Organization for the Advancement of Structured Information Systems

Business Transaction Protocol

An OASIS Committee Specification

CURRENT STATUS : internal committee draft

Version 1.0 [0.9.2.4]
DD Mmm 2002 [3 April 2002 16:19]

| Working Draft 0.9                      | 24 October 2001 |
| Working Draft 0.9.0.1 – minor editorials issues applied | 16 November 2001 |
| Working Draft 0.9.0.2 – issue resolutions balloting to 10 Dec 2001 | 4 December 2001 |
| Working Draft 0.9.0.3 – possible solution to msging issues | 11 December 2001 |
| Working Draft 0.9.0.4 – issue 79 solution, revise msging issues | 12 January 2002 |
| Working Draft 0.9.1 – includes all issues agreed 16 Jan 2002, and 82 (deferred) | 18 January 2002 |
| Working Draft 0.9.1.1 – format changes and proposed soln 77,78, 17. | 27 January 2002 |
| Working Draft 0.9.1.2 – xml changes, new schema, and issue 74 | 30 January 2002 |
| Working Draft 0.9.1.3 – corrections, issue 30, state table – 81, 104 | 8 February 2002 |
| Working Draft 0.9.2 – all issues as agreed 13 February 2002 | 13 February 2002 |
| Working Draft 0.9.2.1 – issues 2, 3, 15, 19, 50, 67, 95 | 26 February 2002 |
| Working Draft 0.9.2.2 – as accepted 27 Feb 2002 + corrections, issues 29, 60, 97, 99 | 12 March 2002 |
| Working Draft 0.9.2.3 – 0.9.2.2 and issue 106, 96, 98 | 18 March 2002 |
| Working Draft 0.9.2.4 – as accepted 27 Mar 2002 + 61, 100, 109, inclusion of model | 3 April 2002 |

Change marks relative to 0.9.2.3 all accepted
Copyright and related notices

Copyright © The Organization for the Advancement of Structured Information Standards (OASIS), 2002. 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 may 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.

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 Executive Director.

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 Executive Director.
Acknowledgements

The members of the OASIS Business Transactions Technical Committee contributed to the development of this specification. The following were members of the committee for at least part of the time from July 2001 until the agreement of the specification are listed below. Some TC members changed their affiliation to OASIS members, but remained members of the TC:

Mike Abbott  CodeMetamorphosis
Alex Berson   Entrust, Inc
Geoff Brown  Oracle
Doug Bunting  Sun Microsystems
Fred Carter  Sun Microsystems; individual
Alex Ceponkus  Bowstreet Inc.; individual
Pyounguk Cho  Iona
Victor Corrales  Hewlett-Packard Co.
Bill Cox  BEA Systems, Inc.
Sanjay Dalal  BEA Systems, Inc.
Alan Davies  SeeBeyond Inc.
Hatem El-Sebaaly  IPNet
Ed Felt  BEA Systems, Inc.
Tony Fletcher  Choreology Ltd
Bill Flood  Sybase
Peter Furniss  Choreology Ltd
Alastair Green  Choreology Ltd
Mark Hale  Interwoven Inc.
Gordon Hamilton  AppliedTheory, individual
Roddy Herries  Choreology Ltd
Mark Little  Hewlett-Packard Co.
Anne Manes  Systinet
Savas Parastatidis  Hewlett-Packard Co.
Bill Pope  Bowstreet, individual
Mark Potts  Individual, Talking Blocks
Pal Takacsi-Nagy  BEA Systems, Inc.
James Tauber  Bowstreet, individual
Sazi Temel  BEA Systems, Inc.
Steve Viens  individual
Jim Webber  Hewlett-Packard Co.
Steve White  SeeBeyond Inc.

The primary authors and editors of the main body of the specification were, in alphabetical order

Alex Ceponkus  alex@ceponkus.org
Sanjay Dalal  sanjay.dalal@bea.com
We thank Pal Takacsi-Nagy of BEA Systems Inc and Bill Pope for their efforts in chairing
the Technical Committee, and Karl Best of OASIS for his guidance on the organization of the
Committee’s work.

In memory of Ed Felt

Ed Felt of BEA Systems Inc. was an active and highly valued contributor to the work of the
OASIS Business Transactions Technical Committee.

His many years of design and implementation experience with the Tuxedo system,
Weblogic’s Java transactions, and Weblogic Integration’s Conversation Management
Protocol were brought to bear in his comments on and proposals for this specification.

He was killed in the crash of the hijacked United Airlines flight 93 near to Pittsburgh,
Typographical and Linguistic Conventions and Style

The initial letters of words in terms which are defined (at least in their substantive or infinitive form) in the Glossary are capitalized whenever the term used with that exact meaning, thus:

Cancel
Participant
Application Message

The first occurrence of a word defined in the Glossary is given in bold, thus:

Coordinator

Such words may be given in bold in other contexts (for example, in section headings or captions) to emphasize their status as formally defined terms.

The names of abstract BTP protocol messages are given in upper-case throughout:

BEGIN
CONTEXT
RESIGN

The values of elements within a BTP protocol message are indicated thus:

BEGIN/atom

BTP protocol messages that are related semantically are joined by an ampersand:

BEGIN/atom & CONTEXT

BTP protocol messages that are transmitted together in a compound are joined by a + sign:

ENROL + VOTE

XML schemata and instances are given in Courier:

<btp:begin> ... </btp:begin>

Terms such as MUST, MAY and so on, which are defined in RFC [TBD number], “[TBD title]” are used with the meanings given in that document but are given in lowercase bold, rather than in upper-case:

An Inferior must send one of RESIGN, PREPARED or CANCELLED to its Superior.
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copyright and related notices</td>
<td>2</td>
</tr>
<tr>
<td>Acknowledgements</td>
<td>3</td>
</tr>
<tr>
<td>Typographical and Linguistic Conventions and Style</td>
<td>4</td>
</tr>
<tr>
<td>Contents</td>
<td>5</td>
</tr>
<tr>
<td>Part 1. Purpose and Features of BTP</td>
<td>11</td>
</tr>
<tr>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>Development and Maintenance of the Specification</td>
<td>13</td>
</tr>
<tr>
<td>Structure of this specification</td>
<td>14</td>
</tr>
<tr>
<td>Conceptual Model</td>
<td>15</td>
</tr>
<tr>
<td>Example Core</td>
<td>18</td>
</tr>
<tr>
<td>Business transactions</td>
<td>16</td>
</tr>
<tr>
<td>External Effects</td>
<td>17</td>
</tr>
<tr>
<td>Two-phase outcome</td>
<td>18</td>
</tr>
<tr>
<td>Actors and roles</td>
<td>19</td>
</tr>
<tr>
<td>Superior:Inferior relationship</td>
<td>19</td>
</tr>
<tr>
<td>Business transaction trees</td>
<td>20</td>
</tr>
<tr>
<td>Atoms and Cohesions</td>
<td>21</td>
</tr>
<tr>
<td>Participants, Sub-Coordinators and Sub-Composers</td>
<td>22</td>
</tr>
<tr>
<td>Business transaction creation</td>
<td>22</td>
</tr>
<tr>
<td>Business transaction propagation</td>
<td>22</td>
</tr>
<tr>
<td>Creation of Intermediates (Sub-Coordinators and Sub-Composers)</td>
<td>23</td>
</tr>
<tr>
<td>Checking and context-reply</td>
<td>23</td>
</tr>
<tr>
<td>Message sequence</td>
<td>24</td>
</tr>
<tr>
<td>Control of inferiors</td>
<td>24</td>
</tr>
<tr>
<td>Evolution of confirm-set</td>
<td>25</td>
</tr>
<tr>
<td>Confirm-set of intermediates</td>
<td>25</td>
</tr>
<tr>
<td>Optimisations and variations</td>
<td>26</td>
</tr>
<tr>
<td>Spontaneous prepared</td>
<td>26</td>
</tr>
<tr>
<td>One-shot</td>
<td>27</td>
</tr>
<tr>
<td>Resignation</td>
<td>27</td>
</tr>
<tr>
<td>One-phase confirmation</td>
<td>28</td>
</tr>
<tr>
<td>Autonomous cancel, autonomous confirm and contradictions</td>
<td>28</td>
</tr>
<tr>
<td>Recovery and failure handling</td>
<td>29</td>
</tr>
<tr>
<td>Types of failure</td>
<td>30</td>
</tr>
<tr>
<td>Consistent information</td>
<td>31</td>
</tr>
<tr>
<td>Recovery messages</td>
<td>31</td>
</tr>
<tr>
<td>Redirection</td>
<td>32</td>
</tr>
<tr>
<td>Terminator:Decider failures and transaction timelimit</td>
<td>33</td>
</tr>
<tr>
<td>Contradictions and hazard</td>
<td>34</td>
</tr>
<tr>
<td>Relation of BTP to application and carrier protocols</td>
<td>35</td>
</tr>
<tr>
<td>Other elements</td>
<td>36</td>
</tr>
<tr>
<td>Identifiers</td>
<td>37</td>
</tr>
<tr>
<td>Addresses</td>
<td>38</td>
</tr>
</tbody>
</table>
### Part 2. Normative Specification of BTP

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actors, Roles and Relationships</td>
<td>58</td>
</tr>
<tr>
<td>Relationships</td>
<td>58</td>
</tr>
<tr>
<td>Roles</td>
<td>60</td>
</tr>
<tr>
<td>Roles involved in the outcome relationships</td>
<td>6</td>
</tr>
<tr>
<td>Superior</td>
<td>6</td>
</tr>
<tr>
<td>Inferior</td>
<td>62</td>
</tr>
<tr>
<td>Enroller</td>
<td>64</td>
</tr>
<tr>
<td>Participant</td>
<td>6</td>
</tr>
<tr>
<td>Sub-coordinator</td>
<td>65</td>
</tr>
<tr>
<td>Sub-composer</td>
<td>66</td>
</tr>
<tr>
<td>Roles involved in the control relationships</td>
<td>6</td>
</tr>
<tr>
<td>Decider</td>
<td>66</td>
</tr>
<tr>
<td>Coordinator</td>
<td>6</td>
</tr>
<tr>
<td>Composer</td>
<td>64</td>
</tr>
<tr>
<td>Terminator</td>
<td>68</td>
</tr>
<tr>
<td>Initiator</td>
<td>70</td>
</tr>
<tr>
<td>Factory</td>
<td>69</td>
</tr>
<tr>
<td>Other roles</td>
<td>70</td>
</tr>
<tr>
<td>Redirector</td>
<td>70</td>
</tr>
<tr>
<td>Status Requestor</td>
<td>70</td>
</tr>
<tr>
<td>Summary of relationships</td>
<td>7</td>
</tr>
<tr>
<td>Abstract Messages and Associated Contracts</td>
<td>73</td>
</tr>
<tr>
<td>Addresses</td>
<td>73</td>
</tr>
<tr>
<td>Request/response pairs</td>
<td>75</td>
</tr>
<tr>
<td>Compounding messages</td>
<td>7</td>
</tr>
<tr>
<td>Extensibility</td>
<td>73</td>
</tr>
<tr>
<td>Messages</td>
<td>77</td>
</tr>
<tr>
<td>Qualifiers</td>
<td>78</td>
</tr>
<tr>
<td>Messages not restricted to outcome or control relationships</td>
<td>7</td>
</tr>
<tr>
<td>CONTEXT</td>
<td>78</td>
</tr>
<tr>
<td>CONTEXT_REPLY</td>
<td>80</td>
</tr>
<tr>
<td>REQUEST_STATUS</td>
<td>81</td>
</tr>
<tr>
<td>STATUS</td>
<td>81</td>
</tr>
<tr>
<td>FAULT</td>
<td>85</td>
</tr>
<tr>
<td>REQUEST_INFERIOR_STATUSES, INFERIOR_STATUSES</td>
<td>86</td>
</tr>
<tr>
<td>Messages used in the outcome relationships</td>
<td>86</td>
</tr>
<tr>
<td>ENROL</td>
<td>86</td>
</tr>
<tr>
<td>ENROLLED</td>
<td>86</td>
</tr>
<tr>
<td>RESIGN</td>
<td>88</td>
</tr>
<tr>
<td>RESIGNED</td>
<td>89</td>
</tr>
<tr>
<td>PREPARE</td>
<td>91</td>
</tr>
<tr>
<td>PREPARED</td>
<td>91</td>
</tr>
<tr>
<td>CONFIRM</td>
<td>92</td>
</tr>
<tr>
<td>CONFIRMED</td>
<td>93</td>
</tr>
<tr>
<td>Page 8 of 202</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CANCEL</td>
<td>94</td>
</tr>
<tr>
<td>CANCELLED</td>
<td>95</td>
</tr>
<tr>
<td>CONFIRM ONE PHASE</td>
<td>96</td>
</tr>
<tr>
<td>HAZARD</td>
<td>97</td>
</tr>
<tr>
<td>CONTRADICTION</td>
<td>98</td>
</tr>
<tr>
<td>SUPERIOR_STATE</td>
<td>99</td>
</tr>
<tr>
<td>INFERIOR_STATE</td>
<td>100</td>
</tr>
<tr>
<td>REDIRECT</td>
<td>101</td>
</tr>
<tr>
<td>Messages used in control relationships</td>
<td>103</td>
</tr>
<tr>
<td>BEGIN</td>
<td>103</td>
</tr>
<tr>
<td>BEGIN</td>
<td>103</td>
</tr>
<tr>
<td>PREPARE_INFERIORS</td>
<td>104</td>
</tr>
<tr>
<td>CONFIRM_TRANSACTION</td>
<td>106</td>
</tr>
<tr>
<td>TRANSACTION_CONFIRMED</td>
<td>107</td>
</tr>
<tr>
<td>CANCEL TRANSACTION</td>
<td>107</td>
</tr>
<tr>
<td>CANCEL_INFERIORS</td>
<td>107</td>
</tr>
<tr>
<td>TRANSACTION_CANCELLED</td>
<td>108</td>
</tr>
<tr>
<td>REQUEST_INFERIOR_STATUSES</td>
<td>109</td>
</tr>
<tr>
<td>INFERIOR_STATUSES</td>
<td>110</td>
</tr>
<tr>
<td>Groups – combinations of related messages</td>
<td>110</td>
</tr>
<tr>
<td>CONTEXT &amp; application message</td>
<td>116</td>
</tr>
<tr>
<td>CONTEXT_REPLY &amp; ENROL</td>
<td>116</td>
</tr>
<tr>
<td>CONTEXT_REPLY (&amp; ENROL) &amp; PREPARED / &amp; CANCELLED</td>
<td>117</td>
</tr>
<tr>
<td>CONTEXT_REPLY &amp; ENROL &amp; application message (&amp; PREPARED)</td>
<td>117</td>
</tr>
<tr>
<td>BEGIN &amp; CONTEXT</td>
<td>118</td>
</tr>
<tr>
<td>BEGIN &amp; CONTEXT</td>
<td>118</td>
</tr>
<tr>
<td>Standard qualifiers</td>
<td>119</td>
</tr>
<tr>
<td>Transaction timelimit</td>
<td>120</td>
</tr>
<tr>
<td>Inferior timeout</td>
<td>120</td>
</tr>
<tr>
<td>Minimum inferior timeout</td>
<td>121</td>
</tr>
<tr>
<td>Inferior name</td>
<td>122</td>
</tr>
<tr>
<td>State Tables</td>
<td>123</td>
</tr>
<tr>
<td>Status queries</td>
<td>124</td>
</tr>
<tr>
<td>Decision events</td>
<td>125</td>
</tr>
<tr>
<td>Disruptions – failure events</td>
<td>125</td>
</tr>
<tr>
<td>Invalid cells and assumptions of the communication mechanism</td>
<td>126</td>
</tr>
<tr>
<td>Meaning of state table events</td>
<td>126</td>
</tr>
<tr>
<td>Persistent information</td>
<td>127</td>
</tr>
<tr>
<td>Superior state table</td>
<td>127</td>
</tr>
<tr>
<td>Inferior state table</td>
<td>128</td>
</tr>
<tr>
<td>Persistent information</td>
<td>129</td>
</tr>
<tr>
<td>XML representation of Message Set</td>
<td>130</td>
</tr>
<tr>
<td>Addresses</td>
<td>130</td>
</tr>
<tr>
<td>Identifiers</td>
<td>130</td>
</tr>
<tr>
<td>Message References</td>
<td>131</td>
</tr>
<tr>
<td>Messages</td>
<td>131</td>
</tr>
<tr>
<td>Messages used in control relationships</td>
<td>131</td>
</tr>
<tr>
<td>Page</td>
<td>Section</td>
</tr>
<tr>
<td>------</td>
<td>---------</td>
</tr>
<tr>
<td>149</td>
<td>CONTEXT...</td>
</tr>
<tr>
<td>150</td>
<td>CONTEXT_REPLY</td>
</tr>
<tr>
<td>150</td>
<td>REQUEST_STATUS</td>
</tr>
<tr>
<td>150</td>
<td>STATUS</td>
</tr>
<tr>
<td>150</td>
<td>FAULT</td>
</tr>
<tr>
<td>151</td>
<td>ENROL</td>
</tr>
<tr>
<td>152</td>
<td>ENROLLED</td>
</tr>
<tr>
<td>152</td>
<td>RESIGN</td>
</tr>
<tr>
<td>153</td>
<td>RESIGNED</td>
</tr>
<tr>
<td>153</td>
<td>PREPARE</td>
</tr>
<tr>
<td>153</td>
<td>PREPARED</td>
</tr>
<tr>
<td>153</td>
<td>CONFIRM</td>
</tr>
<tr>
<td>154</td>
<td>CONFIRMED</td>
</tr>
<tr>
<td>155</td>
<td>CANCEL</td>
</tr>
<tr>
<td>155</td>
<td>CANCELLED</td>
</tr>
<tr>
<td>155</td>
<td>CONFIRM_ONE_PHASE</td>
</tr>
<tr>
<td>156</td>
<td>HAZARD</td>
</tr>
<tr>
<td>156</td>
<td>CONTRADICTION</td>
</tr>
<tr>
<td>156</td>
<td>SUPERIOR_STATE</td>
</tr>
<tr>
<td>156</td>
<td>INFERIOR_STATE</td>
</tr>
<tr>
<td>157</td>
<td>REDIRECT</td>
</tr>
<tr>
<td>157</td>
<td>BEGIN</td>
</tr>
<tr>
<td>157</td>
<td>BEGUN</td>
</tr>
<tr>
<td>158</td>
<td>PREPARE_INFERIORS</td>
</tr>
<tr>
<td>158</td>
<td>CONFIRM_TRANSACTION</td>
</tr>
<tr>
<td>159</td>
<td>TRANSACTION_CONFIRMED</td>
</tr>
<tr>
<td>159</td>
<td>CANCEL.Transaction</td>
</tr>
<tr>
<td>160</td>
<td>CANCEL_INFERIORS</td>
</tr>
<tr>
<td>160</td>
<td>TRANSACTION_CANCELED</td>
</tr>
<tr>
<td>160</td>
<td>REQUEST_INFERIOR_STATUSES</td>
</tr>
<tr>
<td>160</td>
<td>INFERIOR_STATUSES</td>
</tr>
<tr>
<td>160</td>
<td>Standard qualifiers</td>
</tr>
<tr>
<td>161</td>
<td>Transaction timelimit</td>
</tr>
<tr>
<td>161</td>
<td>Minimum inferior timeout</td>
</tr>
<tr>
<td>161</td>
<td>Inferior name</td>
</tr>
<tr>
<td>162</td>
<td>Compounding of Messages</td>
</tr>
<tr>
<td>162</td>
<td>XML Schemas</td>
</tr>
<tr>
<td>162</td>
<td>XML schema for BTP messages</td>
</tr>
<tr>
<td>162</td>
<td>XML schema for standard qualifiers</td>
</tr>
<tr>
<td>178</td>
<td>Carrier Protocol Bindings</td>
</tr>
<tr>
<td>178</td>
<td>Carrier Protocol Binding Proforma</td>
</tr>
<tr>
<td>179</td>
<td>Bindings for request/response carrier protocols</td>
</tr>
<tr>
<td>180</td>
<td>Request/response exploitation rules</td>
</tr>
<tr>
<td>183</td>
<td>Example scenario using SOAP binding</td>
</tr>
<tr>
<td>185</td>
<td>SOAP + Attachments Binding</td>
</tr>
<tr>
<td>Conformance</td>
<td>Part 3: Glossary</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Part 1. Purpose and Features of BTP

Introduction

This document, which describes and defines the Business Transaction Protocol (BTP), is a Committee Specification of the Organization for the Advancement of Structured Information Standards (OASIS). The standard has been authored by the collective work of representatives of thirteen software product companies (listed on page 3), grouped in the Business Transactions Technical Committee (BT TC) of OASIS.

The OASIS BTP Technical Committee began its work at an inaugural meeting in San Jose, Calif. on 13 March 2001, and this specification was endorsed as a Committee Specification by a [*** unanimous] vote on [*** date].

BTP is designed to allow coordination of application work between multiple participants owned or controlled by autonomous organizations. BTP uses a two-phase outcome coordination protocol to ensure the overall application achieves a consistent result, create atomic effects (results of computations). BTP also permits the consistent outcome to be defined a priori -- all the work is confirmed or none is -- (an atomic business transaction or atom) or for the composition of such atomic units of work (atoms) into cohesive business transactions (cohesions), which allow application intervention into the selection of the atoms work which will to be confirmed, and of those which will be cancelled (a cohesive business transaction or cohesion).

BTP’s ability to coordinate between is designed to allow transactional coordination of participants, which are part of services offered by multiple autonomous organizations (as well as within a single organization). It is therefore makes it ideally suited for use in a Web Services environment. For this reason this specification defines communications protocol bindings which target the emerging Web Services arena, while preserving the capacity to carry BTP messages over other communication protocols. Protocol message structure and content constraints are schematized in XML, and message content is encoded in XML instances.

The BTP allows great flexibility in the implementation of business transaction participants. Such participants enable the consistent reversal of the effects of atoms. BTP participants may use recorded before- or after-images, or compensation operations to provide the “roll-forward, roll-back” capacity which enables their subordination to the overall outcome of an atomic business transaction.

The BTP is an interoperation protocol which defines the roles which software agents (actors) may occupy, the messages that pass between such actors, and the obligations upon and commitments made by actors-in-roles. It does not define the programming interfaces to be used by application programmers to stimulate message flow or associated state changes.
The BTP is based on a permissive and minimal approach, where constraints on implementation choices are avoided. The protocol also tries to avoid unnecessary dependencies on other standards, with the aim of lowering the hurdle to implementation.
Development and Maintenance of the Specification

For more information on the genesis and development of BTP, please consult the OASIS BT Technical Committee’s website, at

http://www.oasis-open.org/committees/business-transactions/

As of the date of adoption of this specification the OASIS BT Technical Committee is still in existence, with the charter of

- maintaining the specification in the light of implementation experiences
- coordinating publicity for BTP
- liaising with other standards bodies whose work affects or may be affected by BTP
- reviewing the appropriate time, in the light of implementation experience and user support, to put BTP forward for adoption as a full OASIS standard

If you have a question about the functionality of BTP, or wish to report an error or to suggest a modification to the specification, please subscribe to:

bt-spec@lists.oasis-open.org

Any employee of a corporate member of OASIS, or any individual member of OASIS, may subscribe to OASIS mail lists, and is also entitled to apply to join the Technical Committee.

The main list of the committee is:

business-transaction@lists.oasis-open.org
Structure of this specification

This specification document includes, in Part 1, an explanation and description of the conceptual model of BTP, and, in Part 2, a fully normative specification of the protocol.

The use and definition of terms in the model can be regarded as authoritative but should not be taken to restrict implementations or uses of BTP. In case of (unintended) disagreement between the parts, Part 2 takes precedence over Part 1.

Part 1 contains
- Executive Summary
- This document structure description
- Conceptual Model

Part 2 contains the following sections:
- Actors, roles and relationships: defines the model entities used in the specification, their relationships to each other and indicates the correspondence of these to real implementation constructs; this section also lists which messages are sent and received for each role.
- Abstract message set: defines a set of abstract messages that are exchanged between software agents performing the various roles to create, progress and complete the relationships between those roles. For each abstract message the parameters are defined and the associated “contract” is stated – the contract defines the meaning of the message in terms of what the receiver can infer of the sender’s state and the intended effect on the receiver. This section does not itself specify a particular encoding or representation of the messages nor a single mechanism for communicating the messages
- State tables: specifies the state transitions for the Superior and Inferior roles, detailing when particular messages may be sent and when internal decisions may be made that affect the state
- XML representation: defines an XML representation of the message set. Other representations of the message set, or parts of it are possible – these may or may not be suitable for interoperation between heterogeneous implementations.
- Carrier protocol bindings: defines a “carrier binding proforma” that details the information required to specify the mapping to a particular carrier protocol such that independent implementations can interoperate. The proforma requires an identification for the binding, the nature of the addressing information used with the binding, how the messages are represented and encoded and how they are carried (e.g. which carrier protocol messages or fields they are in) and may include other requirements.
- Using the carrier protocol proforma, this section fully specifies bindings to SOAP 1.1, using the XML representation of the abstract message set.
- Conformance definitions: defines combinations of facilities (expressed as roles) that an implementation can declare it supports
Part 3 contains a glossary that provides succinct definitions of terms used in the rest of the document.

Conceptual Model

This section introduces the concepts of BTP. Its use and definition of terms can be regarded as authoritative but should not be taken to restrict implementations or uses of BTP. Part 2 of the specification is fully normative and in case of disagreement takes precedence over statements or examples in this section.

BTP is designed to make minimal assumptions about the implementation structure and the properties of the carrier protocols. This allows BTP to be bound to more than one carrier protocol. BTP implementations built in quite different ways should be able to interoperate if they are bound to the same carrier protocol. This flexibility requires that much of the text is abstract and may be difficult to visualise in the absence of a particular implementation pattern or carrier protocol. To aid understanding some possible implementation examples are presented in the following text.

Example Core

An advanced manufacturing company (Manufacturer A) orders the parts and services it needs on-line. It has existing relationships with parts suppliers and providers of services such as shipping and insurance. All of the communications between these organizations is via XML messages. The interactions of these business transactions include:

1. Manufacturer A’s production scheduling system sends an Order message to a Supplier.
2. The Supplier’s order processing system sends back an order confirmation with the details of the order.
3. Manufacturer A orders delivery from a Shipper for the ordered parts.
4. The Shipper evaluates the request and based on its truck schedule it sends back a positive or negative reply.
5. Some shipments need to be insured based on their value, where they are shipped from, and method of transportation. Manufacturer A sends an Order message to an Insurer when this is necessary.
6. The Insurer responds with a bid or a no-bid response.

Problems have arisen with some of these interactions.

- Manufacturer A had ordered parts from a supplier and contacted shipper M about delivering the goods. Shipper M was busy and agreed to the contract but only for a scheduled delivery the day after the parts were needed. By the time this was addressed it was too late to schedule alternate shipping.
- There were communications problems with supplier Z that resulted in an order not being confirmed. The shipper arrived to pick up the order and supplier Z knew nothing about it.
• Goods have been shipped without insurance when company policy dictated that insurance was required.

These problems occur because of the unreliable nature of the Internet and the lack of visibility a company has into the workings and state of an outside organization. By using BTP in support of this supply application, these problems can be ameliorated.

BTP is a protocol, that is, a set of specific messages that get exchanged between computer systems supporting an application, with rules about the meaning and use of the messages. The computer systems will also exchange application-specific messages. Thus, within the example, the Manufacturer’s system and the Supplier’s system (say), will exchange messages detailing what the goods are, how many, what price and will also exchange BTP messages.

The parts of the application in both systems that handle these different sets of messages can be distinguished, as in Figure 1. In each BTP-using party there is an **application element** and a **BTP element**. The application elements exchange the order information and cause the associated business functions to be performed. The BTP elements, which send and receive the BTP messages, perform specific roles in the protocol. These BTP elements assist the application in getting the work of the application done. The application element, as understood by this model, may include supporting infrastructure elements, such as containers or interceptors, as well as application-specific code.

**Figure 1 – Manufacturer Example**

**Business transactions**
A Business Transaction can be defined as a consistent change in the state of a business relationship between two or more parties. A business relationship is any distributed state held by the parties which is subject to contractual constraints agreed by those parties. For example, an master purchasing agreement, which permits the placing of orders for components by known buying organizations allows a buyer and a seller to create and subsequently exchange meaningful information about the creation and processing of an order. Such agreements (and the consequent specification of shared or canonical data formats and of the messages that carry those formats, and their permitted sequences, all of which are needed for an automated implementation of an agreement) stem from business negotiations and are specific to a particular trading or information exchange community (group of potential parties). This definition of a business relationship is deliberately silent on the nature of the “business” transacted between the parties: it might be trading for profit, verification of authorizations for expenditure or loans, consistent publication (replication) of government ordinances to multiple sites, or any other computerized interaction where the parties require high confidence of consistent delivery or processing of data. In each party or site where business relationship state resides an application system must exist which can maintain that state and communicate it as needed to other parties. The Business Transaction Protocol (BTP) assists the application systems of the various parties to bring about consistent and coordinated changes in the relationship as viewed from each party. BTP assumes that for a given business transaction, state changes occur, or are desired, in computer systems controlled by some set of parties, and that these changes are related in some application-defined manner. BTP assumes that the parties involved in a business transaction have distinct and autonomous application systems, which do not require knowledge of each others’ implementation or internal state representations in volatile or persistent storage. Access to such loosely coupled application systems is assumed to occur only through service interfaces.

Thus the state changes that BTP is concerned with are only those affecting the immediate business relationship. Although these externally visible changes will typically correspond to internal state changes of the parties, use of BTP does not itself imply any constraints or requirements on the internal state.

External Effects

BTP coordinates the state changes caused by the exchange of application messages. These state changes are part of the contract between BTP-using parties. In the manufacturing example, an interaction between the manufacturer and the supplier might involve the supplier receiving the order (an application message), checking to ensure that it had enough product on hand, reserving the product in the manufacturer’s name and replying. When the manufacturer agrees to the purchase (assuming the shipping and insurance are also reserved), BTP messages are sent to confirm the purchase. In this case, the supplier is offering a BTP-enabled service – the application element and its supporting BTP elements together offer this service.

---

1 Although a Business Transaction is defined as concerning a business relationship, the facilities of BTP make it suitable for other environments where loosely coupled systems require coordination and consistency.
In general, to be able to satisfy such contracts a BTP-enabled service must support in some manner provisional or tentative state changes (the transaction’s provisional effect) and completion either through confirmation (final effect) or cancellation (counter-effect). The meaning of provisional, final, and counter-effect are specific to the application and to the implementation of the application. In the example, the reservation of the order is the provisional effect, the completion of the purchase is the final effect. Some of the implementation approaches are shown in Table 1. From the perspective of BTP and the initiator application, all these are considered equivalent. Outside of BTP the underlying business relationship (or contract) between the parties can constrain the degree to which the effects are visible.

Table 1 Some alternatives for provisional, final and counter effects

<table>
<thead>
<tr>
<th>provisional effect</th>
<th>final effect</th>
<th>counter effect</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Store intended changes without performing them</td>
<td>Perform the changes</td>
<td>Delete the stored changes, unperformed</td>
<td>Provisional effect may include checking for validity</td>
</tr>
<tr>
<td>Perform the changes, making them visible; store information to undo the changes</td>
<td>Delete undo information</td>
<td>Perform undo action</td>
<td>One form of compensation approach</td>
</tr>
<tr>
<td>Store original state, prevent outside access, perform changes</td>
<td>Allow access</td>
<td>Restore original state; allow access</td>
<td>a typical database approach</td>
</tr>
</tbody>
</table>

These alternatives are not the only ones – they can be combined or varied. The visible state of the application information prior to confirmation or cancellation may be different from both the original state and the final state.

Especially in the compensation approach, if the changes are cancelled, the counter-effect may be a precise inversion or removal of provisional changes, or it may be the processing of operations that in some way compensate for, make good, alleviate or supplement their effect. There may be side-effects of various kinds from a counter-effected operation – such as levying of cancellation charges or the record of the operation may be visible, but marked as cancelled. The possibility of these side-effects is considered to be part of the overarching contract.

Two-phase outcome

The BTP protocol coordinates the transitions into and out of the event states described above by sending messages between the transaction parties. This involves a two-phase exchange. First the application elements exchange messages that determine the characteristics and cause
the performance of the provisional effect; then a separate message, to the BTP element, asking for the performance of the final or the counter effect.

In general, the application elements in the systems involved having first communicated the application messages, each system that has to make changes in its own state:

- determines whether it is able achieve its provisional effect and then ensure it will be able either to cancel (counter-effect) its operation or to confirm (give final effect to) its operation, whichever is subsequently instructed, and
- reports its ability to confirm-or-cancel (its preparedness) to a central coordinating entity.

And, after receiving these reports, the coordinating entity:

- determines which of the systems should be instructed to confirm and which should be instructed to cancel
- informs each system whether it should confirm or cancel (the “outcome”) by sending a message to its BTP element

When there is more than one system that has to make changes such a two-phase exchange mediated by a coordinator is required to achieve a consistent outcome for a set of operations. The two-phases of the BTP protocol ensure that either the entire attempted transaction is abandoned or a consistent set of participants is confirmed.

**Actors and roles**

BTP centres on the bilateral relationship between the computer systems of the coordinating entity and those of one of the parties in the overall business transaction. For each bilateral relationship in a business transaction, a software agent within the coordinating entity’s systems plays the BTP role of Superior and a software agent within the systems of the party play the BTP role of Inferior. The concept “role” refers strictly to the participation in a particular relationship in a particular business transaction. The software agent performing a role is termed an Actor. An Actor is distinguished from other Actors by being distinguishably addressable. The same Actor may perform multiple roles in the same business transaction (including the case where a Superior is also an Inferior), and may also perform the same or different roles in multiple business transactions, either concurrently or consecutively.

**Superior:Inferior relationship**

A basic case of a single Superior:Inferior relationship, including the association with application elements, is illustrated in Figure 2. In many cases, including the manufacturer supply example, the application element associated with the superior will directly initiate the application exchanges –as does the manufacturer’s application client to the supplier’s server, for example – but this is not invariably the case. It is possible that the first direct communication between the application elements is from one associated with an
inferior to the one associated with the superior – for example, with an application that
requested quotes by advertising the identity and location of the Superior along with invitation
to quote; incoming quotes would be the first direct application message exchanged. In all
cases the topmost application element in a tree or subtree will be aware of the business
transaction first. How the identity of the transaction and the address of the BTP Superior are
communicated to the secondary application element is a matter for the application protocol
and not strictly part of BTP, although it will commonly be done by associating a BTP
CONTEXT message with application messages.

Figure 2 Basic Superior:Inferior relationship for BTP

An Inferior is associated with some set of application activities that create effects within the
party, for a given business transaction. As stated above, commonly, though not invariably,
this application activity within the party will be a result of some operation invocations from
elsewhere (shown as the “initiating application element” in Figure 2, associated with
the Superior to an application element associated with the Inferior (shown as “Service
application element”). This second application element determines what activities the Inferior
is responsible for, and then the Inferior is responsible for reporting to the Superior whether
the associated operations’ provisional effect can be confirmed/cancelled – this is called
“becoming prepared”, because the Inferior has to remain prepared to receive whichever order
eventually arrives (subject to various exceptions and exclusions, detailed below).

Business transaction trees

There are many patterns in which the service provider participants involved in a business
transaction may be arranged in respect of the two-phase exchange and the determination of
which are eventually confirmed. The simplest is shown in Figure 3, involving only
two parties – one (B) making itself subject to the decision of confirm-or-cancel made by the
other (A). This basic bilateral relationship, in which one side makes itself inferior to the other,
is the building block used in all business transaction patterns. In this simplest case, the
“coordination” by the superior, A, is just that A can be sure whether the operations at the
inferior, B were eventually cancelled or confirmed.
In the next simplest case, as in figure Figure 4, a bilateral, Superior:Inferior relationship appears twice, with two Inferiors, D and E, both making themselves inferior to a single Superior, C. From the perspective of either D or E, they are in the same position as B in the previous case—they are unaware of and unaffected (directly) by each other. It is only within C that there is any linkage between the confirm-or-cancel outcomes that apply to D and E.

The same Superior:Inferior relationship is used in business transaction trees that are both “wider” – with more Inferiors reporting their preparedness to be confirm-or-canceled to a single Superior – and “deeper”. In a “deeper” tree, as in figure Figure 5, an entity (G) that is Superior to one or more Inferiors (H, J), is itself Inferior to another entity (F) – it is said to be interposed or is an Intermediate (either term can be used). In this case, G will collect the information on preparedness of its Inferiors before passing on its own report to its Superior, F, and awaiting the outcome as advised by F.
A business transaction tree, made up of these bilateral Superior:Inferior relationships can, in theory, be arbitrarily “wide” or “deep” – there are no fixed limits to how many Inferiors a single Superior can have, or how many levels of intermediates there are between the top-most Superior (that is Inferior to none) and the bottom-most leaf Inferior. The actual creation of the tree depends on the behaviour and requirements of the application. Given the (potentially) inter-organisational nature of business transactions, there may be no overall design or control of the structure of the tree.

