Part 1: Messaging Framework", <http://www.w3.org/TR/soap12-part1/>, June 2003.
- [XML]** W3C Recommendation, "Extensible Markup Language (XML) 1.0 (Fourth Edition)," <http://www.w3.org/TR/2006/REC-xml-20060816>, 16 August 2006.
- [XML-ns]** W3C Recommendation, "Namespaces in XML 1.0 (Second Edition)," <http://www.w3.org/TR/2006/REC-xml-names-20060816>, 16 August 2006.
- [XML-Schema1]** W3C Recommendation, "XML Schema Part 1: Structures Second Edition," <http://www.w3.org/TR/2004/REC-xmlschema-1-20041028>, 28 October 2004.
- [XML-Schema2]** W3C Recommendation, "XML Schema Part 2: Datatypes Second Edition," <http://www.w3.org/TR/2004/REC-xmlschema-2-20041028>, 28 October 2004.
- [WSCOOR]** Web Services Coordination (WS-Coordination), "<http://docs.oasis-open.org/ws-tx/wscoor/2006/06>"

128	[WSDL]	Web Services Description Language (WSDL) 1.1
129		"http://www.w3.org/TR/2001/NOTE-wsdl-20010315"
130	[WSADDR]	Web Services Addressing (WS-Addressing) 1.0, W3C Recommendation,
131		http://www.w3.org/2005/08/addressing
132	[WSPOLICY]	Web Services Policy 1.2 – Framework (WS-Policy),
133		http://www.w3.org/Submission/2006/SUBM-WS-Policy-20060425/ , W3C Member
134		Submission, 25 April 2006.
135	[WSPOLICYATTACH]	Web Services Policy 1.2 – Attachment (WS-PolicyAttachment),
136		http://www.w3.org/Submission/2006/SUBM-WS-PolicyAttachment-20060425/ ,
137		W3C Member Submission, 25 April 2006.
138	[WSSec]	OASIS Standard 200401, March 2004, "Web Services Security: SOAP Message
139		Security 1.0 (WS-Security 2004), " http://docs.oasis-open.org/wss/2004/01/oasis-
140		200401-wss-soap-message-security-1.0.pdf
141	[WSSecPolicy]	Web Services Security Policy Language (WS-SecurityPolicy),
142		http://schemas.xmlsoap.org/ws/2005/07/securitypolicy/ , Microsoft, VeriSign, IBM,
143		RSA Security, December 2002
144	[WSSecConv]	Web Services Secure Conversation Language (WS-SecureConversation),
145		http://schemas.xmlsoap.org/ws/2005/02/sc/ , OpenNetwork, Layer7, Netegrity,
146		Microsoft, Reactivity, IBM, VeriSign, BEA Systems, Oblix, RSA Security, Ping
147		Identity, Westbridge, Computer Associates, February 2005
148	[WSTrust]	Web Services Trust Language (WS-Trust),
149		http://schemas.xmlsoap.org/ws/2005/02/trust/ , OpenNetwork, Layer7, Netegrity,
150		Microsoft, Reactivity, VeriSign, IBM, BEA Systems, Oblix, RSA Security, Ping
151		Identity, Westbridge, Computer Associates, February 2005
152		

153 2 Business Activity Context

154 This section describes the Business Activity usage of WS-Coordination protocols.

155 WS-BusinessActivity builds on WS-Coordination **[WSCOOR]**, which defines an Activation service, a
156 Registration service, and a CoordinationContext type. Example message flows and a complete
157 description of creating and registering for coordinated activities is found in WS-Coordination **[WSCOOR]**.

158 The Business Activity coordination context is a CoordinationContext type with a coordination type defined
159 in this specification. Business Activity application messages that propagate a coordination context **MUST**
160 use a Business Activity coordination context. If these application messages use a SOAP binding, the
161 Business Activity coordination context **MUST** flow as a SOAP header in the message.

162 WS-BusinessActivity adds the following semantics to the CreateCoordinationContext operation on the
163 Activation service:

- 164 • If the request includes the CurrentContext element, the target coordinator is interposed as a
165 subordinate to the coordinator stipulated inside the CurrentContext element.
- 166 • If the request does not include a CurrentContext element, the target coordinator creates a new
167 activity and acts as the root.

168 A coordination context **MAY** have an Expires element. This element specifies the period, measured from
169 the point in time at which the context was first created or received, after which a business activity **MAY** be
170 terminated solely due to its length of operation. From that point forward, the coordinator **MAY** elect to
171 unilaterally cancel or compensate the activity, as appropriate, so long as it has not made a close decision.
172 Similarly, a participant **MAY** elect to exit the activity so long as it has not already decided to complete.

173 A coordination context **MAY** have additional elements for extensibility.

174 3 Coordination Types and Protocols

175 Business Activities support two coordination types and two protocol types. Either protocol type MAY be
176 used with either coordination type.

177 One of the following two URIs MUST be used to specify a Business Activity CoordinationContext type:

```
178 http://docs.oasis-open.org/ws-tx/wsba/2006/06/AtomicOutcome  
179 http://docs.oasis-open.org/ws-tx/wsba/2006/06/MixedOutcome
```

180 A coordinator for an AtomicOutcome coordination type MUST direct all participants either to close or to
181 compensate. A coordinator for a MixedOutcome coordination type MUST direct all participants to an
182 outcome but MAY direct each individual participant to close or compensate. All Business Activity
183 coordinators MUST implement the AtomicOutcome coordination type. A Business Activity coordinator
184 MAY implement the MixedOutcome coordination type.

185 The Coordination protocols for business activities are summarized below with names relative to the wsba
186 base name:

- 187 • **BusinessAgreementWithParticipantCompletion:** A participant registers for this protocol with its
188 coordinator, so that its coordinator can manage it. A participant knows when it has completed all
189 work for a business activity.
- 190 • **BusinessAgreementWithCoordinatorCompletion:** A participant registers for this protocol with
191 its coordinator, so that its coordinator can manage it. A participant relies on its coordinator to tell it
192 when it has received all requests to perform work within the business activity.

193 3.1 Preconditions

194 The correct operation of the protocols requires that a number of preconditions must be established prior
195 to the processing:

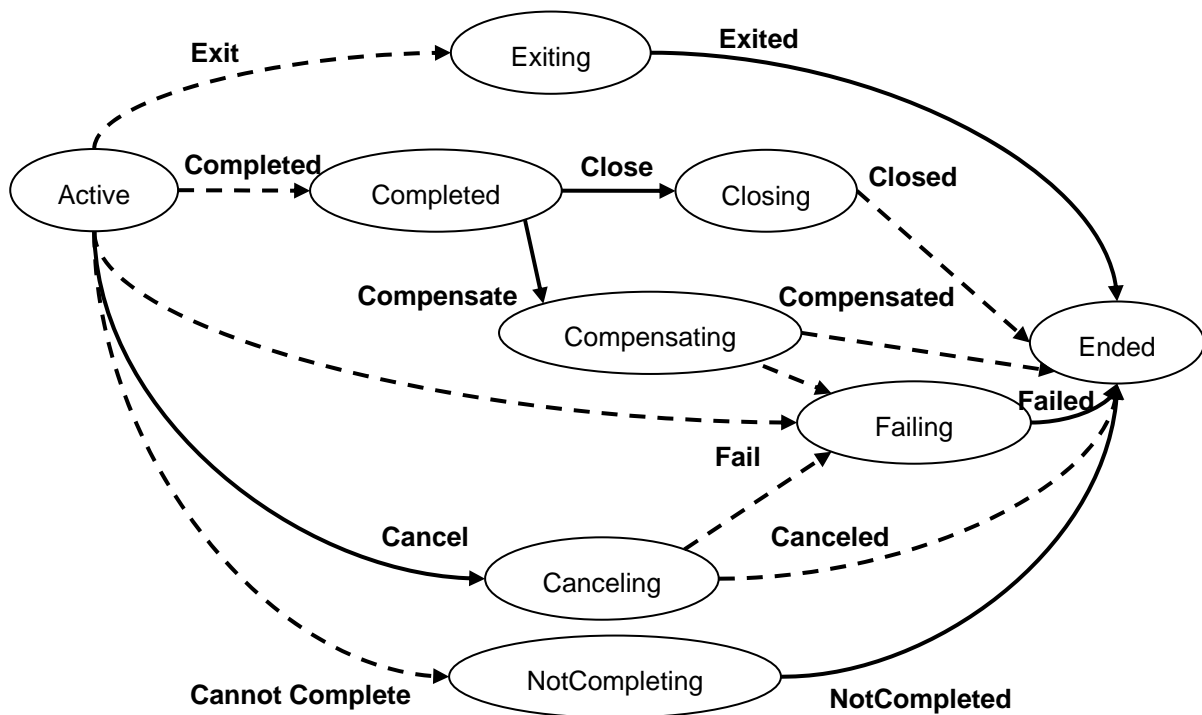
- 196 1. The source SHOULD have knowledge of the destination's policies, if any, and the source
197 SHOULD be capable of formulating messages that adhere to this policy.
- 198 2. If a secure exchange of messages is required, then the source and destination MUST have
199 appropriate security credentials (such as transport-level security credentials or security tokens) in
200 order to protect messages.

201 3.2 BusinessAgreementWithParticipantCompletion Protocol

202 The state diagram in [Figure 1](#) illustrates the abstract behavior of the protocol between a coordinator and a
203 participant. The states in the [Figure 1](#) reflect the view an individual participant or coordinator has of its
204 state in the protocol at a given point in time. As messages take time to be delivered, the views of the
205 coordinator and a participant may temporarily differ. Omitted are details such as resending of messages
206 or the exchange of error messages due to protocol error. Refer to [Appendix B: State Tables for the
207 Agreement Protocols](#) for a detailed description of this protocol.

208 Participants that register for this protocol MUST use the following protocol identifier:

```
209 http://docs.oasis-open.org/ws-tx/wsba/2006/06/ParticipantCompletion
```



Coordinator generated →
— Participant generated →

Figure 1: BusinessAgreementWithParticipantCompletion abstract state diagram

210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234

The coordinator accepts:

Completed

Upon receipt of this notification, the coordinator knows that the participant has completed all processing related to the protocol instance. For the next protocol message the coordinator **MUST** send a Close or Compensate notification to indicate the final outcome of the protocol instance. After sending the Completed notification, a participant **MUST NOT** participate in any further work under that activity.

Fail

Upon receipt of this notification, the coordinator knows that the participant has failed during the Active, Canceling or Compensating states; the state of the work performed by the participant is undetermined. For the next protocol message the coordinator **MUST** send a Failed notification. This notification carries a QName defined in schema indicating the cause of the failure.

Compensated

After transmitting this notification, the participant **SHOULD** forget about the activity. Upon receipt of this notification, the coordinator knows that the participant has successfully compensated all processing related to the protocol instance; the coordinator **SHOULD** forget its state about that participant.

Closed

After transmitting this notification, the participant **SHOULD** forget about the activity. Upon receipt of this notification, the coordinator knows that the participant has finalized the protocol instance successfully; the coordinator **SHOULD** forget its state about that participant.

Canceled

235 After transmitting this notification, the participant SHOULD forget about the activity. Upon receipt
236 of this notification, the coordinator knows that the participant has successfully canceled all
237 processing related to the protocol instance; the coordinator SHOULD forget its state about that
238 participant.

239 **Exit**

240 Upon receipt of this notification, the coordinator knows that the participant will no longer
241 participate in the business activity, and any pending work was discarded by the participant and
242 any work performed by the participant related to the protocol instance was successfully canceled.
243 For the next protocol message the coordinator MUST send an Exited notification. The Exit
244 message MAY be sent by a participant only from the Active or Completing states.

245 **CannotComplete**

246 Upon receipt of this notification, the coordinator knows that the participant has determined that it
247 cannot successfully complete all processing related to the protocol instance. Any pending work
248 was discarded by the participant and any work performed by the participant related to the protocol
249 instance was successfully canceled. For the next protocol message the coordinator MUST send a
250 NotCompleted notification. After sending the CannotComplete notification, a participant MUST
251 NOT participate in any further work under that activity. The CannotComplete message MAY be
252 sent by a participant only from the Active state.

253 The participant accepts:

254 **Close**

255 Upon receipt of this notification, the participant knows the protocol instance is to be ended
256 successfully. For the next protocol message the participant MUST send a Closed notification to
257 end the protocol instance.

258 **Cancel**

259 Upon receipt of this notification, the participant knows that the work being done has to be
260 canceled. For the next protocol message, the participant MUST send either a Canceled or Fail
261 message. A Canceled message SHOULD be sent by the participant if the work is successfully
262 canceled; this also ends the protocol instance. A Fail message SHOULD be sent by the
263 participant if the work was not successfully canceled.