Each Inferior has only one Superior. However, a single Superior may (and commonly does) have multiple relationships with Inferiors, and may have such relationships with multiple Inferiors within each party to the transaction, and with Inferiors within multiple parties.

**Atoms and Cohesions**

As described in the previous section, the Superior receives reports from its Inferiors as to whether they are prepared. It gathers these reports in order to ascertain which Inferiors should be cancelled and which confirmed - those that cannot prepare will have already cancelled themselves. This determined, directly or indirectly, by the application element responsible of the creation and control of the Superior, which determines the nature of the Superior. There are two dimensions of variation in the Superior: is it an Inferior to another Superior; does it treat its own Inferiors atomically or cohesively. The distinction between atomic and cohesive behaviour is whether the Superior will choose or allow some Inferiors to cancel while others confirm – this is not allowed for atomic behaviour, in which all must confirm or all must cancel, but is for cohesive.

The possible cases for a Superior, given these two dimensions of variation, are:

Figure 5 Business transaction with an Intermediate (interposition)
a) the application element initiated the business transaction (causing the creation of the Superior), and instructed that all Inferiors of the Superior should confirm or all should cancel; the Superior is an Atom Coordinator;

b) the application element initiated the business transaction, but deferred the choice of which Inferiors should confirm until later, allowing it (the application element) to choose some subset to be confirmed, others to cancel; the Superior is a Cohesion Composer;

c) the application element was itself involved in an existing business transaction, and the Superior in this relationship is the Inferior in another one; this application element instructed that all Inferiors of this Superior should confirm, but only if confirmation is instructed from above or all should cancel; the Superior is an (atomic) Sub-coordinator;

d) the application element was itself involved in an existing business transaction, and the Superior in this relationship is the Inferior in another one; this application element deferred the choice of which Inferiors should be candidates to confirm until later, allowing it (the application element) to choose some subset to be confirmed, given that confirmation is instructed from above, others to cancel; the Superior is a (cohesive) Sub-composer.

In the atomic case, the two-phase outcome exchange means a Superior acting as an atomic Coordinator or sub-coordinator will treat any Inferior which cannot prepare to cancel/confirm as having veto power, causing the Superior to instruct all its Inferiors to cancel. A business transaction whose topmost Superior is atomic is an Atomic Business Transaction, or Atom – the superior is the Atom Coordinator.

In the cohesion case, with the Superior acting as a cohesive Composer or Sub-Composer, the controlling application element will determine the implications of an Inferior’s failure to be prepared to confirm-or-cancel; the application element may cancel some or all other Inferiors, do other application work, which may involve new Inferiors or may just accept the cancellation of that one Inferior and carry on. A business transaction whose topmost Superior is cohesive is a Cohesive Business Transaction, or Cohesion – the Superior is the Cohesion Composer.

For a cohesion, the set of Inferiors that eventually confirm is called the confirm-set. The term is also used to mean the set of Inferiors that have been chosen to (potentially) confirm before the final outcome is decided – if the cohesion is eventually cancelled, then confirm-set cancels. (See section “Evolution of confirm-set”). The confirm-set of an Atom is all of the Inferiors.

If the Superior is itself an Inferior, its own action of becoming prepared, and reporting this to its own Superior will depend on the receipt of prepared reports from its Inferiors. If it is atomic (i.e. is a sub-coordinator), it will only become prepared if all Inferiors reported preparedness to it; if it is cohesive (i.e. is a sub-composer), the controlling application element will determine whether the set of Inferiors that have reported as prepared is sufficient.
If the Superior is not an Inferior, the determination of when, if and, for a Cohesion, what it
should confirm depends on the controlling application. This “top-most” Superior has a
different relationship to the controlling application to that of an Inferior to its Superior: an
Inferior reports that it is prepared to the Superior, which instructs it whether to cancel or to
confirm; the top-most Superior is asked by the application element to attempt to confirm, but,
dependent on the preparedness of its Inferiors, the top-most Superior makes the final
decision. Consequently the top-most Superior is termed the Decider; the application element
that asks it to confirm is the Terminator.

Participants, Sub-Coordinators and Sub-Composers

An Inferior may directly be responsible for applying the confirm-or-cancel decision to some
application effects, or may in turn be a BTP Superior to which others will enrol. If it only
handles application effects it is called a Participant, in the latter case it is called a Sub-
coordinator or a Sub-composer, depending on whether it is atomic or cohesive with respect
to its own future Inferiors. (If an Inferior is both responsible for application effects, and is a
BTP Superior, it is not considered a Participant, according to the strict definitions, though
informally it may be referred to as such.) The Superior is unaware, via the BTP exchanges,
whether the Inferior is a Participant, Sub-coordinator or Sub-composer. This specification
does not define messages or interfaces for the creation of Participants or for the application
element to tell the Participant what the application effects are or how they are to be confirmed
or cancelled as necessary. (Although out-of-scope for this specification, one or more APIs
could be standardised.)

Business transaction creation

This section describes in some detail how a BTP business transaction is created. The
interaction diagram in Figure 6 also shows this sequence. The messages shown in
lower-case italics (between Factory and Coordinator) represent interactions that are not
specified in BTP.

199199
A business transaction is started at the initiative of an application element, which causes the creation of a Coordinator or Composer. Any Inferiors participating in this transaction will enrol with this Superior. BTP defines abstract messages (BEGIN, BEGUN) to request this but the equivalent function can also be achieved using proprietary means, especially if the Factory or Coordinator is an internal component of the initiating application. If the BTP messages are used, the application element performs the role of Initiator and sends BEGIN to a Factory. The BEGIN message identifies whether a Coordinator (for an atom) or a Composer (for a cohesion) is desired. The Factory, after the creation of the new Coordinator or Composer, replies with related BEGUN and CONTEXT messages. "Related" means they are sent together in a manner that has semantic significance; how this is represented is determined by the binding in use. The Coordinator's or Composer's creation is the establishment of a new instance of a BTP role. It may involve only the assignment of a new identifier within an existing Actor (which may also be performing the Factory role, for example). Alternatively a new Actor with a distinct address may be instantiated. These and other alternatives are implementation choices, and BTP ensures other Actors are unaffected by the choice made.

The BEGUN message provides the addressing and identification information needed for a Terminator to access the new Coordinator or Composer as Decider; the application element performing the Initiator role may itself act as Terminator, or may pass this information to some other application element.

Whether this interoperable BTP Initiator:Factory relationship or some other mechanism is used to initiate the business transaction, a CONTEXT is made available. This identifies the
Coordinator or Composer as a Superior – containing both addressing information and the identification of the relevant state information. The CONTEXT is also marked as to whether or not this Superior will behave atomically with respect to its Inferiors (i.e. is it a Coordinator or Composer).

**Business transaction propagation**

The propagation of the business transaction from one party to another, to establish the Superior:Inferior relationships involves the transmission of the CONTEXT. This is commonly in association with, or related to, one or more application messages between the parties. In a typical case, an application message is sent from the application element that performed the Initiator role (the “sending application” in Figure 2) to some other element (the receiving application). The CONTEXT is sent with the application message in such a way that the application elements understand that work performed as a result of the application message is to be the subject of a confirm-or-cancel decision of the Superior. The receiving application element causes the creation of an Inferior (which, as for the Superior may involve just assignment on a new identifier, or instantiation of a new Actor) and ensures the new Inferior is enrolled with the Superior identified in the received CONTEXT, using an ENROL message sent to the Superior using the address in that CONTEXT.

Figure 7 shows a sequence diagram of the propagation of a business transaction. It is assumed the transaction has already been created, and thus the application element and Coordinator exist. The diagram shows the Enroller as a distinct role, with non-standardised interactions between the application element, the Enroller and the new Inferior. The Enroller role may in fact be performed by the application element, by the Inferior or by a distinct entity. At least the Superior-identifier and Superior-address from the CONTEXT has to be passed the Enroller and to the Inferior so they can communicate with the Coordinator (whose identifier and address these are).

---

2 The relationship between the application activity and BTP is subtle, and summarised in this sentence.
Creation of Intermediates (Sub-Coordinators and Sub-Composers)

If the new Inferior is to be a Sub-coordinator or Sub-composer, this can be created using a non-standard mechanism or the Initiator:Factory relationship can be used again. Figure 8 shows a sequence diagram, using the latter mechanism. The application element, having received an application message and a CONTEXT from some Superior – shown as a Coordinator/a in the diagram – wants to create the new Inferior and acting in the Initiator role, issues BEGIN to the Factory, but the CONTEXT for the original Superior (Coordinator/a) is “related” to the BEGIN. The Factory is responsible for enrolling the new Sub-coordinator or Sub-composer as an Inferior of the Superior identified by the received CONTEXT. The reply from the Factory is a related BEGUN and CONTEXT – this being the CONTEXT for the new Sub-coordinator (‘b’) or Sub-composer as a Superior. The Sub-coordinator/Sub-composer is not a Decider, as its decision is subordinated to the outcome received from the Superior. For a Sub-coordinator, further control by the application is primarily a matter of relating the new CONTEXT to appropriate application activity. For a Sub-composer there is in addition a requirement for the application to determine which of the Inferiors of the Sub-composer must have reported they are prepared before the Sub-composer can report that it is itself prepared to its own Superior, and then which of these Inferiors are to be ordered to confirm if the Sub-composer is ordered to confirm. This specification does not provide an interface or interoperable message to control this; like the relationship between application element and Participant, it is left to the implementation or independent standardisation.
The creation of a new Inferior and establishment of a Superior:Inferior relationship does not always imply that the BTP Actors are under the control of different business parties or application elements. In particular, an application element may begin a Cohesion, then create and enrol (atomic) Sub-coordinators as Inferiors of the Composer, then associate a different Sub-coordinator’s CONTEXT with each of several aspects of the application work, transmitting that CONTEXT with the application messages for that aspect to the other parties in the business transaction. Those parties can then create Participants (or other Inferiors) that are enrolled with the appropriate Sub-coordinator. Later, the application element (as Terminator, or its equivalent) can choose which of the Cohesion Composers’ Inferiors to cancel and which to confirm. By interposing its own atomic Sub-coordinator the initiating application element can indicate to the other parties that some associated set of application work will be confirmed or cancelled as a unit. This may allow the receiving parties to share information between application operations and to make one Participant responsible for applying the outcome to several operations.

“Checking” and context-reply

In BTP, enrolment is at the initiative of an application element that has received or has access to the CONTEXT which creates an Inferior (BTP uses a “pull” paradigm for enrolment). An application element in possession of a CONTEXT can choose, perhaps constrained by an overarching business and application understanding, whether and how many Inferiors to create and enrol. Consequently, in general, an application element which propagates a CONTEXT to another (via whatever mechanisms it choose), cannot be sure how many Inferiors will be enrolled as a result. Without further controls, there would be a possibility that an application element receiving a CONTEXT might attempt to enrol an Inferior with a Superior after the Superior had been asked to confirm, or even had completed confirmation.
In such a case application work that should have been part of a confirmed atomic business 
transaction could be cancelled, violating the atomicity in a manner that will not be apparent to 
the application.

To avoid this, whenever a CONTEXT is transmitted to another party by or on behalf of the 
application, the transmission of the CONTEXT itself can be replied to with a 
CONTEXT_REPLY message – this is required for an Atom, allowed for a Cohesion. An 
application element that has received a BTP CONTEXT is able, because it knows the 
Superior’s identification and address in the CONTEXT, to enrol Inferiors (Figure 9). 
Reprising with CONTEXT_REPLY means that the sender (the earlier receiver of a 
CONTEXT) will not enrol any more Inferiors. Consequently the sender of a CONTEXT can 
keep track of whether there are any outstanding (un-replied to) CONTEXTs that could be 
used for an enrolment and can avoid requesting or permitting confirmation until everything is 
safe. This check is required for an Atom, but is not always essential when the CONTEXT is 
for a Cohesion. For a Cohesion, it is a matter for the controlling application whether all 
would-be Inferiors must be enrolled before a confirmation decision can be made; or whether 
it is acceptable to proceed to confirmation at some point in time with the already enrolled 
Inferiors (or a subset thereof), accepting the automatic cancellation of any late arrivals.

CONTEXT_REPLY can also indicate that attempted enrollments failed. This can occur if the 
Enroller is unable to contact the Superior, but it able to return a CONTEXT_REPLY to 
where-ever the CONTEXT came from.

Section explaining becoming prepared?

Message sequence

BTP messages are used in relationships between several pairs of roles. These particular pair-
wise relationships can be categorised into:

• Outcome relationships : the Superior:Inferior relationship (i.e. between BTP actors 
within the transaction tree) and the Enroller:Superior relationship used in establishing it 
• Control relationships : the application:BTP actor relationships that create the nodes of 
the transaction tree (Initiator:Factory) and drive the completion (Terminator:Decider).

The outcome relationships and the messages used in them an essential part of BTP. For the 
control relationships, it would be possible to achieve the same general function using non-
standardised messages or API mechanisms. There are other distinguishable relationships 
between roles defined by BTP that are not standardised in this specification.

Figure 9 shows the message exchange for the conventional progression of a simple 
transaction to confirmation with a single Superior:Inferior relationship, assuming the standard 
control relationship. Two application elements using a request/response application message 
exchange are involved – the first is represented as the Initiator and Terminator, the second as 
the Service and Enroller. The Decider/Superior is shown as a Coordinator, but with only one 
Inferior there would be no difference with a cohesion Composer. The Factory:Coordinator

3 The “application element” from the perspective of BTP may include infrastructure software such as 
containers or interceptors, as well the application-specific code itself.
events are non-standardised, but represent interactions that must occur in some form. There are other interactions between the various application groups – Initiator-Terminator and Participant-Enroller-Service that are not shown – in particular the Service:Participant relationship.

The message sequence is shown is the “conventional” sequence, with all messages explicitly present and sent separately. There are several variations and optimisations possible – these are discussed below.
Figure 9 A conventional message sequence for a simple transaction
Note that CONTEXT has a “related” (&) relationship to BEGUN and to the application request (although in the latter case the meaning of this is defined by the application, not by BTP. The response + CONTEXT_REPLY has no semantic significance, and could be sent separately; provided the CONTEXT_REPLY is not sent until the ENROLLED has returned.

The progression of a single instance of the central outcome (Superior:Inferior) relationship can also be presented as a set of state transitions. The normative part of the specification includes state tables for the Superior side of such a relationship and for the Inferior. Since a single Superior (Coordinator, Composer, Sub-coordinator, Sub-composer) can have multiple Inferiors, each Superior will have multiple instances of the “Superior state”. How these link together is discussed below in the section “Evolution of confirm-set”, but the state transitions for the individual Superior:Inferior relationships include “decision events” which constrain the behaviour of the business transaction tree node as a whole, and thus define the semantics of the BTP messages.

The normative state tables distinguish some states that differ only in which messages can be received and thus allow for a level of error checking. The progress of the outcome relationship can be followed without dropping to such a detailed level, and the state diagrams shown here aggregate some of the states that are distinguished in the state tables. The single letters in parentheses in the diagrams correspond to the state names used in the tables. For simplicity, the state diagrams do not include the events leading to the sending of a HAZARD message – the detection and recording of a “problem” – meaning that the Inferior is unable to cleanly confirm or cleanly cancel the operations it is responsible for. As is specified in the state tables, such a problem can be detected in most states, and reported with a HAZARD message.

It should be noted that, with some exceptions, the transmission of a message from a Superior or Inferior does not cause a state change at that side. State changes are normally caused either by the receipt of a message from the peer, or by a “decision event” – which may be an internal change, including a change in the persistent information for the transactions, or may be the receipt of a message on another relationship (e.g. as when a Sub-coordinator receives CANCEL from its Superior, which is a decision event as perceived on the relationships to its Inferiors). It would be normal for an implementation on entering a new state to send the message it can now send (there will be only one). It may repeat this message at any interval – in practice only if there is reason to believe (due to lower-layer errors, timeout or known recovery events) that messages may have got lost.
Figure 10  State diagram for Superior side of a Superior:Inferior relationship
Figure 11  State diagram for Inferior side of Superior:Inferior relationship

Control of inferiors

In the case as shown in Figure 12, where the CONTEXT has been propagated from one application element (A) to others (B, C, and from C to D, E), the determination of whether to create and enrol Inferiors is, in general, up to the receiving application element – this is an aspect of the fundamental autonomy of the parties involved in a business transaction. This
autonomy may be constrained in particular situations, by inter-party agreement or where the application elements are in fact under common control.

Figure 12  Transaction tree showing various application:Participant relationships

The relationship between the application messages and either the propagated CONTEXT or the ENROL message(s) sent to the Superior is strictly part of the application protocol (or the application-with-BTP combination protocol). However defined, this allows the Superior-side application element to be aware of what application work will be confirmed or cancelled under the control of an Inferior. However, from the perspective of the Superior, and the application element controlling it, the Inferior is opaque – it is not in general possible for the Superior or its controlling application element to determine whether is a Sub-composer or Sub-coordinator (i.e. has Inferiors of its own) or is a Participant, with no further BTP relationships. Thus, if the Inferior is a Sub-composer or Sub-coordinator, the Superior has no visibility or control of its “grand-children” – the Inferiors of its Inferior (thus, in Figure 12, the Composer at A is unaware of D and E)
The opacity of an Inferior does not however apply to the control exercised by the
immediately controlling application element. An application element, acting as Terminator to
a Decider (i.e. to a Composer or Coordinator), can be aware of and distinguish the different
Inferiors enrolled with that Decider (i.e. Inferiors enrolled with the Decider in its role as
Superior). (E.g. in Figure 12, application element A knows of the Inferiors at C, B1
and B2) This is especially the case for a Cohesion Composer, where the Terminator will be
able to control which of the enrolled Inferiors of the Composer are eventually confirmed –
more exactly, the application will have control of the confirm-set for the Cohesion. For an
Atom Coordinator, visibility of the Inferiors is useful but less important, since no selection
can be made among which will be in the confirm-set – for an Atom, all Inferiors are ipso
facto members of the confirm-set.

For this control of the Inferiors to be useful, the Terminator application element will need to
be able to associate particular parts of the application work with each Inferior. This can be
achieved by various means. Taking the case of an application element controlling a Cohesion
Composer:

a) The application element can create an Atom Sub-coordinator as an immediate
Inferior of the Cohesion Composer and propagate the Sub-coordinator’s CONTEXT
associate with application messages concerned with the particular part of the
application work; any Inferiors (however many there may be) enrolled with Sub-
coordinator can be assumed to be responsible for (some of) that part of the
application, and the Terminator application element can just deal with the immediate
Inferior of the Composer that it created.

b) The application element can propagate the Composer’s own CONTEXT, and the
receiving application element can create its own Inferior which will be responsible
for some part of the application, and send ENROL to the Composer (as Superior).
Application messages concerned with that part of the application are associated with
the ENROL, and the Terminator application element can thus determine what the
Inferior is responsible for.

In both cases, the means by which the application message and the BTP CONTEXT or
ENROL are associated is ultimately application-specific. At the abstract message level, BTP
defines the concept of transmitting “related” BTP and application messages – particular
bindings to carrier protocols can specify interoperable ways to represent this relatedness. BTP
messages, including CONTEXT and ENROL, can also carry “qualifiers” – extension fields
that are not core parts of BTP or are not defined by BTP at all. The standard qualifier
“inferior-name” or application-specific qualifiers can be used to associate application
information and the BTP message, allowing a Terminator to determine which parts of the
application work are associated with each Inferior.

These considerations about control of the Inferiors of a Decider also apply to the control of
the Inferiors of a Sub-composer (and, again of less importance, a Sub-coordinator).

Evolution of confirm-set
As mentioned above, the set of Inferiors of a Cohesion that will eventually confirm is called the Confirm-set. The determination of the Confirm-set is made by the controlling application, but is affected by events from the Inferiors themselves. If the standard control relationship is used, the control of the Cohesion Composer is expressed by the Terminator:Decider exchanges, and the progressive determination of the confirm-set (its evolution) is effectively the event sequence for the Terminator:Decider relationship.

An Atom also has a confirm-set, but this always includes all the Inferiors and so does not evolve in the same way as Cohesion’s. With some exceptions, the Terminator:Decider relationship is the same for Atom Coordinators as for Cohesion Composers; this section deals with both, noting the exceptions.

The event sequence for a Composer or Coordinator is summarised in the state diagram in Figure 13. The step-by-step description refers to “Composer”, but should be read as referring to Coordinators as well, unless stated otherwise.

Initially, the Composer is created (by the Factory, using BEGIN with no related CONTEXT), and has no Inferiors. The Composer is now in the active state.
While in the active state, the following may occur, in any order and with any repetition or overlapping:

- Inferiors are enrolled – ENROL is received by the Composer – adding to the set of Inferiors of the Composer.

- Inferiors may resign - RESIGN is received from an Inferior (see section Resignation below). The Inferior is immediately removed from the set of Inferiors, as if it had never been enrolled (a RESIGNED message may be sent to the Inferior, but it no longer “counts” in any of the Composer-wide considerations here.

- CANCELLED may be received from an Inferior; there is no required immediate effect, but if this is a Coordinator the Atom will certainly cancel eventually (and an implementation may choose to initiate cancellation immediately).
• PREPARED may be received; there is no immediate effect

• The Terminator may issue PREPARE_INFERIORS to the Composer (as Decider) for some subset of the Inferiors; PREPARE is sent to each and any of the Inferiors in the subset, excluding any from RESIGN, CANCELLED or PREPARED has been received; the sending of PREPARE will induce the Inferiors to reply with PREPARED, CANCELLED or RESIGN; when replies have been received from all, the Composer (as Decider) replies to the Terminator with INFERIOR_STATUSES, reporting the replies received (which may in fact have been received before the PREPARE_INFERIORS). PREPARE_INFERIORS is not issued to Atom Coordinators.

• The Terminator may issue CANCEL_INFERIORS to the Composer (as Decider) for some subset of the Inferiors; CANCEL is sent to each and any of the Inferiors in the subset, excluding any from RESIGN or CANCELLED has been received; the sending of CANCEL will normally induce the Inferiors to reply with CANCELLED – there are some exception cases; when replies have been received from all, the Composer (as Decider) replies to the Terminator with INFERIOR_STATUSES, reporting the replies received. CANCEL_INFERIORS is not issued to Atom Coordinators. CANCEL_INFERIORS may be issued for an Inferior regardless of whether PREPARED has been received from it.

• The Terminator may issue REQUEST_INFERIOR_STATUSES to the Composer (as Decider) for all or some subset of the Inferiors; the Composer immediately replies with INFERIOR_STATUSES, reporting the current state of the Inferiors as known to the Superior.

Eventually, the Terminator issues one of the completion messages – CANCEL_TRANSACTION or CONFIRM_TRANSACTION. These messages have a flag that determines whether the Terminator wishes to be informed of contradictory and heuristic decisions or hazards within the transaction – this affects when the reply from the Composer (as Decider) is sent to the Terminator. (See section “Autonomous cancel, autonomous confirm and contradictions” for details on contradictory and heuristic cases).

If the message is CANCEL_TRANSACTION, CANCEL is sent to all Inferiors that it has not already been sent to, and from which neither RESIGN or CANCELLED have been received. If the Terminator indicates it does not want to be informed of contradictions, the Composer will immediately reply with TRANSACTION_CANCELLED. Otherwise, if and when CANCELLED or RESIGN has been received from all Inferiors, the Composer replies to the Terminator with TRANSACTION_CANCELLED; but if HAZARD or CONFIRMED is received from any Inferior, the reply is INFERIOR_STATUSES, identifying which Inferior(s) had problems.

If the completion message is CONFIRM_TRANSACTION, the inferiors-list parameter of the message defines the confirm-set. If the parameter is absent (which it must be for an atom
Coordinator), then all Inferiors (excluding only those that have resigned) are the confirm-set; otherwise the confirm-set is only the Inferiors identified in the inferiors-list parameter (less any from which RESIGN has been received). The processing to arrive at the confirm decision is:

- If at the point of receiving CONFIRM_TRANSACTION or at any point before making the confirm decision (see below), CANCELLED is received, then the transaction is cancelled and processing continues as if CANCEL_TRANSACTION had been received.
- If there any Inferiors not in the confirm-set from which neither CANCELLED or RESIGN has been received, CANCEL is sent to them (this cannot happen for Atom Coordinators).
- If initially or later, there is exactly one Inferior in the confirm-set, and either PREPARE has not been sent to it, or PREPARED has been received from it, then at implementation or configuration option, CONFIRM_ONE_PHASE can be sent to that Inferior. This delegates the confirm decision to the Inferior.
- If at any point, RESIGN is received from an Inferior, it is immediately removed from the confirm-set (this may trigger the decision making).
- If there are any Inferiors in the confirm-set from which none of PREPARED, CANCELLED has been received and to which PREPARE has not yet been sent, PREPARE is sent to that Inferior.
- If initially or later, PREPARED has been received from all Inferiors in the confirm-set, the Composer makes the confirm decision; it persists (or attempts to persist) information identifying the Inferiors in the confirm-set; if this fails, the transaction is cancelled and processing continues as if CANCEL_TRANSACTION had been received; if the information is persisted, the confirm decision has been made.

When the confirm decision is made, CONFIRM is sent to all the Inferiors in the confirm-set. And, if on the CONFIRM_TRANSACTION the Terminator indicated it did not wish to be informed of contradictions, TRANSACTION_CONFIRMED is sent to the Terminator.

If the Terminator indicated it wanted to be informed of contradictions, the Composer replies to it with TRANSACTION_CONFIRMED if and when CONFIRMED has been received from all the Inferiors in the confirm-set and CANCELLED or RESIGN has been received from any other Inferiors. If other replies (CANCELLED from a confirm-set Inferior, CONFIRMED from other Inferiors, HAZARD from any) are received, the reply to the Terminator is INFERIOR_STATUSES, identifying which Inferior(s) had problems.

**Confirm-set of intermediates**

An Intermediate, that is a Superior that is also an Inferior, also has a confirm-set, but this is controlled rather differently to the top-most Superior (Decider) described above.

As an Inferior, the interface between the application and BTP elements is not fully defined in this specification. However, within the standard control relationship, issuing BEGIN with a related CONTEXT to a Factory will cause the creation of a Sub-coordinator or Sub-composer (depending on whether the BEGIN parameter asked for atomic or cohesive behaviour). Initially, of course, the new Intermediate has no Inferiors – however, unlike a Participant (in

---

**OASIS BTPDraft Specification 0.9.2.4, 3 April 2002**

Page 40 of
the strict sense of the term), it has a “superior-address” to which ENROL can be sent to enrol
Inferiors. This address is a field of the new CONTEXT.

The behaviour of the Intermediate towards its Inferiors, during the active phase, is basically
the same as for the Decider:

- ENROL messages can be received, adding a new Inferior
- Inferiors may resign - RESIGN is received from an Inferior. The Inferior is
  immediately removed from the set of Inferiors
- CANCELLED may be received from an Inferior
- PREPARED may be received from an Inferior

In some circumstances, receipt of an incoming message allows an Intermediate to
determine that a state change for the whole transaction node takes place. The
Intermediate is able to send messages to its Superior at its own initiative (whereas a
Decider can only respond to a received message from the Terminator), so the receipt of
a message from an Inferior can trigger the sending of messages. This is especially the
case if the Intermediate knows (from application knowledge, perhaps involving
received or sent CONTEXT_REPLY messages) that there will be no further
enrolments. In particular:

- If CANCELLED is received from an Inferior, and this is a Sub-coordinator, the Sub-
  coordinator can itself cancel - CANCEL is sent to other Inferiors, and CANCELLED
to the Superior
- If RESIGN is received from the only Inferior and there will be no other enrolments,
  the Intermediate can itself resign, sending RESIGN to the Superior
- If PREPARED is received from the Superior, it is known there will be no other
  enrolments and this is a Sub-coordinator, the Sub-coordinator can become prepared
  (assuming successful persistence of the appropriate information) and send
  PREPARED to the Superior.

For a Sub-composer, application logic will invariably be involved in determining what effect
a CANCELLED and PREPARED from an Inferior have – though in a real implementation,
this logic may be delegated to the BTP-support software.

The Intermediate may initiate cancellation or the two-phase outcome exchange, either as a
result of receiving the corresponding message (CANCEL, PREPARE) from the Superior, or
triggered by its own controlling application element. For a Sub-composer, this may be partial
- a Sub-composer might be instructed by the application element to cancel some Inferiors and
  send PREPARE to others. Receipt of PREPARE from the Superior will often have a similar
effect to a Decider receiving CONFIRM_TRANSACTION – PREPARE is propagated to all
Inferiors that have not indicated they are PREPARED. However, exactly what happens on
receiving PREPARE will depend on the application – receipt of the PREPARE may be
visible to the application element and cause it to initiate further application activity (perhaps
causing enrolment of new Inferiors) before it is determined whether to propagate PREPARE, and with a Sub-composer, some of the Inferiors may be instructed to cancel instead.

Assuming the Intermediate does not cancel as a whole (in which case CANCEL would be sent to all Inferiors), the Intermediate will at some point attempt to become prepared. If it is a Sub-coordinator, this will require that PREPARED has been received from all Inferiors. For a Sub-composer, application logic will determine from which Inferiors PREPARED is required, with the others being cancelled. In either case, the Intermediate will persist the information about the Inferiors that are to be in the confirm-set and about the Superior, if this persisting is successful, send PREPARED to its own Superior.

If CANCEL is subsequently received from the Superior, this is propagated to all the Inferiors and the persistent information removed (or effectively removed as far as recovery is concerned). It is not important which order this is done in, since the recovery sequence will ensure that a cancel outcome is eventually delivered anyway.

If CONFIRM is received from the Superior (which can only be after sending PREPARED to the Superior), this is likewise propagated to the Inferiors. For a Sub-coordinator, CONFIRM is invariably sent to all Inferiors. However, for a Sub-composer it is possible further application logic intervenes and some of the Inferiors are rejected from the confirm-set at this late stage. (This can only occur when the application work, as defined by the contract to the Superior, can be performed by some sub-set of the Inferiors.) The Intermediate may, but is not required, to change the persistent information to reflect the confirm outcome (though a Sub-composer that selects only some Inferiors probably will need to re-write the information to ensure the correct subset are confirmed despite possible failures). If the information is not changed, then, on recovery, the Intermediate will find itself to be in a prepared state and will interrogate the Superior to re-determine the outcome. If the information is changed, a recovered Intermediate can immediately continue with ordering confirmation to its Inferiors.

If CONFIRM_ONE_PHASE is received from the Superior, either before or after the Intermediate has become PREPARED, the effect is very similar to a Decider receiving CONFIRM_TRANSACTION. If there is only one Inferior, the CONFIRM_ONE_PHASE may be propagated to that Inferior. Otherwise, the Intermediate behaves as a Decider, making a confirm decision if it can.

If one or more Inferiors make contradictory autonomous decisions, or HAZARD is received from an Inferior, the Intermediate may report this to the Superior using HAZARD. However, BTP does not require this. Since the Superior may be owned and controlled by a different organisation, there may be business reasons not to report such problems.

Optimisations and variations

Spontaneous prepared

As described above, before a Superior can order confirmation to an Inferior, the Inferior must become “prepared”, meaning that it is ready to confirm or to cancel as it so ordered and send the PREPARED message as a report of this. In the conventional message sequence, as shown...
above, the Inferior attempts to become prepared when it receives a PREPARE message from the Superior. The PREPARE in turn is sent by the Superior when it receives an appropriate request from its controlling application (or from its own Superior, if there is one). The application controlling the Superior will request the sending of PREPARE when it determines that no further application work associated with this Inferior (or, perhaps with the whole business transaction) will occur.

However, for some applications, the application element controlling the Inferior will know that the application work for which the Inferior will be responsible is complete before a PREPARE is sent from the Superior. In fact, because the application element has autonomy in determining how application work is to be allocated to Inferiors, it is possible for the Inferior-side application element to know the work is complete for a particular Inferior when Superior-side application element will be sending more message to the Inferior-side. (The future work will, probably, require the enrollment of additional Inferiors.)

BTP consequently allows the application element controlling an Inferior to cause the Inferior to become prepared, and to send PREPARED to the Superior without PREPARE having been received from the Superior. From the perspective of the BTP Superior the Inferior sends PREPARED spontaneously. Apart from this, a spontaneous PREPARED message is the same as, and has the same effect and implications as one induced by a PREPARE message.

**One-shot**

In the “conventional” message sequence shown above and assuming the Initiator, Terminator and Coordinator on the one side, and “Service”, Enroller and Participant on the other are located within their respective parties, there are eight messages passed in one direction or the other between the two parties. There are four round-trip exchanges: the application request and response exchange, the ENROL/ENROLLED exchange (going in the opposite direction and overlapped with the application exchange), then PREPARE/PREPARED and the CONFIRM/CONFIRMED. However, if the application exchange is a single request/response, it is possible to reduce these eight to two round-trips– the first of which merges the first three of the conventional sequence. The fundamental two-phase nature of BTP (or any coordination mechanism) means there have to be at least two round trips – one before the confirm-or-cancel decision is made at the Superior, one after. This merging of the exchanges is termed “one-shot”, as it requires only one exchange to take the relationship from non-existent to waiting for the confirm-or-cancel decision.

Figure 14 shows a typical “one-shot” message sequence. The diagram distinguishes an additional aspect of the application elements, labelled “context-handler”. This is not a role in the BTP model, but is used only to distinguish a set of responsibilities and actions. In a real implementation these might be performed by the user application itself, or might be performed by the BTP-supporting infrastructure on the path between the application elements. (Figure 9 could be redrawn to show the context-handlers, but to no particular benefit) As in the conventional case, the CONTEXT is sent related to the application request (the creation of the CONTEXT by the Factory is not shown and is the
same as the conventional case). The “context-handler” is aware of the sending of the CONTEXT.

On the responder (service side), however, when the application element creates the Inferior, the ENROL is not sent immediately, but retained. The application performs the “provisional effect” implied by the received message and the Inferior becomes prepared and issues a PREPARED message, which is also retained. When the application response is available, it is sent with the retained messages and the CONTEXT_REPLY (which indicates that the related ENROL will complete the enrolments implied by the earlier transmission of the CONTEXT.

When this group of messages is received by the context-handler on the client side, the contained ENROL and PREPARED messages are forwarded to the Superior (whose address was on the original CONTEXT and so is known to the context-handler). An ENROLLED message is sent back to the context-handler, assuring it that the enrolment was successful and the application can progress. If enrollment fails and the business transaction is atomic, confirmation must be prevented – this responsibility falls on the context-handler and the client application, since the failure of the enrolment implies that Superior itself is inaccessible. If enrolment fails and the business transaction is a cohesion, the appropriate response is a matter for the application.

With “one-shot”, if there are multiple Inferiors created as a result of a single application message, there is an ENROL and PREPARED message for each one sent related with the CONTEXT_REPLY. If an operation fails, a CANCELLED message may be sent instead of a PREPARED – if the Superior is atomic, this will ensure it cancels, if cohesive, the client application will be aware of this and behave appropriately.

Whether the “one-shot” mechanism is used is determined by the implementation on the responding (Inferior) side. This may be subject to configuration and may also be constrained by the application or by the binding in use.
Resignation

After an Inferior is enrolled, it may be determined that the application work it is responsible for has no real effect – more exactly, that the counter-effect, if cancelled, and the final effect, if confirmed, will be identical. In such a case the Inferior can effectively un-enrol itself by sending a RESIGN message to the Superior. This can be done “spontaneously” (as far as BTP is concerned) or as a response to a received PREPARE message. It cannot be done after the Inferior has become prepared.

An Inferior from which RESIGN has been received is not considered an Inferior in discussion of the confirm-set – the phrase “remaining Inferiors” is used to mean only non-resigned Inferiors.
One-phase confirmation

If a Coordinator or Composer that has been requested to confirm has only one (remaining) Inferior in the confirm-set, it may delegate the confirm-or-cancel decision to that Inferior, just requesting it to confirm rather than performing the two-phase exchange. This is done by sending the CONFIRM_ONE_PHASE message. Unlike the two-phase exchange (PREPARED received, CONFIRM sent), it is possible with CONFIRM_ONE_PHASE for a failure to occur that leads to the original Coordinator or Composer (and its controlling application element – the Terminator) being uncertain whether the outcome was confirmation or cancellation.

Autonomous cancel, autonomous confirm and contradictions

As described above, BTP does not require a Participant, while it is responsible for holding application resources such that can be confirmed or cancelled, to use any particular mechanism for maintaining this state. A Participant that “becomes prepared” may choose to let the “provisional effect” be identical to the “final effect”, and hold a compensating “counter effect” ready to implement cancellation; or it may make the provisional effect effectively null, and only perform the real application work as the final effect if confirmed; or the “provisional effect” may involve performance of the application work and locking application data against other access; or other patterns, as may be constrained or permitted by the application.