264 **Compensate**

265 Upon receipt of this notification, the participant knows that the work being done should be
266 compensated. For the next protocol message the participant MUST send a Compensated or Fail
267 notification. A Compensated message SHOULD be sent by the participant if the work is
268 successfully compensated; this also ends the protocol instance. A Fail message SHOULD be
269 sent by the participant if the work was not successfully compensated.

270 **Failed**

271 After transmitting this notification, the coordinator SHOULD forget about the participant. Upon
272 receipt of this notification, the participant knows that the coordinator is aware of a failure and no
273 further actions are required of the participant; the participant SHOULD forget the activity.

274 **Exited**

275 After transmitting this notification, the coordinator SHOULD forget about the participant. Upon
276 receipt of this notification, the participant knows that the coordinator is aware the participant will
277 no longer participate in the activity; the participant SHOULD forget the activity.

278 **NotCompleted**

279 After transmitting this notification, the coordinator SHOULD forget about the participant. Upon
280 receipt of this notification, the participant knows that the coordinator is aware that the participant
281 cannot complete all processing related to the protocol instance and that the participant will no
282 longer participate in the activity; the participant SHOULD forget the activity.

283

284 Both the coordinator and participant accept:

285 **GetStatus**

286 This message requests the current state of a coordinator or participant. In response the
287 coordinator or participant returns a Status message containing a QName indicating which column
288 of the state table [[Appendix B: State Tables for the Agreement Protocols](#)] the coordinator or
289 participant is currently in. GetStatus never provokes a state change.

290 For example, a coordinator that is waiting for a participant to initiate the
291 BusinessAgreementWithParticipantCompletion may use this message to confirm that the
292 participant is in one of the expected states: wsba:Active or wsba:Completed. If the participant has
293 forgotten the activity the Status response MUST be wsba:Ended.

294 **Status**

295 This message is received in response to a GetStatus request. The message includes a QName
296 indicating the state of the coordinator or participant to which the request was sent. For example, if
297 a participant is in the closing state as indicated by the state table, it would return wsba:Closing.

298

299 The coordinator may enter a condition in which it has sent a protocol message and it receives a protocol
300 message from the participant that is consistent with the former state, not the current state. In this case,
301 the coordinator MUST revert to the prior state, accept the notification from the participant, and continue
302 the protocol from that point. If the participant detects this condition, it MUST discard the inconsistent
303 protocol message from the coordinator.

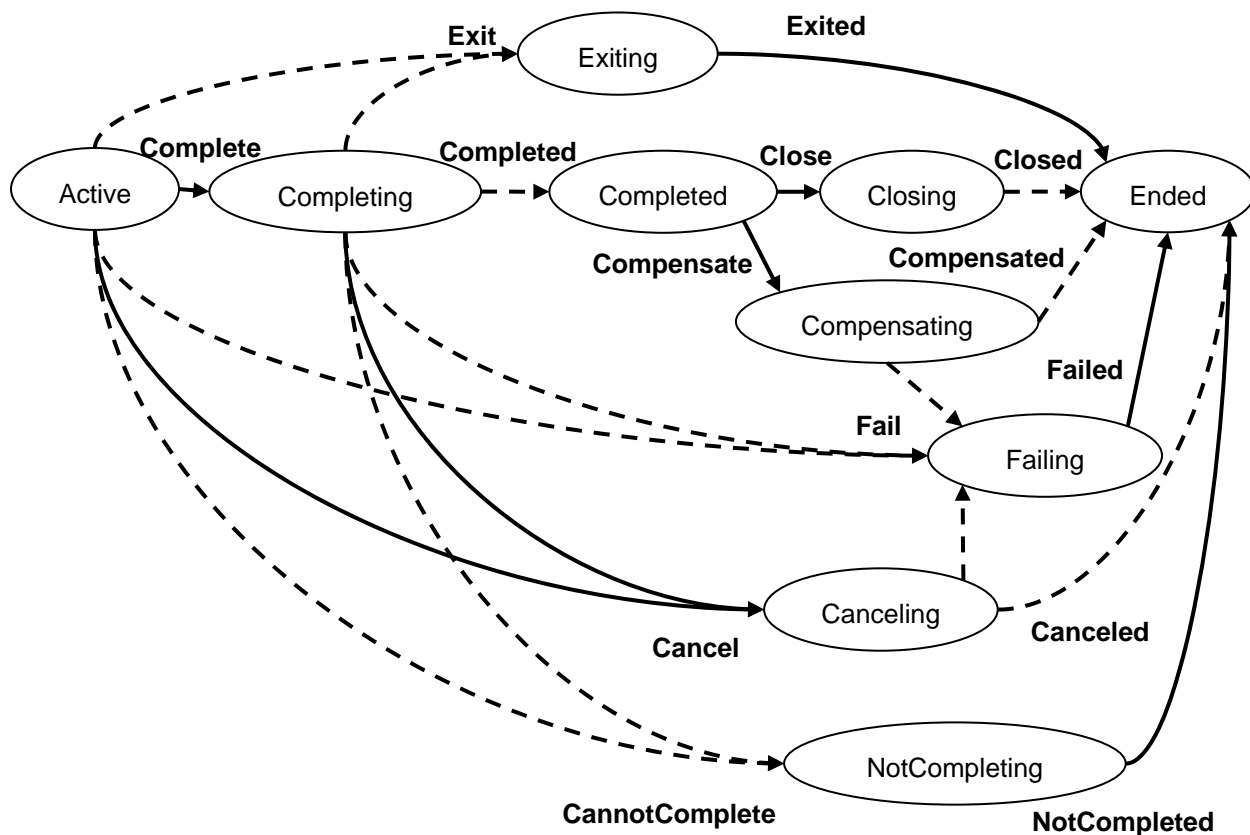
304 A party MUST be prepared to receive duplicate notifications. If a duplicate message is received it MUST
305 be treated as specified in the state tables [[Appendix B: State Tables for the Agreement Protocols](#)].

306 **3.3 BusinessAgreementWithCoordinatorCompletion Protocol**

307 The BusinessAgreementWithCoordinatorCompletion protocol is the same as the
308 BusinessAgreementWithParticipantCompletion protocol, except that a participant relies on its coordinator
309 to tell it when it has received all requests to do work within the business activity.

310 Participants that register for this protocol MUST use the following protocol identifier:

311 `http://docs.oasis-open.org/ws-tx/wsba/2006/06/CoordinatorCompletion`



312 → Coordinator generated - - - Participant generated

313 Figure 2: BusinessAgreementWithCoordinatorCompletion abstract state diagram

314

315 The BusinessAgreementWithCoordinatorCompletion protocol redefines the following notifications in

316 [Section 3.2](#) above:

317

318 The coordinator accepts:

319 **Fail**

320 Upon receipt of this notification, the coordinator knows that the participant has failed during the

321 Active, Canceling, Completing or Compensating states; the state of the work performed by the

322 participant is undetermined. For the next protocol message the coordinator MUST send a Failed

323 notification. This notification carries a QName defined in schema indicating the cause of the

324 failure.

325 **CannotComplete**

326 Upon receipt of this notification, the coordinator knows that the participant has determined that it

327 cannot successfully complete all processing related to the protocol instance. Any pending work

328 was discarded by the participant and any work performed by the participant related to the protocol

329 instance was successfully canceled. For the next protocol message the coordinator MUST send a

330 NotCompleted notification. After sending the CannotComplete notification, a participant MUST

331 NOT participate in any further work under that activity. The CannotComplete message MAY be

332 sent by a participant only from the Active or Completing states.

333 In addition to the notifications in [Section 3.2](#) above, the BusinessAgreementWithCoordinatorCompletion
334 protocol adds the following notification:

335

336 The participant accepts:

337 **Complete**

338 Upon receipt of this notification the participant knows that it will receive no new requests for work
339 within the business activity. The participant completes application processing and if successful
340 MUST transmit a Completed notification. If unsuccessful the participant MUST transmit an Exit,
341 Fail, or CannotComplete notification.

342

4 Policy Assertions

343
344
345
346
347
348

WS-Policy Framework **[WSPOLICY]** and WS-Policy Attachment **[WSPOLICYATTACH]** collectively define a framework, model and grammar for expressing the capabilities, requirements, and general characteristics of entities in an XML Web services-based system. To enable a Web service to describe Business Activity related capabilities and requirements of a service and its operations, this specification defines a pair of Business Agreement policy assertions that leverage the WS-Policy framework **[WSPOLICY]**.

349

4.1 Assertion Models

350
351
352

The Business Activity policy assertions are provided by a Web service to qualify the Business Activity related processing of messages associated with the particular operation to which the assertions are scoped. The Business Activity policy assertions indicate:

353
354

- Whether the sender of an input message MAY or MUST include an AtomicOutcome coordination context flowed with the message. The coordination type of such a context MUST be the following:

355

```
http://docs.oasis-open.org/ws-tx/wsba/2006/06/AtomicOutcome
```

356
357

- Whether the sender of an input message MAY or MUST include a MixedOutcome coordination context flowed with the message. The coordination type of such a context MUST be the following:

358

```
http://docs.oasis-open.org/ws-tx/wsba/2006/06/MixedOutcome
```

359

4.2 Normative Outlines

360

The normative outlines for the Business Activity policy assertions are:

361
362
363

```
<wsba:BAAtomicOutcomeAssertion [wsp:Optional="true"]? ... >  
...  
</wsba:BAAtomicOutcomeAssertion>
```

364

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

365

/wsba:BAAtomicOutcomeAssertion

366
367
368
369
370
371

A policy assertion that specifies that the sender of an input message MUST include a coordination context for a Business Activity with AtomicOutcome coordination type flowed with the message. From the perspective of the requester, the target service that processes the activity MUST behave as if it had participated in the activity. For application messages that use a SOAP binding, the Business Activity coordination context MUST flow as a SOAP header in the message.

372

/wsba:BAAtomicOutcomeAssertion/@wsp:Optional="true"

373
374

Per WS-Policy **[WSPOLICY]**, this is compact notation for two policy alternatives, one with and one without the assertion.

375
376
377

```
<wsba:BAMixedOutcomeAssertion [wsp:Optional="true"]? ... >  
...  
</wsba:BAMixedOutcomeAssertion>
```

378

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

379

/wsba:BAMixedOutcomeAssertion

380
381
382
383

A policy assertion that specifies that the sender of an input message MUST include a coordination context for a Business Activity with MixedOutcome coordination type flowed with the message. From the perspective of the requester, the target service that processes the activity MUST behave as if it had participated in the activity. For application messages that use a SOAP

384 binding, the Business Activity coordination context MUST flow as a SOAP header in the
385 message.

386 **/wsba: BAMixedOutcomeAssertion/@wsp:Optional="true"**

387 Per WS-Policy [WSPOLICY], this is compact notation for two policy alternatives, one with and
388 one without the assertion.

389 4.3 Assertion Attachment

390 Because the Business Activity policy assertions indicate Business Activity related behavior for a single
391 operation, the assertions have an Operation Policy Subject [WSPOLICYATTACH].

392 WS-PolicyAttachment [WSPOLICYATTACH] defines two WSDL [WSDL] policy attachment points with
393 an Operation Policy Subject:

- 394 • wsdl:portType/wsdl:operation – A policy expression containing a Business Activity policy
395 assertion MUST NOT be attached to a wsdl:portType; the Business Activity policy assertions
396 specify a concrete behavior whereas the wsdl:portType is an abstract construct.
- 397 • wsdl:binding/wsdl:operation – A policy expression containing a Business Activity policy assertion
398 SHOULD be attached to a wsdl:binding.

399 4.4 Assertion Example

400 An example use of the Business Activity policy assertion follows:

```
401 (01) <wsdl:definitions
402 (02)     targetNamespace="hotel.example.com"
403 (03)     xmlns:tns="hotel.example.com"
404 (04)     xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
405 (05)     xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
406 (06)     xmlns:wsba="http://docs.oasis-open.org/ws-tx/wsba/2006/06"
407 (07)     xmlns:wssu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
408 wssecurity-utility-1.0.xsd" >
409 (08)     <wsp:Policy wsu:Id="BAAtomicPolicy" >
410 (09)         <wsba:BAAtomicOutcomeAssertion/>
411 (10)         <!-- omitted assertions -->
412 (11)     </wsp:Policy>
413 (12) <!-- omitted elements -->
414 (13) <wsdl:binding name="HotelBinding" type="tns:HotelPortType" >
415 (14)     <!-- omitted elements -->
416 (15)     <wsdl:operation name="ReserveRoom" >
417 (16)         <wsp:PolicyReference URI="#BAAtomicPolicy" wsdl:required="true"/>
418 (17)         <!-- omitted elements -->
419 (18)     </wsdl:operation>
420 (19) </wsdl:binding>
421 (20) </wsdl:definitions>
```

422

423 Lines (8-11) are a policy expression that includes a Business Activity policy assertion (Line 9) to indicate
424 that a coordination context for a Business Activity with an AtomicOutcome, expressed in WS-Coordination
425 [WS-COOR] format, MUST be used.