Although a Participant is not required to lock data (as would be the case with some other transaction specifications) on becoming prepared, it is nevertheless in a state of doubt, and this doubt may have application or business implications. Accordingly it is recognised that a Participant (or, rather the business party controlling the application element and the Participant) may need to limit the promise made by sending PREPARED, and retain the right to apply its own decision to confirm or cancel to the Participant and the application effects it is responsible for. This is described as an “autonomous” decision. It is closely analogous to the heuristic decisions recognised in other transaction specifications. The only difference is the conceptual one that heuristic decisions are typically considered to occur only as a result of rare and unpredictable failure, whereas BTP recognises that the right to take an autonomous decision may be critical to the willingness of a business party to be involved in the business transaction at all. BTP therefore allows Participants (and all Inferiors) to indicate that there are limits on how long they are willing to promise to remain in the prepared state, and that after that time they may invoke their right of taking an autonomous decision.

Taking an autonomous decision will of course run the risk of breaking the intended consistency of outcome across the business transaction, if the autonomous decision of the Inferior contradicts the decision (for this Inferior) made by the Superior. The Superior will have received the PREPARED message and thus be permitted to make a confirm decision (directly, or through exchanges with a Terminator application element or with its own Superior). An Inferior taking an autonomous decision informs the Superior by sending CONFIRMED or CANCELLED, as appropriate, without waiting for an outcome order from the Superior. This may cross the outcome message from the Superior, or the Superior may not make its decision till later. If the decisions agree, the normal CONFIRM or CANCEL message is sent. In the case of CANCEL, this completes the relationship – the CANCEL and
CANCELLED messages acknowledge each other, regardless of which travels first. In the case of CONFIRM, another CONFIRMED message is needed.

If the Superior’s decision is contradicted by the autonomous decision, the Superior may need to record this, report it to management systems or inform the Terminator application or its own Superior. When this has been done (details are implementation-specific, but may be constrained by the application), the Superior sends a CONTRADICTION message to the Inferior. If an outcome message was sent earlier (crossing the announcement of the autonomous decision), the Inferior will already know there was a contradiction, but the receipt of the CONTRADICTION message informs the Inferior that the Superior knows and has done whatever it considers necessary to cope.

As mentioned, BTP allows an Inferior to inform the Superior, with a qualifier on the PREPARED message, that the promise to remain in the prepared state will expire. In turn this allows the application on the Superior side to avoid risking a contradictory decision by making and sending its own decision in time. The Superior side can also indicate, with another qualifier, a minimum time for which it expects the prepared promise to remain valid.

As well as deliberate and forewarned autonomous decisions, BTP recognises that failures and exceptional conditions may force unplanned autonomous decisions. In the protocol sequence these are treated exactly like planned autonomous decisions – if they contradict, the Superior will be informed and a CONTRADICTION message sent to the Inferior.

Autonomous decisions, planned or unplanned, are equivalent to the heuristic decisions of other transaction systems. The term is avoided in BTP since it may carry implications that it only occurs in an unplanned manner.

Recovery and failure handling

Types of failure

BTP is designed to ensure the delivery of a consistent decision for a business transaction to the parties involved, even in the event of failure. Failures can be classified as:

Communication failure: messages between BTP actors are lost and not delivered. BTP assumes the carrier protocol ensures that messages are either delivered correctly (without corruption) or are lost, but does not assume that all losses are reported nor that messages sent separately are delivered in the order of sending.

Node failure (system failure, site failure): a machine hosting one or more BTP actors stops processing and all its volatile data is lost. BTP assumes a site fails by stopping – it either operates correctly or not at all, it never operates incorrectly.

Communication failure may become known to a BTP implementation by an indication from the lower layers or may be inferred (or suspected) by the expiry of a timeout. Recovery from a communication failure requires only that the two actors can again send messages to each other and continue or complete the progress of the business transaction.
A node failure is distinguished from communication failure because there is loss of volatile state. To ensure consistent application of the decision of a business transaction, BTP requires that some state information will be persisted despite node failure. Exactly what real events correspond to node failure but leave the persistent information undamaged is a matter for implementation choice, depending on application requirements; however, for most application uses, power failure should be survivable (an exception would be if the data manipulated by the associated operations was volatile). In all cases, there will be some level of event sufficiently catastrophic to lose persistent information and the ability to recover—destruction of the computer or bankruptcy of the organisation, for example.

Recovery from node failure involves recreating an accessible communications endpoint in a network node that has access to the persistent information for incomplete transactions. This may be a recreation of the original actor using the same addresses; or using a different address; or there may be a distinct recovery entity, which can access the persistent data, but has a different address; other implementation approaches are possible. The recovered, and possibly relocated actor may or may not be capable of performing new application work. Restoration of the actor from persistent information will often result in a partial loss of state, relative to the volatile state reached before the failure. In some states, there may be total loss of knowledge of the business transaction, including particular Superior:Inferior relationships. After recovery from node failure, the implementation behaves much as if a communication failure had occurred.

**Persistent information**

BTP requires that certain state information is persisted—these are information that records an Inferior’s decision to be prepared, a Superior’s decision to confirm, and an Inferior’s autonomous decision. Requiring the first two to be persistent ensures that a consistent decision can be reached for the business transaction and that it is delivered to all involved nodes, despite failure. Requiring an Inferior’s autonomous decision to be persistent allows BTP to ensure that, if the autonomous decision is contradictory (i.e., opposite to the decision at the Superior), the contradiction will be reported to the Superior, despite failures.

BTP also permits, but does not require, recovery of the Superior:Inferior relationship in the active state (unlike many transaction protocols, where a communication or node failure in active state would invariably cause rollback of the transaction). Recovery in the active state may require that the application exchange is resynchronised as well—BTP does not directly support this, but allows continuation of the business transaction if the application desires it. Apart from the (optional) recovery in active state, BTP follows the well-known presume-abort model—it is only required that information be persisted when decisions are made (and not, for example, on enrolment). This means that on recovery one side may have persistent information while the other does not. This occurs, among other cases, when an Inferior has decided to be prepared but the Superior never confirmed (so the decision is “presumed” to be cancelled), and when the Superior did confirm, the Inferior applied the confirmation and removed its persistent information but the acknowledgement message (CONFIRMED) was never received by the Superior.
Information to be persisted when an Inferior decides to be prepared has to be sufficient to re-establish communication with the Superior, to apply a confirm decision and to apply a cancel decision. It will thus need to include the addressing and identification information for the Superior. The information needed to apply the confirm or cancel decision will depend on the application and the associated operations.

A Superior must persist the corresponding information to allow it to re-establish communication with the Inferior. When it must persist this information depends on its position within the transaction tree. If it is the top of the tree – i.e. it is the Decider for the business transaction -- it need only persist this information if and when it makes a decision to confirm (and, for a Cohesion, only if this Inferior is in the confirm-set). A Superior that is an intermediate in the tree – i.e. it is an Inferior to some other Superior – must persist the information about each of its own Inferiors as part of (or before) persisting its own decision to be prepared. For such an intermediate, the “decision to confirm” as Superior is made when either CONFIRM is received from its Superior or it makes an autonomous decision to confirm. If CONFIRM is received, the persistent information may be changed to show the confirm decision, but alternatively, the receipt of the CONFIRM can be treated as the decision itself and the CONFIRM message propagated to the Inferiors without changing the persistent information. If the persistent information is left unchanged and there is a node failure, on recovery the entity (as an Inferior) will be in a prepared state, and will rediscover the confirm decision (using the recovery exchanges to its Superior) before propagating it to its Inferior(s).

Since BTP messages may carry application-specified qualifiers, and the BTP messages may be repeated if they are lost in transit (see next section), the persistent information may need to include sufficient to recreate the qualifiers, to allow them to be resent with their carrying BTP message. This applies both to qualifiers on PREPARED (which would be persisted by the Inferior) and on CONFIRM (which would be persisted by the Superior).

In some cases, an implementation may not need to make an active change to have a persistent record of a decision, provided that the implementation will restore itself to the appropriate state on recovery. For example, an implementation that, as Inferior, always used the default-is-cancel mechanism, and recorded the timeout (to cancel) in the persistent information on becoming prepared, and always updated or removed that record when it applied a confirm instruction could treat the presence of an expired record as effectively a record of an autonomous cancel decision.

Recovery messages

Once the Superior:Inferior relationship has entered the completion phase – BTP does not generally use special messages in recovery, but merely permits the resending of the previous message – thus, for example, PREPARE, PREPARED, CANCEL, CONFIRM can all be sent repeatedly. Resending the previous message means a possible loss of the original message may be invisible to the receiver. The trigger for this re-sending is implementation dependent – a reported communication failure, a timeout expiry while waiting for a reply, the re-
establishment of communications or the general restoration of function after a node failure.

are all possible triggers. An incoming repetition of the last message received, if it has already
been replied to (e.g. receiving PREPARE after PREPARED has been sent), should normally
trigger a resending of the last message sent – since that sent message may have got lost.

While in the active phase – i.e. prior to entering completion – there is no appropriate last
message that can be sent. However, for active-phase recovery there needs to be some way for
the BTP actors to determine that the peer is still there and still aware of the Superior:Inferior
relationship. In this case, the peers can interrogate each other using the INFERIOR_STATE
or SUPERIOR_STATE messages, informing the peer of their own state and requesting a
response – which may be the opposite message, or one of the main BTP messages (which
perhaps had been lost). If it is another SUP|INFERIOR_STATE message, that reply does not
ask for a response. Receiving a SUP|INFERIOR_STATE messages that asks for a response
does not require an immediate response – especially if an implementation is waiting to
determine a decision (perhaps because it is itself waiting for a decision from elsewhere), an
implementation may choose not to reply until it wishes to.

The SUP|INFERIOR_STATE messages are also used as replies when the receiver of any of
the Superior:Inferior message has determined that there is no corresponding state information
– the targeted Superior or Inferior does not exist (or is known to have completed and is no
longer an active entity). The SUP|INFERIOR_STATE messages with a status of “unknown”
is the indication that the state information does not exist.

The SUP|INFERIOR_STATE messages are also available as replies to any Superior:Inferior
message in the (transient, one hopes) case where, after failure an implementation cannot
currently determine whether the persistent information exists or not, or what its state is, and
so cannot give a definitive answer. The SUP|INFERIOR_STATE messages with a status of
“inaccessible” is the indication that the existence of state information cannot be determined.
The receiver of such a message should normally treat it as a “retry later” suggestion.

Redirection

As described above, BTP uses the presume-abort model for recovery. A corollary of this is
that there are cases where one side will attempt to re-establish communication when there is
no persistent information for the relationship at the far-end, because that side either never
reached a state where the state was persisted, or had been persisted, but then progressed to
remove the state information. In such cases, it is important the side that is attempting
recovery can distinguish between unsuccessful attempts to connect to the holder of the
persistent information and when the information no longer exists. If the peer information does
not exist, the side that is attempting recovery can draw appropriate conclusions (that the peer
either was never prepared, never confirmed or has already completed) and complete its part
of the transaction; if it merely fails to get through, it is stuck in attempting recovery.

---

4 BTP’s capability of binding to alternative carrier protocols is part of the motivation for not having a
distinct recovery message sequence, since the carrier binding does not necessarily have a well-defined
communication failure indication.
Two mechanisms are provided to assist implementation flexibility while allowing completion of Superior:Inferior relationships when only one side has any persistent information. The mechanisms are:

- Address fields which provide the address that will be used by the peer to send messages to an actor (effectively a “callback address”) can be a set of addresses, which are alternatives, one of which is chosen as the target address for the future message. If the sender of that message finds the address does not work, it can try a different alternative.

- The REDIRECT message can be used to inform the peer that an address previously given is no longer valid and to supply a replacement address (or set of addresses). REDIRECT can be issued either as a response to receipt of a message or spontaneously.

The two mechanisms can be used in combination, with one or more of the original set of addresses just being a redirector, which does not itself ever have direct access to the state information for the transaction, but will respond to any message with an appropriate REDIRECT.

REDIRECT as a message is only used on the Superior:Inferior relationship, where each side holds the address of the other. On the other relationships (e.g. Terminator:Decider), one side (e.g. Terminator) has the address of the other, and initiates all the message exchanges. However, the entity whose address is known to the other may itself move - e.g. if a Coordinator, which will be both Decider and Superior changes its address as a Superior, it will probably change its address as a Decider too. In this case, a FAULT reply to a misdirected message can be used, assuming there is some entity available at, or on the path to the old address that understands BTP sufficiently to provide the redirection information.

Some implementations, in which a single addressable entity with one, constant address deals with all transactions, distinguishing them by identifier, will not need to supply “backup” addresses (and would only use REDIRECT if permanently migrated).

**Terminator:Decider failures and transaction timelimit**

BTP does not provide facilities or impose requirements on the recovery of Terminator:Decider relationships, other than allowing messages to be repeated. A Terminator may survive failures (by retaining knowledge of the Decider’s address and identifier), but this is an implementation option. Although a Decider (if it decides to confirm) will persist information about the confirm decision, it is not required, after failure, to remain accessible using the address it originally gave to the Initiator (and used by the Terminator). Any such recovery is an implementation option.

A Decider has no way of initiating a call to a Terminator to ensure that it is still active, and thus no way of detecting that a Terminator has failed. The Decider always has the right to initiate cancellation, but if the application (Terminator) and the Decider have different views about how long a “long time” is, then either the Decider might wait unnecessarily for a completion request (e.g. CONFIRM_TRANSACTION) that will never arrive, or it might
initiate cancellation while the application is still active. To avoid these irritations, a standard qualifier “Transaction timelimit” can be used (by the Initiator) to inform the Decider when it can assume the Terminator will not request confirmation and so it (the Decider) should initiate cancellation.

**Contradictions and hazard**

As described above (see “Autonomous cancel, autonomous confirm and contradictions”), in some circumstances an Inferior may apply a decision that is contradictory to the decision of the Superior. This can occur in a semi-planned manner, when the Inferior has announced a timeout on the PREPARED message but no outcome message has been received, or as a result of an exceptional condition that forces the Inferior to break the promise implicit in PREPARED, regardless of timers. In both cases, this is considered an autonomous decision by the Inferior. An autonomous decision, of itself, does not imply a contradiction – it only results in a contradiction if the decision is opposite to that of the Superior (in the case of a cohesive Superior, opposite to the decision that applies to this Inferior).

In order to ensure that a contradiction is detected despite node and communication failures, it is required that information about the taking of the autonomous decision be persisted until a BTP message received from the Superior indicates either that there was no contradiction (the decisions were in line – CANCEL is received after an autonomous cancel or CONFIRM is received after an autonomous confirm) or that the Superior is aware of the contradiction (CONTRADICTION is received). Note that the Inferior will become aware of the fact of the contradiction when it receives the “wrong” message, but must retain the record of its own decision until it receives the CONTRADICTION message, which tells it the Superior knows too.

The Superior’s action on becoming aware of the contradiction is not determined by this specification. In particular, if the Superior is a Sub-coordinator or Sub-composer, it is not required by this specification to report the contradiction to its own Superior (which may, for example, be controlled by a different organisation). The Superior may report the problem to management systems or record it for manual repair. However, BTP does provide mechanisms to report the contradiction to the next higher Superior (if there is one) or to the Terminator application element.

A contradiction occurring in an Inferior will usually mean the immediate Superior has a “mixed” condition – some of the application work it was responsible for has confirmed, some has cancelled (and contrary to any cohesion confirm-set selection). If the Superior is a Sub-coordinator or Sub-composer, it can report the mixed condition to its own Superior with the HAZARD message. If the Superior is the top-most in the tree, it can report the problem with the INFERIOR_STATUSES message, which will detail the state of all the Inferiors.

If a Sub-coordinator or Sub-composer having sent (or attempted to send) the outcome message to its Inferiors, is temporarily unable to get a response (CONFIRMED or CANCELLED), it may either wait until a response does come back or choose to reply to its own Superior with a HAZARD message indicating that a contradiction is “possible”. If it does choose to send HAZARD, it is required to persist a record of this until it receives a
CONTRADICTION message from the Superior, or a message from the Inferior indicating there was no contradiction in fact.

HAZARD is also used to indicate that it has become impossible to cleanly and consistently achieve either a confirmed or a cancelled state for the application work. In this case, there is no guarantee that the problem will be reliably reported – especially because it may be the inability to persist information that is the cause of the problem.

Relation of BTP to application and carrier protocols

BTP messages are communicated between actors in two distinguishable circumstances:

a) in establishing and progressing the outcome and control relationships between BTP actors, and between application elements and BTP actors – Initiator:Factory, Terminator:Decider, Superior:Inferior etc.

b) in association with application messages that are communicated between application elements.

In the first case, interoperable communication requires a specification of how the abstract BTP messages are represented and encoded, and how they are transmitted. This specification is a carrier protocol binding (or just “binding”, if the context is clear). BTP allows bindings to a multiplicity of carrier protocols. The only requirement that BTP makes is that the transmission of a message either delivers an uncorrupted message or fails. BTP does not require that the carrier report failure to deliver a message, to either side, nor that messages are delivered in the order they are sent (though implementations can take advantage of information from a richer carrier, which can improve performance in various ways). BTP messages communicated in this way have semantics that are defined in this specification – a PREPARE message (for example), refers back to the ENROL via the “inferior-identifier” parameter and is an instruction to the Inferior to become and report that it is prepared.

In the second case, the full semantics cannot be defined in this specification. Interoperation with BTP requires that the parties have a common understanding of what is being confirmed or cancelled, but this mutual understanding is defined by the contract of the application, not by BTP. (The contract may be explicit or implicit, declared by one side as take-it-or-leave-it, or may be negotiated in some way.) Part of this contract will include how the combination of the application protocol (i.e. the application messages and their sequencing) and BTP operate such that the two sides are agreed as to which application operations are part of which business transaction. This will often be achieved by sending application messages and BTP messages in “association” in some way – thus an application message sent in association with a CONTEXT can be specified (by the application contract) to mean that if work is done as result of the receipt of the message, one or more Inferiors should be enrolled to apply the confirm/cancel decision to that work. Similarly, an application message may be sent associated with an ENROL with the contractual understanding that the message refers to some application work that has been made the responsibility of the Inferior being enrolled.

The concrete representation of this “association” is also a matter for the application protocol specification. There are several ways this can be done, including:
• the BTP message is contained within the application message, or both are contained within a larger construct;
• the application message contains a field that is the superior-identifier or inferior-identifier that is also present on the CONTEXT or the ENROL;
• the BTP message contains a qualifier that references (a field of) the application message in some way (e.g. if the application message is an invoice, the qualifier might contain the invoice number);
• the encoding of the BTP and application messages reference each other (e.g. using XML id and refid attributes).

In all cases, the application specification will need to define the mechanism so that both parties have common understanding. Many applications will use the same mechanism and their specifications can therefore take advantage of standard patterns, and their implementations of standard tools.

The association of an application message with a BTP message is analogous to the concept of “related” BTP messages. “Related” BTP messages are sent as a group, with a declared and defined semantic for the group. Associated application and BTP messages can be considered as “related”, with the proviso that the semantic is defined by the application, not by BTP.

There is no necessary relationship between how the application messages and any associated BTP messages are transmitted by carrier protocols, and the carrier binding for the BTP messages. BTP messages are invariably sent to a BTP actor whose address has been passed to the sender by some means – thus a CONTEXT contains the address of the Superior to which ENROLs will be sent, and the ENROL contains the address of the Inferior. Similarly, BEGUN contains the address (as Decider) of the new Composer or Coordinator. These addresses are all sets of addresses (possibly of cardinality one), and each individual address identifies which binding is to be used. Thus, for example, when a CONTEXT is sent associated with an application message, the ENROL will travel on a carrier binding identified by the particular address from the CONTEXT that the Enroller chooses to use – which may have no relationship to how the application message arrived.

Despite this, it will be common that the application binding and the BTP binding will use the same carrier. This is the case in the bindings specified in this edition of the specification, which define a binding of BTP to SOAP 1.1 over HTTP. Included in this SOAP/HTTP binding specification, are rules that allow an application to associate (relate) a single CONTEXT or a single ENROL (carried in the SOAP header) with the application message(s) carried in the SOAP body.

Other elements

Identifiers

5 The “application specification”, or “application protocol specification” may be very informal or may be a standardised agreement.
An Identifier is a globally unambiguous identification of the state corresponding to one of
Decider, Superior or Inferior. Where a single entity has more than one of these roles (at the
same node in the same transaction, as with a Sub-coordinator that is both Superior and
Inferior), the Identifiers may be the same or different, at implementation option - they are
distinguished by which messages the Identifier is used on. (A Superior has only one Superior-
identifier, although it may be in multiple Superior:Inferior relationships, each with a separate
state in terms of the state table).

The state identified by an Identifier can be accessed by BTP messages sent to any of the
addresses supplied with the Identifier in the appropriate message (CONTEXT, BEGUN,
ENROL), or as updated by REDIRECT. An Identifier itself has no location implications.
(Identifiers are specified, in the XML representation, as syntactically URIs - by their use as
names of BTP entities, they are URNs. If an Identifier happens to specify an network location
(i.e. it is a URL), it is treated as an opaque value by BTP)

Identifiers are specified as being globally unambiguous - the same Identifier only ever
identifies one Decider, Superior or Inferior over all systems and all time. In practice, an
Identifier could be re-used if there is no possibility of the colliding values being confused.
However implementations are recommended to use truly unambiguous Identifiers (that is to
use them as URNs).

Addresses
In most cases, BTP actors that need to communicate are informed of each others addresses
from received BTP messages. When an Inferior is to be enrolled, a CONTEXT message
which contains the address of the Superior will have been received or otherwise passed to the
Enroller and the Inferior. The ENROL message received by the Superior contains the address
of the Inferior. The BEGUN returned from a Factory to the Initiator contains the address of
the Decider, and this can be passed to the Terminator or any Status Requestor.

The addresses carried in these messages (which are effectively “call-back” addresses, to be
used as the destination of future messages) are sets of tripartite addresses. Each contains an
identifier (binding name) for the binding to an underlying transport, or carrier protocol, a
“binding address”, in a format specific to the carrier which is the information necessary to
connect using that carrier, and an optional additional information field. This additional
information is opaque to all but the future destination (which also created this address for
itself) and is used however the implementation there wishes (e.g. it can be used to distinguish
a particular program object, or to relay on, perhaps over a different protocol). The multiple
members of the set allow support of multiple carrier bindings (including both different
versions of standard bindings and proprietary bindings) and for relocation of the BTP actor.

When a message is actually to be sent, the sender, possessing the set of addresses for the
destination, chooses one - restricting its choice to bindings that it supports obviously, but not
otherwise constrained by the specification. The binding address will be used by the senders
carrier implementation (depending on the protocol, the address may or may not be transmitted
– with http, for example, it is). The additional information, if present, will be included in the
BTP message. The chosen address is considered the “target-address” when considering the
abstract message, but only the additional information will normally appear within the
encoded BTP-message (the encoding used is part of the binding specification, which could
require that all of the address is (redundantly) transmitted, if the specifier so chose).

Where a BTP message invokes a reply – as with the Initiator:Factory, Terminator:Decider
and Status Requestor:various roles – the receiver (Factory, Decider, etc) of the message will
not know a priori the address of the sender. Accordingly, in these cases the abstract messages
are specified as containing a single “reply-address”. Depending on the binding, and the
particular use of the binding, the “reply-address” may be directly represented in the encoding
of the BTP message, or may be implicit in the carrier protocol. Similar considerations apply
in the Superior:Inferior relationship, where although the addresses are normally known by the
other side, there are cases when a message is received, and must be responded to, but the peer
is unknown. Accordingly, the Superior:Inferior messages contain (in abstract) a single
“senders-address”. As with the the “reply-address”es, it may be implicit in the carrier
protocol.

The CONTEXT message does not contain a “target-address”, even as an abstract message, as
it is never transmitted between BTP actors on its own – it is always either related to a BTP
BEGIN or BEGUN message, or is passed between application elements with some
(application-detailed) association with application messages.

Qualifiers
Qualifiers are elements of the BTP messages used to exchange additional information
between the actors. Qualifiers can be specified in the BTP specification (“standard
qualifiers”), by industry groups, by BTP implementors or for the purposes of particular
applications. Of the standard qualifiers in this version of the specification some are
constraints on the BTP contract, such as time limits, and some are further identifiers used to
distinguish specific parties in the BTP interchange. Non-standard qualifiers could extend the
protocol or carry application-specific information.

Overview of the Business Transaction Protocol

A Business Transaction is a consistent change in the state of a business relationship between
two or more parties. BTP provides means to allow the consistent and coordinated changes in
the relationship as viewed from each party.

BTP assumes that for a given business transaction state changes occur, or are desired, in some
set of parties, and that these changes are related in some business-defined manner.

Typically business-defined messages (“application messages”) are exchanged between the
parties to the transaction, which result in the performance of some set of operations. These
operations create provisional or tentative state changes (the transaction’s effect). The
provisional changes of each party must either be confirmed (given final effect), or must be
cancelled (counter-effected). Those parties which are confirmed create an atomic unit, within
which the business transaction should have a consistent final effect.
The meaning of “effect”, “final effect” and “counter-effect” is specific to each business transaction and to each party’s role within it. A party may log intended changes (as its effect) and only process them as visible state changes on confirmation (its final effect). Or it may make visible state changes and store the information needed to cancel (its effect), and then simply delete the information needed for cancellation (its final effect). A counter-effect may be a precise inversion or removal of provisional changes, or it may be the processing of operations that in some way compensate for, make good, alleviate or supplement their effect.

To ensure that confirmation or cancellation of the provisional effect within different parties can be consistently performed, it is necessary that each party should

- determine whether it is able both to cancel (counter-effect) and to confirm (give final effect to) its effect
- report its ability or inability to cancel or confirm (its preparedness) to a central coordinating entity

After receiving these reports, the coordinating entity is responsible for determining which of the parties should be instructed to confirm and which should be instructed to cancel.

Such a two-phase exchange (ask, instruct) mediated by a central coordinator is required to achieve a consistent outcome for a set of operations. BTP defines the means for software agents executing on network nodes to interoperate using a two-phase coordination protocol, leading either to the abandonment of the entire attempted transaction, or to the selection of an internally consistent set of confirmed operations.

BTP centres on the bilateral relationship between the computer systems of the coordinating entity and those of one of the parties in the overall business transaction. In that relationship a software agent within the coordinating entity’s systems plays the BTP role of Superior for a given transaction and one or more software agents within the systems of the party play the BTP role of Inferior. Each Inferior has one Superior, therefore, while a single Superior may have multiple Inferiors within each party to the transaction, and may be related to Inferiors within multiple parties. Each Superior-Inferior pair exchanges protocol-defined messages.

An Inferior is associated with some set of operation invocations that creates effect (provisional or tentative changes) within the party, for a given business transaction. The Inferior is responsible for reporting to its related Superior whether its associated operations’ effect can be confirmed/cancelled. A Superior is responsible for gathering the reports of all of its Inferiors, in order to ascertain which should be cancelled or confirmed. For example, if a Superior is acting as an atomic Coordinator it will treat any Inferior which cannot prepare to cancel/confirm as having veto power over the whole business transaction, causing the Superior to instruct all its Inferiors to cancel. A Superior may, under the dictates of a controlling application, increase or reduce the set of Inferiors to which a common confirm or cancel outcome may be delivered. Thus, the set of prepared Inferiors may be larger than the set of confirmed Inferiors.
An Inferior:Superior relationship is typically established in relation to one or more
application messages sent from one part of the application (linked to the Superior) to some
other part of the application to request the performance of operations that are to be subject to
the confirm or cancel decision of the Superior. If an application is divided between a client
and a service, which use RPCs to communicate application requests and responses, then the
client would typically be associated with the Superior and the service would typically host the
Inferior(s). (BTP does not mandate such an application topology nor does it require the use of
RPC or any other application communication paradigm.)

BTP defines a CONTEXT message that can be sent “in relation to” such application
messages. On receipt of a CONTEXT, one or more Inferiors may be created and “enrolled”
with the Superior, establishing the Superior:Inferior relationships. The particular mechanisms
by which a CONTEXT is “related” to application messages is an issue for the application
protocol and its binding to carrier mechanisms. BTP does not require that the enrolment is
requested by any particular entity—in a particular implementation this may be done by the
Inferior itself, by parts of the application or by other entities involved in the transmission of
the CONTEXT and the application messages. BTP defines a CONTEXT_REPLY message
that can be sent on the return path of the CONTEXT to indicate whether the enrolment was
successful. Without CONTEXT_REPLY it would be possible for a Superior to have an
incorrect view of which Inferiors it was supposed to involve in its confirm decision.

It should be noted that this BTP specification recognises that:

— an Inferior may itself be a Superior to other BTP Inferiors; this occurs when some of
the operations associated with the Inferior involve other application elements whose
operations are to be subject to the confirm/cancel instruction sent to the Inferior. The
specification treats any lower Inferiors as part of the associated operations;

— the requirement on an Inferior to be able to confirm or cancel does not include any
specific mechanism to determine the isolation of the effects of operations; the
requirement is only that the Inferior is able to confirm or cancel the operations, as
their effects are known to the Superior and the application directly in contact with the
Superior. Thus the confirm or cancel requirement may be achieved by performing all
the operations and remembering a compensating counter operation (that will be
triggered by a cancel order); or by remembering the operations (having checked they
are valid) and performing them only if a confirm order is received; or by forbidding
any other access to data changed by the operations and releasing them in their
unchanged state (if cancelled) or their changed state (if confirmed); or by various
combinations of these. In addition, a cancellation may not return data to their original
state, but only to a state accepted by the application as appropriate to a cancelled
operation.

Actors, Roles and Relationships

Actors are software agents which process computations. BTP actors are addressable for the purposes of receiving application and BTP protocol messages transmitted over some underlying communications or carrier protocol. (See section “Addressing” for more detail.)

BTP actors play roles in the sending, receiving and processing of messages. These roles are associated with responsibilities or obligations under the terms of software contracts defined by this specification. (These contracts are stated formally in the sections entitled “Abstract Messages and Associated Contracts” and “State Tables”.) A BTP actor’s computations put the contracts into effect.

A role is defined and described in terms of a single business transaction. An implementation supporting a role may, as an addressable entity, play the same role in multiple business transactions, simultaneously or consecutively, or a separate addressable entity may be created for each transaction. This is a choice for the implementer, and the addressing mechanisms allow interoperation between implementations that make different choices.

Within a single transaction, one actor may play several roles, or each role may be assigned to a distinct actor. This is again a choice for the implementer. An actor playing a role is termed an “actor-in-role”.

Actors may interoperate, in the sense that the roles played by actors may be implemented using software created by different vendors for each actor-in-role. The section “Conformance”, gives guidelines on the groups of roles that may be implemented in a partial, interoperable implementation of BTP.

The descriptions of the roles concentrate on the normal progression of a business transaction, and some of the more important divergences from this. They do not cover all exception cases – the message set definition and the state tables provide a more comprehensive specification.

Note – A BTP role is approximately equivalent to an interface in some distributed computing mechanisms, or a port-type in WSDL. The definition of a role includes behaviour.

Relationships

There are two primary relationships in BTP.
Between an application element that determines that a business transaction should be completed (the role of Terminator) and the BTP actor at the top of the transaction tree (the role of Decider);

Between BTP actors within the tree, where one (the Superior) will inform the other (the Inferior) what the outcome decision is.

These primary relationships are involved in arriving at a decision on the outcome of a business transaction, and propagating that decision to all parties to the transaction. Taking the path that is followed when a business transaction is confirmed:

1. The Terminator determines that the business transaction should confirm, if it can; or (for a Cohesion), which parts should confirm
2. The Terminator asks the Decider to apply the desired outcome to the tree, if it can guarantee the consistency of the confirm decision
3. The Decider, which is Superior to one or more Inferiors, asks its Inferiors if they can agree to a confirm decision (for a Cohesion, this may not be all the Inferiors)
4. If any of those Inferiors are also Superiors, they ask their Inferiors and so on down the tree
5. Inferiors that are not Superiors report if they can agree to a confirm to their Superior
6. Inferiors that are also Superiors report their agreement only if they received such agreement from their Inferiors, and can agree themselves
7. Eventually agreement (or not) is reported to the Decider. If all have agreed, the Decider makes and persists the confirm decision (hence the term “Decider” – it decides, everything else just asked); if any have disagreed, or if the confirm decision cannot be persisted, a cancel decision is made
8. The Decider, as Superior tells its Inferiors of the outcome
9. Inferiors that are also Superiors tell their Inferiors, recursively down the tree
10. The Decider replies to the Terminator’s request to confirm, reporting the outcome decision

There are other relationships that are secondary to Terminator:Decider, Superior:Inferior, mostly involved in the establishment of the primary relationships. The various particular relationships can be grouped as the “control” relationships – primarily Terminator:Decider, but also Initiator:Factory; and the “outcome” relationships – primarily Superior:Inferior, but also Enroller:Superior.

The two groups of relationships are linked in that a Decider is a Superior to one or more Inferiors. There are also similarities in the semantics of some of the exchanges (messages) within the relationships. However they differ in that
1. All exchanges between Terminator and Decider are initiated by the Terminator (it is essentially a request/response relationship); either of Superior or Inferior may initiate messages to the other.

2. The Superior:Inferior relationship is recoverable – depending on the progress of the relationship, the two sides will re-establish their shared state after failure; the Terminator:Decider relationship is not recoverable.

3. The nature of the Superior:Inferior relationship requires that the two parties know of each other’s addresses from when the relationship is established; the Decider does not need to know the address of the Terminator (provided it has some way of returning the response to a received message).

Roles

Figure 15 and Figure 16 show the BTP roles that are specialisations of the central Superior and Inferior roles.

![Diagram of BTP roles]

Figure 15: Superior and derived roles
In the following sections, the responsibility of each role is defined, and the messages that are sent or received by that role are listed. Note that some roles exist only to have a name for an actor that issues a message and receives a reply to that message. Some of these roles may be played by several actors in the course of a single business transaction.

For each role, a table shows which messages are received and sent. Where the messages appear on the same line, the second is a reply to the first. (Consequently the columns are sometimes sent first, received second, sometimes vice versa.)

Roles involved in the outcome relationships

Superior

Accepts enrolments from Inferiors from Enrollers, establishing a Superior:Inferior relationship with each. In cooperation with other actors and constrained by the messages exchanged with the Inferior, the Superior determines the Outcome applicable to the Inferior and informs the Inferior by sending CONFIRM or CANCEL. This outcome can be confirm only if a PREPARED message is received from the Inferior, and if a record, identifying the Inferior can be persisted. (Whether this record is also a record of a confirm decision depends on the Superior’s position in the business transaction as a whole.). The Superior must retain this persistent record until it receives a CONFIRMED (or, in exceptional cases, CANCELLED or HAZARD) from the Inferior.

A Superior may delegate the taking of the confirm or cancel decision to an Inferior, if there is only one Inferior, by sending CONFIRM_ONE_PhASE.

A Superior may be Atomic or Cohesive; an Atomic Superior will apply the same decision to all of its Inferiors; a Cohesive Superior may apply confirm to some Inferiors and cancel to others, or may confirm some after others have reported cancellation. The set of Inferiors that the Superior confirms (or attempts to confirm) is called the “confirm-set”.

If RESIGN is received from an Inferior, the Superior:Inferior relationship is ended; the Inferior has no further effect on the behaviour of the Superior as a whole.
<table>
<thead>
<tr>
<th>Superior receives</th>
<th>Superior sends</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENROL</td>
<td>ENROLLED</td>
</tr>
<tr>
<td>PREPARE</td>
<td></td>
</tr>
<tr>
<td>CONFIRM</td>
<td></td>
</tr>
<tr>
<td>CANCEL</td>
<td></td>
</tr>
<tr>
<td>RESIGNED</td>
<td></td>
</tr>
<tr>
<td>CONFIRM_ONE_PHASE</td>
<td></td>
</tr>
<tr>
<td>CONTRADICTION</td>
<td></td>
</tr>
<tr>
<td>SUPERIOR_STATE</td>
<td></td>
</tr>
<tr>
<td>PREPARED</td>
<td></td>
</tr>
<tr>
<td>CONFIRMED</td>
<td></td>
</tr>
<tr>
<td>CANCELLED</td>
<td></td>
</tr>
<tr>
<td>HAZARD</td>
<td></td>
</tr>
<tr>
<td>RESIGN</td>
<td></td>
</tr>
<tr>
<td>INFERIOR_STATE</td>
<td></td>
</tr>
<tr>
<td>REQUEST_STATUS</td>
<td>STATUS</td>
</tr>
<tr>
<td>REQUEST_INFERIORS_STATUS</td>
<td>INFERIOR_STATUSES</td>
</tr>
</tbody>
</table>

Receipt of ENROL establishes a new Superior:Inferior relationship (unless the ENROL is a duplicate). ENROLLED is sent only if a reply is asked for on the ENROL. A Superior receives ENROL to enrol a new Inferior, establishing a new Superior:Inferior relationship. A Superior sends ENROLLED in reply to ENROL, if the appropriate parameter on the ENROL asked for the reply. A Superior sends PREPARE, CONFIRM, CANCEL, RESIGNED, CONFIRM_ONE_PHASE, SUPERIOR_STATE to an enrolled Inferior. A Superior receives
Inferior

Responsible for applying the Outcome to some set of associated operations – the application determines which operations are the responsibility of a particular Inferior.