426 Lines (13-19) are a WSDL [**WSDL**] binding. Line (16) indicates that the policy in Lines (8-11) applies to
427 this binding, specifically indicating that a coordination context for a Business Activity with an
428 AtomicOutcome MUST flow inside "ReserveRoom" messages.

429

5 Security Considerations

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

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

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

- 442 • Attaching a nonce to each message and using it in a derived key function with the shared secret
- 443 • Using a derived key sequence and switch "generations"
- 444 • Closing and re-establishing a security context (not possible for delegated keys)
- 445 • Exchanging new secrets between the parties (not possible for delegated keys)

446 It should be noted that the mechanisms listed above are independent of the Security Context Token
447 (SCT) and secret returned when the coordination context is created. That is, the keys used to secure the
448 channel may be independent of the key used to prove the right to register with the activity.

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

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

- 456 • **Message alteration** – Alteration is prevented by including signatures of the message information
457 using WS-Security [WSSec].
- 458 • **Message disclosure** – Confidentiality is preserved by encrypting sensitive data using WS-
459 Security [WSSec].
- 460 • **Key integrity** – Key integrity is maintained by using the strongest algorithms possible (by
461 comparing secured policies – see WS-Policy [WSPOLICY] and WS-SecurityPolicy
462 [WSSecPolicy]).
- 463 • **Authentication** – Authentication is established using the mechanisms described in WS-Security
464 [WSSec] and WS-Trust [WSTrust]. Each message is authenticated using the mechanisms
465 described in WS-Security [WSSec].
- 466 • **Accountability** – Accountability is a function of the type of and string of the key and algorithms
467 being used. In many cases, a strong symmetric key provides sufficient accountability. However, in
468 some environments, strong PKI signatures are required.
- 469 • **Availability** – Many services are subject to a variety of availability attacks. Replay is a common
470 attack and it is RECOMMENDED that this be addressed as described in the next bullet. Other
471 attacks, such as network-level denial of service attacks are harder to avoid and are outside the
472 scope of this specification. That said, care should be taken to ensure that minimal processing be
473 performed prior to any authenticating sequences.
- 474 • **Replay** – Messages may be replayed for a variety of reasons. To detect and eliminate this attack,
475 mechanisms should be used to identify replayed messages such as the timestamp/nonce

476
477

outlined in WS-Security [**WS**Sec]. Alternatively, and optionally, other technologies, such as sequencing, can also be used to prevent replay of application messages.

478 6 Use of WS-Addressing Headers

479 The protocols defined in WS-BusinessActivity use a "one way" message exchange pattern consisting of a
480 sequence of notification messages between a coordinator and a participant. There are two types of
481 notification messages used in these protocols:

- 482 • A notification message is a terminal message when it indicates the end of a
483 coordinator/participant relationship. **Closed, Compensated, Canceled, Exited, NotCompleted**
484 and **Failed** are terminal messages as are the protocol faults defined in WS-Coordination
485 **[WSCOOR]**.
- 486 • A notification message is a non-terminal message when it does not indicate the end of a
487 coordinator/participant relationship. **Complete, Completed, Close, Compensate, Cancel, Exit,**
488 **CannotComplete** and **Fail** are non-terminal messages.

489 The following statements define addressing interoperability requirements for the respective Business
490 Activity message types:

491 Non-terminal notification messages

- 492 • MUST include a [source endpoint] property whose [address] property is not set to
493 'http://www.w3.org/2005/08/addressing/anonymous' or
494 'http://www.w3.org/2005/08/addressing/none'

495 Both terminal and non-terminal notification messages

- 496 • MUST include a [reply endpoint] property whose [address] property is set to
497 'http://www.w3.org/2005/08/addressing/none'

498 Notification messages used in WS-BusinessActivity protocols MUST include as the [action] property an
499 action URI that consists of the wsba namespace URI concatenated with the "/" character and the element
500 name of the message. For example:

501 `http://docs.oasis-open.org/ws-tx/wsba/2006/06/Complete`

502 Notification messages are normally addressed according to section 3.3 of WS-Addressing 1.0 – Core
503 **[WSADDR]** by both coordinators and participants using the Endpoint References initially obtained during
504 the Register-RegisterResponse exchange. If a [source endpoint] property is present in a notification
505 message, it MAY be used by the recipient. Cases exist where a coordinator or participant has forgotten
506 an activity that is completed and needs to respond to a resent protocol message. In such cases, the
507 [source endpoint] property SHOULD be used as described in section 3.3 of WS-Addressing 1.0 – Core
508 **[WSADDR]**. Permanent loss of connectivity between a coordinator and a participant in an in-doubt state
509 can result in data corruption.

510 Protocol faults raised by a coordinator or participant during the processing of a notification message are
511 terminal notifications and MUST be composed using the same mechanisms as other terminal notification
512 messages.

513 All messages are delivered using connections initiated by the sender.

514 A. Acknowledgements

515 This document is based on initial contribution to OASIS WS-TX Technical Committee by the following
516 authors: Luis Felipe Cabrera (Microsoft), George Copeland (Microsoft), Max Feingold (Microsoft), Robert
517 W Freund (Hitachi), Tom Freund (IBM), Sean Joyce (IONA), Johannes Klein (Microsoft), David
518 Langworthy (Microsoft), Mark Little (JBoss Inc.), Frank Leymann (IBM), Eric Newcomer (IONA), David
519 Orchard (BEA Systems), Ian Robinson (IBM), Tony Storey (IBM), Satish Thatte (Microsoft).

520

521 The following individuals have provided invaluable input into the initial contribution: Francisco Curbera
522 (IBM), Doug Davis (IBM), Gert Drapers (Microsoft), Don Ferguson (IBM), Kirill Gavrylyuk (Microsoft), Dan
523 House (IBM), Oisin Hurley (IONA), Thomas Mikalsen (IBM), Jagan Peri (Microsoft), John Shewchuk
524 (Microsoft), Stefan Tai (IBM).

525

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

527 **Participants:**

528 Charlton Barreto, Adobe Systems, Inc.
529 Martin Chapman, Oracle
530 Kevin Conner, JBoss Inc.
531 Paul Cotton, Microsoft Corporation
532 Doug Davis, IBM
533 Colleen Evans, Microsoft Corporation
534 Max Feingold, Microsoft Corporation
535 Thomas Freund, IBM
536 Robert Freund, Hitachi, Ltd.
537 Peter Furniss, Choreology Ltd.
538 Marc Goodner, Microsoft Corporation
539 Alastair Green, Choreology Ltd.
540 Daniel House, IBM
541 Ram Jeyaraman, Microsoft Corporation
542 Paul Knight, Nortel Networks Limited
543 Mark Little, JBoss Inc.
544 Jonathan Marsh, Microsoft Corporation
545 Monica Martin, Sun Microsystems
546 Joseph Fialli, Sun Microsystems
547 Eric Newcomer, IONA Technologies
548 Eisaku Nishiyama, Hitachi, Ltd.
549 Alain Regnier, Ricoh Company, Ltd.
550 Ian Robinson, IBM
551 Tom Rutt, Fujitsu Limited
552 Andrew Wilkinson, IBM

553

B. State Tables for the Agreement Protocols

554 The following state tables show state transitions that occur in the receiver when a protocol message is
555 received or in the sender when a protocol message is sent.

556 Each cell in the tables uses the following convention:

557

Legend
<i>Action to take</i>
Next state

558

559 Each state supports a number of possible events. Expected events are processed by taking the
560 prescribed action and transitioning of the next state. Unexpected protocol messages MUST result in a
561 fault message as defined in the state tables. These faults MUST use a standard fault code defined in
562 WS-Coordination [WS-COOR].

563 The following rules need to be applied when reading the state tables in this document:

564 • For the period of time that a protocol message is in transit the sender and recipient states will be
565 different.

566 The sender of a protocol message transitions to the "next state" when the message is first sent.

567 The recipient of a protocol message transitions to the "next state" when the message is first
568 received.

569 • As described earlier in this document, if the coordinator receives a protocol message from the
570 participant that is consistent with the former state of the coordinator then the coordinator reverts
571 to its prior state, accepts the notification from the participant, and continues the protocol from that
572 point.

573 The GetStatus and Status protocol messages are not included in the tables as these never result in a
574 change of state.

575 These tables present the view of a coordinator or participant with respect to a single partner. A
576 coordinator with multiple participants can be understood as a collection of independent coordinator state
577 machines, each with its own state.

578

B.1 Participant view of BusinessAgreementWithParticipantCompletion

579

BusinessAgreementWithParticipantCompletion protocol (Participant View)										
Inbound Events	States									
	Active	Canceling	Completed	Closing	Compensating	Failing (Active, Canceling)	Failing (Compensating)	NotCompleting	Exiting	Ended
Cancel	Canceling	Ignore Canceling	Resend Completed Completed	Ignore Closing	Ignore Compensating	Resend Fail Failing-*	Ignore Failing- Compensating	Resend CannotComplete NotCompleting	Resend Exit Exiting	Send Canceled Ended
Close	Invalid State Active	Invalid State Canceling	Closing	Ignore Closing	Invalid State Compensating	Invalid State Failing-*	Invalid State Failing- Compensating	Invalid State NotCompleting	Invalid State Exiting	Send Closed Ended
Compensate	Invalid State Active	Invalid State Canceling	Compensating	Invalid State Closing	Ignore Compensating	Invalid State Failing-*	Resend Fail Failing- Compensating	Invalid State NotCompleting	Invalid State Exiting	Send Compensated Ended
Failed	Invalid State Active	Invalid State Canceling	Invalid State Completed	Invalid State Closing	Invalid State Compensating	Forget Ended	Forget Ended	Invalid State NotCompleting	Invalid State Exiting	Ignore Ended
Exited	Invalid State Active	Invalid State Canceling	Invalid State Completed	Invalid State Closing	Invalid State Compensating	Invalid State Failing-*	Invalid State Failing- Compensating	Invalid State NotCompleting	Forget Ended	Ignore Ended
NotCompleted	Invalid State Active	Invalid State Canceling	Invalid State Completed	Invalid State Closing	Invalid State Compensating	Invalid State Failing-*	Invalid State Failing- Compensating	Forget Ended	Invalid State Exiting	Ignore Ended

580

581

582

583

BusinessAgreementWithParticipantCompletion protocol (Participant View)									
Outbound Events	States								
	Active	Canceling	Completed	Closing	Compensating	Failing (Active, Canceling, Compensating)	NotCompleting	Exiting	Ended
Exit	Exiting	<i>Invalid State</i> Canceling	<i>Invalid State</i> Completed	<i>Invalid State</i> Closing	<i>Invalid State</i> Compensating	<i>Invalid State</i> Failing-*	<i>Invalid State</i> NotCompleting	Exiting	<i>Invalid State</i> Ended
Completed	Completed	<i>Invalid State</i> Canceling	Completed	<i>Invalid State</i> Closing	<i>Invalid State</i> Compensating	<i>Invalid State</i> Failing-*	<i>Invalid State</i> NotCompleting	<i>Invalid State</i> Exiting	<i>Invalid State</i> Ended
Fail	Failing- Active	Failing- Canceling	<i>Invalid State</i> Completed	<i>Invalid State</i> Closing	Failing- Compensating	Failing-*	<i>Invalid State</i> NotCompleting	<i>Invalid State</i> Exiting	<i>Invalid State</i> Ended
CannotComplete	NotCompleting	<i>Invalid State</i> Canceling	<i>Invalid State</i> Completed	<i>Invalid State</i> Closing	<i>Invalid State</i> Compensating	<i>Invalid State</i> Failing-*	NotCompleting	<i>Invalid State</i> Exiting	<i>Invalid State</i> Ended
Canceled	<i>Invalid State</i> Active	<i>Forget</i> Ended	<i>Invalid State</i> Completed	<i>Invalid State</i> Closing	<i>Invalid State</i> Compensating	<i>Invalid State</i> Failing-*	<i>Invalid State</i> NotCompleting	<i>Invalid State</i> Exiting	Ended
Closed	<i>Invalid State</i> Active	<i>Invalid State</i> Canceling	<i>Invalid State</i> Completed	<i>Forget</i> Ended	<i>Invalid State</i> Compensating	<i>Invalid State</i> Failing-*	<i>Invalid State</i> NotCompleting	<i>Invalid State</i> Exiting	Ended
Compensated	<i>Invalid State</i> Active	<i>Invalid State</i> Canceling	<i>Invalid State</i> Completed	<i>Invalid State</i> Closing	<i>Forget</i> Ended	<i>Invalid State</i> Failing-*	<i>Invalid State</i> NotCompleting	<i>Invalid State</i> Exiting	Ended