An Inferior is **Enrolled** with a single Superior (hereafter referred to as “its Superior”), establishing a Superior:Inferior relationship. If the Inferior is able to ensure that either a confirm or cancel decision can be applied to the associated operations, and can persist information to retain that condition, it sends a PREPARED message to the Superior. When the Outcome is received from the Superior, the Inferior applies it, deletes the persistent information, and replies with CANCELLED or CONFIRMED as appropriate.

If an Inferior is unable to come to a prepared state, it cancels the associated operations and informs the Superior with a CANCELLED message. If it is unable to either come to a prepared state, or to cancel the associated operations, it informs the Superior with a HAZARD message.

An Inferior that has become prepared may, exceptionally, make an autonomous decision to be applied to the associated operations, without waiting for the Outcome from the Superior. It is required to persist this autonomous decision and report it to the Superior with CONFIRMED or CANCELLED as appropriate. If, when CONFIRM or CANCEL is received, the autonomous decision and the decision received from the Superior are contradictory, the Inferior must retain the record of the autonomous decision until receiving a CONTRADICTION message.

<table>
<thead>
<tr>
<th>Inferior receives</th>
<th>Inferior sends</th>
</tr>
</thead>
<tbody>
<tr>
<td>PREPARE</td>
<td>PREPARED</td>
</tr>
<tr>
<td>CONFIRM</td>
<td>CONFIRMED</td>
</tr>
<tr>
<td>CANCEL</td>
<td>CANCELLED</td>
</tr>
<tr>
<td>RESIGNED</td>
<td></td>
</tr>
<tr>
<td>CONFIRM_ONE_PHASE</td>
<td></td>
</tr>
<tr>
<td>CONTRADICTION</td>
<td></td>
</tr>
<tr>
<td>SUPERIOR_STATE</td>
<td>PREPARED</td>
</tr>
<tr>
<td></td>
<td>CONFIRMED</td>
</tr>
<tr>
<td></td>
<td>CANCELLED</td>
</tr>
</tbody>
</table>

from an enrolled Inferior.
An Inferior receives

- PREPARE
- CONFIRM
- CANCEL
- RESIGNED
- CONFIRM_ONE_PHASE
- SUPERIOR_STATE

from its Superior.

An Inferior sends

- PREPARED
- CANCELLED
- CONFIRMED
- HAZARD
- RESIGN
- INFERIOR_STATE

to its Superior.

Enroller

Causes the enrolment of an Inferior with a Superior. This role is distinguished because in some implementations the enrolment request will be performed by the application, in some the application will ask the actor that will play the role of Inferior to enrol itself, and a Factory may enrol a new Inferior (which will also be Superior) as a result of receiving BEGIN&CONTEXT.

<table>
<thead>
<tr>
<th>Enroller sends</th>
<th>Enroller receives</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENROL</td>
<td>ENROLLER</td>
</tr>
</tbody>
</table>

An Enroller sends

- ENROL

to a Superior.

An Enroller receives
ENROLLED is received only if the
in reply to ENROL if the Enroller asked for a response when the ENROL was sent.
An ENROL message sent from an Enroller that did not require an ENROLLED response may
be modified en route to the Superior by an intermediate actor to ask for an ENROLLED
response to be sent to the intermediate. (This may occur in the “one-shot” scenario, where an
ENROL/no-rsp-req is received in relation to a CONTEXT_REPLY/related; the receiver of
the CONTEXT_REPLY will need to ensure the enrolment is successful).

Participant
An Inferior which is specialized for the purposes of an application. Some application
operations are associated directly with the Participant, which is responsible for determining
whether a prepared condition is possible for them, and for applying the outcome. (“associated
directly” as opposed to involving another BTP Superior:Inferior relationship, in which this
actor is the Superior).
The associated operations may be performed by the actor that has the role of Participant, or
they may be performed by another actor, and only the confirm/cancel application is
performed by the Participant.
In either case, the Participant, as part of becoming prepared (i.e. before it can send
PREPARED to the Superior), will persist information allowing it apply a confirm decision to
the operations and to apply a cancel decision. The nature of this information depends on the
operations.

Note – Possible approaches are:

- The operations may be performed completely and the
  Participant persists information to perform counter-effect
  operations (compensating operations) to apply
  cancellation;
- The operations may be just checked and not performed at
  all; the Participant persists information to perform them to
  apply confirmation;
- The Participants persists the prior state of data affected by
  the operations and the operations are performed; the
  Participant restores the prior state to apply cancellation;
- As the previous, but other access to the affected data is
  forbidden until the decision is known
Since a Participant is an Inferior, it sends and receives the messages for an Inferior.

Sub-coordinator

An Inferior which is also an Atomic Superior.

A sub-coordinator is the Inferior in one Superior:Inferior relationship and the Superior in one or more Superior:Inferior relationships.

From the perspective of its Superior (the one the sub-coordinator is Inferior to), there is no difference between a sub-coordinator and any other Inferior. From this perspective, the “associated operations” of the sub-coordinator as an Inferior include the relationships with its Inferiors.

A sub-coordinator does not become prepared (and send PREPARED to its Superior) until and unless it has received PREPARED (or RESIGN) from all its Inferiors. The outcome is propagated to all Inferiors.

Since a Sub-coordinator is both an Inferior and a Superior, it sends and receives the messages for both.

Sub-composer

An Inferior which is also a Cohesive Superior.

Like a sub-coordinator, a sub-composer cannot be distinguished from any other Inferior from the perspective of its Superior.

A sub-composer is similar to a sub-coordinator, except that the constraints linking the different Inferiors concern only those Inferiors in the confirm-set. How the confirm-set is controlled, and when, is not defined in this specification.

If the sub-composer is instructed to cancel, by receiving a CANCEL message from its Superior, the cancellation is propagated to all its Inferiors.

Since a Sub-composer is both an Inferior and a Superior, it sends and receives the messages for both.

Roles involved in the control relationships

Decider

A Superior that is not also the Inferior on a Superior:Inferior relationship. It is the top-node in the transaction tree and receives requests from a Terminator as to the desired outcome for the business transaction. If the Terminator asks the Decider to confirm the business transaction, it is the responsibility of the Decider to finally take the confirm decision. The taking of the
A Decider is instructed to cancel by receiving CANCEL_TRANSACTION.

A Decider that is an Atomic Superior (all Inferiors will have the same outcome) is a Coordinator. A Decider that is a Cohesive Superior (some Inferiors may cancel, some confirm) is a Cohesion.

<table>
<thead>
<tr>
<th>Decider receives</th>
<th>Decider sends</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONFIRM_TRANSACTION</td>
<td>TRANSACTION_CONFIRMED</td>
</tr>
<tr>
<td></td>
<td>TRANSACTION_CANCELLED</td>
</tr>
<tr>
<td></td>
<td>INFERIOR_STATUSES</td>
</tr>
<tr>
<td>CANCEL_TRANSACTION</td>
<td>TRANSACTION_CANCELLED</td>
</tr>
<tr>
<td></td>
<td>INFERIOR_STATUSES</td>
</tr>
<tr>
<td>REQUEST_INFERIOR_STATUSES</td>
<td>INFERIOR_STATUSES</td>
</tr>
</tbody>
</table>

A Decider is also a Superior and thus sends and receives the messages for a Superior.

All Deciders receive
- CONFIRM_TRANSACTION
- CANCEL_TRANSACTION
- REQUEST_INFERIOR_STATUSES

All Deciders send
- TRANSACTION_CONFIRMED
- TRANSACTION_CANCELLED
- INFERIOR_STATUSES

Coordinator

A Decider that is an Atomic Superior. The same outcome decision will be applied to all Inferiors (excluding any from which RESIGN is received).

PREPARED must be received from all remaining Inferiors for a confirm decision to be taken.

A Coordinator must make a cancel decision if
- it is instructed to cancel by the Terminator
- if CANCELLED is received from any Inferior
- if it is unable to persist a confirm decision

Since a Coordinator is a Decider, it receives the messages appropriate for a Decider and a Superior.

Composer
A Decider that is a Cohesive Superior. If the Terminator requests confirmation of the Cohesion, that request will determine the confirm-set of the Cohesion.

PREPARED must be received from all Inferiors in the confirm-set (excluding any from which RESIGN is received) for a confirm decision to be taken.

A Composer must make a cancel decision (applying to all Inferiors) if it is instructed to cancel by the Terminator if CANCELLED is received from any Inferior in the confirm-set if it is unable to persist a confirm decision

A Composer may be asked to prepare some or all of its Inferiors by receiving PREPARE_INFERIORS. It issues PREPARE to any of those Inferiors from which none of PREPARED, CANCELLED or RESIGN have been received, and replies to the PREPARE_INFERIORS with INFERIOR_STATUSES.

A Composer may be asked to cancel some of its Inferiors, but not itself, by receiving CANCEL_INFERIORS.

Terminator

Asks a Decider to confirm the business transaction, or instructs it to cancel all or (for a Cohesion) part of the business transaction.

All communications between Terminator and Decider are initiated by the Terminator. A Terminator is usually an application element.

A request to confirm is made by sending CONFIRM_TRANSACTION to the target Decider. If the Decider is a Cohesion Composer, the Terminator may select which of the Composer’s Inferiors are to be included in the confirm-set. If the Decider is an Atom Coordinator, all Inferiors are included. After applying the decision, the Decider replies with TRANSACTION_CONFIRMED, TRANSACTION_CANCELLED or (in the case of problems) INFERIOR_STATUSES.

A Terminator may ask a Composer (but not a Coordinator) to prepare some or all of its Inferiors with PREPARE_INFERIORS. The Composer replies with INFERIOR_STATUSES.

A Terminator may send CANCEL_TRANSACTION to instruct the Decider to cancel the whole business transaction... The Decider replies with CANCEL_COMPLETE if all Inferiors cancel successfully, and with INFERIOR_STATUSES in the case of problems.. If the
Decider is a Cohesion Composer, the Terminator may send CANCEL_INFERIORS to cancel some of the Inferiors; the Decider always replies with INFERIOR_STATUSES.

A Terminator may check the status of the Inferiors of the Decider by sending REQUEST_INFERIOR_STATUSES. The Decider replies with INFERIOR_STATUSES.

<table>
<thead>
<tr>
<th>Terminator sends</th>
<th>Terminator receives</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONFIRM_TRANSACTION</td>
<td>TRANSACTION_CONFIRMED</td>
</tr>
<tr>
<td></td>
<td>TRANSACTION_CANCELLED</td>
</tr>
<tr>
<td></td>
<td>INFERIOR_STATUSES</td>
</tr>
<tr>
<td>CANCEL_TRANSACTION</td>
<td>TRANSACTION_CANCELLED</td>
</tr>
<tr>
<td></td>
<td>INFERIOR_STATUSES</td>
</tr>
<tr>
<td>PREPARE_INFERIORS</td>
<td>INFERIOR_STATUSES</td>
</tr>
<tr>
<td>CANCEL_INFERIORS</td>
<td>INFERIOR_STATUSES</td>
</tr>
<tr>
<td>REQUEST_INFERIOR_STATUSES</td>
<td>INFERIOR_STATUSES</td>
</tr>
</tbody>
</table>

A Terminator sends
- CONFIRM_TRANSACTION
- CANCEL_TRANSACTION
- CANCEL_INFERIORS
- PREPARE_INFERIORS
- REQUEST_INFERIOR_STATUSES

A Terminator receives
- TRANSACTION_CONFIRMED
- TRANSACTION_CANCELLED
- INFERIOR_STATUSES

Initiator

Requests a Factory to create a Superior – this will either be a Decider (representing a new top-level business transaction) or a sub-coordinator or sub-composer to be the Inferior of an existing business transaction.

<table>
<thead>
<tr>
<th>Initiator sends</th>
<th>Initiator receives</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGIN</td>
<td>BEGUN &amp; CONTEXT</td>
</tr>
<tr>
<td>BEGIN &amp; CONTEXT</td>
<td>BEGUN &amp; CONTEXT</td>
</tr>
</tbody>
</table>

The received CONTEXT is that for the new Superior. An Initiator sends
- BEGIN
- BEGIN & CONTEXT
to a Factory, and receives in reply
- BEGUN & CONTEXT
Factory

Creates Superiors and returns the CONTEXT for the new Superior. The following types of Superior are created:

- Decider, which is either
  - Composer or
  - Coordinator
- Sub-composer
- Sub-coordinator

<table>
<thead>
<tr>
<th>Factory receives</th>
<th>Factory sends</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGIN</td>
<td>BEGUN &amp; CONTEXT</td>
</tr>
<tr>
<td>BEGIN &amp; CONTEXT</td>
<td>BEGUN &amp; CONTEXT</td>
</tr>
</tbody>
</table>

A Factory receives

- BEGIN
- BEGIN & CONTEXT

and replies with

- BEGUN & CONTEXT

If the BEGIN has no related CONTEXT, the Factory creates a Decider, either a Cohesion Composer or an Atom Coordinator, as determined by the “superior type” parameter on the BEGIN.

If the BEGIN has a related CONTEXT, the new Superior is also enrolled as an Inferior of the Superior identified by the CONTEXT. The new Superior is thus a sub-composer or sub-coordinator, as determined by the “superior type” parameter on the BEGIN.

Other roles

Redirector

Sends a REDIRECT message to inform a Superior or Inferior that an address previously supplied for the peer (i.e. an Inferior or Superior, respectively) is no longer appropriate, and to supply a new address or set of addresses to replace the old one.

A Redirector may send a REDIRECT message in response to receiving a message using the old address, or may send REDIRECT at its own initiative.

If a Superior moves from the superior-address in its CONTEXT, or an Inferior moves from the inferior-address in the ENROL message, the implementation must ensure that a
Redirector catches any inbound messages using the old address and replies with a REDIRECT message giving the new address. (Note that the inbound message may itself be a REDIRECT message, in which case the Redirector shall use the new address in the received message as the target for the REDIRECT that it sends.)

After receiving a REDIRECT message, the BTP actor must use the new address not the old one, unless failure prevents it updating its information.

<table>
<thead>
<tr>
<th>Redirector receives</th>
<th>Redirector sends</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any message for Superior or Inferior</td>
<td>REDIRECT</td>
</tr>
</tbody>
</table>

**Status Requestor**

Requests and receives the current status of a transaction tree node – any of an Inferior, Superior or Decider, or the current status of the nodes relationships with its Inferiors, if any. The role of Status Requestor has no responsibilities – it is just a name for where the REQUEST_STATUS and REQUEST_INFERIOR_STATUSES comes from (REQUEST_INFERIOR_STATUSES is also issued by a Terminator to a Decider).

<table>
<thead>
<tr>
<th>Status Requestor sends</th>
<th>Status Requestor receives</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQUEST_STATUS</td>
<td>STATUS</td>
</tr>
<tr>
<td>REQUEST_INFERIOR_STATUS</td>
<td>INFERIOR_STATUSES</td>
</tr>
</tbody>
</table>

A Status Requestor sends

REQUEST_STATUS
REQUEST_INFERIOR_STATUSES

and receives

STATUS
INFERIOR_STATUSES

in response.

The receiver of the request can refuse to provide the status information by replying with FAULT(StatusRefused). The information returned in STATUS will always relate to the transaction tree node as a whole (e.g. as an Inferior, even if it is also a Superior).

**Summary of relationships**

[Figure 17][Figure 3] [Figure 3][Figure 3][Figure 3] summarises the relationships between the BTP roles. BTP can be implemented using proprietary equivalents of the Terminator and Decider roles.
Figure 173  Summary of relationships between roles
Abstract Messages and Associated Contracts

BT Protocol Messages are defined in this section in terms of the abstract information that has to be communicated. These abstract messages will be mapped to concrete messages communicated by a particular carrier protocol (there can be several such mappings defined).

The abstract message set and the associated state table assume the carrier protocol will deliver messages completely and correctly, or not at all (corrupted messages will not be delivered);

- report some communication failures, but will not necessarily report all (i.e. not all message deliveries are positively acknowledged within the carrier);

- sometimes deliver successive messages in a different order than they were sent;

and

- does not have built-in mechanisms to link a request and a response

Note that these assumptions would be met by a mapping to SMTP and more than met by mappings to SOAP/HTTP.

However, when the abstract message set is mapped to a carrier protocol that provides a richer service (e.g. reports all delivery failures, guarantees ordered delivery or offers a request/response mechanism), the mapping can take advantage of these features. Typically in such cases, some of the parameters of an abstract message will be implicit in the carrier mechanisms, while the values of other parameters will be directly represented in transmitted elements.

The abstract messages include <b>Delivery parameters</b> that are concerned with the transmission and delivery of the messages as well as <b>Payload parameters</b> directly concerned with the progression of the BTP relationships. When bound to a particular carrier protocol and for particular implementation configurations, parts or all of the Delivery parameters may be implicit in the carrier protocol and will not appear in the "on-the-wire" representation of the BTP messages as such. Delivery parameters are defined as being only those parameters that are concerned with the transmission of this message, or of an immediate reply (thus address parameters to be used in repeated later messages and the identifiers of both sender and receiver are Payload parameters). In the tables in this section, Delivery parameters are shown in shaded cells.

Addresses

All of the messages except CONTEXT have a “target address” parameter and many also have other address parameters. These latter identify the desired target of other messages in the set.
In all cases, the exact value will \textit{invariably} have been originally determined by the implementation that is the target or \textit{desired future} \textit{intended} target.

The detailed format of the address will depend on the particular carrier protocol, but at this abstract level is considered to have three parts. The first part, the “binding name”, identifies the binding to a particular carrier protocol – some bindings are specified in this document, others can be specified elsewhere. The second part of the address, the “binding address”, is meaningful to the carrier protocol itself, which will use it for the communication (i.e. it will permit a message to be delivered to a receiver). The third part, “additional information”, is not used or understood by the carrier protocol. The “additional information” may be a structured value.

When a message is actually transmitted, the “binding name” of the target address will identify which carrier protocol is in use and the “binding address” will identify the destination, as known to the carrier protocol. The entire binding address is considered to be “consumed” by the carrier protocol implementation. All of it may be used by the sending implementation, or some of it may be transmitted in headers, or as part of a URL in the carrier protocol, but then used or consumed by the receiving implementation of the carrier protocol to direct the BTP message to a BTP-aware entity (BTP-aware in that it is capable of interpreting the BTP messages). The “additional information” of the target address will be part of the BTP message itself and used in some way by the receiving BTP-aware entity (it could be used to route the message on to some other BTP entity). Thus, for the target address, only the “additional information” field is transmitted in the BTP message and the “additional information” is opaque to parties other than the recipient.

For other addresses in BTP messages, all three components will be within the message.

All messages that concern a particular Superior:Inferior relationship have an identifier parameter for the target side as well as the target address. This allows full flexibility for implementation choices – an implementation can:

\begin{itemize}
\item[a)] Use the same binding address and additional information for multiple business transactions, using the identifier parameter to locate the relevant state information;
\item[b)] Use the same binding address for multiple business transactions and use the additional information to locate the information; or
\item[c)] Use a different binding address for each business transaction.
\end{itemize}

Which of these choices is used is opaque to the entity sending the message – both parts of the address and the identifier originated at the recipient of this message (and were transmitted as parameters of earlier messages in the opposite direction).

BTP recovery requires that the state information for a Superior or Inferior is accessible after failure and that the peer can distinguish between temporary inaccessibility and the permanent non-existence of the state information. As is explained in \textit{below}, BTP provides mechanisms – having a set of BTP addresses for some parameters, and the REDIRECT
message – that make this possible, even if the recovered state information is on a different address to the original one (as may be the case if case c) above is used).

Request/response pairs

Many of the messages combine in pairs as a request and its response. However, in some cases the response message is sent without a triggering request, or as a possible response to more than one type of request. To allow for this, the abstract message set treats each message as standalone; but where a request does expect a reply, a “reply-address” parameter will be present. For any message with a reply address parameter, in the case of certain errors, a FAULT message will be sent to the reply address instead of the expected reply.

Between Superior and Inferior the address of the peer is normally known (from the “superior-address” on an earlier CONTEXT or the “inferior-address” on a received ENROL). However, in some cases a message will be received for a Superior or Inferior that is not known – the state information no longer exists. This is not an exceptional condition but occurs when one side has either not created or has removed its persistent state in accordance with the procedures, but a message has got lost in a failure, and the peer still has state information. The response to a message for an unknown (and logically non-existent) Superior is SUPERIOR_STATE/unknown, for an unknown Inferior it is INFERIOR_STATE/unknown. However, since the intended target is unknown, there is no information to locate the peer, which sent the undeliverable message. To enable the receiver to reply with the appropriate STATE/unknown, all the messages between Superior and Inferior have a “senders-address” parameter. If a FAULT message is to be sent in response to message which (as an abstract message) has a “senders-address” parameter, the FAULT message is sent to that address.

Note – Both reply-address and senders-address may be absent when the carrier protocol itself has a request/response pattern. In these cases, the reply or sender address is implicitly that of the sender of the request (and thus the destination of a response)

Compounding messages

BTP messages may be sent in combination with each other, or with other (application) messages. There are two cases:

a) Sending the messages together where the combination has semantic significance. One message is said to be “related to” the other – the combination is termed a “group”.

b) Sending of the messages where the combination has no semantic significance, but is merely a convenience or optimisation. This is termed “bundling” – the combination is termed a “bundle”. 
The form A&B is used to refer to a combination (group) where message B is sent in relation to A (“relation” is asymmetric). The form A+B is used to refer to A and B bundled together — the transmission of the bundle "A+B" is semantically identical to the transmission of A followed by the transmission of B.

Only certain combinations of messages are possible in a group, and the meaning of the relation is specifically defined for each such combination in the next section. A particular group is treated as a unit for transmission — it has a single target address. This is usually that of one of the messages in the group — the specification for the group defines which.

A “bundle” of messages may contain both unrelated messages and groups of related messages. The only constraint on which messages and groups can be bundled is that all have the same binding address, but may have different “additional information” values. (Messages within a related group may have different addresses, where the rules of their relatedness permit this). Unless constrained by the binding, any messages or groups that are to be sent to the same binding address may be bundled — the fact that the binding addresses are the same is a necessary and sufficient condition for the sender to determine that the messages can be bundled.

A particular and important case of related messages is where a BTP CONTEXT message is sent related to an application message. In this case, the target of the application message defines the destination of the CONTEXT message. The receiving implementation may in fact remove the CONTEXT before delivering the application message to the application (Service) proper, but from the perspective of the sender, the two are sent to the same place. The compounding mechanisms, and the multi-part address structures, support the “one-wire” and “one-shot” communication patterns.

In “one-wire”, all message exchanges between two sides of a Superior:Inferior relationship, including the associated application messages, pass via the same “endpoints”. These “endpoints” may in fact be relays, routing messages on to particular actors within their domain. The onward routing will require some further addressing, but this has to be opaque to the sender. This can be achieved if the relaying endpoint ensures that all addresses for actors in its domain have the relay’s address as their binding address, and any routing information it will need in its own domain is placed in the additional information. (This may involve the relay changing addresses in messages as they pass through it on the way out). On receiving a message, it determines the within-domain destination from the received additional information (which is thus rewritten) and forwards the message appropriately. The sender is unaware of this, and merely sees addresses with the same binding address, which it is permitted to bundle. The content of the “additional information” is a matter only for the relay — it could put an entire BTP address in there, or other implementation-defined information. Note that a quite different one-wire implementation can be constructed where there is no relaying, but the receiving entity effectively performs all roles, using the received identifiers to locate the appropriate state.

“One-shot” communication makes it possible to send an application message, receive the application reply, enrol an Inferior to be responsible for the confirm/cancel of the operations of those message and inform the Superior that the Inferior is prepared, all in one two-way
exchange across the network (e.g. one request/reply of a carrier protocol). The application request is sent with a related CONTEXT message. The application response is sent with a relation group of CONTEXT_REPLY/related, ENROL/no-rsp-req message and a PREPARED message. This is possible even if the Superior address is different from the address of the application element that sends the original message (if the application exchange is request/reply, there may not even be an identifiable address for the application element). The target addresses of the ENROL and PREPARED (the Superior address) are not transmitted; the actor that was originally responsible for adding the CONTEXT to the outbound application message remembers the Superior address and forwards the ENROL and PREPARED appropriately.

With “one-shot”, if there are multiple Inferiors created as a result of a single application message, there is an ENROL and PREPARED message for each sent related to the CONTEXT_REPLY. If an operation fails, a CANCELLED message is sent instead of a PREPARED.

If the CONTEXT has “superior-type” of “atom”, then subsequent messages to the same Service, with the same related CONTEXT/atom, can have their associated operations put under the control of the same Inferior, and only a CONTEXT_REPLY/completed is sent back with the response (if the new operations fail, it will be necessary to send back CONTEXT_REPLY/repudiated, or send CANCELLED). If the “superior type” on the CONTEXT is “cohesive”, each operation will require separate enrolment.

Whether the “one-shot” mechanism is used is determined by the implementation on the responding (Inferior) side. This may be subject to configuration and may also be constrained by the application or by the binding in use.

### Extensibility

To simplify interoperation between implementations of this edition of BTP with implementations of future editions, the “must-be-understood” sub-parameter as specified for Qualifiers may be defined for use with any parameter added to an existing message in a future revision of this specification. The default for “must-be-understood” shall be “true”, so an implementation receiving an unrecognised parameter without a “false” value for “must-be-understood” shall not accept it (the FAULT value “UnrecognisedParameter” is available, but other errors, including lower-layer parsing/ unmarshalling errors may be reported instead). If “must-be-understood” with the value “false” is present as a sub-parameter of a parameter in any message, a receiving implementation should ignore the parameter.

How the sub-parameter is associated with the new parameter is determined by the particular binding.

No special mechanism is provided to allow for the introduction of completely new messages.

### Messages
Qualifiers

All messages have a Qualifiers parameter which contains zero or more Qualifier values. A Qualifier has sub-parameters:

- **Sub-parameter**
  - qualifier name: string
  - qualifier group: URI
  - must-be-understood: Boolean
  - to-be-propagated: Boolean
  - content: Arbitrary – depends on type

**Qualifier group** ensures the Qualifier name is unambiguous. Qualifiers in the same group need not have any functional relationship. The qualifier group will typically be used to identify the specification that defines the qualifier’s meaning and use. Qualifiers may be defined in this or other standard specifications, in specifications of a particular community of users or of implementations or by bilateral agreement.

**Qualifier name** this identifies the meaning and use of the Qualifier, using a name that is unambiguous within the scope of the Qualifier group.

**Must-be-understood** if this has the value “true” and the receiving entity does not recognise the Qualifier type (or does not implement the necessary functionality), a FAULT “UnsupportedQualifier” shall be returned and the message shall not be processed. Default is “true”.

**To-be-propagated** if this has the value “true” and the receiving entity passes the BTP message (which may be a CONTEXT, but can be other messages) onwards to other entities, the same Qualifier value shall be included. If the value is “false”, the Qualifier shall not be automatically included if the BTP message is passed onwards. (If the receiving entity does support the qualifier type, it is possible a propagated message may contain another instance of the same type, even with the same Content – this is not considered propagation of the original qualifier.). Default is “false”.

**Content** the type (which may be structured) and meaning of the content is defined by the specification of the Qualifier.

Messages not restricted to outcome or control relationships.
The messages in this section are used between various roles. CONTEXT message is used in
the Initiator:Factory relationship (when it is related to BEGIN or to BEGUN), and related to
an application ‘message’ to propagate the business transaction between parts of the
application. CONTEXT_REPLY is used as the reply to a CONTEXT.REQUEST_STATUS
can be issued to, and STATUS returned by any of Decider, Superior or Inferior. FAULT can
be used on any relationship to indicate an error condition back to the sender of a message.

CONTEXT

A CONTEXT is supplied by (or on behalf of) a Superior and related to one or more
application messages. (The means by which this relationship is represented is determined by
the binding and the binding mechanisms of the application protocol.) The “superior-type”
parameter identifies whether the Superior will apply the same decision to all Inferiors
enrolled using the same superior identifier (“superior-type” is “atom”) or whether it may
apply different decisions (“superior-type” is “cohesion”).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>superior-address</td>
<td>Set of BTP addresses</td>
</tr>
<tr>
<td>superior-identifier</td>
<td>Identifier</td>
</tr>
<tr>
<td>superior-type</td>
<td>cohesion/atom</td>
</tr>
<tr>
<td>qualifiers</td>
<td>List of qualifiers</td>
</tr>
<tr>
<td>reply-address</td>
<td>BTP address</td>
</tr>
</tbody>
</table>

superior-address the address to which ENROL and other messages from an
enrolled Inferior are to be sent. This can be a set of alternative addresses.

superior-identifier identifies the Superior. This shall be globally unambiguous.

superior-type identifies whether the CONTEXT refers to a Cohesion or an
Atom. Default is atom.

qualifiers standardised or other qualifiers. The standard qualifier “Transaction
timelimit” is carried by CONTEXT.

reply-address the address to which a replying CONTEXT_REPLY is to be sent.
This may be different each time the CONTEXT is transmitted – it refers to the
destination of a replying CONTEXT_REPLY for this particular transmission of
the CONTEXT.

There is no “target-address” parameter for CONTEXT as it is only transmitted in relation to
the application messages, BEGIN and BEGUN.
The forms CONTEXT/cohesion and CONTEXT/atom refer to CONTEXT messages with the “superior-type” with the appropriate value.

**CONTEXT_REPLY**

CONTEXT_REPLY is sent after receipt of CONTEXT (related to application message(s)) to indicate whether all necessary enrolments have already completed (ENROLLED has been received) or will be completed by ENROL messages sent in relation to the CONTEXT_REPLY or if an enrolment attempt has failed. CONTEXT_REPLY may be sent related to an application message (typically the response to the application message related to the CONTEXT). In some bindings the CONTEXT_REPLY may be implicit in the application message. **CONTEXT_REPLY is used in some of the related groups to allow BTP messages to be sent to a Superior with an application message.**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>superior-identifier</td>
<td>Identifier</td>
</tr>
<tr>
<td>completion-status</td>
<td>complete/related/repudiated</td>
</tr>
<tr>
<td>qualifiers</td>
<td>List of qualifiers</td>
</tr>
<tr>
<td>target-address</td>
<td>BTP address</td>
</tr>
</tbody>
</table>

**superior-identifier** the “superior-identifier” from the CONTEXT

**completion-status:** reports whether all enrol operations made necessary by the receipt of the earlier CONTEXT message have completed. Values are

<table>
<thead>
<tr>
<th>Value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>completed</td>
<td>All enrolments (if any) have succeeded already</td>
</tr>
<tr>
<td>incomplete</td>
<td>Further enrolments are possible (used only in related groups with other BTP messages)</td>
</tr>
<tr>
<td>related</td>
<td>At least some enrolments are to be performed by ENROL messages related to the CONTEXT_REPLY. All other enrolments (if any) have succeeded already.</td>
</tr>
<tr>
<td>repudiated</td>
<td>At least one enrolment has failed. The implications of receiving the CONTEXT have not been honoured.</td>
</tr>
</tbody>
</table>

**qualifiers** standardised or other qualifiers.

**target-address** the address to which the CONTEXT_REPLY is sent. This shall be the “reply-address” from the CONTEXT.
The form CONTEXT_REPLY/completed, CONTEXT_REPLY/related and CONTEXT_REPLY/repudiated refer to CONTEXT_REPLY messages with status having the appropriate value. The form CONTEXT_REPLY/ok refers to either of CONTEXT_REPLY/completed or CONTEXT_REPLY/related.

If there are no necessary enrolments (e.g. the application messages related to the received CONTEXT did not require the enrolment of any Inferiors), then CONTEXT_REPLY/completed is used.

If a CONTEXT_REPLY/repudiated is received, the receiving implementation must ensure that the business transaction will not be confirmed.

REQUEST_STATUS

Sent to an Inferior, Superior or to a Decider to ask it to reply with STATUS. The receiver may reject the request with a FAULT(StatusRefused).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>target-identifier</td>
<td>Identifier</td>
</tr>
<tr>
<td>qualifiers</td>
<td>List of qualifiers</td>
</tr>
<tr>
<td>target-address</td>
<td>BTP address</td>
</tr>
<tr>
<td>reply-address</td>
<td>BTP address</td>
</tr>
</tbody>
</table>

**target identifier** The identifier for the business transaction, or part of business transaction whose status is sought. If the target-address is a “decider-address”, this parameter shall be the “transaction-identifier” on the BEGUN message. If the “target-address” is an “inferior-address”, this parameter shall be the “inferior-identifier” on the ENROL message. If the “target-address” is a “superior-address”, this parameter shall be the “superior-identifier” on the CONTEXT.

**qualifiers** standardised or other qualifiers.

**target-address** the address to which the REQUEST_STATUS message is sent. This can be any of “decider-address”, “inferior-address” or “superior-address”.

**reply-address** the address to which the replying STATUS should be sent.

Types of FAULT possible (sent to “reply-address”)

**General**
Redirect – if the intended target now has a different address
StatusRefused – if the receiver is not prepared to report its status to the
sender of this message
UnknownTransaction – if the target-identifier is unknown

STATUS

Sent by a Inferior, Superior or Decider in reply to a REQUEST_STATUS, reporting the
overall state of the transaction tree node represented by the sender.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>responders-identifier</td>
<td>Identifier</td>
</tr>
<tr>
<td>status</td>
<td>See below</td>
</tr>
<tr>
<td>qualifiers</td>
<td>List of qualifiers</td>
</tr>
<tr>
<td>target-address</td>
<td>BTP address</td>
</tr>
</tbody>
</table>

responders-identifier the identifier of the state, identical to the “target-
identifier” on the REQUEST_STATUS.

status states the current status of the transaction tree node represented by the
sender. Some of the values are only issued if the sender is an Inferior. If the
transaction tree node is both Superior and Inferior (i.e. a sub-coordinator or
sub-composer), and two status values would be valid for the current state, it is the
sender’s option which one is used.

<table>
<thead>
<tr>
<th>status value</th>
<th>Meaning from Superior</th>
<th>Meaning from Inferior</th>
</tr>
</thead>
</table>
| Created      | Not applicable        | The Inferior exists (and is
                        | addressable) but it has not been
                        | enrolled with a Superior |
| Enrolling    | Not applicable        | ENROL has been sent, but
                        | ENROLLED is awaited     |
| Active       | New enrolment of inferiors is possible | The Inferior is enrolled |
| Resigning    | Not applicable        | RESIGN has been sent;
                        | RESIGNED is awaited     |
| Resigned     | Not applicable        | RESIGNED has been received |
| Preparing    | Not applicable        | PREPARE has been received;
<pre><code>                    | PREPARED has not been sent |
</code></pre>
<p>| Prepared     | Not applicable        | PREPARED has been sent; no |</p>
<table>
<thead>
<tr>
<th>status value</th>
<th>Meaning from Superior</th>
<th>Meaning from Inferior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Confirming</td>
<td>Confirm decision has been made or CONFIRM has been received as Inferior but responses from inferiors are pending</td>
<td>CONFIRM has been received; CONFIRMED/response has not been sent</td>
</tr>
<tr>
<td>Confirmed</td>
<td>CONFIRMED/responses have been received from all Inferiors</td>
<td>CONFIRMED/response has been sent</td>
</tr>
<tr>
<td>Cancelling</td>
<td>Cancel decision has been made but responses from inferiors are pending</td>
<td>CANCEL has been received or auto-cancel has been decided</td>
</tr>
<tr>
<td>Cancelled</td>
<td>CANCELLED has been received from all Inferiors</td>
<td>CANCELLED has been sent</td>
</tr>
<tr>
<td>cancel-contradiction</td>
<td>Not applicable │ Autonomous cancel decision was made, CONFIRM received; CONTRADICTION has not been received</td>
<td></td>
</tr>
<tr>
<td>confirm-contradiction</td>
<td>Not applicable │ Autonomous confirm decision was made, CANCEL received; CONTRADICTION has not been received</td>
<td></td>
</tr>
<tr>
<td>Hazard</td>
<td>A hazard has been reported from at least one Inferior</td>
<td>A hazard has been discovered; CONTRADICTION has not been received</td>
</tr>
<tr>
<td>Contradicted</td>
<td>Not applicable</td>
<td>CONTRADICTION has been received</td>
</tr>
<tr>
<td>Unknown</td>
<td>No state information for the target-identifier exists</td>
<td>No state information for the target-identifier exists</td>
</tr>
<tr>
<td>Inaccessible</td>
<td>There may be state information for this target-identifier but it cannot be reached/existence cannot be determined</td>
<td>There may be state information for this target-identifier but it cannot be reached/existence cannot be determined</td>
</tr>
</tbody>
</table>

**qualifiers** standardised or other qualifiers.

**target-address** the address to which the STATUS is sent. This will be the “reply-address” on the REQUEST_STATUS message
General

FAULT

Sent in reply to various messages to report an error condition. The FAULT message is used on all the relationships as a general negative reply to a message.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>superior-identifier</td>
<td>Identifier</td>
</tr>
<tr>
<td>inferior-identifier</td>
<td>Identifier</td>
</tr>
<tr>
<td>fault-type</td>
<td>See below</td>
</tr>
<tr>
<td>fault-data</td>
<td>See below</td>
</tr>
<tr>
<td>fault-text</td>
<td>Text string</td>
</tr>
<tr>
<td>qualifiers</td>
<td>List of qualifiers</td>
</tr>
<tr>
<td>target-address</td>
<td>BTP address</td>
</tr>
</tbody>
</table>

superior-identifier the “superior-identifier” as on the CONTEXT message and as used on the ENROL message (present only if the FAULT is sent to the superior).

inferior-identifier the “inferior-identifier” as on the ENROL message (present only if the FAULT is sent to the inferior)

fault-type identifies the nature of the error, as specified for each of the main messages.

fault-data information relevant to the particular error. Each “fault-type” defines the content of the “fault-data”:

<table>
<thead>
<tr>
<th>fault-type</th>
<th>meaning</th>
<th>fault-data</th>
</tr>
</thead>
<tbody>
<tr>
<td>CommunicationFailure</td>
<td>Any fault arising from the carrier mechanism and communication infrastructure.</td>
<td>Determined by the carrier mechanism and binding specification</td>
</tr>
<tr>
<td>DuplicateInferior</td>
<td>An inferior with the same address and identifier is already enrolled with this Superior</td>
<td>The identifier</td>
</tr>
<tr>
<td>General</td>
<td>Any otherwise unspecified problem</td>
<td>None</td>
</tr>
<tr>
<td>InvalidDecider</td>
<td>The address the message was sent to is not valid (at all or for this Terminator and transaction identifier)</td>
<td>The address</td>
</tr>
<tr>
<td>InvalidInferior</td>
<td>The “inferior-identifier” in the message or at least one “inferior-identifier”’s in an “inferior-list” parameter is not known or does not identify a known Inferior</td>
<td>One or more invalid identifiers</td>
</tr>
<tr>
<td>InvalidSuperior</td>
<td>The received identifier is not known or does not identify a known Superior</td>
<td>The identifier</td>
</tr>
<tr>
<td>StatusRefused</td>
<td>The receiver will not report the requested status (or inferior statuses) to this StatusRequestor</td>
<td>Free text explanation</td>
</tr>
<tr>
<td>InvalidTerminator</td>
<td>The address the message was sent to is not valid (at all or for this Decider and transaction identifier)</td>
<td>The address</td>
</tr>
<tr>
<td>UnknownParameter</td>
<td>A BTP message has been received with an unrecognised parameter</td>
<td>None</td>
</tr>
<tr>
<td>UnknownTransaction</td>
<td>The transaction-identifier is unknown</td>
<td>The transaction-identifier</td>
</tr>
<tr>
<td>UnsupportedQualifier</td>
<td>A qualifier has been received that is not recognised and on which “must-be-Understood” is “true”.</td>
<td>Qualifier group and name</td>
</tr>
<tr>
<td>WrongState</td>
<td>The message has arrived when the recipient or the transaction identified by a related CONTEXT is in an invalid state.</td>
<td>None</td>
</tr>
</tbody>
</table>

WrongState

The message has arrived when the recipient or the transaction identified by a related CONTEXT is in an invalid state. Set of BTP addresses, to be used instead of the address the BTP message was received on
qufault-text Free text describing the fault or providing more information. Whether this parameter is present, and exactly what it contains are an implementation option.

qualifiers standardised or other qualifiers.

target-address the address to which the FAULT is sent. This may be the “reply-address” from a received message or the address of the opposite side (superior/inferior) as given in a CONTEXT or ENROL message.

Note – If the carrier mechanism used for the transmission of BTP messages is capable of delivering messages in a different order than they were sent in, the “WrongState” FAULT is not sent and should be ignored if received.

REQUEST_INFERIOR_STATUSES, INFERIOR_STATUSES

REQUEST_INFERIOR_STATUSES may be sent to and INFERIOR_STATUSES sent from any Decider, Superior or Inferior, asking it to report on the status of its relationships with Inferiors (if any). Since Deciders are required to respond to REQUEST_INFERIOR_STATUSES with INFERIOR_STATUSES but non-Deciders may just issue FAULT(StatusRefused), and INFERIOR_STATUSES is also used as a reply to other messages from Terminator to Decider, these messages are described below under the messages used in the control relationships.

Messages used in the outcome relationships

ENROL

A request to a Superior to ENROL an Inferior. This is typically issued after receipt of a CONTEXT message in relation to an application request.

The actor issuing ENROL plays the role of Enroller.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>superior-identifier</td>
<td>Identifier</td>
</tr>
<tr>
<td>response-requested</td>
<td>Boolean</td>
</tr>
<tr>
<td>inferior-address</td>
<td>Set of BTP addresses</td>
</tr>
<tr>
<td>inferior-identifier</td>
<td>Identifier</td>
</tr>
</tbody>
</table>
qualifiers | List of qualifiers
---|---
target-address | BTP address
reply-address | BTP address

**superior-identifier.** The “superior-identifier” as on the CONTEXT message

**response-requested** true if an ENROLLED response is required, false otherwise. Default is false.

**inferior-address** the address to which PREPARE, CONFIRM, CANCEL and SUPERIOR_STATE messages for this Inferior are to be sent.

**inferior-identifier** an identifier that identifies this Inferior. This shall be globally unambiguous..

**qualifiers** standardised or other qualifiers. The standard qualifier “Inferior name” may be present.

**target-address** the address to which the ENROL is sent. This will be the “superior-address” from the CONTEXT message.

**reply-address** the address to which a replying ENROLLED is to be sent, if “response-requested” is true. If this field is absent and “response-requested” is true, the ENROLLED should be sent to the “inferior-address” (or one of them, at sender’s option)

Types of FAULT possible (sent to “reply-address”)

**General**

**InvalidSuperior** – if “superior-identifier” is unknown

**Redirect** – if the Superior now has a different superior-address

**DuplicateInferior** – if inferior with at least one of the set “inferior-address” the same and the same “inferior-identifier” is already enrolled

**WrongState** – if it is too late to enrol new Inferiors (generally if the Superior has already sent a PREPARED message to its superior or terminator, or if it has already issued CONFIRM to other Inferiors).

The form ENROL/rsp-req refers to an ENROL message with “response-requested” having the value “true”; ENROL/no-rsp-req refers to an ENROL message with “response-requested” having the value “false”

ENROL/no-rsp-req is typically sent in relation to CONTEXT_REPLY/related. ENROL/rsp-req is typically when CONTEXT_REPLY/completed will be used (after the ENROLLED message has been received.)
ENROLLED

Sent from Superior in reply to an ENROL/rsp-req message, to indicate the Inferior has been successfully enrolled (and will therefore be included in the termination exchanges)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>inferior-identifier</td>
<td>Identifier</td>
</tr>
<tr>
<td>qualifiers</td>
<td>List of qualifiers</td>
</tr>
<tr>
<td>target-address</td>
<td>BTP address</td>
</tr>
<tr>
<td>sender-address</td>
<td>BTP address</td>
</tr>
</tbody>
</table>

inferior-identifier The “inferior-identifier” as on the ENROL message

qualifiers standardised or other qualifiers.

target-address the address to which the ENROLLED is sent. This will be the “reply-address” from the ENROL message (or one of the “inferior-address”s if the “reply-address” was empty)

sender-address the address from which the ENROLLED is sent. This is an address of the Superior.

No FAULT messages are issued on receiving ENROLLED.

RESIGN

Sent from an enrolled Inferior to the Superior to remove the Inferior from the enrolment. This can only be sent if the operations of the business transaction have had no effect as perceived by the Inferior.

RESIGN may be sent at any time prior to the sending of a PREPARED or CANCELLED message (which cannot then be sent). RESIGN may be sent in response to a PREPARE message.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>superior-identifier</td>
<td>identifier</td>
</tr>
<tr>
<td>inferior-identifier</td>
<td>identifier</td>
</tr>
<tr>
<td>response-requested</td>
<td>Boolean</td>
</tr>
<tr>
<td>qualifiers</td>
<td>List of qualifiers</td>
</tr>
</tbody>
</table>
superior-identifier The “superior-identifier” as on the ENROL message

inferior-identifier The “inferior-identifier” as on the earlier ENROL message

response-requested is set to “true” if a RESIGNED response is required. Default is “false”.

qualifiers standardised or other qualifiers.

target-address the address to which the RESIGN is sent. This will be the superior address as used on the ENROL message.

sender-address the address from which the RESIGN is sent. This is an address of the Inferior.

Note -- RESIGN is equivalent to readonly vote in some other protocols, but can be issued early.

Types of FAULT possible (sent to “sender-address”)

<table>
<thead>
<tr>
<th>General</th>
</tr>
</thead>
<tbody>
<tr>
<td>InvalidSuperior – if “superior-identifier” is unknown</td>
</tr>
<tr>
<td>InvalidInferior – if no ENROL had been received for this “inferior-identifier”</td>
</tr>
<tr>
<td>WrongState – if a PREPARED or CANCELLED has already been received by the Superior from this Inferior</td>
</tr>
</tbody>
</table>

The form RESIGN/rsp-req refers to an RESIGN message with “response-requested” having the value “true”; RESIGN/no-rsp-req refers to an RESIGN message with “response-requested” having the value “false”

RESIGNED

Sent in reply to a RESIGN/rsp-req message.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>inferior-identifier</td>
<td>Identifier</td>
</tr>
<tr>
<td>qualifiers</td>
<td>List of qualifiers</td>
</tr>
<tr>
<td>target-address</td>
<td>BTP address</td>
</tr>
</tbody>
</table>
sender-address  BTP address

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>inferior-identifier</td>
<td>Identifier</td>
</tr>
<tr>
<td>qualifiers</td>
<td>List of qualifiers</td>
</tr>
<tr>
<td>target-address</td>
<td>BTP address</td>
</tr>
<tr>
<td>sender-address</td>
<td>BTP address</td>
</tr>
</tbody>
</table>

inferior-identifier  The “inferior-identifier” as on the earlier ENROL message for this Inferior.

qualifiers  standardised or other qualifiers.

target-address  the address to which the RESIGNED is sent. This will be the “inferior-address” from the ENROL message.

sender-address  the address from which the RESIGNED is sent. This is an address of the Superior.

After receiving this message the Inferior will not receive any more messages with this “inferior-identifier”.

Types of FAULT possible (sent to “sender-address”)

General
WrongState - if RESIGN has not been sent

PREPARE

Sent from Superior to an Inferior from whom ENROL but neither CANCELLED nor RESIGN have been received, requesting a PREPARED message. PREPARE can be sent after receiving a PREPARED message.

inferior-identifier  the “inferior-identifier” as on the earlier ENROL message.

qualifiers  standardised or other qualifiers. The standard qualifier “Minimal inferior timeout” is carried by PREPARE.

target-address  the address to which the PREPARE message is sent. This will be the “inferior-address” from the ENROL message.
sender-address the address from which the PREPARE is sent. This is an address of the Superior.

On receiving PREPARE, an Inferior should reply with a PREPARED, CANCELLED or RESIGN.

Types of FAULT possible (sent to “sender-address”)

General
InvalidInferior – if “inferior-identifier” is unknown, or an inferior-handle on the inferiors-list is unknown
WrongState – if a CONFIRM or CANCEL has already been received by this Inferior.

PREPARED

Sent from Inferior to Superior, either unsolicited or in response to PREPARE, but only when the Inferior has determined the operations associated with the Inferior can be confirmed and can be cancelled, as may be instructed by the Superior. The level of isolation is a local matter (i.e. it is the Inferiors choice, as constrained by the shared understanding of the application exchanges) – other access may be blocked, may see applied results of operations or may see the original state.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>superior-identifier</td>
<td>Identifier</td>
</tr>
<tr>
<td>inferior-identifier</td>
<td>Identifier</td>
</tr>
<tr>
<td>default-is cancel</td>
<td>Boolean</td>
</tr>
<tr>
<td>qualifiers</td>
<td>List of qualifiers</td>
</tr>
<tr>
<td>target-address</td>
<td>BTP address</td>
</tr>
<tr>
<td>sender-address</td>
<td>BTP address</td>
</tr>
</tbody>
</table>

superior-identifier the “superior-identifier” as on the ENROL message
inferior-identifier The “inferior-identifier” as on the ENROL message
default-is cancel if “true”, the Inferior states that if the outcome at the Superior is to cancel the operations associated with this Inferior, no further messages need be sent to the Inferior. If the Inferior does not receive a CONFIRM message, it will cancel the associated operations. The value “true” will invariably be used with a qualifier indicating under what circumstances (usually a timeout) an autonomous decision to cancel will be made. If “false”, the Inferior will expect
a CONFIRM or CANCEL message as appropriate, even if qualifiers indicate that
an autonomous decision will be made.

**qualifiers** standardised or other qualifiers. The standard qualifier “Inferior
timeout” may be carried by PREPARED.

target-address the address to which the PREPARED is sent. This will be the
Superior address as on the ENROL message.

**sender-address** the address from which the PREPARED is sent. This is an
address of the Inferior.

On sending a PREPARED, the Inferior undertakes to maintain its ability to confirm or cancel
the effects of the associated operations until it receives a CONFIRM or CANCEL message.
Qualifiers may define a time limit or other constraints on this promise. The “default-is
cancel” parameter affects only the subsequent message exchanges and does not of itself state
that cancellation will occur.

Types of FAULT possible (sent to “sender-address”)

**General**

*InvalidSuperior* – if “superior-identifier” is unknown

*InvalidInferior* – if no ENROL has been received for this “inferior-
identifier”, or if RESIGN has been received from this Inferior

The form PREPARED/cancel refers to a PREPARED message with “default-is cancel” =
“true”. The unqualified form PREPARED refers to a PREPARED message with “default-is
cancel” = “false”.

**CONFIRM**

Sent by the Superior to an Inferior from whom PREPARED has been received.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>inferior-identifier</td>
<td>Identifier</td>
</tr>
<tr>
<td>qualifiers</td>
<td>List of qualifiers</td>
</tr>
<tr>
<td>target-address</td>
<td>BTP address</td>
</tr>
<tr>
<td>sender-address</td>
<td>BTP address</td>
</tr>
</tbody>
</table>

**in inferior-identifier** The “inferior-identifier” as on the earlier ENROL message for
this Inferior.
qualifiers standardised or other qualifiers.

target-address the address to which the CONFIRM message is sent. This will be the “inferior-address” from the ENROL message.

sender-address the address from which the CONFIRM is sent. This is an address of the Superior.

On receiving CONFIRM, the Inferior is released from its promise to be able to undo the operations of associated with the Inferior. The effects of the operations can be made available to everyone (if they weren’t already).

Types of FAULT possible (sent to “sender-address”)

General
InvalidInferior – if “inferior-identifier” is unknown
WrongState – if no PREPARED has been sent by, or if CANCEL has been received by this Inferior.

CONFIRMED
Sent after the Inferior has applied the confirmation, both in reply to CONFIRM or when the Inferior has made an autonomous confirm decision, and in reply to a CONFIRM_ONE_PHASE if the Inferior decides to confirm its associated operations.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>superior-identifier</td>
<td>Identifier</td>
</tr>
<tr>
<td>inferior-identifier</td>
<td>Identifier</td>
</tr>
<tr>
<td>confirm-received</td>
<td>Boolean</td>
</tr>
<tr>
<td>qualifiers</td>
<td>List of qualifiers</td>
</tr>
<tr>
<td>target-address</td>
<td>BTP address</td>
</tr>
<tr>
<td>sender-address</td>
<td>BTP address</td>
</tr>
</tbody>
</table>

superior-identifier the “superior-identifier” as on the CONTEXT message.

inferior-identifier the “inferior-identifier” as on the earlier ENROL message.

confirm-received “true” if CONFIRMED is sent after receiving a CONFIRM message; “false” if an autonomous confirm decision has been made and either if
no CONFIRM message has been received or the implementation cannot
determine if CONFIRM has been received (due to loss of state information in a
failure).

**qualifiers** standardised or other qualifiers.

**target-address** the address to which the CONFIRMED is sent. This will be the
Superior address as on the CONTEXT message.

**sender-address** the address from which the CONFIRMED is sent. This is an
address of the Inferior.

Types of FAULT possible (sent to “sender-address”)

- **General**
  - **InvalidSuperior** – if “superior-identifier” is unknown
  - **InvalidInferior** – if no ENROL has been received for this “inferior-
    identifier”, or if RESIGN has been received from this Inferior.

Note – A CONFIRMED message arriving before a CONFIRM message is
sent, or after a CANCEL has been sent will occur when the Inferior has
taken an autonomous decision and is not regarded as occurring in the wrong
state. (The latter will cause a CONTRADICTION message to be sent.)

The form CONFIRMED/auto refers to a CONFIRMED message with “confirm-
received” = “false”; CONFIRMED/response refers to a CONFIRMED message
with “confirm-received” = ”true”.

**CANCEL**

Sent by the Superior to an Inferior at any time before (and unless) CONFIRM has been sent.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>inferior-identifier</td>
<td>Identifier</td>
</tr>
<tr>
<td>qualifiers</td>
<td>List of qualifiers</td>
</tr>
<tr>
<td>target-address</td>
<td>BTP address</td>
</tr>
<tr>
<td>sender-address</td>
<td>BTP address</td>
</tr>
</tbody>
</table>

**inferior-identifier** the “inferior-identifier” as on the earlier ENROL message.
qualifiers  standardised or other qualifiers.

target-address  the address to which the CANCEL message is sent. This will be the “inferior-address” from the ENROL message.

sender-address  the address from which the CANCEL is sent. This is an address of the Superior.

When received by an Inferior, the effects of any operations associated with the Inferior should be undone. If the Inferior had sent PREPARED, the Inferior is released from its promise to be able to confirm the operations.

Types of FAULT possible (sent to “sender-address”)

**General**

*InvalidInferior* – if “inferior-identifier” is unknown, or an inferior-handle on the inferiors-list is unknown

*WrongState* – if a CONFIRM has been received by this Inferior.

**CANCELLED**

Sent when the Inferior has applied (or is applying) cancellation of the operations associated with the Inferior. CANCELLED is sent from Inferior to Superior in the following cases:

1. before (and instead of) sending PREPARED, to indicate the Inferior is unable to apply the operations in full and is cancelling all of them;
2. in reply to CANCEL, regardless of whether PREPARED has been sent;
3. after sending PREPARED and then making and applying an autonomous decision to cancel.
4. in reply to CONFIRM_ONE_PHASE if the Inferior decides to cancel the associated operations

As is specified in the state tables, cases 1, 2 and 3 are not always distinct in some circumstances of recovery and resending of messages.

**Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>superior-identifier</td>
<td>Identifier</td>
</tr>
<tr>
<td>inferior-identifier</td>
<td>Identifier</td>
</tr>
<tr>
<td>qualifiers</td>
<td>List of qualifiers</td>
</tr>
<tr>
<td>target-address</td>
<td>BTP address</td>
</tr>
</tbody>
</table>
sender-address  BTP address

superior-identifier  the “superior-identifier” as on the CONTEXT message.

in inferior-identifier  the inferior identifier as on the earlier ENROL message.

qualifiers  standardised or other qualifiers.

target-address  the address to which the CANCELLED is sent. This will be the Superior address as on the CONTEXT message.

sender-address  the address from which the CANCELLED is sent. This is an address of the Inferior.

Types of FAULT possible (sent to “sender-address”)

General
InvalidSuperior  – if “superior-identifier” is unknown
InvalidInferior  – if no ENROL has been received for this “inferior-identifier”, or if RESIGN has been received from this Inferior
WrongState  – if CONFIRM has been sent

Note – A CANCELLED message arriving before a CANCEL message is sent, or after a CONFIRM has been sent will occur when the Inferior has taken an autonomous decision and is not regarded as occurring in the wrong state. (The latter will cause a CONTRADICTION message to be sent.)

CONFIRM_ONE_PHASE

Sent from a Superior to an enrolled Inferior, when there is only one such enrolled Inferior. In this case the two-phase exchange is not performed between the Superior and Inferior and the outcome decision for the operations associated with the Inferior is determined by the Inferior.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>inferior-identifier</td>
<td>Identifier</td>
</tr>
<tr>
<td>report-hazard</td>
<td>boolean</td>
</tr>
<tr>
<td>qualifiers</td>
<td>List of qualifiers</td>
</tr>
<tr>
<td>target-address</td>
<td>BTP address</td>
</tr>
<tr>
<td>sender-address</td>
<td>BTP address</td>
</tr>
</tbody>
</table>
inferior-identifier The “inferior-identifier” as on the earlier ENROL message for this Inferior.

report hazard Defines whether the superior wishes to be informed if a mixed condition occurs for the operations associated with the Inferior. If “report-hazard” is “true”, the Inferior will reply with HAZARD if a mixed condition occurs, or if the Inferior cannot determine that a mixed condition has not occurred. If “report-hazard” is false, the Inferior will report only its own decision, regardless of whether that decision was correctly and consistently applied. Default is false.

qualifiers standardised or other qualifiers.

target-address the address to which the CONFIRM_ONE_PHASE message is sent. This will be the “inferior-address” on the ENROL message.

sender-address the address from which the CONFIRM_ONE_PHASE is sent. This is an address of the Superior.

CONFIRM_ONE_PHASE can be issued by a Superior to an Inferior from whom PREPARED has been received (subject to the requirement that there is only one enrolled Inferior).

Types of FAULT possible (sent to “sender-address”)

**General**

- **InvalidInferior** – if “inferior-identifier” is unknown
- **WrongState** – if a PREPARE has already been sent to this Inferior

**HAZARD**

Sent when the Inferior has either discovered a “mixed” condition: that is unable to correctly and consistently cancel or confirm the operations in accord with the decision, or when the Inferior is unable to determine that a “mixed” condition has not occurred.

HAZARD is also used to reply to a CONFIRM_ONE_PHASE if the Inferior determines there is a mixed condition within its associated operations or is unable to determine that there is not a mixed condition.

Note - If the Inferior makes its own autonomous decision then it signals that decision with CONFIRMED or CANCELLED and waits to receive a confirmatory CONFIRM or CANCEL, or a CONTRADICTION if the autonomous decision by the Inferior was the opposite of that made by the Superior.
**Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>superior-identifier</td>
<td>Identifier</td>
</tr>
<tr>
<td>inferior-identifier</td>
<td>Identifier</td>
</tr>
<tr>
<td>level</td>
<td>mixed/possible</td>
</tr>
<tr>
<td>qualifiers</td>
<td>List of qualifiers</td>
</tr>
<tr>
<td>target-address</td>
<td>BTP address</td>
</tr>
<tr>
<td>sender-address</td>
<td>BTP address</td>
</tr>
</tbody>
</table>

**superior-identifier** The “superior-identifier” as on the ENROL message.

**inferior-identifier** The “inferior-identifier” as on the earlier ENROL message.

**level** indicates, with value “mixed” that a mixed condition has definitely occurred; or, with value “possible” that it is unable to determine whether a mixed condition has occurred or not.

**qualifiers** standardised or other qualifiers.

**target-address** the address to which the HAZARD is sent. This will be the superior address from the ENROL message.

**sender-address** the address from which the HAZARD is sent. This is an address of the Inferior.

Types of FAULT possible (sent to “sender-address”)

**General**

*InvalidSuperior* – if “superior-identifier” is unknown

*InvalidInferior* – if no ENROL has been received for this “inferior-identifier”, or if RESIGN has been received from this Inferior

The form HAZARD/mixed refers to a HAZARD message with “level” = “mixed”, the form HAZARD/possible refers to a HAZARD message with “level” = “possible”.

**CONTRADICTION**

Sent by the Superior to an Inferior that has taken an autonomous decision contrary to the decision for the atom. This is detected by the Superior when the ‘wrong’ one of
CONFIRMED or CANCELLED is received. CONTRADICTION is also sent in response to a HAZARD message.

### Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>inferior-identifier</td>
<td>Identifier</td>
</tr>
<tr>
<td>qualifiers</td>
<td>List of qualifiers</td>
</tr>
<tr>
<td>target-address</td>
<td>BTP address</td>
</tr>
<tr>
<td>sender-address</td>
<td>BTP address</td>
</tr>
</tbody>
</table>

**inferior-identifier** The “inferior-identifier” as on the earlier ENROL message for this Inferior.

**qualifiers** standardised or other qualifiers.

**target-address** the address to which the CONTRADICTION message is sent. This will be the “inferior-address” from the ENROL message.

**sender-address** the address from which the CONTRADICTION is sent. This is an address of the Superior.

Types of FAULT possible (sent to “sender-address”)

**General**

**InvalidInferior** – if “inferior-identifier” is unknown

**WrongState** – if neither CONFIRMED or CANCELLED has been sent by this Inferior

### SUPERIOR.STATE

Sent by a Superior as a query to an Inferior when

1. in the active state
2. there is uncertainty what state the Inferior has reached (due to recovery from previous failure or other reason).

Also sent by the Superior to the Inferior in response to a received INFERIOR_STATE, in particular states.

### Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>inferior-identifier</td>
<td>Identifier</td>
</tr>
<tr>
<td>status</td>
<td>see below</td>
</tr>
<tr>
<td>Parameter</td>
<td>Type</td>
</tr>
<tr>
<td>------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>response-requested</td>
<td>Boolean</td>
</tr>
<tr>
<td>qualifiers</td>
<td>List of qualifiers</td>
</tr>
<tr>
<td>target-address</td>
<td>BTP address</td>
</tr>
<tr>
<td>sender-address</td>
<td>BTP address</td>
</tr>
</tbody>
</table>

**inferior-identifier** The “inferior-identifier” as on the earlier ENROL message for this Inferior.

**status** states the current state of the Superior, in terms of its relation to this Inferior only.

<table>
<thead>
<tr>
<th>status value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>active</td>
<td>The relationship with the Inferior is in the active state from the perspective of the Superior; ENROLLED has been sent, PREPARE has not been sent and PREPARED has not been received (as far as the Superior knows)</td>
</tr>
<tr>
<td>prepared-received</td>
<td>PREPARED has been received from the Inferior, but no outcome is yet available</td>
</tr>
<tr>
<td>inaccessible</td>
<td>The state information for the Superior, or for its relationship with this Inferior, if it exists, cannot be accessed at the moment. This should be a transient condition</td>
</tr>
<tr>
<td>unknown</td>
<td>The Inferior is not known – it does not exist from the perspective of the Superior. The Inferior can treat this as an instruction to cancel any associated operations</td>
</tr>
</tbody>
</table>

**response-requested** true, if SUPERIOR_STATE is sent as a query at the Superior’s initiative; false, if SUPERIOR_STATE is sent in reply to a received INFERIOR_STATE or other message. Can only be true if status is active or prepared-received. Default is “false”

**qualifiers** standardised or other qualifiers.

**target-address** the address to which the SUPERIOR_STATE message is sent. This will be the “inferior-address” from the ENROL message.

**sender-address** the address from which the SUPERIOR_STATE is sent. This is an address of the Superior.
The Inferior, on receiving SUPERIOR_STATE with “response-requested = true, should reply
in a timely manner by (depending on its state) repeating the previous message it sent or by
sending INFERIOR_STATE with the appropriate status value.

A status of unknown shall only be sent if it has been determined for certain that the Superior
has no knowledge of the Inferior, or (equivalently) it can be determined that the relationship
with the Inferior was cancelled. If there could be persistent information corresponding to the
Superior, but it is not accessible from the entity receiving an INFERIOR_STATE/*/y (or
other) message targeted to the Superior or that entity cannot determine whether any such
persistent information exists or not, the response shall be Inaccessible.

SUPERIOR_STATE/unknown is also used as a response to messages, other than
INFERIOR_STATE/*/y that are received when the Inferior is not known (and it is known
there is no state information for it).

The form SUPERIOR_STATE/abcd refers to a SUPERIOR_STATE message status having a
value equivalent to “abcd” (for active, prepared-received, unknown and inaccessible) and
with “response-requested” = “false”. SUPERIOR_STATE/abcd/y refers to a similar message,
but with “response-requested” = “true”. The form SUPERIOR_STATE/*/y refers to a
SUPERIOR_STATE message with “response-requested” = “true” and any value for status.

INFERIOR_STATE

Sent by an Inferior as a query when in the active state to a Superior, when (due recovery from
previous failure or other reason) there is uncertainty what state the Superior has reached.

Also sent by the Inferior to the Superior in response to a received SUPERIOR_STATE, in
particular states.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>superior-identifier</td>
<td>Identifier</td>
</tr>
<tr>
<td>inferior-identifier</td>
<td>Identifier</td>
</tr>
<tr>
<td>status</td>
<td>see below</td>
</tr>
<tr>
<td>response-requested</td>
<td>Boolean</td>
</tr>
<tr>
<td>qualifiers</td>
<td>List of qualifiers</td>
</tr>
<tr>
<td>target-address</td>
<td>BTP address</td>
</tr>
<tr>
<td>sender-address</td>
<td>BTP address</td>
</tr>
</tbody>
</table>

superior-identifier The “superior-identifier” as used on the ENROL message
inferior-identifier The “inferior-identifier” as on the ENROL message
status states the current state of the Inferior for the atomic business transaction, which corresponds to the last message sent to the Superior by (or in the case of ENROL for) the Inferior

<table>
<thead>
<tr>
<th>status value</th>
<th>meaning/previous message sent</th>
</tr>
</thead>
<tbody>
<tr>
<td>active</td>
<td>The relationship with the Superior is in the active state from the perspective of the Inferior; ENROL has been sent, a decision to send PREPARED has not been made.</td>
</tr>
<tr>
<td>inaccessible</td>
<td>The state information for the relationship with the Superior, if it exists, cannot be accessed at the moment. This should be a transient condition</td>
</tr>
<tr>
<td>unknown</td>
<td>The Inferior is not known – it does not exist from the perspective of the Superior. The Inferior can be treated as cancelled</td>
</tr>
</tbody>
</table>

response-requested “true” if INFERIOR_STATE is sent as a query at the Superior’s initiative; “false” if INFERIOR_STATE is sent in reply to a received SUPERIOR_STATE or other message. Can only be “true” if “status” is “active” or “prepared-received”. Default is “false”

qualifiers standardised or other qualifiers.

target-address the address to which the INFERIOR_STATE is sent. This will be the “target-address” as used the original ENROL message.

sender-address the address from which the INFERIOR_STATE is sent. This is an address of the Inferior.

The Superior, on receiving INFERIOR_STATE with “response-requested” = “true”, should reply in a timely manner by (depending on its state) repeating the previous message it sent or by sending SUPERIOR_STATE with the appropriate status value.

A status of “unknown” shall only be sent if it has been determined for certain that the Inferior has no knowledge of a relationship with the Superior. If there could be persistent information corresponding to the Superior, but it is not accessible from the entity receiving an SUPERIOR_STATE/*/y (or other) message targeted on the Inferior or the entity cannot determine whether any such persistent information exists, the response shall be “inaccessible”.

INFERIOR_STATE/unknown is also used as a response to messages, other than SUPERIOR_STATE/*/y that are received when the Inferior is not known (and it is known there is no state information for it).

A SUPERIOR_STATE/INFERIOR_STATE exchange that determines that one or both sides are in the active state does not require that the Inferior be cancelled (unlike some other two-
phase commit protocols). The relationship between Superior and Inferior, and related
tool elements may be continued, with new tool messages carrying the same
CONTEXT. Similarly, if the Inferior is prepared but the Superior is active, there is no
required impact on the progression of the relationship between them.

The form INFERIOR_STATE/abcd refers to a INFERIOR_STATE message status having a
value equivalent to “abcd” (for active, unknown and inaccessible) and with “response-
requested” = “false”. INFERIOR_STATE/abcd/y refers to a similar message, but with
“response-requested” = “true”. The form INFERIOR_STATE/*/y refers to a
INFERIOR_STATE message with “response-requested” = “true” and any value for status.

REDIRECT

Sent when the address previously given for a Superior or Inferior is no longer valid and the
relevant state information is now accessible with a different address (but the same superior or
“inferior-identifier”).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>superior-identifier</td>
<td>Identifier</td>
</tr>
<tr>
<td>inferior-identifier</td>
<td>Identifier</td>
</tr>
<tr>
<td>old-address</td>
<td>Set of BTP addresses</td>
</tr>
<tr>
<td>new-address</td>
<td>Set of BTP addresses</td>
</tr>
<tr>
<td>qualifiers</td>
<td>List of qualifiers</td>
</tr>
</tbody>
</table>

**superior-identifier** The “superior-identifier” as on the CONTEXT message and
used on an ENROL message. (present only if the REDIRECT is sent from the
Inferior).

**inferior-identifier** The “inferior-identifier” as on the ENROL message

**old-address** The previous address of the sender of REDIRECT. A match is
considered to apply if any of the “old-address” values match one that is already
known.

**new-address** The (set of alternatives) “new-address” values to be used for
messages sent to this entity.

**qualifiers** standardised or other qualifiers.
**target-address** the address to which the REDIRECT is sent. This is the address of the opposite side (superior/inferior) as given in a CONTEXT or ENROL message.

If the actor whose address is changed is an Inferior, the “new-address” value replaces the “inferior-address” as present in the ENROL.

If the actor whose address is changed is a Superior, the “new-address” value replaces the Superior address as present in the CONTEXT message (or as present in any other mechanism used to establish the Superior:Inferior relationship).

**Messages used in control relationships**

**BEGIN**

A request to a Factory to create a new Business Transaction. This may either be a new top-level transaction, in which case the Composer or Coordinator will be the Decider, or the new Business Transaction may be immediately made the Inferior within an existing Business Transaction (thus creating a sub-Composer or sub-Coordinator).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>transaction-type</td>
<td>cohesion/atom</td>
</tr>
<tr>
<td>qualifiers</td>
<td>List of qualifiers</td>
</tr>
<tr>
<td>target-address</td>
<td>BTP address</td>
</tr>
<tr>
<td>reply-address</td>
<td>BTP address</td>
</tr>
</tbody>
</table>

**transaction-type** identifies whether a new Cohesion or new Atom is to be created; this value will be the “superior-type” in the new CONTEXT.

**qualifiers** standardised or other qualifiers. The standard qualifier “Transaction timelimit” may be present on BEGIN, to set the timelimit for the new business transaction and will be copied to the new CONTEXT. The standard qualifier “Inferior name” may be present if there is a CONTEXT related to the BEGIN.

**target-address** the address of the entity to which the BEGIN is sent. How this address is acquired and the nature of the entity are outside the scope of this specification.

**reply-address** the address to which the replying BEGUN and related CONTEXT message should be sent.
A new top-level Business Transaction is created if there is no CONTEXT related to the BEGIN. A Business Transaction that is to be Inferior in an existing Business Transaction is created if the CONTEXT message for the existing Business Transaction is related to the BEGIN. In this case, the Factory is responsible for enrolling the new Composer or Coordinator as an Inferior of the Superior identified in that CONTEXT.

Note – This specification does not provide a standardised means to determine which of the Inferiors of a sub-Composer are in its confirm set. This is considered part of the application:inferior relationship.

The forms BEGIN/cohesion and BEGIN/atom refer to BEGIN with “transaction-type” having the corresponding value.

Types of FAULT possible (sent to “reply-address”)

- **General**
- **Redirect** – if the Factory now has a different address
- **WrongState** - only issued if there is a related CONTEXT, and the Superior identified by the CONTEXT is in the wrong state to enrol new Inferiors

**BEGUN**

BEGUN is a reply to BEGIN. There is always a related CONTEXT, which is the CONTEXT for the new business transaction.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>decider-address</td>
<td>Set of BTP addresses</td>
</tr>
<tr>
<td>inferior-address</td>
<td>Set of BTP addresses</td>
</tr>
<tr>
<td>transaction-identifier</td>
<td>Identifier</td>
</tr>
<tr>
<td>qualifiers</td>
<td>List of qualifiers</td>
</tr>
<tr>
<td>target-address</td>
<td>BTP address</td>
</tr>
</tbody>
</table>

**decider-address** for a top-most transaction (no CONTEXT related to the BEGIN), this is the address to which PREPARE_INFERIORS, CONFIRM_TRANSACTION, CANCEL_TRANSACTION, CANCEL_INFERIORS and REQUEST_INFERIOR_STATUSES messages are to be sent; if a CONTEXT was related to the BEGIN this parameter is absent

**inferior-address** for a non-top-most transaction (a CONTEXT was related to the BEGIN), this is the “inferior-address” used in the enrolment with the Superior
identified by the CONTEXT related to the BEGIN. The parameter is optional
(implementation’s choice) if this is not a top-most transaction; it shall be absent if
this is a top-most transaction.

**transaction-identifier** if this is a top-most transaction, this is an globally-
unambiguous identifier for the new Decider (Composer or Coordinator). If this is
not a top-most transaction, the transaction-identifier shall be the inferior-
identifier used in the enrolment with the Superior identified by the CONTEXT
related to the BEGIN.

---

Note – The “transaction-identifier” may be identical to the “superior-
identifier” in the CONTEXT that is related to the BEGUN

---

**qualifiers** standardised or other qualifiers.

**target-address** the address to which the BEGUN is sent. This will be the “reply-
address” from the BEGIN.

At implementation option, the “decider-address” and/or “inferior-address” and the “superior-
address” in the related CONTEXT may be the same or may be different. There is no general
requirement that they even use the same bindings. Any may also be the same as the “target-
address” of the BEGIN message (the identifier on messages will ensure they are applied to
the appropriate Composer or Coordinator).