584

585

586

B.2 Coordinator view of BusinessAgreementWithParticipantCompletion

587

BusinessAgreementWithParticipantCompletion protocol (Coordinator View)										
Inbound Events	States									
	Active	Canceling	Completed	Closing	Compensating	Failing (Active, Canceling)	Failing (Compensating)	NotCompleting	Exiting	Ended
Exit	Exiting	Exiting	<i>Invalid State</i> Completed	<i>Invalid State</i> Closing	<i>Invalid State</i> Compensating	<i>Invalid State</i> Failing-*	<i>Invalid State</i> Failing-Compensating	<i>Invalid State</i> NotCompleting	<i>Ignore</i> Exiting	<i>Resend Exited</i> Ended
Completed	Completed	Completed	<i>Ignore</i> Completed	<i>Resend Close</i> Closing	<i>Resend Compensate</i> Compensating	<i>Invalid State</i> Failing-*	<i>Ignore</i> Failing-Compensating	<i>Invalid State</i> NotCompleting	<i>Invalid State</i> Exiting	<i>Ignore</i> Ended
Fail	Failing-Active	Failing-Canceling	<i>Invalid State</i> Completed	<i>Invalid State</i> Closing	Failing-Compensating	Failing-*	Failing-Compensating	<i>Invalid State</i> NotCompleting	<i>Invalid State</i> Exiting	<i>Resend Failed</i> Ended
CannotComplete	NotCompleting	NotCompleting	<i>Invalid State</i> Completed	<i>Invalid State</i> Closing	<i>Invalid State</i> Compensating	<i>Invalid State</i> Failing-*	<i>Invalid State</i> Failing-Compensating	<i>Ignore</i> NotCompleting	<i>Invalid State</i> Exiting	<i>Resend NotCompleted</i> Ended
Canceled	<i>Invalid State</i> Active	<i>Forget</i> Ended	<i>Invalid State</i> Completed	<i>Invalid State</i> Closing	<i>Invalid State</i> Compensating	<i>Invalid State</i> Failing-*	<i>Invalid State</i> Failing-Compensating	<i>Invalid State</i> NotCompleting	<i>Invalid State</i> Exiting	<i>Ignore</i> Ended
Closed	<i>Invalid State</i> Active	<i>Invalid State</i> Canceling	<i>Invalid State</i> Completed	<i>Forget</i> Ended	<i>Invalid State</i> Compensating	<i>Invalid State</i> Failing-*	<i>Invalid State</i> Failing-Compensating	<i>Invalid State</i> NotCompleting	<i>Invalid State</i> Exiting	<i>Ignore</i> Ended
Compensated	<i>Invalid State</i> Active	<i>Invalid State</i> Canceling	<i>Invalid State</i> Completed	<i>Invalid State</i> Closing	<i>Forget</i> Ended	<i>Invalid State</i> Failing-*	<i>Invalid State</i> Failing-Compensating	<i>Invalid State</i> NotCompleting	<i>Invalid State</i> Exiting	<i>Ignore</i> Ended

588

589

590

BusinessAgreementWithParticipantCompletion protocol (Coordinator View)									
Outbound Events	States								
	Active	Canceling	Completed	Closing	Compensating	Failing (Active, Canceling, Compensating)	NotCompleting	Exiting	Ended
Cancel	Canceling	Canceling	Invalid State Completed	Invalid State Closing	Invalid State Compensating	Invalid State Failing-*	Invalid State NotCompleting	Invalid State Exiting	Invalid State Ended
Close	Invalid State Active	Invalid State Canceling	Closing	Closing	Invalid State Compensating	Invalid State Failing-*	Invalid State NotCompleting	Invalid State Exiting	Invalid State Ended
Compensate	Invalid State Active	Invalid State Canceling	Compensating	Invalid State Closing	Compensating	Invalid State Failing-*	Invalid State NotCompleting	Invalid State Exiting	Invalid State Ended
Failed	Invalid State Active	Invalid State Canceling	Invalid State Completed	Invalid State Closing	Invalid State Compensating	Forget Ended	Invalid State NotCompleting	Invalid State Exiting	Ended
Exited	Invalid State Active	Invalid State Canceling	Invalid State Completed	Invalid State Closing	Invalid State Compensating	Invalid State Failing-*	Invalid State NotCompleting	Forget Ended	Ended
NotCompleted	Invalid State Active	Invalid State Canceling	Invalid State Completed	Invalid State Closing	Invalid State Compensating	Invalid State Failing-*	Forget Ended	Invalid State Exiting	Ended

591

592

593

B.3 Participant view of BusinessAgreementWithCoordinatorCompletion

594

BusinessAgreementWithCoordinatorCompletion protocol (Participant View)											
Inbound Events	States										
	Active	Canceling	Completing	Completed	Closing	Compensating	Failing (Active, Canceling, Completing)	Failing (Compensating)	NotCompleting	Exiting	Ended
Cancel	Canceling	Ignore Canceling	Ignore Canceling	Resend Completed Completed	Ignore Closing	Ignore Compensating	Resend Fail Failing-*	Ignore Failing-Compensating	Resend CannotComplete NotCompleting	Resend Exit Exiting	Send Canceled Ended
Complete	Completing	Ignore Canceling	Ignore Completing	Resend Completed Completed	Ignore Closing	Ignore Compensating	Resend Fail Failing-*	Ignore Failing-Compensating	Resend CannotComplete NotCompleting	Resend Exit Exiting	Send Fail Ended
Close	Invalid State Active	Invalid State Canceling	Invalid State Completing	Invalid State Closing	Ignore Closing	Invalid State Compensating	Invalid State Failing-*	Invalid State Failing-Compensating	Invalid State NotCompleting	Invalid State Exiting	Send Closed Ended
Compensate	Invalid State Active	Invalid State Canceling	Invalid State Completing	Invalid State Compensating	Invalid State Closing	Ignore Compensating	Invalid State Failing-*	Resend Fail Failing-Compensating	Invalid State NotCompleting	Invalid State Exiting	Send Compensated Ended
Failed	Invalid State Active	Invalid State Canceling	Invalid State Completing	Invalid State Completed	Invalid State Closing	Invalid State Compensating	Forget Ended	Forget Ended	Invalid State NotCompleting	Invalid State Exiting	Ignore Ended
Exited	Invalid State Active	Invalid State Canceling	Invalid State Completing	Invalid State Completed	Invalid State Closing	Invalid State Compensating	Invalid State Failing-*	Invalid State Failing-Compensating	Invalid State NotCompleting	Forget Ended	Ignore Ended
NotCompleted	Invalid State Active	Invalid State Canceling	Invalid State Completing	Invalid State Completed	Invalid State Closing	Invalid State Compensating	Invalid State Failing-*	Invalid State Failing-Compensating	Forget Ended	Invalid State Exiting	Ignore Ended