No FAULT messages are issued on receiving BEGUN.

**PREPARE_INFERIORS**

Sent from a Terminator to a Decider, but only if it is a Cohesion Composer, to tell it to
prepare all or some of its inferiors, by sending PREPARE to any that have not already sent
PREPARED, RESIGN or CANCELLED to the Decider (Composer) on its relationships as
Superior. If the inferiors-list parameter is absent, the request applies to all the inferiors; if the
parameter is present, it applies only to the identified inferiors of the Decider (Composer).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>transaction-identifier</td>
<td>Identifier</td>
</tr>
<tr>
<td>inferiors-list</td>
<td>List of Identifiers</td>
</tr>
<tr>
<td>qualifiers</td>
<td>List of qualifiers</td>
</tr>
<tr>
<td>target-address</td>
<td>BTP address</td>
</tr>
<tr>
<td>reply-address</td>
<td>BTP address</td>
</tr>
</tbody>
</table>
transaction identifier identifies the Decider and will be the transaction-identifier from the BEGUN message.

inferiors-list defines which of the Inferiors of this Decider preparation is requested for, using the “inferior-identifiers” as on the ENROL received by the Decider (in its role as Superior). If this parameter is absent, the PREPARE applies to all Inferiors.

qualifiers standardised or other qualifiers.

target-address the address to which the PREPARE_INFERIORS message is sent. This will be the decider-address from the BEGUN message.

reply-address the address of the Terminator sending the PREPARE_INFERIORS message.

For all Inferiors identified in the inferiors-list parameter (all Inferiors if the parameter is absent), from which none of PREPARED, CANCELLED or RESIGNED has been received, the Decider shall issue PREPARE. It will reply to the Terminator, using the “reply-address” on the PREPARE_INFERIORS message, sending an INFERIOR_STATUSES message giving the status of the Inferiors identified on the inferiors-list parameter (all of them if the parameter was absent).

If one or more of the “inferior-identifier”s in the "inferior-list" is unknown (does not correspond to an enrolled Inferior), a FAULT/Invalid-inferior shall be returned. It is an implementation option whether CANCEL is sent to any of the Inferiors that are validly identified in the "inferiors-list".

Types of FAULT possible (sent to Superior address)

General

InvalidDecider – if Decider address is unknown
Redirect – if the Decider now has a different “decider-address”
UnknownTransaction – if the transaction-identifier is unknown
InvalidInferior – if one or more inferior-handles on the inferiors-list is unknown

WrongState – if a CONFIRM_TRANSACTION or CANCEL_TRANSACTION has already been received by this Composer.

The form PREPARE_INFERIORS/all refers to a PREPARE_INFERIORS message where the “inferiors-list” parameter is absent. The form PREPARE_INFERIORS/specific refers to a PREPARE_INFERIORS message where the “inferiors-list” parameter is present.
CONFIRM_TRANSACTION

Sent from a Terminator to a Decider to request confirmation of the business transaction. If the business transaction is a Cohesion, the confirm-set is specified by the “inferiors-list” parameter.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>transaction-identifier</td>
<td>Identifier</td>
</tr>
<tr>
<td>inferiors-list</td>
<td>List of Identifiers</td>
</tr>
<tr>
<td>report-hazard</td>
<td>Boolean</td>
</tr>
<tr>
<td>qualifiers</td>
<td>List of qualifiers</td>
</tr>
<tr>
<td>target-address</td>
<td>BTP address</td>
</tr>
<tr>
<td>reply-address</td>
<td>BTP address</td>
</tr>
</tbody>
</table>

**transaction-identifier** identifies the Decider. This will be the transaction-identifier from the BEGUN message.

**inferiors-list** defines which Inferiors enrolled with the Decider, if it is a Cohesion Composer, are to be confirmed, using the “inferior-identifiers” as on the ENROL received by the Decider (in its role as Superior). Shall be absent if the Decider is an Atom Coordinator.

**report-hazard** defines whether the Terminator wishes to be informed of hazard events and contradictory decisions within the business transaction. If “report-hazard” is “true”, the receiver will wait until responses (CONFIRMED, CANCELLED or HAZARD) have been received from all of its inferiors, ensuring that any hazard events are reported. If “report-hazard” is “false”, the Decider will reply with TRANSACTION_CONFIRMED or TRANSACTION_CANCELLED as soon as the decision for the transaction is known.

**qualifiers** standardised or other qualifiers.

**target-address** the address to which the CONFIRM_TRANSACTION message is sent. This will be the “decider-address” on the BEGUN message.

**reply-address** the address of the Terminator sending the CONFIRM_TRANSACTION message.

If the “inferiors-list” parameter is present, the Inferiors identified shall be the “confirm-set” of the Cohesion. If the parameter is absent and the business transaction is a Cohesion, the “confirm-set” shall be all remaining Inferiors. If the business transaction is an Atom, the “confirm-set” is automatically all the Inferiors.
Any Inferiors from which RESIGN is received are not counted in the confirm-set.

If, for each of the Inferiors in the confirm-set, PREPARE has not been sent and PREPARED has not been received, PREPARE shall be issued to that Inferior.

NOTE -- If PREPARE has been sent but PREPARED not yet received from an Inferior in the confirm-set, it is an implementation option whether and when to re-send PREPARE. The Superior implementation may choose to re-send PREPARE if there are indications that the earlier PREPARE was not delivered.

A confirm decision may be made only if PREPARED has been received from all Inferiors in the “confirm-set”. The making of the decision shall be persistent (and if it is not possible to persist the decision, it is not made). If there is only one remaining Inferior in the “confirm set” and PREPARE has not been sent to it, CONFIRM_ONE PHASE may be sent to it.

All remaining Inferiors that are not in the confirm set shall be cancelled.

If a confirm decision is made and “report-hazard” was “false”, a TRANSACTION_CONFIRMED message shall be sent to the “reply-address”.

If a cancel decision is made and “report-hazard” was “false”, a TRANSACTION_CANCELLED message shall be sent to the “reply-address”.

If "report-hazard" was "true", TRANSACTION_CONFIRMED shall be sent to the "reply-address" after CONFIRMED has been received from each Inferior in the confirm-set and CANCELLED or RESIGN from each and any Inferior not in the confirm-set.

If “report-hazard” was “true” and any HAZARD or contradictory message was received (i.e. CANCELLED from an Inferior in the confirm-set or CONFRIMED from an Inferior not in the confirm-set), an INFERIOR_STATUSES reporting the status for all Inferiors shall be sent to the “reply-address”.

If one or more of the "inferior-identifier"s in the "inferior-list" is unknown (does not correspond to an enrolled Inferior), a FAULT/Invalid-inferior shall be returned. The Decider shall not make a confirm decision and shall not send CONFIRM to any Inferior.

Types of FAULT possible (sent to “reply-address”)

General
InvalidDecider – if Decider address is unknown
Redirect – if the Decider now has a different “decider-address”
UnknownTransaction – if the transaction-identifier is unknown
InvalidInferior – if one or more inferior handles in the inferiors-list is unknown
WrongState – if a CANCEL_TRANSACTION has already been received.

The form CONFIRM_TRANSACTION/all refers to a CONFIRM_TRANSACTION message where the “inferiors-list” parameter is absent. The form CONFIRM_TRANSACTION/specific refers to a CONFIRM_TRANSACTION message where the “inferiors-list” parameter is present.

TRANSACTION_CONFIRMED

A Decider sends TRANSACTION_CONFIRMED to a Terminator in reply to CONFIRM_TRANSACTION if all of the confirm-set confirms (and, for a Cohesion, all other Inferiors cancel) without reporting hazards, or if the Decider made a confirm decision and the CONFIRM_TRANSACTION had a “report-hazards” value of “false”.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>transaction-identifier</td>
<td>identifier</td>
</tr>
<tr>
<td>qualifiers</td>
<td>List of qualifiers</td>
</tr>
<tr>
<td>target-address</td>
<td>BTP address</td>
</tr>
</tbody>
</table>

transaction-identifier the “transaction-identifier” as on the BEGUN message (i.e. the identifier of the Decider as a whole).

qualifiers standardised or other qualifiers.

target-address the address to which the TRANSACTION_CONFIRMED is sent, this will be the “reply-address” from the CONFIRM_TRANSACTION message

Types of FAULT possible (sent to “decider-address”)

General
InvalidTerminator – if Terminator address is unknown
UnknownTransaction – if the transaction-identifier is unknown

CANCEL_TRANSACTION

Sent by a Terminator to a Decider at any time before CONFIRM_TRANSACTION has been sent.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
</table>

OASIS BTP Draft Specification 0.9.2.4, 3 April 2002
transaction-identifier Identifier
report-hazard Boolean
qualifiers List of qualifiers
target-address BTP address
reply-address BTP address

**transaction-identifier** identifies the Decider and will be the transaction-identifier from the BEGUN message.

**report-hazard** Defines whether the Terminator wishes to be informed of hazard events and contradictory decisions within the business transaction. If “report-hazard” is “true”, the receiver will wait until responses (CONFIRMED, CANCELLED or HAZARD) have been received from all of its inferiors, ensuring that any hazard events are reported. If “report-hazard” is “false”, the Decider will reply with TRANSACTION_CANCELLED immediately.

**qualifiers** standardised or other qualifiers.

**target-address** the address to which the CANCEL_TRANSACTION message is sent. This will be the decider-address from the BEGUN message.

**reply-address** the address of the Terminator sending the CANCEL_TRANSACTION message.

The business transaction is cancelled – this is propagated to any remaining Inferiors by issuing CANCEL to them. No more Inferiors will be permitted to enrol.

If "report-hazard" was "false", a TRANSACTION_CANCELLED message shall be sent to the "reply-address".

If "report-hazard" was "true" and any HAZARD or CONFIRMED message was received, an INFERIOR_STATUSES reporting the status for all Inferiors shall be sent to the "reply-address".

If "report-hazard" was "true", TRANSACTION_CANCELLED shall be sent to the "reply-address" after CANCELLED or RESIGN has been received from each Inferior.

Types of FAULT possible (sent to Superior address)

**General**

**InvalidDecider** – if Decider address is unknown

**Redirect** – if the Decider now has a different “decider-address”

**UnknownTransaction** – if the transaction-identifier is unknown
WrongState – if a CONFIRM_TRANSACTION has been received by this Composer.

CANCEL_INFERIORS

Sent by a Terminator to a Decider, but only if is a Cohesion Composer, at any time before CONFIRM_TRANSACTION or CANCEL_TRANSACTION has been sent.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>transaction-identifier</td>
<td>Identifier</td>
</tr>
<tr>
<td>inferiors-list</td>
<td>List of Identifiers</td>
</tr>
<tr>
<td>qualifiers</td>
<td>List of qualifiers</td>
</tr>
<tr>
<td>target-address</td>
<td>BTP address</td>
</tr>
<tr>
<td>reply-address</td>
<td>BTP address</td>
</tr>
</tbody>
</table>

**transaction-identifier** identifies the Decider and will be the transaction-identifier from the BEGUN message.

**inferiors-list** defines which of the Inferiors of this Decider are to be cancelled, using the “inferior-identifiers” as on the ENROL received by the Decider (in its role as Superior).

**qualifiers** standardised or other qualifiers.

**target-address** the address to which the CANCEL_TRANSACTION message is sent. This will be the decider-address from the BEGUN message.

**reply-address** the address of the Terminator sending the CANCEL_TRANSACTION message.

Only the Inferiors identified in the inferiors-list are to be cancelled. Any other inferiors are unaffected by a CANCEL_INFERIORS. Further Inferiors may be enrolled.

---

Note – A CANCEL_INFERIORS for all of the currently enrolled Inferiors will leave the cohesion ‘empty’, but permitted to continue with new Inferiors, if any enrol.

If one or more of the "inferior-identifier"s in the "inferior-list" is unknown (does not correspond to an enrolled Inferior), a FAULT/Invalid-inferior shall be returned. It is an...
implementation option whether CANCEL is sent to any of the Inferiors that are validly identified in the "inferiors-list".

Types of FAULT possible (sent to Superior address)

General
InvalidDecider – if Decider address is unknown
Redirect – if the Decider now has a different “decider-address”
UnknownTransaction – if the transaction-identifier is unknown
InvalidInferior – if one or more inferior-handle on the inferiors-list is unknown
WrongState – if a CONFIRM_TRANSACTION or CANCEL_TRANSACTION has been received by this Composer.

TRANSACTION_CANCELLED

A Decider sends TRANSACTION_CANCELLED to a Terminator in reply to CANCEL_TRANSACTION or in reply to CONFIRM_TRANSACTION if the Decider decided to cancel. In both cases, TRANSACTION_CANCELLED is used only if all Inferiors cancelled without reporting hazards or the CANCEL_TRANSACTION or CONFIRM_TRANSACTION had a "report-hazard" value of “false.

Parameter

<table>
<thead>
<tr>
<th>transaction-identifier</th>
<th>identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>qualifiers</td>
<td>List of qualifiers</td>
</tr>
<tr>
<td>target-address</td>
<td>BTP address</td>
</tr>
</tbody>
</table>

transaction-identifier the “transaction-identifier” as on the BEGUN message (i.e. the identifier of the Decider as a whole).

qualifiers standardised or other qualifiers.

target-address the address to which the TRANSACTION_CANCELLED is sent. This will be the “reply-address” from the CANCEL_TRANSACTION or CONFIRM_TRANSACTION message.

Types of FAULT possible (sent to “decider-address”)

General
InvalidTerminator – if Terminator address is unknown
UnknownTransaction – if the transaction-identifier is unknown

OASIS BTP Draft Specification 0.9.2.4, 3 April 2002
REQUEST_INFERIOR_STATUSES

Sent to a Decider to ask it to report the status of its Inferiors with an INFERIOR_STATUSES message. It can also be sent to any actor with a “superior-address” or “inferior-address”, asking it about the status of that transaction tree nodes Inferiors, if there are any. In this latter case, the receiver may reject the request with a FAULT(StatusRefused). If it is prepared to reply, but has no Inferiors, it replies with an INFERIOR_STATUSES with an empty “status-list” parameter.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>target-identifier</td>
<td>Identifier</td>
</tr>
<tr>
<td>inferiors-list</td>
<td>List of Identifiers</td>
</tr>
<tr>
<td>qualifiers</td>
<td>List of qualifiers</td>
</tr>
<tr>
<td>target-address</td>
<td>BTP address</td>
</tr>
<tr>
<td>reply-address</td>
<td>BTP address</td>
</tr>
</tbody>
</table>

**target-identifier** identifies the transaction (or transaction tree node). When the message is used to a Decider, this will be the transaction-identifier from the BEGUN message. Otherwise it will be the superior-identifier from a CONTEXT or an inferior-identifier from an ENROL message.

**inferiors-list** defines which inferiors enrolled with the target are to be included in the INFERIOR_STATUSES, using the “inferior-identifiers” as on the ENROL received by the Decider (in its role as Superior). If the list is absent, the status of all enrolled Inferiors will be reported.

**qualifiers** standardised or other qualifiers.

**target-address** the address to which the REQUEST_STATUS message is sent. When used to a Decider, this will be the “decider-address” from the BEGUN message. Otherwise it may be a “superior-address” from a CONTEXT or “inferior-address” from an ENROL message.

**reply-address** the address to which the replying INFERIOR_STATUSES is to be sent.

Types of FAULT possible (sent to reply-address)

**General**

**Redirect** – if the intended target now has a different address

**StatusRefused** – if the receiver is not prepared to report its status to the
sender of this message. This “fault-type” shall not be issued when a Decider
receives REQUEST_STATUSES from the Terminator.

UnknownTransaction – if the transaction-identifier is unknown

The form REQUEST_INFERIOR_STATUSES/all refers to a REQUEST_STATUS with the
instructors-list absent. The form REQUEST_INFERIOR_STATUSES/specific refers to a
REQUEST_INFERIOR_STATUSES with the instructors-list present.

INFERIOR_STATUSES

Sent by a Decider to report the status of all or some of its inferiors in response to a
REQUEST_INFERIOR_STATUSES, PREPARE_INFERIORS, CANCEL_INFERIORS,
CANCEL_TRANSACTION with “report-hazard” value of “true” and
CONFIRM_TRANSACTION with “report-hazard” value of “true”. It is also used by any
actor in response to a received REQUEST_INFERIOR_STATUSES to report the status of
inferiors, if there are any.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>responders-identifier</td>
<td>Identifier</td>
</tr>
<tr>
<td>status-list</td>
<td>Set of Status items - see below</td>
</tr>
<tr>
<td>general-qualifiers</td>
<td>List of qualifiers</td>
</tr>
<tr>
<td>target-address</td>
<td>BTP address</td>
</tr>
</tbody>
</table>

responders-identifier  the target-identifier used on the
REQUEST_INFERIOR_STATUSES.

status-list contains a number of Status-items, each reporting the status of one of
the inferiors of the Decider. The fields of a Status-item are

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>inferior-identifier</td>
<td>Inferior-identifier, identifying which inferior this</td>
</tr>
<tr>
<td></td>
<td>Status-item contains information for.</td>
</tr>
<tr>
<td>status</td>
<td>One of the status values below (these are a</td>
</tr>
<tr>
<td></td>
<td>subset of those for STATUS)</td>
</tr>
<tr>
<td>qualifiers</td>
<td>A list of qualifiers as received from the particular</td>
</tr>
<tr>
<td></td>
<td>inferior or associated with the inferior in earlier</td>
</tr>
<tr>
<td></td>
<td>messages (e.g. an Inferior name qualifier).</td>
</tr>
</tbody>
</table>

The status value reports the current status of the particular inferior, as known to
the Decider (Composer or Coordinator). Values are:
<table>
<thead>
<tr>
<th>status value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>active</td>
<td>The Inferior is enrolled</td>
</tr>
<tr>
<td>resigned</td>
<td>RESIGNED has been received from the Inferior</td>
</tr>
<tr>
<td>preparing</td>
<td>PREPARE has been sent to the Inferior, none of PREPARED, RESIGNED, CANCELLED, HAZARD have been received</td>
</tr>
<tr>
<td>prepared</td>
<td>PREPARED has been received</td>
</tr>
<tr>
<td>autonomously confirmed</td>
<td>CONFIRMED/auto has been received, no completion message has been sent</td>
</tr>
<tr>
<td>autonomously cancelled</td>
<td>PREPARED had been received, and since then CANCELLED has been received but no completion message has been sent</td>
</tr>
<tr>
<td>confirming</td>
<td>CONFIRM has been sent, no outcome reply has been received</td>
</tr>
<tr>
<td>confirmed</td>
<td>CONFIRMED/response has been received</td>
</tr>
<tr>
<td>cancelling</td>
<td>CANCEL has been sent, no outcome reply has been received</td>
</tr>
<tr>
<td>cancelled</td>
<td>CANCELLED has been received, and PREPARED was not received previously</td>
</tr>
<tr>
<td>cancel-contradiction</td>
<td>Confirm had been ordered (and may have been sent), but CANCELLED was received</td>
</tr>
<tr>
<td>confirm-contradiction</td>
<td>Cancel had been ordered (and may have been sent) but CONFIRM/auto was received</td>
</tr>
<tr>
<td>hazard</td>
<td>A HAZARD message has been received</td>
</tr>
<tr>
<td>invalid</td>
<td>No such Inferior is enrolled (used only in reply to a REQUEST_INFERIOR_STATUSES/specific)</td>
</tr>
</tbody>
</table>

**general-qualifiers** standardised or other qualifiers applying to the INFERIOR_STATUSES as a whole. Each Status-item contains a “qualifiers” field containing qualifiers applying to (and received from) the particular Inferior.

**target-address** the address to which the INFERIOR_STATUSES is sent. This will be the “reply-address” on the received message.

If the inferiors-list parameter was present on the received message, only the inferiors identified by that parameter shall have their status reported in status-list of this message. If the inferiors-list parameter was absent, the status of all enrolled inferiors shall be reported, except that an inferior that had been reported as cancelled or resigned on a previous INFERIOR_STATUSES message may be omitted (sender’s option).
Types of FAULT possible (sent to “decider-address”)

General
Invalid Terminator – if Terminator address is unknown
Unknown Transaction – if the transaction-identifier is unknown

Groups – combinations of related messages

The following combinations of messages form related groups, for which the meaning of the group is not just the aggregate of the meanings of the messages. The “&” notation is used to indicate relatedness. Messages appearing in parentheses in the names of groups in this section indicate messages that may or may not be present. The notation A & B / & C in a group name in this section indicates a group that contains A and B or A and C or A, B and C, possibly with any of those appearing more than once.

CONTEXT & application message

Meaning: the transmission of the application message is deemed to be part of the business transaction identified by the CONTEXT. The exact effect of this for application work implied by the transmission of the message is determined by the application – in many cases, it will mean the effects of the application message are to be subject to the outcome delivered to an enrolled Inferior, thus requiring the enrolment of a new Inferior if no appropriate Inferior is enrolled or if the CONTEXT is for cohesion.

target-address: the “target-address” is that of the application message. It is not required that the application address be a BTP address (in particular, there is no BTP-defined “additional information” field – the application protocol (and its binding) may or may not have a similar construct).

There may be multiple application messages related to a single CONTEXT message. All the application messages so related are deemed to be part of the business transaction identified by the CONTEXT. This specification does not imply any further relatedness among the application messages themselves (though the application might).

The actor that sends the group shall retain knowledge of the Superior address in the CONTEXT. If the CONTEXT is a CONTEXT/atom, the actor shall also keep track of transmitted CONTEXTs for which no CONTEXT_REPLY has been received.

If the CONTEXT is a CONTEXT/atom, the actor receiving the CONTEXT shall ensure that a CONTEXT_REPLY message is sent back to the “reply-address” of the CONTEXT with the appropriate completion status.
Note – The representation of the relation between CONTEXT and one or more application messages depends on the binding to the carrier protocol. It is not necessary that the CONTEXT and application messages be closely associated “on the wire” (or even sent on the same connection) – some kind of referencing mechanism may be used.

**CONTEXT_REPLY & ENROL**

**Meaning:** the enrolment of the Inferior identified in the ENROL is to be performed with the Superior identified in the CONTEXT message this CONTEXT_REPLY is replying to. If the “completion-status” of CONTEXT_REPLY is “related”, failure of this enrolment shall prevent the confirmation of the business transaction.

**target-address:** the “target-address” is that of the CONTEXT_REPLY. This will be the “reply-address” of the CONTEXT message (in many cases, including request/reply application exchanges, this address will usually be implicit).

The “target-address” of the ENROL message is omitted.

The actor receiving the related group will use the retained Superior address from the CONTEXT sent earlier to forward the ENROL. When doing so, it changes the ENROL to ask for a response (if it was an ENROL/no-rsp-req) and supplies its own address as the “reply-address”, remembering the original “reply-address” if there was one.

If ENROLLED is received and the original received ENROL was ENROL/rsp-req, the ENROLLED is forwarded back to the original “reply-address”.

If this attempt fails (i.e. ENROLLED is not received), and the “completion-status” of the CONTEXT_REPLY was “related”, the actor is required to ensure that the Superior does not proceed to confirmation. How this is achieved is an implementation option, but must take account of the possibility that direct communication with the Superior may fail. (One method is to prevent CONFIRM_TRANSACTION being sent to the Superior (in its role as Decider); another is to enrol as another Inferior before sending the original CONTEXT out with an application message). If the Superior is a sub-coordinator or sub-composer, an enrolment failure must ensure the sub-coordinator does not send PREPARED to its own Superior.

If the actor receiving the related group is also the Superior (i.e. it has the same binding address), the explicit forwarding of the ENROL is not required, but the resultant effect – that if enrolment fails the Superior does not confirm or issue PREPARED – shall be the same.

A CONTEXT_REPLY & ENROL group may contain multiple ENROL messages, for several Inferiors. Each ENROL shall be forwarded and an ENROLLED reply received
before the Superior is allowed to confirm if the “completion-status” in the
CONTEXT_REPLY was “related”.

When the group is constructed, if the CONTEXT had “superior-type” value of “atom”,
the “completion-status” of the CONTEXT_REPLY shall be “related”. If the “superior-
type” was “cohesive”, the “completion-status” shall be “incompleted” or “related” (as
required by the application). If the value is “completed incomplete”, the actor receiving
the group shall forward the ENROLs, but is not required to (though it may) prevent
confirmation (though it may do so).

**CONTEXT_REPLY (& ENROL) & PREPARED / & CANCELLED**

This combination is characterised by a related CONTEXT_REPLY and either or both of
PREPARED and CANCELLED, with or without ENROL.

**Meaning:** If ENROL is present, the meaning and required processing is the same as for
CONTEXT_REPLY & ENROL. The PREPARED or CANCELLED message(s) are
forwarded to the Superior identified in the CONTEXT message this CONTEXT_REPLY
is replying to.

Note – the combination of CONTEXT_REPLY & ENROL & CANCELLED
may be used to force cancellation of an atom

**target-address:** the “target-address” is that of the CONTEXT_REPLY. This will be the
“reply-address” of the CONTEXT message (in many cases, including request/reply
application exchanges, this address will usually be implicit).

The “target-address” of the PREPARED and CANCELLED message is omitted – they
will be sent to the Superior identified in the earlier CONTEXT message.

The actor receiving the group forwards the PREPARED or CANCELLED message to the
Superior in as for an ENROL, using the retained Superior address from the CONTEXT
sent earlier, except there is no reply required from the Superior.

If (as is usual) an ENROL and PREPARED or CANCELLED message are for the same
Inferior, the ENROL shall be sent first, but the actor need not wait for the ENROLLED to
come back before sending the PREPARED or CANCELLED (so an
ENROL+PREPARED bundle from this actor to the Superior could be used).

The group can contain multiple ENROL, PREPARED and CANCELLED messages.
Each PREPARED and CANCELLED message will be for a different Inferior.. There is
no constraint on the order of their forwarding, except that ENROL and PREPARED or
CANCELLED for the same Inferior shall be delivered to the Superior in the order
ENROL first, followed by the other message for that Inferior.
CONTEXT_REPLY & ENROL & application message (& PREPARED)

This combination is characterised by a related CONTEXT_REPLY, ENROL and an application message. PREPARED may or may not be present in the related group.

Meaning: the relation between the BTP messages is as for the preceding groups. The transmission of the application message (and application effects implied by its transmission) has been associated with the Inferior identified by the ENROL and will be subject to the outcome delivered to that Inferior.

target-address: the “target-address” of the group is the “target-address” of the CONTEXT_REPLY which shall also be the “target-address” of the application message. The ENROL and PREPARED messages do not contain their “target-address” parameters.

The processing of ENROL and PREPARED messages is the same as for the previous groups.

This group can be used when participation in business transaction (normally a cohesion), is initiated by the service (Inferior) side, which fetches or acquires the CONTEXT, with some associated application semantic, performs some work for the transaction and sends an application message with a related ENROL. The CONTEXT_REPLY allows the addressing of the application (and the CONTEXT_REPLY) to be distinct from that of the Superior.

The actor receiving the group may associate the “inferior-identifier” received on the ENROL with the application message in a manner that is visible to the application receiving the message (e.g. for subsequent use in Terminator:Decider exchanges).

BEGUN & CONTEXT

Meaning: the CONTEXT is that for the new business transaction, containing the Superior address.

target-address: the “target-address” is that of the BEGUN message – this will be the “reply-address” of the earlier BEGIN message.

BEGIN & CONTEXT

Meaning: the new business transaction is to be an Inferior (sub-coordinator or sub-composer) of the Superior identified by the CONTEXT. The Factory (receiver of the BEGIN) will perform the enrolment.

target-address: the “target-address” is that of the BEGIN – this will be the address of the Factory.
Standard qualifiers

The following qualifiers are expected to be of general use to many applications and environments. The URI “urn:oasis:names:tc:BTP:1.0:qualifiers” is used in the Qualifier group value for the qualifiers defined here.

Transaction timelimit

The transaction timelimit allows the Superior (or an application element initiating the business transaction) to indicate the expected length of the active phase, and thus give an indication to the Inferior of when it would be appropriate to initiate cancellation if the active phase appears to continue too long. The time limit ends (the clock stops) when the Inferior decides to be prepared and issues PREPARED to the Superior.

It should be noted that the expiry of the time limit does not change the permissible actions of the Inferior. At any time prior to deciding to be prepared (for an Inferior), the Inferior is permitted to initiate cancellation for internal reasons. The timelimit gives an indication to the entity of when it will be useful to exercise this right.

The qualifier is propagated on a CONTEXT message.

The “Qualifier name” shall be “transaction-timelimit”.

The “Content” shall contain the following field:

<table>
<thead>
<tr>
<th>Content field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timelimit</td>
<td>Integer</td>
</tr>
</tbody>
</table>

Timelimit indicates the maximum (further) duration, expressed as whole seconds from the time of transmission of the containing CONTEXT, of the active phase of the business transaction.

Inferior timeout

This qualifier allows an Inferior to limit the duration of its “promise”, when sending PREPARED, that it will maintain the ability to confirm or cancel the effects of all associated operations. Without this qualifier, an Inferior is expected to retain the ability to confirm or cancel indefinitely. If the timeout does expire, the Inferior is released from its promise and can apply the decision indicated in the qualifier.

It should be noted that BTP recognises the possibility that an Inferior may be forced to apply a confirm or cancel decision before the CONFIRM or CANCEL is received and before this timeout expires (or if this qualifier is not used). Such a decision is termed a heuristic decision,
and (as with other transaction mechanisms), is considered to be an exceptional event. As with
heuristic decisions, the taking of an autonomous decision by a Inferior subsequent to the
expiry of this timeout, is liable to cause contradictory decisions across the business
transaction. BTP ensures that at least the occurrence of such a contradiction will be
(eventually) reported to the Superior of the business transaction. BTP treats “true” heuristic
decisions and autonomous decisions after timeout the same way – in fact, the expiry in this
timeout does not cause a qualitative (state table) change in what can happen, but rather a step
change in the probability that it will.

The expiry of the timeout does not strictly require that the Inferior immediately invokes the
intended decision, only that is at liberty to do so. An implementation may choose to only
apply the decision if there is contention for the underlying resource, for example.

Nevertheless, Superiors are recommended to avoid relying on this and ensure decisions for
the business transaction are made before these timeouts expire (and allow a margin of error
for network latency etc.).

The qualifier may be present on a PREPARED message. If the PREPARED message has the
“default-is cancel” parameter “true”, then the “IntendedDecision” field of this qualifier shall
have the value “cancel”.

The “Qualifier name” shall be “inferior-timeout”.

The “Content” shall contain the following fields:

<table>
<thead>
<tr>
<th>Content field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timeout</td>
<td>Integer</td>
</tr>
<tr>
<td>IntendedDecision</td>
<td>“confirm” or “cancel”</td>
</tr>
</tbody>
</table>

Timeout indicates how long, expressed as whole seconds from the time of transmission of the
carrying message, the Inferior intends to maintain its ability to either confirm or cancel the
effects of the associated operations, as ordered by the receiving Superior.

IntendedDecision indicates which outcome will be applied, if the timeout completes and an
autonomous decision is made.

Minimum inferior timeout

This qualifier allows a Superior to constrain the Inferior timeout qualifier received from the
Inferior. If a Superior knows that the decision for the business transaction will not be
determined for some period, it can require that Inferiors do not send PREPARED messages
with Inferior timeouts that would expire before then. An Inferior that is unable or unwilling to
send a PREPARED message with a longer (or no) timeout should cancel, and reply with
CANCELLED.
The qualifier may be present on a CONTEXT, ENROLLED or PREPARE message. If present on more than one, and with different values of the MinimumTimeout field, the value on ENROLLED shall prevail over that on CONTEXT and the value on PREPARE shall prevail over either of the others.

The “Qualifier name” shall be “minimum-inferior-timeout”.

The “Content” shall contain the following field:

<table>
<thead>
<tr>
<th>Content field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>MinimumTimeout</td>
<td>Integer</td>
</tr>
</tbody>
</table>

**Minimum Timeout** is the minimum value of timeout, expressed as whole seconds, that will be acceptable in the Inferior timeout qualifier on an answering PREPARED message.

**Inferior name**

This qualifier allows an Enroller to supply a name for the Inferior that will be visible on INFERIOR_STATUSES and thus allow the Terminator to determine which Inferior (of the Composer or Coordinator) is related to which application work. This is in addition to the “inferior-identifier” field. The name can be human-readable and can also be used in fault tracing, debugging and auditing.

The name is never used by the BTP actors themselves to identify each other or to direct messages. (The BTP actors use the addresses and the identifiers in the message parameters for those purposes.)

This specification makes no requirement that the names are unambiguous within any scope (unlike the globally unambiguous “inferior-identifier” on ENROLLED and BEGUN). Other specifications, including those defining use of BTP with a particular application may place requirements on the use and form of the names. (This may include reference to information passed in application messages or in other, non-standardised, qualifiers.)

The qualifier may be present on BEGIN, ENROL and in the “qualifiers” field of a Status-item in INFERIOR_STATUSES. It is present on BEGIN only if there is a related CONTEXT; if present, the same qualifier value **should** be included in the consequent ENROL. If INFERIOR_STATUSES includes a Status-item for an Inferior whose ENROL had an inferior-name qualifier, the same qualifier value **should** be included in the Status-item.

The “Qualifier -name” shall be “inferior-name”

The “Content” shall contain the following fields:

<table>
<thead>
<tr>
<th>Content field</th>
<th>Type</th>
</tr>
</thead>
</table>
inferior-name  String

4309
4310 Inferior name the name assigned to the enrolling Inferior.
4311
Explanation of the state tables

The state tables deal with the state transitions of the Superior and Inferior roles and which message can be sent and received in each state. The state tables directly cover only a single, bi-lateral Superior:Inferior relationship. The interactions between, for example, multiple Inferiors of a single Superior that will apply the same decision to all or some (of them), are dealt with in the definitions of the “decision” events which also specify when changes are made to persistent state information (see below).

There are two state tables, one for Superior, one for Inferior. States are identified by a letter-digit pair, with upper-case letters for the superior, lower-case for the inferior. The same letter is used to group states which have the same, or similar, persistent state, with the digit indicating volatile state changes or minor variations. Corresponding upper and lower-case letters are used to identify (approximately) corresponding Superior and Inferior states.

The Inferior table includes events occurring both at the Inferior as such and at the associated Enroller, as the Enroller’s actions are constrained by and constrain the Inferior role itself.

In the state tables, each side is either waiting to make a decision or can send a message. For some states, the message to be sent is a repetition of a regular message; for other states, the INFERIOR_STATE or SUPERIOR_STATE message can be sent, requesting a response. Normally, on entry to a state that allows the sending of any message other than one of the *_STATE messages, the implementation will send that message – failure to do so will cause the relationship to lock up. The message can be resent if the implementation determines that the original message (or the next message sent in reply) may have been lost.

Status queries

In BTP the messages SUPERIOR_STATE and INFERIOR_STATE are available to prompt the peer to report its current state by repeating the previous message (when this is allowed) or by sending the other *_STATE message. The “reply_requested” parameter of these messages distinguishes between their use as a prompt and as a reply. An implementation receiving a *_STATE message with “reply_requested” as “true” is not required to reply immediately – it may choose to delay any reply until a decision event occurs and then send the appropriate new message (e.g. on receiving INFERIOR_STATE/prepared/y while in state E1, a superior is permitted to delay until it has performed “decide to confirm” or “decide to cancel”). However, this may cause the other side to repeatedly send interrogatory *_STATE messages.

Note that a Superior (or some entity standing in for a now-extinct Superior) uses SUPERIOR_STATE/unknown to reply to messages received from an Inferior where the Superior:Inferior relationship is in an unknown (using state “Y1”). The *_STATE messages with a “state” value “inaccessible” can be used as a reply when any message is received and the implementation is temporarily unable to determine whether the relationship is known or what the state is. Other than these cases, the *_STATE messages with “response_requested” equal to “false” are only sent when the other message with “response_requested” equal to
“true” has been received and no other message has been sent. Receipt of the
* STATE/inaccessible messages is not shown in the tables and has no effect on the state at
the receiving side (though it may cause the implementation to resend its own message after
some interval of its own choosing).

Decision events

The persistent state changes (equivalent to logging in a regular transaction system) and some
other events are modelled as “decision events” (e.g. “decide to confirm”, “decide to be
prepared”). The exact nature of the real events and changes in an implementation that are
modelled by these events depends on the position of the Superior or Inferior within the
business transaction and on features of the implementation (e.g. making of a persistent record
of the decision means that the information will survive at least some failures that otherwise
lose state information, but the level of survival depends on the purpose of the
implementation). Table 3 and Table 4 define the decision events.

In some cases, an implementation may not need to make an active change to have a persistent
record of a decision, provided that the implementation will restore itself to the appropriate
state on recovery. For example, an (inferior) implementation that “decided to be prepared”,
and recorded a timeout (to cancel) in the persistent information for that decision (signalled via
the appropriate qualifier on PREPARED), could treat the presence of an expired record as a
record of “decide to cancel autonomously”, provided it always updated such a record as part
of the “apply ordered confirmation” decision event.