595

596

597

BusinessAgreementWithCoordinatorCompletion protocol (Participant View)										
Outbound Events	States									
	Active	Canceling	Completing	Completed	Closing	Compensating	Failing (Active, Canceling, Completing, Compensating)	NotCompleting	Exiting	Ended
Exit	Exiting	Invalid State Canceling	Exiting	Invalid State Completed	Invalid State Closing	Invalid State Compensating	Invalid State Failing-*	Invalid State NotCompleting	Exiting	Invalid State Ended
Completed	Invalid State Active	Invalid State Canceling	Completed	Completed	Invalid State Closing	Invalid State Compensating	Invalid State Failing-*	Invalid State NotCompleting	Invalid State Exiting	Invalid State Ended
Fail	Failing-Active	Failing-Canceling	Failing-Completing	Invalid State Completed	Invalid State Closing	Failing-Compensating	Failing-*	Invalid State NotCompleting	Invalid State Exiting	Invalid State Ended
CannotComplete	NotCompleting	Invalid State Canceling	NotCompleting	Invalid State Completed	Invalid State Closing	Invalid State Compensating	Invalid State Failing-*	NotCompleting	Invalid State Exiting	Invalid State Ended
Canceled	Invalid State Active	Forget Ended	Invalid State Completing	Invalid State Completed	Invalid State Closing	Invalid State Compensating	Invalid State Failing-*	Invalid State NotCompleting	Invalid State Exiting	Ended
Closed	Invalid State Active	Invalid State Canceling	Invalid State Completing	Invalid State Completed	Forget Ended	Invalid State Compensating	Invalid State Failing-*	Invalid State NotCompleting	Invalid State Exiting	Ended
Compensated	Invalid State Active	Invalid State Canceling	Invalid State Completing	Invalid State Completed	Invalid State Closing	Forget Ended	Invalid State Failing-*	Invalid State NotCompleting	Invalid State Exiting	Ended

598

599

B.4 Coordinator view of BusinessAgreementWithCoordinatorCompletion

BusinessAgreementWithCoordinatorCompletion protocol (Coordinator View)												
Inbound Events	States											
	Active	Canceling (Active)	Canceling (Completing)	Completing	Completed	Closing	Compensating	Failing (Active, Canceling, Completing)	Failing (Compensating)	NotCompleting	Exiting	Ended
Exit	Exiting	Exiting	Exiting	Exiting	Invalid State Completed	Invalid State Closing	Invalid State Compensating	Invalid State Failing-*	Invalid State Failing-Compensating	Invalid State NotCompleting	Ignore Exiting	Resend Exited Ended
Completed	Invalid State Active	Invalid State Canceling-Active	Completed	Completed	Ignore Completed	Resend Close Closing	Resend Compensate Compensating	Invalid State Failing-*	Ignore Failing-Compensating	Invalid State NotCompleting	Invalid State Exiting	Ignore Ended
Fail	Failing-Active	Failing-Canceling	Failing-Canceling	Failing-Completing	Invalid State Completed	Invalid State Closing	Failing-Compensating	Ignore Failing-*	Ignore Failing-Compensating	Invalid State NotCompleting	Invalid State Exiting	Resend Failed Ended
CannotComplete	NotCompleting	NotCompleting	NotCompleting	NotCompleting	Invalid State Completed	Invalid State Closing	Invalid State Compensating	Invalid State Failing-*	Invalid State Failing-Compensating	Ignore NotCompleting	Invalid State Exiting	Resend NotCompleted Ended
Canceled	Invalid State Active	Forget Ended	Forget Ended	Invalid State Completing	Invalid State Completed	Invalid State Closing	Invalid State Compensating	Invalid State Failing-*	Invalid State Failing-Compensating	Invalid State NotCompleting	Invalid State Exiting	Ignore Ended
Closed	Invalid State Active	Invalid State Canceling-Active	Invalid State Canceling-Completing	Invalid State Completing	Invalid State Completed	Forget Ended	Invalid State Compensating	Invalid State Failing-*	Invalid State Failing-Compensating	Invalid State NotCompleting	Invalid State Exiting	Ignore Ended
Compensated	Invalid State Active	Invalid State Canceling-Active	Invalid State Canceling-Completing	Invalid State Completing	Invalid State Completed	Invalid State Closing	Forget Ended	Invalid State Failing-*	Invalid State Failing-Compensating	Invalid State NotCompleting	Invalid State Exiting	Ignore Ended

604

605

BusinessAgreementWithCoordinatorCompletion protocol (Coordinator View)										
Outbound Events	States									
	Active	Canceling (Active, Completing)	Completing	Completed	Closing	Compensating	Failing (Active, Canceling, Completing, Compensating)	NotCompleting	Exiting	Ended
Cancel	Canceling- Active	Canceling-*	Canceling- Completing	Invalid State Completed	Invalid State Closing	Invalid State Compensating	Invalid State Failing-*	Invalid State NotCompleting	Invalid State Exiting	Invalid State Ended
Complete	Completing	Invalid State Canceling-*	Completing	Invalid State Completed	Invalid State Closing	Invalid State Compensating	Invalid State Failing-*	Invalid State NotCompleting	Invalid State Exiting	Invalid State Ended
Close	Invalid State Active	Invalid State Canceling-*	Invalid State Completing	Invalid State Closing	Invalid State Closing	Invalid State Compensating	Invalid State Failing-*	Invalid State NotCompleting	Invalid State Exiting	Invalid State Ended
Compensate	Invalid State Active	Invalid State Canceling-*	Invalid State Completing	Invalid State Compensating	Invalid State Closing	Invalid State Compensating	Invalid State Failing-*	Invalid State NotCompleting	Invalid State Exiting	Invalid State Ended
Failed	Invalid State Active	Invalid State Canceling-*	Invalid State Completing	Invalid State Completed	Invalid State Closing	Invalid State Compensating	Forget Ended	Invalid State NotCompleting	Invalid State Exiting	Invalid State Ended
Exited	Invalid State Active	Invalid State Canceling-*	Invalid State Completing	Invalid State Completed	Invalid State Closing	Invalid State Compensating	Invalid State Failing-*	Invalid State NotCompleting	Forget Ended	Invalid State Ended
NotCompleted	Invalid State Active	Invalid State Canceling-*	Invalid State Completing	Invalid State Completed	Invalid State Closing	Invalid State Compensating	Invalid State Failing-*	Forget Ended	Invalid State Exiting	Invalid State Ended

606

607