The Superior event “decide to prepare” is considered semi-persistent. Since the sending of
PREPARE indicates that the application exchange (to associate operations with the Inferior)
is complete, it is not meaningful for the Superior:Inferior relationship to revert to an earlier
state corresponding to an incomplete application exchange. However, implementations are
not required to make the sending of PREPARE persistent in terms of recovery – a Superior
that experiences failure after sending PREPARE may, on recovery, have no information
about the transaction, in which case it is considered to be in the completed state (Z), which
will imply the cancellation of the Inferior and its associated operations.

Where a Superior is an Intermediate (i.e. is itself an Inferior to another Superior entity), in a
transaction hierarchy tree, its “decide to confirm” and “decide to cancel” decisions will in fact
be the receipt of a CONFIRM or CANCEL instruction from its own Superior, without
necessary change of local persistent information (which would combine both superior and
inferior information, pointing both up and down the tree).

Disruptions – failure events

Failure events are modelled as “disruption”. A failure and the subsequent recovery will (or
may) cause a change of state. The disruption events in the state tables model different extents
of loss of state information. An implementation is not required to exhibit all the possible
disruption events, but it is not allowed to exhibit state transitions that do not correspond to a possible disruption. The different levels of disruption describe legitimate states for the endpoint to be in after it has been restored to normal functioning. The absence of a destination state for the disruption events means that such a transition is not legitimate – thus, for example, an Inferior that has decided to be prepared will always recover to the same state, by virtue of the information persisted in the “decide to be prepared” event.

In addition to the disruption events in the tables, there is an implicit “disruption 0” event, which involves possible interruption of service and loss of messages in transit, but no change of state (either because no state information was lost, or because recovery from persistent information restores the implementation to the same state). The “disruption 0” event would typically be an appropriate abstraction for a communication failure.

Invalid cells and assumptions of the communication mechanism

The empty cells in state table represent events that cannot happen. For events corresponding to sending a message or any of the decision events, this prohibition is absolute – e.g. a conformant implementation in the Superior active state “B1” will not send CONFIRM. For events corresponding to receiving a message, the interpretation depends on the properties of the underlying communications mechanism.

For all communication mechanisms, it is assumed that
a) the two directions of the Superior:Inferior communication are not synchronised – that is messages travelling in opposite directions can cross each other to any degree; any number of messages may be in transit in either direction; and
b) messages may be lost arbitrarily

If the communication mechanisms guarantee ordered delivery (i.e. that messages, if delivered at all, are delivered to the receiver in the order they were sent), then receipt of a message in a state where the corresponding cell is empty indicates that the far-side has sent a message out of order – a FAULT message with the “fault-type” “WrongState” can be returned.

If the communication mechanisms cannot guarantee ordered delivery, then messages received where the corresponding cell is empty should be ignored. Assuming the far-side is conformant, these messages can assumed to be “stale” and have been overtaken by messages sent later but already delivered. (If the far-side is non-conformant, there is a problem anyway).

Meaning of state table events

The tables in this section define the events (rows) in the state tables. Table 1 defines the events corresponding to sending or receiving BTP messages and the disruption events, Table 2 describes the decision events for an Inferior, Table 3 those for a Superior.

**Table 1**

**Table 2**

**Table 3**

**Table 4**
The decision events for a Superior, defined in Table 4, cannot be specified without reference to other Inferiors to which it is Superior and to its relation with the application or other entity that (acting ultimately on behalf of the application) drives it.

The term “remaining Inferiors” refers to any actors to which this endpoint is Superior and which are to be treated as an atomic decision unit with (and thus including) the Inferior on this relationship. If the CONTEXT for this Superior:Inferior relationship had a “superior-type” of “atom”, this will be all Inferiors established with same Superior address and “superior-identifier” except those from which RESIGN has been received. If the CONTEXT had “superior-type” of “cohesion”, the “remaining Inferiors” excludes any that it has been determined will be cancelled, as well as any that have resigned – in other words it includes only those for which a confirm decision is still possible or has been made. The determination of exactly which Inferiors are “remaining Inferiors” in a cohesion is determined, in some way, by the application. The term “Other remaining Inferiors” excludes this Inferior on this relationship. A Superior with a single Inferior will have no “other remaining Inferiors”.

In order to ensure that the confirmation decision is delivered to all remaining Inferiors, despite failures, the Superior must persistently record which these Inferiors are (i.e. their addresses and identifiers). It must also either record that the decision is confirm, or ensure that the confirm decision (if there is one) is persistently recorded somewhere else, and that it will be told about it. This latter would apply if the Superior were also BTP Inferior to another entity which persisted a confirm decision (or recursively deferred it still higher). However, since there is no requirement that the Superior be also a BTP Inferior to any other entity, the behaviour of asking another entity to make (and persist) the confirm decision is termed “offering confirmation” - the Superior offers the possible confirmation of itself, and its remaining Inferiors to some other entity. If that entity (or something higher up) then does make and persist a confirm decision, the Superior is “instructed to confirm” (which is equivalent BTP CONFIRM).

The application, or an entity acting indirectly on behalf of the application, may request a Superior to prepare an Inferior (or all Inferiors). This typically implies that there will be no more operations associated with the Inferior. Following a request to prepare all remaining Inferiors, the Superior may offer confirmation to the entity that requested the prepare. (If the Superior is also a BTP Inferior, its superior can be considered an entity acting on behalf of the application.)

The application, or an entity acting indirectly on behalf of the application, may also request confirmation. This means the Superior is to attempt to make and persist a confirm decision itself, rather than offer confirmation.

Table 21: send, receive and disruption events

<table>
<thead>
<tr>
<th>Event name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>send/receive ENROL/rsp-req</td>
<td>send/receive ENROL with response-requested = true</td>
</tr>
</tbody>
</table>

OASIS BTPDraft Specification 0.9.2.4, 3 April 2002
<table>
<thead>
<tr>
<th>Event name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>send/receive ENROL/no-rsp-req</td>
<td>send/receive ENROL with response-requested = false</td>
</tr>
<tr>
<td>send/receive RESIGN/rsp-req</td>
<td>send/receive RESIGN with response-requested = true</td>
</tr>
<tr>
<td>send/receive RESIGN/no-rsp-req</td>
<td>send/receive RESIGN with response-requested = false</td>
</tr>
<tr>
<td>send/receive PREPARED</td>
<td>send/receive PREPARED, with default-cancel = false</td>
</tr>
<tr>
<td>send/receive PREPARED/cancel</td>
<td>send/receive PREPARED, with default-cancel = true</td>
</tr>
<tr>
<td>send/receive CONFIRMED/auto</td>
<td>send/receive CONFIRMED, with confirm-received = true</td>
</tr>
<tr>
<td>send/receive CONFIRMED/response</td>
<td>send/receive CONFIRMED, with confirm-received = false</td>
</tr>
<tr>
<td>send/receive HAZARD</td>
<td>send/receive HAZARD</td>
</tr>
<tr>
<td>send/receive INF_STATE/***/y</td>
<td>send/receive INFERIOR_STATE with status *** and response-requested = true</td>
</tr>
<tr>
<td>send/receive INF_STATE/***</td>
<td>send/receive INFERIOR_STATE with status *** and response-requested = false</td>
</tr>
<tr>
<td>send/receive SUP_STATE/***/y</td>
<td>send/receive SUPERIOR_STATE with status *** and response-requested = true (&quot;prepared-rcvd&quot; represents &quot;prepared-received&quot;)</td>
</tr>
<tr>
<td>send/receive SUP_STATE/***</td>
<td>send/receive SUPERIOR_STATE with status *** and response-requested = false (&quot;prepared-rcvd&quot; represents &quot;prepared-received&quot;)</td>
</tr>
<tr>
<td>disruption ***</td>
<td>Loss of state– new state is state applying after any local recovery processes complete</td>
</tr>
</tbody>
</table>

**Table 32: Decision events for Inferior**

<table>
<thead>
<tr>
<th>Event name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>decide to resign</td>
<td>• Any associated operations have had no effect (data state is unchanged)).</td>
</tr>
<tr>
<td>decide to be prepared</td>
<td>• Effects of all associated operations can be confirmed or cancelled; • information to retain confirm/cancel ability has been made persistent</td>
</tr>
<tr>
<td>decide to be prepared/cancel</td>
<td>• As “decide to be prepared”; • the persistent information specifies that the default action will be to cancel</td>
</tr>
<tr>
<td>Event name</td>
<td>Meaning</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>decide to confirm autonomously</td>
<td>• Decision to confirm autonomously has been made persistent;</td>
</tr>
<tr>
<td></td>
<td>• the effects of associated operations will be confirmed regardless of failures</td>
</tr>
<tr>
<td>decide to cancel autonomously</td>
<td>• Decision to cancel autonomously has been made persistent</td>
</tr>
<tr>
<td></td>
<td>• the effects of associated operations will be cancelled regardless of failures</td>
</tr>
<tr>
<td>apply ordered confirmation</td>
<td>• Effects of all associated operations have been confirmed;</td>
</tr>
<tr>
<td></td>
<td>• Persistent information is effectively removed</td>
</tr>
<tr>
<td>remove persistent information</td>
<td>• Persistent information is effectively removed</td>
</tr>
<tr>
<td>detect problem</td>
<td>• For at least some of the associated operations, EITHER</td>
</tr>
<tr>
<td></td>
<td>o they cannot be consistently cancelled or consistently confirmed; OR</td>
</tr>
<tr>
<td></td>
<td>o it cannot be determined whether they will be cancelled or confirmed</td>
</tr>
<tr>
<td></td>
<td>• AND, information about this is not persistent</td>
</tr>
<tr>
<td>detect and record problem</td>
<td>• As for the first condition of “detect problem”</td>
</tr>
<tr>
<td></td>
<td>• information recording this has been persisted (to the degree considered appropriate), or the detection itself is persistent. (i.e. will be re-detected on recovery)</td>
</tr>
</tbody>
</table>

Table 43: Decision events for a Superior

<table>
<thead>
<tr>
<th>Event name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>decide to confirm one-phase</td>
<td>• All associated application messages to be sent to the service have been sent;</td>
</tr>
<tr>
<td></td>
<td>• There are no other remaining Inferiors</td>
</tr>
<tr>
<td></td>
<td>• If an atom, all enrolments that would create other Inferiors have completed (no outstanding CONTEXT_REPLYS)</td>
</tr>
<tr>
<td></td>
<td>• The Superior has been requested to confirm</td>
</tr>
<tr>
<td>decide to prepare</td>
<td>• All associated application messages to be sent to the service have been sent;</td>
</tr>
<tr>
<td>Event name</td>
<td>Meaning</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>decide to confirm</td>
<td>• The Superior has been requested to prepare this Inferior</td>
</tr>
<tr>
<td></td>
<td>• Either • PREPARRED or PREPARRED/cancel has been received from all other remaining Inferiors; AND</td>
</tr>
<tr>
<td></td>
<td>• Superior has been requested to confirm; AND</td>
</tr>
<tr>
<td></td>
<td>• persistent information records the confirm decision and identifies all remaining Inferiors;</td>
</tr>
<tr>
<td></td>
<td>• Or • persistent information records an offer of confirmation and has been instructed to confirm</td>
</tr>
<tr>
<td>decide to cancel</td>
<td>• Superior has not offered confirmation; OR</td>
</tr>
<tr>
<td></td>
<td>• Superior has offered confirmation and has been instructed to cancel; OR</td>
</tr>
<tr>
<td></td>
<td>• Superior has offered confirmation but has made an autonomous cancellation decision</td>
</tr>
<tr>
<td>remove confirm information</td>
<td>• Persistent information has been effectively removed;</td>
</tr>
<tr>
<td>record contradiction</td>
<td>• Information recording the contradiction has been persisted (to the degree considered appropriate)</td>
</tr>
</tbody>
</table>

**Persistent information**

Persisted information (especially prepared information at an Inferior, confirm information at a Superior) may include qualifications of the state carried in Qualifiers of the corresponding message (e.g. inferior timeouts in prepared information). It may also include application-specific information (especially in Inferiors) to allow the future confirmation or cancellation of the associated operations. In some cases it will also include information allowing an application message sent with a BTP message (e.g. PREPARED) to be repeated.

The “effective” removal of persistent information allows for the possibility that the information is retained (perhaps for audit and tracing purposes) but some change to the persistent information (as a whole) means that if there is a failure after such change, on recovery, the persistent information does not cause the endpoint to return the state it would have recovered to before the change.

In all cases, the degree to which information described as “persistent” will survive failure is a configuration and implementation option. An implementation should describe the level of failure that it is capable of surviving. For applications manipulating information that is itself volatile (e.g. network configurations), there is no requirement to make the BTP state information more persistent that than the application information.
The degree of persistence of the recording of a hazard (problem) at an Inferior and recording of a detected contradiction at a Superior may be different from that applying to the persistent prepared and confirm information. Implementations and configuration may choose to pass hazard and contradiction information via management mechanisms rather than through BTP. Such passing of information to a management mechanism could be treated as “record problem” or “record contradiction”.
Table 54: Superior states

<table>
<thead>
<tr>
<th>State</th>
<th>summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>I1</td>
<td>CONTEXT created</td>
</tr>
<tr>
<td>A1</td>
<td>ENROling</td>
</tr>
<tr>
<td>B1</td>
<td>ENROLLED (active)</td>
</tr>
<tr>
<td>C1</td>
<td>resigning</td>
</tr>
<tr>
<td>D1</td>
<td>PREPARE sent</td>
</tr>
<tr>
<td>E1</td>
<td>PREPARED received</td>
</tr>
<tr>
<td>E2</td>
<td>PREPARED/cancel received</td>
</tr>
<tr>
<td>F1</td>
<td>CONFIRM sent</td>
</tr>
<tr>
<td>F2</td>
<td>completed after confirm</td>
</tr>
<tr>
<td>G1</td>
<td>cancel decided</td>
</tr>
<tr>
<td>G2</td>
<td>CANCEL sent</td>
</tr>
<tr>
<td>G3</td>
<td>cancelling, RESIGN received</td>
</tr>
<tr>
<td>G4</td>
<td>both cancelled</td>
</tr>
<tr>
<td>H1</td>
<td>inferior autonomously confirmed</td>
</tr>
<tr>
<td>J1</td>
<td>Inferior autonomously cancelled</td>
</tr>
<tr>
<td>K1</td>
<td>confirmed, contradiction detected</td>
</tr>
<tr>
<td>L1</td>
<td>cancelled, contradiction detected</td>
</tr>
<tr>
<td>P1</td>
<td>hazard reported</td>
</tr>
<tr>
<td>P2</td>
<td>hazard reported in null state</td>
</tr>
<tr>
<td>P3</td>
<td>hazard reported after confirm decision</td>
</tr>
<tr>
<td>P4</td>
<td>hazard reported after cancel decision</td>
</tr>
<tr>
<td>Q1</td>
<td>contradiction detected in null state</td>
</tr>
<tr>
<td>R1</td>
<td>Contradiction or hazard recorded</td>
</tr>
<tr>
<td>R2</td>
<td>completed after contradiction or hazard recorded</td>
</tr>
<tr>
<td>S1</td>
<td>one-phase confirm decided</td>
</tr>
<tr>
<td>Y1</td>
<td>completed queried</td>
</tr>
<tr>
<td>Z</td>
<td>completed and unknown</td>
</tr>
</tbody>
</table>
### Table 6: Inferior states

<table>
<thead>
<tr>
<th>State</th>
<th>summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>i1</td>
<td>aware of CONTEXT</td>
</tr>
<tr>
<td>a1</td>
<td>enrolling</td>
</tr>
<tr>
<td>b1</td>
<td>enrolled</td>
</tr>
<tr>
<td>c1</td>
<td>resigning</td>
</tr>
<tr>
<td>d1</td>
<td>preparing</td>
</tr>
<tr>
<td>e1</td>
<td>prepared</td>
</tr>
<tr>
<td>e2</td>
<td>prepared, default to cancel</td>
</tr>
<tr>
<td>f1</td>
<td>confirming</td>
</tr>
<tr>
<td>f2</td>
<td>confirming after default cancel</td>
</tr>
<tr>
<td>g1</td>
<td>CANCEL received in prepared state</td>
</tr>
<tr>
<td>g2</td>
<td>CANCEL received in prepared/cancel state</td>
</tr>
<tr>
<td>h1</td>
<td>Autonomously confirmed</td>
</tr>
<tr>
<td>h2</td>
<td>autonomously confirmed, superior confirmed</td>
</tr>
<tr>
<td>j1</td>
<td>autonomously cancelled</td>
</tr>
<tr>
<td>j2</td>
<td>autonomously cancelled, superior cancelled</td>
</tr>
<tr>
<td>k1</td>
<td>autonomously cancelled, contradicted</td>
</tr>
<tr>
<td>k2</td>
<td>autonomously cancelled, CONTRADICTION received</td>
</tr>
<tr>
<td>l1</td>
<td>autonomously confirmed, contradicted</td>
</tr>
<tr>
<td>l2</td>
<td>autonomously confirmed, CONTRADICTION received</td>
</tr>
<tr>
<td>m1</td>
<td>confirmation applied</td>
</tr>
<tr>
<td>n1</td>
<td>cancelling</td>
</tr>
<tr>
<td>p1</td>
<td>hazard detected, not recorded</td>
</tr>
<tr>
<td>p2</td>
<td>hazard detected in prepared state, not recorded</td>
</tr>
<tr>
<td>q1</td>
<td>hazard recorded</td>
</tr>
<tr>
<td>s1</td>
<td>CONFIRM_ONE_PHASE received after prepared state</td>
</tr>
<tr>
<td>s2</td>
<td>CONFIRM_ONE_PHASE received</td>
</tr>
<tr>
<td>s3</td>
<td>CONFIRM_ONE_PHASE received, confirming</td>
</tr>
<tr>
<td>s4</td>
<td>CONFIRM_ONE_PHASE received, cancelling</td>
</tr>
<tr>
<td>s5</td>
<td>CONFIRM_ONE_PHASE received, hazard detected</td>
</tr>
<tr>
<td>s6</td>
<td>CONFIRM_ONE_PHASE received, hazard recorded</td>
</tr>
<tr>
<td>x1</td>
<td>completed, presuming abort</td>
</tr>
<tr>
<td>State</td>
<td>Summary</td>
</tr>
<tr>
<td>-------</td>
<td>----------------------------------------------</td>
</tr>
<tr>
<td>x2</td>
<td>completed, presuming abort after prepared/cancel</td>
</tr>
<tr>
<td>y1</td>
<td>completed, queried</td>
</tr>
<tr>
<td>y2</td>
<td>completed, default cancel, a message received</td>
</tr>
<tr>
<td>z</td>
<td>completed</td>
</tr>
</tbody>
</table>
| z1    | completed with default cancel

4529
4530
### Superior state table

#### Table 76: Superior state table – normal forward progression

<table>
<thead>
<tr>
<th>I</th>
<th>A1</th>
<th>B1</th>
<th>B2</th>
<th>C1</th>
<th>D1</th>
<th>E1</th>
<th>E2</th>
<th>F1</th>
<th>F2</th>
</tr>
</thead>
<tbody>
<tr>
<td>receive ENROL/rsp-req</td>
<td>A1</td>
<td>A1</td>
<td>B2</td>
<td>B2</td>
<td>D1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive ENROL/no-rsp-req</td>
<td>B1</td>
<td>B1</td>
<td>B1</td>
<td>B1</td>
<td>D1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive RESIGN/rsp-req</td>
<td>Y1</td>
<td>C1</td>
<td>C1</td>
<td>C1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive RESIGN/no-rsp-req</td>
<td>Z</td>
<td>Z</td>
<td>Z</td>
<td>Z</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive PREPARED</td>
<td>Y1</td>
<td>E1</td>
<td>E1</td>
<td>E1</td>
<td>E1</td>
<td>F1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive PREPARED/cancel</td>
<td>Y1</td>
<td>E2</td>
<td>E2</td>
<td>E2</td>
<td>E2</td>
<td>F1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive CONFIRMED/auto</td>
<td>Q1</td>
<td>H1</td>
<td>H1</td>
<td>H1</td>
<td>H1</td>
<td>F1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive CONFIRMED/response</td>
<td></td>
<td>F2</td>
<td>F2</td>
<td>F2</td>
<td>F2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive CANCELLED</td>
<td>Y1</td>
<td>Z</td>
<td>Z</td>
<td>Z</td>
<td>J1</td>
<td>J1</td>
<td>K1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive HAZARD</td>
<td>P1</td>
<td>P1</td>
<td>P1</td>
<td>P1</td>
<td>P1</td>
<td>F1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive INF_STATE/active/y</td>
<td>Y1</td>
<td>A1</td>
<td>B1</td>
<td>B2</td>
<td>D1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive INF_STATE/active</td>
<td>B1</td>
<td>B2</td>
<td>D1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive INF_STATE/unknown</td>
<td>Z</td>
<td>Z</td>
<td>Z</td>
<td>Z</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send ENROLLED</td>
<td></td>
<td>B1</td>
<td>B1</td>
<td>Z</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send RESIGNED</td>
<td></td>
<td></td>
<td></td>
<td>D1</td>
<td>E1</td>
<td>E2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send PREPARE</td>
<td></td>
<td></td>
<td></td>
<td>B1</td>
<td>B2</td>
<td>D1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send CONFIRM_ONE_PHASE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send CONFIRM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>F1</td>
</tr>
<tr>
<td>send CANCEL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send CONTRADICTION</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send SUP_STATE/active/y</td>
<td></td>
<td>B1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send SUP_STATE/active</td>
<td></td>
<td>B1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send SUP_STATE/prepared-rcvd/y</td>
<td></td>
<td>E1</td>
<td>E2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send SUP_STATE/prepared-rcvd</td>
<td></td>
<td>E1</td>
<td>E2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send SUP_STATE/unknown</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>decide to confirm one-phase</td>
<td></td>
<td>S1</td>
<td>S1</td>
<td>S1</td>
<td>S1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>decide to prepare</td>
<td></td>
<td>D1</td>
<td>D1</td>
<td>D1</td>
<td>D1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>decide to confirm</td>
<td></td>
<td>G1</td>
<td>G1</td>
<td>G1</td>
<td>G1</td>
<td>F1</td>
<td>F1</td>
<td>F1</td>
<td></td>
</tr>
<tr>
<td>decide to cancel</td>
<td></td>
<td>G1</td>
<td>G1</td>
<td>G1</td>
<td>G1</td>
<td>Z</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>remove persistent information</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Z</td>
</tr>
<tr>
<td>record contradiction</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>disruption I</td>
<td></td>
<td>Z</td>
<td>Z</td>
<td>Z</td>
<td>B1</td>
<td>Z</td>
<td>Z</td>
<td>Z</td>
<td>F1</td>
</tr>
<tr>
<td>disruption II</td>
<td></td>
<td>Z</td>
<td></td>
<td></td>
<td>D1</td>
<td>D1</td>
<td>B1</td>
<td>B1</td>
<td></td>
</tr>
<tr>
<td>disruption III</td>
<td></td>
<td>B1</td>
<td>B1</td>
<td>B1</td>
<td>B1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>disruption IV</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 87: Superior state table – cancellation and contradiction

<table>
<thead>
<tr>
<th>Event</th>
<th>G1</th>
<th>G2</th>
<th>G3</th>
<th>G4</th>
<th>H1</th>
<th>J1</th>
<th>K1</th>
<th>L1</th>
</tr>
</thead>
<tbody>
<tr>
<td>receive ENROL/rsp-req</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive ENROL/no-rsp-req</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive RESIGN/rsp-req</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive RESIGN/no-rsp-req</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive PREPARED</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive PREPARED/cancel</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive CONFIRMED/auto</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive CONFIRMED/response</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive CANCELLED</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive HAZARD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive INF_STATE/active/y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive INF_STATE/active</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive INF_STATE/unknown</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send ENROLLED</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send RESIGNED</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send PREPARE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send CONFIRM/ONE_PHASE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send CONFIRM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send CANCEL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send CONTRADICTION</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send SUP_STATE/active/y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send SUP_STATE/active</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send SUP_STATE/prepared-rcvd/y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send SUP_STATE/prepared-rcvd</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send SUP_STATE/unknown</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>decide to confirm one-phase</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>decide to prepare</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>decide to confirm</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>decide to cancel</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>remove persistent information</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>record contradiction</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>disruption I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>disruption II</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>disruption III</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>disruption IV</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**4533**
Table 98: Superior state table – hazard and request confirm

<table>
<thead>
<tr>
<th>Event</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
<th>Q1</th>
<th>R1</th>
<th>R2</th>
<th>S1</th>
</tr>
</thead>
<tbody>
<tr>
<td>receive ENROL/rsp-req</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>S1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive ENROL/no-rsp-req</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>S1</td>
<td></td>
</tr>
<tr>
<td>receive RESIGN/rsp-req</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Z</td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive RESIGN/no-rsp-req</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Z</td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive PREPARED</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Q1</td>
<td>R1</td>
<td>R1</td>
<td>S1</td>
</tr>
<tr>
<td>receive PREPARED/cancel</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>S1</td>
<td></td>
</tr>
<tr>
<td>receive CONFIRMED/auto</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Z</td>
<td></td>
<td>R2</td>
<td>Z</td>
</tr>
<tr>
<td>receive CONFIRMED/response</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive CANCELLED</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>R1</td>
<td>R1</td>
<td>Z</td>
</tr>
<tr>
<td>receive HAZARD</td>
<td></td>
<td>P1</td>
<td>P2</td>
<td>P3</td>
<td>P4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive INF_STATE/active/y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>S1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive INF_STATE/active</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>S1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive INF_STATE/unknown</td>
<td>P1</td>
<td>P2</td>
<td>P4</td>
<td></td>
<td></td>
<td>R2</td>
<td>R2</td>
<td>Z</td>
</tr>
<tr>
<td>send ENROLLED</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send RESIGNED</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send PREPARE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send CONFIRM_ONE_PHASE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>S1</td>
<td></td>
</tr>
<tr>
<td>send CONFIRM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send CANCEL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send CONTRADICTION</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>R2</td>
<td></td>
</tr>
<tr>
<td>send SUP_STATE/active/y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send SUP_STATE/active</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send SUP_STATE/prepared-rcvd/y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send SUP_STATE/prepared-rcvd</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send SUP_STATE/unknown</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>decide to confirm one-phase</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>decide to prepare</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>decide to confirm</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>decide to cancel</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>remove persistent information</td>
<td>R1</td>
<td>R1</td>
<td>R1</td>
<td>R1</td>
<td>R1</td>
<td></td>
<td></td>
<td>Z</td>
</tr>
<tr>
<td>record contradiction</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>disruption I</td>
<td>Z</td>
<td>Z</td>
<td>Z</td>
<td>Z</td>
<td>Z</td>
<td></td>
<td>R1</td>
<td>Z</td>
</tr>
<tr>
<td>disruption II</td>
<td>D1</td>
<td>F1</td>
<td>G2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>disruption III</td>
<td>B1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>disruption IV</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 109: Superior state table – query after completion and completed states

<table>
<thead>
<tr>
<th>Event</th>
<th>Y1</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>receive ENROL/rsp-req</td>
<td>Y1</td>
<td>Y1</td>
</tr>
<tr>
<td>receive ENROL/no-rsp-req</td>
<td>Y1</td>
<td>Y1</td>
</tr>
<tr>
<td>receive RESIGN/rsp-req</td>
<td>Y1</td>
<td>Y1</td>
</tr>
<tr>
<td>receive RESIGN/no-rsp-req</td>
<td></td>
<td>Z</td>
</tr>
<tr>
<td>receive PREPARED</td>
<td>Y1</td>
<td>Y1</td>
</tr>
<tr>
<td>receive PREPARED/cancel</td>
<td>Y1</td>
<td>Y1</td>
</tr>
<tr>
<td>receive CONFIRMED/auto</td>
<td>Q1</td>
<td>Q1</td>
</tr>
<tr>
<td>receive CONFIRMED/response</td>
<td>Z</td>
<td>Z</td>
</tr>
<tr>
<td>receive CANCELLED</td>
<td>Y1</td>
<td>Y1</td>
</tr>
<tr>
<td>receive HAZARD</td>
<td>P2</td>
<td>P2</td>
</tr>
<tr>
<td>receive INF_STATE/active/y</td>
<td>Y1</td>
<td>Y1</td>
</tr>
<tr>
<td>receive INF_STATE/active</td>
<td>Y1</td>
<td>Z</td>
</tr>
<tr>
<td>receive INF_STATE/unknown</td>
<td>Z</td>
<td>Z</td>
</tr>
<tr>
<td>send ENROLLED</td>
<td></td>
<td></td>
</tr>
<tr>
<td>send RESIGNED</td>
<td></td>
<td></td>
</tr>
<tr>
<td>send PREPARE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>send CONFIRM_ONE_PHASE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>send CONFIRM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>send CANCEL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>send CONTRADICTION</td>
<td></td>
<td></td>
</tr>
<tr>
<td>send SUP_STATE/active/y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>send SUP_STATE/active</td>
<td></td>
<td></td>
</tr>
<tr>
<td>send SUP_STATE/prepared-rcvd/y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>send SUP_STATE/prepared-rcvd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>send SUP_STATE/unknown</td>
<td>Z</td>
<td></td>
</tr>
<tr>
<td>decide to confirm one-phase</td>
<td></td>
<td></td>
</tr>
<tr>
<td>decide to prepare</td>
<td></td>
<td></td>
</tr>
<tr>
<td>decide to confirm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>decide to cancel</td>
<td></td>
<td></td>
</tr>
<tr>
<td>remove persistent information</td>
<td></td>
<td></td>
</tr>
<tr>
<td>record contradiction</td>
<td></td>
<td></td>
</tr>
<tr>
<td>disruption I</td>
<td>Z</td>
<td></td>
</tr>
<tr>
<td>disruption II</td>
<td></td>
<td></td>
</tr>
<tr>
<td>disruption III</td>
<td></td>
<td></td>
</tr>
<tr>
<td>disruption IV</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Inferior state table

Table 1140: Inferior state table – normal forward progression

<table>
<thead>
<tr>
<th>send ENROL/rsp-req</th>
<th>send ENROL/no-rsp-req</th>
<th>send RESIGN/rsp-req</th>
<th>send RESIGN/no-rsp-req</th>
<th>send PREPARED</th>
<th>send PREPARED/cancel</th>
<th>send CONFIRMED/auto</th>
<th>send CONFIRMED/response</th>
<th>send CANCELLED</th>
<th>send HAZARD</th>
<th>send INF_STATE/active/y</th>
<th>send INF_STATE/active</th>
<th>send INF_STATE/unknown</th>
</tr>
</thead>
<tbody>
<tr>
<td>i1</td>
<td>a1</td>
<td>b1</td>
<td>c1</td>
<td>d1</td>
<td>e1</td>
<td>e2</td>
<td>f1</td>
<td>f2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send ENROL/rsp-req</td>
<td>send ENROL/no-rsp-req</td>
<td>send RESIGN/rsp-req</td>
<td>send RESIGN/no-rsp-req</td>
<td>send PREPARED</td>
<td>send PREPARED/cancel</td>
<td>send CONFIRMED/auto</td>
<td>send CONFIRMED/response</td>
<td>send CANCELLED</td>
<td>send HAZARD</td>
<td>send INF_STATE/active/y</td>
<td>send INF_STATE/active</td>
<td>send INF_STATE/unknown</td>
</tr>
<tr>
<td>receive ENROLLED</td>
<td>receive RESIGNED</td>
<td>receive PREPARE</td>
<td>receive CONFIRM_ONE_PHASE</td>
<td>receive CONFIRM</td>
<td>receive CANCEL</td>
<td>receive CONTRADICTION</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive SUP_STATE/active/y</td>
<td>receive SUP_STATE/active</td>
<td>receive SUP_STATE/prepared-rcvd/y</td>
<td>receive SUP_STATE/prepared-rcvd</td>
<td>receive SUP_STATE/unknown</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>decide to resign</td>
<td>decide to be prepared</td>
<td>decide to be prepared/cancel</td>
<td>decide to confirm autonomously</td>
<td>decide to cancel autonomously</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>apply ordered confirmation</td>
<td>remove persistent information</td>
<td>detect problem</td>
<td>detect and record problem</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>disruption I</td>
<td>disruption II</td>
<td>disruption III</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>i1</th>
<th>a1</th>
<th>b1</th>
<th>c1</th>
<th>d1</th>
<th>e1</th>
<th>e2</th>
<th>f1</th>
<th>f2</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>a1</td>
<td>b1</td>
<td>b1</td>
<td>c1</td>
<td>z</td>
<td>e1</td>
<td>e2</td>
<td>f1</td>
</tr>
<tr>
<td>b1</td>
<td>b1</td>
<td>c1</td>
<td>c1</td>
<td>d1</td>
<td>z</td>
<td>e1</td>
<td>e2</td>
<td>f1</td>
</tr>
<tr>
<td>d1</td>
<td>d1</td>
<td>c1</td>
<td>c1</td>
<td>d1</td>
<td>e1</td>
<td>e2</td>
<td>f1</td>
<td>f2</td>
</tr>
<tr>
<td>n1</td>
<td>n1</td>
<td>z</td>
<td>n1</td>
<td>g1</td>
<td>g2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>z</td>
<td>z</td>
<td>z</td>
<td>x1</td>
<td>x2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c1</td>
<td>c1</td>
<td>c1</td>
<td>c1</td>
<td>e1</td>
<td>e1</td>
<td>e1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>e1</td>
<td>e1</td>
<td>e1</td>
<td>e1</td>
<td>e2</td>
<td>e2</td>
<td>e2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>h1</td>
<td>j1</td>
<td>z1</td>
<td>m1</td>
<td>m1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p1</td>
<td>p1</td>
<td>p1</td>
<td>p1</td>
<td>p2</td>
<td>p2</td>
<td>p2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>z</td>
<td>z</td>
<td>z</td>
<td>z</td>
<td>e1</td>
<td>e2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 1211: Inferior state table – cancellation and contradiction

<table>
<thead>
<tr>
<th>event</th>
<th>g1</th>
<th>g2</th>
<th>h1</th>
<th>h2</th>
<th>j1</th>
<th>j2</th>
<th>k1</th>
<th>k2</th>
<th>l1</th>
<th>l2</th>
</tr>
</thead>
<tbody>
<tr>
<td>send ENROL/rsp-req</td>
<td>g1</td>
<td>g2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send ENROL/no-rsp-req</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send RESIGN/rsp-req</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send RESIGN/no-rsp-req</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send PREPARED</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>h1</td>
<td></td>
</tr>
<tr>
<td>send PREPARED/cancel</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>l1</td>
<td></td>
</tr>
<tr>
<td>send CONFIRMED/auto</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>j1</td>
<td>k1</td>
</tr>
<tr>
<td>send CONFIRMED/response</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send CANCELLED</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send HAZARD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send INF_STATE/active/y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send INF_STATE/active</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send INF_STATE/unknown</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive ENROLLED</td>
<td></td>
<td></td>
<td>h1</td>
<td></td>
<td>j1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive RESIGNED</td>
<td></td>
<td></td>
<td>h1</td>
<td></td>
<td>j1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive PREPARE</td>
<td></td>
<td></td>
<td>h1</td>
<td></td>
<td>j1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive CONFIRM_ONE_PHASE</td>
<td></td>
<td></td>
<td>s3</td>
<td></td>
<td>s4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive CONFIRM</td>
<td></td>
<td></td>
<td>h2</td>
<td>h2</td>
<td>k1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive CANCEL</td>
<td></td>
<td></td>
<td>g1</td>
<td>g2</td>
<td>l1</td>
<td>j2</td>
<td>j2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive CONTRADICTION</td>
<td></td>
<td></td>
<td>l1</td>
<td>l2</td>
<td>k2</td>
<td>k2</td>
<td>k2</td>
<td>l2</td>
<td>l2</td>
<td>l2</td>
</tr>
<tr>
<td>receive SUP_STATE/active/y</td>
<td></td>
<td></td>
<td>h1</td>
<td></td>
<td>j1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive SUP_STATE/active</td>
<td></td>
<td></td>
<td>h1</td>
<td></td>
<td>j1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive SUP_STATE/prepared-rcvd/y</td>
<td></td>
<td></td>
<td>h1</td>
<td></td>
<td>j1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive SUP_STATE/prepared-rcvd</td>
<td></td>
<td></td>
<td>h1</td>
<td></td>
<td>j1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive SUP_STATE/unknown</td>
<td></td>
<td></td>
<td>x1</td>
<td>x2</td>
<td>l1</td>
<td>j2</td>
<td>j2</td>
<td>k2</td>
<td>k2</td>
<td>l1</td>
</tr>
<tr>
<td>decide to resign</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>decide to be prepared</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>decide to be prepared/cancel</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>decide to confirm autonomously</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>decide to cancel autonomously</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>apply ordered confirmation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>remove persistent information</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>detect problem</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>detect and record problem</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>disruption I</td>
<td></td>
<td></td>
<td>e1</td>
<td>e2</td>
<td>h1</td>
<td>j1</td>
<td>j1</td>
<td>k1</td>
<td>h1</td>
<td></td>
</tr>
<tr>
<td>disruption II</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>disruption III</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Action</td>
<td>m1</td>
<td>n1</td>
<td>p1</td>
<td>p2</td>
<td>q1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>----</td>
<td>----</td>
<td>-----</td>
<td>-----</td>
<td>----</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send ENROL/rsp-req</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send ENROL/no-rsp-req</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send RESIGN/rsp-req</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send RESIGN/no-rsp-req</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send PREPARED</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send PREPARED/cancel</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send CONFIRMED/auto</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send CONFIRMED/response</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send CANCELLED</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send HAZARD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send INF_STATE/active/y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send INF_STATE/active</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send INF_STATE/unknown</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive ENROLLED</td>
<td></td>
<td></td>
<td>p1</td>
<td>p2</td>
<td>q1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive RESIGNED</td>
<td></td>
<td></td>
<td>p1</td>
<td>p2</td>
<td>q1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive PREPARE</td>
<td></td>
<td></td>
<td>p1</td>
<td>p2</td>
<td>q1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive CONFIRM_ONE_PHASE</td>
<td></td>
<td></td>
<td></td>
<td>s5</td>
<td>s6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive CONFIRM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>p2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive CANCEL</td>
<td></td>
<td></td>
<td></td>
<td>n1</td>
<td>p1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive CONTRADICTION</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive SUP_STATE/active/y</td>
<td></td>
<td></td>
<td>p1</td>
<td>p2</td>
<td>q1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive SUP_STATE/active</td>
<td></td>
<td></td>
<td>p1</td>
<td>p2</td>
<td>q1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive SUP_STATE/prepared-rcvd/y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>p2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive SUP_STATE/prepared-rcvd</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>p2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive SUP_STATE/unknown</td>
<td></td>
<td></td>
<td></td>
<td>z</td>
<td>p1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>decide to resign</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>decide to be prepared</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>decide to be prepared/cancel</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>decide to confirm autonomously</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>decide to cancel autonomously</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>apply ordered confirmation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>remove persistent information</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>detect problem</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>detect and record problem</td>
<td></td>
<td></td>
<td></td>
<td>q1</td>
<td>q1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>disruption I</td>
<td></td>
<td></td>
<td>z</td>
<td></td>
<td>z</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>disruption II</td>
<td></td>
<td></td>
<td></td>
<td>d1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>disruption III</td>
<td></td>
<td></td>
<td></td>
<td>b1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 14.3: Inferior state table – request confirm states

<table>
<thead>
<tr>
<th>s1</th>
<th>s2</th>
<th>s3</th>
<th>s4</th>
<th>s5</th>
<th>s6</th>
</tr>
</thead>
<tbody>
<tr>
<td>send ENROL/rsp-req</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send ENROL/no-rsp-req</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send RESIGN/rsp-req</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send RESIGN/no-rsp-req</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send PREPARED</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send PREPARED/cancel</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send CONFIRMED/auto</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send CONFIRMED/response</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send CANCELLED</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>z</td>
</tr>
<tr>
<td>send HAZARD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>z</td>
</tr>
<tr>
<td>send INF_STATE/active/y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send INF_STATE/active</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send INF_STATE/unknown</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive ENROLLED</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive RESIGNED</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive PREPARE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive CONFIRM_ONE_PHASE</td>
<td>s1</td>
<td>s2</td>
<td>s3</td>
<td>s4</td>
<td>s5</td>
</tr>
<tr>
<td>receive CONFIRM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive CANCEL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive CONTRADICTION</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive SUP_STATE/active/y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive SUP_STATE/active</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive SUP_STATE/prepared-rcvd/y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive SUP_STATE/prepared-rcvd</td>
<td>x1</td>
<td>z</td>
<td>z</td>
<td>z</td>
<td>z</td>
</tr>
<tr>
<td>receive SUP_STATE/unknown</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>decide to resign</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>decide to be prepared</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>decide to be prepared/cancel</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>decide to confirm autonomously</td>
<td>s3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>decide to cancel autonomously</td>
<td>s4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>apply ordered confirmation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>remove persistent information</td>
<td>s2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>detect problem</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>detect and record problem</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>disruption I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>disruption II</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>disruption III</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 15.4: Inferior state table – completed states (including presume-abort and queried)

<table>
<thead>
<tr>
<th>Event</th>
<th>x1</th>
<th>x2</th>
<th>y1</th>
<th>y2</th>
<th>z</th>
<th>z1</th>
</tr>
</thead>
<tbody>
<tr>
<td>send ENROL/rsp-req</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send ENROL/no-rsp-req</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send RESIGN/rsp-req</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send RESIGN/no-rsp-req</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send PREPARED</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send PREPARED/cancel</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send CONFIRMED/auto</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send CONFIRMED/response</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send CANCELLED</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send HAZARD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send INF_STATE/active/y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send INF_STATE/active</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>send INF_STATE/unknown</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive ENROLLED</td>
<td>y1</td>
<td>y2</td>
<td>z</td>
<td></td>
<td>z1</td>
<td></td>
</tr>
<tr>
<td>receive RESIGNED</td>
<td>y1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive PREPARE</td>
<td>y1</td>
<td>y2</td>
<td>y1</td>
<td>z1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive CONFIRM_ONE_PHASE</td>
<td>y1</td>
<td>y2</td>
<td>y1</td>
<td>y1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive CONFIRM</td>
<td>y2</td>
<td>m1</td>
<td>y2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive CANCEL</td>
<td>y1</td>
<td>z</td>
<td>y1</td>
<td>y1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive CONTRACTION</td>
<td>z</td>
<td>z</td>
<td>z</td>
<td>z</td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive SUP_STATE/active/y</td>
<td>y1</td>
<td>y2</td>
<td>y1</td>
<td>y2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive SUP_STATE/active</td>
<td>y1</td>
<td>y2</td>
<td>z</td>
<td>z1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive SUP_STATE/prepared-rcvd/y</td>
<td>y2</td>
<td></td>
<td>y2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive SUP_STATE/prepared-rcvd</td>
<td>y2</td>
<td></td>
<td>y2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>receive SUP_STATE/unknown</td>
<td>x1</td>
<td>x2</td>
<td>y1</td>
<td>y2</td>
<td>z</td>
<td>z1</td>
</tr>
<tr>
<td>decide to resign</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>decide to be prepared</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>decide to be prepared/cancel</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>decide to confirm autonomously</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>decide to cancel autonomously</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>apply ordered confirmation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>remove persistent information</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>detect problem</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>detect and record problem</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Disruption</th>
<th>e1</th>
<th>e2</th>
</tr>
</thead>
<tbody>
<tr>
<td>disruption I</td>
<td></td>
<td></td>
</tr>
<tr>
<td>disruption II</td>
<td></td>
<td></td>
</tr>
<tr>
<td>disruption III</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Failure Recovery

Types of failure

BTP is designed to ensure the delivery of a consistent decision for a business transaction to the parties involved, even in the event of failure. Failures can be classified as:

Communication failure: messages between BTP actors are lost and not delivered. BTP assumes the carrier protocol ensures that messages are either delivered correctly (without corruption) or are lost, but does not assume that all losses are reported or that messages sent separately are delivered in the order of sending.

Node failure (system failure, site failure): a machine hosting one or more BTP actors stops processing and all its volatile data is lost. BTP assumes a site fails by stopping—it either operates correctly or not at all, it never operates incorrectly.

Communication failure may become known to a BTP implementation by an indication from the lower layers or may be inferred (or suspected) by the expiry of a timeout. Recovery from a communication failure requires only that the two actors can again send messages to each other and continue or complete the progress of the business transaction. In the state tables for the Superior:Inferior relationship, each side is either waiting to make a decision or can send a message. For some states, the message to be sent is a repetition of a regular message; for other states, the INFERIOR_STATE or SUPERIOR_STATE message can be sent, requesting a response. Thus, following a communication failure, either side can prompt the other to re-establish the relationship. Receiving one of the *_STATE messages asking for a response does not require an immediate response—especially if an implementation is waiting to determine a decision (perhaps because it is itself waiting for a decision from elsewhere), an implementation may choose not to reply until it wishes too.

A node failure is distinguished from communication failure because there is loss of volatile state. To ensure consistent application of the decision of a business transaction, BTP requires that some state information will be persisted despite node failure. Exactly what real events correspond to node failure but leave the persistent information undamaged is a matter for implementation choice, depending on application requirements; however, for most application uses, power failure should be survivable (an exception would be if the data manipulated by the associated operations was volatile). There will always be some level of event sufficiently catastrophic to lose persistent information and the ability to recover—destruction of the computer or bankruptcy of the organisation, for example.

Recovery from node failure involves recreating the endpoint in a node that has access to the persistent information for incomplete transactions. This may be a recreation of the original node (including the ability to perform application work) using the same addresses; or there may be a distinct recovery entity, which can access the persistent data, but has a different address; other implementation approaches are possible. Restoration of the endpoint from persistent information will often result in a partial loss of state, relative to the volatile state reached before the failure. This is modelled in the state tables by the “disruption” events.
After recovery from node failure, the implementation behaves much as if a communication failure had occurred.

**Persistent information**

BTP requires that some decision events are persisted—that information recording an Inferior’s decision to be prepared, a Superior’s decision to confirm and an Inferior’s autonomous decision survive failure. Making the first two decisions persistent ensures that a consistent decision can be reached for the business transaction and that it is delivered to all involved nodes. Requiring an Inferior’s autonomous decision to be persistent allows BTP to ensure that, if this decision is contradictory (i.e. opposite to the decision at the Superior), the contradiction will be reported to the Superior, despite failures.

BTP also permits, but does not require, recovery of the Superior:Inferior relationship in the active state (unlike many transaction protocols, where a communication or endpoint failure in active state would invariably cause rollback of the transaction). Recovery in the active state may require that the application exchange is resynchronised as well—BTP does not directly support this, but does allow continuation of the business transaction as such. In the state tables, from some states, there are several levels of disruption, distinguished by which state the implementation transits to—this represents the survival of different extents of state information over failure and recovery. The different levels of disruption describe legitimate states for the endpoint to be in after it has recovered—they do not require that all implementations are able to exhibit the appropriate partial loss of state information.

The absence of a destination state for the disruption events means that such a transition is not legitimate—thus, for example, an Inferior that has decided to be prepared will always recover to the same state, by virtue of the information persisted in the “decide to be prepared” event.

Apart from the (optional) recovery in active state, BTP follows the well-known presume-abort model—it is only required that information be persisted when decisions are made (and not, e.g. on enrolment). This means that on recovery, one side may have persistent information but the other does not. This occurs when an Inferior has decided to be prepared but the Superior never confirmed (so the decision is “presumed” to be cancel), or because the Superior did confirm, and the Inferior applied the confirm, removed its persistent information but the acknowledgement (CONFIRMED) was never received by the Superior (or, at least, it still had the persistent information when the failure occurred).

The BTP recovery mechanisms require that information is persisted by the BTP actors that perform the Superior and Inferior roles. To ensure consistent application of the outcome, despite failures, the Inferior must persist some state information at the point of becoming prepared, and the Superior at the point of making a confirm decision. If the Superior is a Sub-coordinator or Sub-composer, it must persist information when, as an Inferior it becomes prepared. The minimum information to be persisted is the identifiers and addresses of the peer Inferiors and Supeior—the fact of the persistence being itself an indication of the preparedness or confirm decision. However, BTP allows recovery of a Superior:Inferior relationship to occur in other cases—during the active phase, and before a confirm decision has been made. Thus, in general, the BTP actors will need to persist the current state of the relationships.
Since BTP messages may carry application-specified qualifiers, which may need to be re-sent in the case of failure (because the first attempt got lost). BTP actors should be prepared to persist such qualifiers as well.

A Participant will normally also need to persist some information concerning the application work whose final or counter effect it is responsible for. The nature of this information is not considered further in this specification.

Information to be persisted for an Inferior’s “decision to be prepared” must be sufficient to re-establish communication with the Superior, to apply a confirm decision and to apply a cancel decision. It will thus need to include:

- Inferior identity (this may be an index used to locate the information)
- “superior-address” (as on CONTEXT as updated by REDIRECT)
- “superior-identifier” (as on CONTEXT)
- “default-is-cancel” value (as on PREPARED)

The information needed to apply confirm/cancel decisions will depend on the application and the associated operations. It may also normally be necessary to persist any qualifiers that were sent with the PREPARED message or application messages sent with the PREPARED, since the PREPARED message will be repeated if a failure occurs.

A Superior must record corresponding information to allow it to re-establish communication with the Inferior. Thus, for each Inferior:

- “inferior-address” (as on ENROL as updated by REDIRECT)
- “inferior-identifier” (as on ENROL)

In order to recover their own function, both Superior and Inferior will need to persist their own Identifier (“superior-identifier” and “inferior-identifier”) and, depending on the implementation, may need to persist their original “superior-address” or “inferior-address”.

A Superior that is the Decider for the business transaction need only persist this information if it makes a decision to confirm (and this Inferior is in the confirm set, for a Cohesion). A Superior that is also an Inferior to some other entity (i.e. it is an intermediate in a tree, as atom in a cohesion, sub-coordinator or sub-composer) must persist this information as Superior (to this Inferior) as part of the persistent information of its decision to be prepared (as an Inferior). For such an entity, the “decision to confirm” as Superior is made when (and if) CONFIRM is received from its Superior or it makes an autonomous decision to confirm. If CONFIRM is received, the persistent information may be changed to show the confirm decision, but alternatively, the receipt of the CONFIRM can be treated as the decision itself. If the persistent information is left unchanged and there is a node failure, on recovery the entity (as an Inferior) will be in a prepared state, and will rediscover the confirm decision (using the recovery exchanges to its Superior) before propagating it to its Inferior(s).

After failure, an implementation may not be able to restore an endpoint to the appropriate state immediately—in particular, the necessary persistent information may be inaccessible, although the implementation can respond to received BTP messages. In such a case, a
Superior may reply to any BTP message except INFERIOR_STATE/* (i.e. with a “response-requested” value “false”) with SUPERIOR_STATE/inaccessible and an Inferior to any BTP message except SUPERIOR_STATE/* with “INFERIOR_STATE/inaccessible. Receipt of the *_STATE/inaccessible messages has no effect on the endpoint state.

Redirection

As described above, BTP uses the presume-abort model for recovery. A corollary of this is that there are cases where one side will attempt to re-establish communication when there is no persistent information for the relationship at the far-end. In such cases, it is important the side that is attempting recovery can distinguish between unsuccessful attempts to connect to the holder of the persistent information and when the information no longer exists. If the peer information does not exist, this side can draw conclusions and complete appropriately; if they merely fail to get through they are stuck in attempting recovery.

Two mechanisms are provided to make it possible that even when one side of a Superior:Inferior relationship has completed, that a message can eventually get through to something that can definitively report the status, distinguishing this case from a temporary inability to access the state of a continuing transaction element. The mechanisms are:

- Address fields which provide a “callback address” can be a set of addresses, which are alternatives one of which is chosen as the “target-address” for the future message. If the sender of that message finds the address does not work, it can try a different alternative.
- The REDIRECT message can be used to inform the peer that an address previously given is no longer valid and to supply a replacement address (or set of addresses). REDIRECT can be issued either as a response to receipt of a message or spontaneously.

The two mechanisms can be used in combination, with one or more of the original set of addresses just being a redirector, which does not itself ever have direct access to the state information for the transaction, but will respond to any message with an appropriate REDIRECT.

An alternative implementation approach is to have a single addressable entity that uses the same address for all transactions, distinguishing them by identifier, and which always recovers to use the same address. Such an implementation would not need to supply “backup” addresses (and would only use REDIRECT if it was being permanently migrated).

Terminator:Decider failures

BTP does not provide facilities or impose requirements on the recovery of Terminator:Decider relationships, other than allowing messages to be repeated. A Terminator may survive failures (by retaining knowledge of the Decider’s address and identifier), but this is an implementation option. Although a Decider (if it decides to confirm) will persist information about the confirm decision, it is not required, after failure, to remain accessible.
using the inferior address it offered to the Terminator. Any such recovery is an implementation option.

A Decider’s address (as returned on BEGUN) may be a set of addresses, allowing a failed Decider to be recovered at a different address.

A Decider has no way of initiating a call to a Terminator to ensure that it is still active, and thus no way of detecting that a Terminator has failed. To avoid a Decider waiting for ever for a CONFIRM_TRANSACTION that will never arrive, the standard qualifier “Transaction timelimit” can be used (by the Initiator) to inform the Decider when it can assume the Terminator will not issue CONFIRM_TRANSACTION and so it (the Decider) should initiate cancellation.

XML representation of Message Set

This section describes the syntax for BTP messages in XML. These XML messages represent a midpoint between the abstract messages and what actually gets sent on the wire.

All BTP related URIs have been created using Oasis URI conventions as specified in RFC 3121.

The XML Namespace for the BTP messages is urn:oasis:names:tc:BTP:1.0:core.

In addition to an XML schema, this specification uses an informal syntax to describe the structure of the BTP messages. The syntax appears as an XML instance, but the values contain data types instead of values. The following symbols are appended to some of the XML constructs: ? (zero or one), * (zero or more), + (one or more.) The absence of one of these symbols corresponds to “one and only one.”

The Delivery parameters are shown in the XML with a darker background.

Addresses

As described in the “Abstract Message and Associated Contracts – Addresses” section, a BTP address comprises three parts, and for a “target-address” only the “additional information” field is inside the BTP messages. For all BTP messages whose abstract form includes a “target-address” parameter, the corresponding XML representation includes a “target-additional-information” element. This element may be omitted if it would be empty.

For other addresses, all three fields are represent, as in:

```xml
<btp:some-address>
  <btp:binding-name>...carrier binding URI...</btp:binding-name>
  <btp:binding-address>...carrier specific address...</btp:binding-address>
</btp:some-address>
```
A "published" address can be a set of <some-address>, which are alternatives which can be
chosen by the peer (sender.) Multiple addresses are used in two cases: different bindings to
same endpoint, or backup endpoints. In the former, the receiver of the message has the choice
of which address to use (depending on which binding is preferable.) In the case where
multiple addresses are used for redundancy, a priority attribute can be specified to help the
receiver choose among the addresses- the address with the highest priority should be used.
other things being equal. The priority is used as a hint and does not enforce any behaviour in
the receiver of the message. Default priority is a value of 1.

Qualifiers
The “Qualifier name” is used as the element name, within the namespace of the “Qualifier
group”.

Examples:

```xml
<btpq:inferior-timeout
   xmlns:btpq="urn:oasis:names:tc:BTP:1.0:qualifiers"
   xmlns:btp="urn:oasis:names:tc:BTP:1.0:core"
   btp:must-be-understood="false"
   btp:to-be-propagated="false">1800</btpq:inferior-timeout>

<auth:username
   xmlns:auth="http://www.example.com/ns/auth"
   xmlns:btp="urn:oasis:names:tc:BTP:1.0:core"
   btp:must-be-understood="true"
   btp:to-be-propagated="true">jtauber</auth:username>
```

Attributes must-be-understood has default value “true” and to-be-propagated has default
value “false”.

Identifiers

Identifiers shall be URIs

```
Note – Identifiers need to be globally unambiguous. Apart from their
generation, the only operation the BTP implementations have to perform on
identifiers is to match them.
```

Message References

Each BTP message has an optional id attribute to give it a unique identifier. An application
can make use of those identifiers, but no processing is enforced.
Messages

CONTEXT

```xml
<btp:context id?>
  <btp:superior-address> +
      ... address...
  </btp:superior-address>
  <btp:superior-identifier>... URI...</btp:superior-identifier>
  <btp:superior-type>cohesion|atom</btp:superior-type>
  <btp:qualifiers> ?
      ... qualifiers...
  </btp:qualifiers>
  <btp:reply-address> ?
      ... address...
  </btp:reply-address>
</btp:context>
```

CONTEXT_REPLY

```xml
<btp:context-reply id?>
  <btp:superior-identifier>...URI...</btp:superior-identifier>
  <btp:completion-status>completed|incomplete|related|repudiated</btp:completion-status>
  <btp:qualifiers> ?
      ... qualifiers...
  </btp:qualifiers>
  <btp:target-additional-information> ?
      ... additional address information...
  </btp:target-additional-information>
</btp:context-reply>
```

REQUEST_STATUS

```xml
<btp:request-status id?>
  <btp:target-identifier>...URI...</btp:target-identifier>
  <btp:qualifiers> ?
      ... qualifiers...
  </btp:qualifiers>
  <btp:target-additional-information> ?
      ... additional address information...
  </btp:target-additional-information>
  <btp:reply-address> ?
      ... address...
  </btp:reply-address>
</btp:request-status>
```

STATUS
<btp:status id?>
  <btp:responders-identifier>...URI...</btp:responders-identifier>
  <btp:status-value>created|enrolling|active|resigning|
  resigned|preparing|prepared|
  confirming|confirmed|cancelling|cancelled|
  cancel-contradiction|confirm-contradiction|
  hazard|contradicted|unknown|inaccessible</btp:status-value>
  <btp:qualifiers> ?
  ...qualifiers...
  </btp:qualifiers>
</btp:status>

<btp:fault id?>
  <btp:superior-identifier>...URI...</btp:superior-identifier> ?
  <btp:inferior-identifier>...URI...</btp:inferior-identifier> ?
  <btp:fault-type>...fault type name...</btp:fault-type>
  <btp:fault-data>...fault data...</btp:fault-data> ?
  <btp:fault-text>...string data ...</btp:fault-data> ?
  <btp:qualifiers> ?
  ...qualifiers...
  </btp:qualifiers>
</btp:fault>

The following fault type names are represented by simple strings, corresponding to the entries defined in the abstract message set:

- communication-failure
- duplicate-inferior
- general
- invalid-decider
- invalid-inferior
- invalid-superior
- status-refused
- invalid-terminator
- unknown-parameter
- unknown-transaction
- unsupported-qualifier
- wrong-state
- redirect
Revisions of this specification may add other fault type names, which shall be simple strings of letters, numbers and hyphens. If other specifications define fault type names to be used with BTP, the names shall be URIs.

Fault data can take on various forms:

Identifier:

```xml
<btp:fault-data>...URI...</btp:fault-data>
```

Inferior Identity:

```xml
<btp:fault-data>
  <btp:inferior-address> +
   ...address...
  </btp:inferior-address>
  <btp:inferior-identifier>...URI...</btp:inferior-identifier>
</btp:fault-data>
```

ENROL

```xml
<btp:enrol id?>
  <btp:superior-identifier>...URI...</btp:superior-identifier>
  <btp:response-requested>true|false</btp:response-requested>
  <btp:inferior-address> +
   ...address...
  </btp:inferior-address>
  <btp:inferior-identifier>...URI...</btp:inferior-identifier>
  <btp:qualifiers> ?
   ...qualifiers...
  </btp:qualifiers>
  <btp:target-additional-information> ?
   ...additional address information...
  </btp:target-additional-information>
  <btp:reply-address> ?
   ...address...
  </btp:reply-address>
</btp:enrol>
```

ENROLLED

```xml
<btp:enrolled id?>
  <btp:sender-address> ?
   ...address...
  </btp:sender-address>
  <btp:inferior-identifier>...URI...</btp:inferior-identifier>
  <btp:qualifiers> ?
```
...qualifiers...
</btp:qualifiers>

```xml
<btp:target-additional-information> ?
...additional address information...
</btp:target-additional-information>
</btp:enrolled>
```

### RESIGN

```xml
<btp:resign id?>
  <btp:sender-address> </btp:sender-address>
  <btp:superior-identifier>...URI...</btp:superior-identifier>
  <btp:inferior-identifier>...URI...</btp:inferior-identifier>
  <btp:response-requested>true|false</btp:response-requested>
  <btp:qualifiers> ?
...qualifiers...
</btp:qualifiers>
  <btp:target-additional-information> ?
...additional address information...
</btp:target-additional-information>
</btp:resign>
```

### RESIGNED

```xml
<btp:resigned id?>
  <btp:sender-address> </btp:sender-address>
  <btp:superior-identifier>...URI...</btp:superior-identifier>
  <btp:inferior-identifier>...URI...</btp:inferior-identifier>
  <btp:response-requested>true|false</btp:response-requested>
  <btp:qualifiers> ?
...qualifiers...
</btp:qualifiers>
  <btp:target-additional-information> ?
...additional address information...
</btp:target-additional-information>
</btp:resigned>
```

### PREPARE

```xml
<btp:prepare id?>
  <btp:sender-address> </btp:sender-address>
  <btp:superior-identifier>...URI...</btp:superior-identifier>
  <btp:inferior-identifier>...URI...</btp:inferior-identifier>
  <btp:response-requested>true|false</btp:response-requested>
  <btp:qualifiers> ?
...qualifiers...
</btp:qualifiers>
  <btp:target-additional-information> ?
```
...additional address information...
</btp:target-additional-information>
<btp:sender-address> ?
...address...
</btp:sender-address>
</btp:prepare>

**PREPARED**

```xml
<btp:prepared id?>
  <btp:superior-identifier>...URI...</btp:superior-identifier>
  <btp:inferior-identifier>...URI...</btp:inferior-identifier>
  <btp:default-is-cancel>true|false</btp:default-is-cancel>
  <btp:qualifiers> ?
    ...qualifiers...
  </btp:qualifiers>
  <btp:target-additional-information> ?
    ...additional address information...
  </btp:target-additional-information>
  <btp:sender-address> ?
    ...address...
  </btp:sender-address>
</btp:prepared>
```

**CONFIRM**

```xml
<btp:confirm id?>
  <btp:inferior-identifier>...URI...</btp:inferior-identifier>
  <btp:qualifiers> ?
    ...qualifiers...
  </btp:qualifiers>
  <btp:target-additional-information> ?
    ...additional address information...
  </btp:target-additional-information>
  <btp:sender-address> ?
    ...address...
  </btp:sender-address>
</btp:confirm>
```

**CONFIRMED**

```xml
<btp:confirmed id?>
  <btp:superior-identifier>...URI...</btp:superior-identifier>
  <btp:inferior-identifier>...URI...</btp:inferior-identifier>
  <btp:confirmed-received>true|false</btp:confirmed-received>
  <btp:qualifiers> ?
    ...qualifiers...
  </btp:qualifiers>
```
CANCEL

```xml
<btp:cancel id?>
  <btp:inferior-identifier>...URI...</btp:inferior-identifier>
  <btp:reply-address> ?
    ...address...
  </btp:reply-address>
  <btp:qualifiers> ?
    ...qualifiers...
  </btp:qualifiers>
  <btp:target-additional-information> ?
    ...additional address information...
  </btp:target-additional-information>
  <btp:sender-address> ?
    ...address...
  </btp:sender-address>
</btp:cancel>
```

CANCELLED

```xml
<btp: cancelled id?>
  <btp:superior-identifier>...URI...</btp:superior-identifier>
  <btp:inferior-identifier>...URI...</btp:inferior-identifier> ?
  <btp:qualifiers> ?
    ...qualifiers...
  </btp:qualifiers>
  <btp:target-additional-information> ?
    ...additional address information...
  </btp:target-additional-information>
  <btp:sender-address> ?
    ...address...
  </btp:sender-address>
</btp: cancelled>
```

CONFIRM_ONE_PHASE

```xml
<btp:confirm-one-phase id?>
```
HAZARD

```
      <btp:inferior-identifier>...URI...</btp:inferior-identifier>
      <btp:report-hazard>true|false</btp:report-hazard>
      <btp:qualifiers> ?
      ...qualifiers...
      </btp:qualifiers>
      <btp:target-additional-information> ?
      ...additional address information...
      </btp:target-additional-information>
      <btp:sender-address> ?
      ...address...
      </btp:sender-address>
      </btp:confirm-one-phase>

      <btp:hazard id?>
        <btp:superior-identifier>...URI...</btp:superior-identifier>
        <btp:inferior-identifier>...URI...</btp:inferior-identifier>
        <btp:level>mixed|possible</btp:level>
        <btp:qualifiers> ?
        ...qualifiers...
        </btp:qualifiers>
        <btp:target-additional-information> ?
        ...additional address information...
        </btp:target-additional-information>
        <btp:sender-address> ?
        ...address...
        </btp:sender-address>
      </btp:hazard>

CONTRADICTION

```

```
      <btp:contradiction id?>
        <btp:inferior-identifier>...URI...</btp:inferior-identifier>
        <btp:qualifiers> ?
        ...qualifiers...
        </btp:qualifiers>
        <btp:target-additional-information> ?
        ...additional address information...
        </btp:target-additional-information>
        <btp:sender-address> ?
        ...address...
        </btp:sender-address>
      </btp:contradiction>

SUPERIOR_STATE

```
<btp:superior-state id?>
  <btp:inferior-identifier>...URI...</btp:inferior-identifier>
  <btp:status>active|prepared-received|inaccessible|unknown</btp:status>
  <btp:response-requested>true|false</btp:response-requested>
  <btp:qualifiers> ?
    ...qualifiers...
  </btp:qualifiers>
  <btp:target-additional-information> ?
    ...additional address information...
  </btp:target-additional-information>
  <btp:sender-address> ?
    ...address...
  </btp:sender-address>
</btp:superior-state>

INFERIOR_STATE

<btp:inferior-state id?>
  <btp:superior-identifier>...URI...</btp:superior-identifier>
  <btp:inferior-identifier>...URI...</btp:inferior-identifier>
  <btp:status>active|inaccessible|unknown</btp:status>
  <btp:response-requested>true|false</btp:response-requested>
  <btp:qualifiers> ?
    ...qualifiers...
  </btp:qualifiers>
  <btp:target-additional-information> ?
    ...additional address information...
  </btp:target-additional-information>
  <btp:sender-address> ?
    ...address...
  </btp:sender-address>
</btp:inferior-state>

REDIRECT

<btp:redirect id?>
  <btp:superior-identifier>...URI...</btp:superior-identifier> ?
  <btp:inferior-identifier>...URI...</btp:inferior-identifier>
  <btp:old-address> +
    ...address...
  </btp:old-address>
  <btp:new-address> +
    ...address...
  </btp:new-address>
  <btp:qualifiers> ?
    ...qualifiers...
  </btp:qualifiers>
BEGIN

<BTP:begin id?>
  <btp:transaction-type>cohesion|atom</btp:transaction-type>
  <btp:qualifiers> ?
    ...qualifiers...
  </btp:qualifiers>
  <btp:target-additional-information> ?
    ...additional address information...
  </btp:target-additional-information>
  <btp:reply-address> ?
    ...address...
  </btp:reply-address>
</btp:begin>

BEGIN

<BTP:begun id?>
  <btp:decider-address> *
    ...address...
  </btp:decider-address>
  <btp:inferior-address> *
    ...address...
  </btp:inferior-address>
  <btp:transaction-identifier>...URI...</btp:transaction-identifier>
  <btp:qualifiers> ?
    ...qualifiers...
  </btp:qualifiers>
  <btp:target-additional-information> ?
    ...additional address information...
  </btp:target-additional-information>
</btp:begun>

PREPARE_INFERIORS

<BTP:prepare-inferiors id?>
  <btp:transaction-identifier>...URI...</btp:transaction-identifier>
  <btp:inferiors-list> ?
    <btp:inferior-handle>...URI...</btp:inferior-handle> +
  </btp:inferiors-list>
  <btp:qualifiers> ?
</btp:prepare-inferiors>
CONFIRM_TRANSACTION

```xml
<btp:confirm-transaction id?>
    <btp:transaction-identifier>...URI...</btp:transaction-identifier>
    <btp:inferiors-list>
        <btp:inferior-handle>...URI...</btp:inferior-handle>
        <btp:inferiors-list>
            <btp:report-hazard>true|false</btp:report-hazard>
            <btp:qualifiers>...
            </btp:qualifiers>
        </btp:inferiors-list>
    </btp:inferiors-list>
    <btp:target-additional-information>...
    </btp:target-additional-information>
    <btp:reply-address>...
</btp:confirm-transaction>
```

TRANSACTION_CONFIRMED

```xml
<btp:transaction-confirmed id?>
    <btp:transaction-identifier>...URI...</btp:transaction-identifier>
    <btp:qualifiers>...
    </btp:qualifiers>
    <btp:target-additional-information>...
    </btp:target-additional-information>
</btp:transaction-confirmed>
```

CANCEL_TRANSACTION

```xml
<btp:cancel-transaction id?>
```
<btp:transaction-identifier>...URI...</btp:transaction-identifier>
  <btp:report-hazard>true|false</btp:report-hazard>
  <btp:qualifiers> ?? ...qualifiers... </btp:qualifiers>
  <btp:target-additional-information> ?? ...additional address information... </btp:target-additional-information>
  <btp:reply-address> ?? ...address... </btp:reply-address>
</btp:cancel-transaction>

CANCEL_INFERIORS

<btp:cancel-inferiors id?>
  <btp:transaction-identifier>...URI...</btp:transaction-identifier> ??
  <btp:inferiors-list>
    <btp:inferior-handle>...URI...</btp:inferior-handle> +-
  </btp:inferiors-list>
  <btp:qualifiers> ?? ...qualifiers... </btp:qualifiers>
  <btp:target-additional-information> ?? ...additional address information... </btp:target-additional-information>
  <btp:reply-address> ?? ...address... </btp:reply-address>
</btp:cancel-inferiors>

TRANSACTION_CANCELLED

<btp:transaction-cancelled id?>
  <btp:transaction-identifier>...URI...</btp:transaction-identifier>
  <btp:qualifiers> ?? ...qualifiers... </btp:qualifiers>
  <btp:target-additional-information> ?? ...additional address information... </btp:target-additional-information>
</btp:transaction-cancelled>

REQUEST_INFERIOR_STATUSES
<btp:request-inferior-statuses id?>
  <btp:target-identifier>...URI...</btp:target-identifier>
  <btp:inferiors-list> ?
    <btp:inferior-handle>...URI...</btp:inferior-handle> +
  </btp:inferiors-list>
  <btp:qualifiers> ?
    ...qualifiers...
  </btp:qualifiers>
  <btp:target-additional-information> ?
    ...additional address information...
  </btp:target-additional-information>
  <btp:reply-address> ?
    ...address...
  </btp:reply-address>
</btp:request-inferior-statuses>

INFERIOR_STATUSES

<bp:request-inferior-statuses id?>
  <btp:responders-identifier>...URI...</btp:responders-identifier>
  <btp:status-list>
    <btp:status-item> +
      <btp:inferior-handle>...URI...</btp:inferior-handle>
      <btp:status>active|resigned|preparing|prepared|
      autonomously-confirmed|autonomously-cancelled|
      confirming|confirmed|cancelling|cancelled|
      cancel-contradiction|confirm-contradiction|
      hazard|invalid</btp:status>
      <btp:qualifiers> ?
        ...qualifiers...
      </btp:qualifiers>
    </btp:status-item>
  </btp:status-list>
  <btp:qualifiers> ?
    ...qualifiers...
  </btp:qualifiers>
  <btp:target-additional-information> ?
    ...additional address information...
  </btp:target-additional-information>
  <btp:reply-address> ?
    ...address...
  </btp:reply-address>
</btp:request-inferior-statuses>

Standard qualifiers
The informal syntax for these messages assumes the namespace prefix “btpq” is associated with the URI “urn:oasis:names:tc:BTP:1.0:qualifiers”.

Transaction timelimit

<btpq:transaction-timelimit>
  <btpq:timelimit>
</btpq:transaction-timelimit>

OASIS BTPDraft Specification 0.9.2.4, 3 April 2002
...time in seconds...
</btpq:timelimit>
</btpq:transaction-timelimit>

**Inferior timeout**

```xml
<btpq:inferior-timeout>
  <btpq:timeout>
    ...time in seconds...
  </btpq:timeout>
  <btpq:intended-decision>confirm|cancel</btpq:intended-decision>
</btpq:inferior-timeout>
```

**Minimum inferior timeout**

```xml
<btpq:minimum-inferior-timeout>
  <btpq:minimum-timeout>
    ...time in seconds...
  </btpq:minimum-timeout>
</btpq:minimum-inferior-timeout>
```

**Inferior name**

```xml
<btpq:inferior-name>
  <btpq:inferior-name>
    ...string...
  </btpq:inferior-name>
</btpq:inferior-name>
```

### Compounding of Messages

Relating BTP to one another, in a “group” is represented by containing them within the `btp:related-group` element, with the related messages as child elements. The processing for the group is defined in the section “Groups – combinations of related messages”. For example:

```xml
<btp:related-group>
  <btp:context-reply>
    ...<completion-status>related</completion-status> ...
  </btp:context-reply>
  <btp:enrol>...</btp:enrol>
  <btp:prepared>...</btp:prepared>
</btp:related-group>
```

If the rules for the group state that the “target-address” of the abstract message is omitted, the corresponding target-address-information element shall be absent in the message in the related-group. The carrier protocol binding specifies how a relation between application and BTP messages is represented.

Bundling (semantically insignificant combination) of BTP messages and related groups is indicated with the "btp:messages" element, with the bundled messages and related groups as child elements. For example (confirming one and cancelling another inferiors of a cohesion):

```xml
<btp:related-group>
  <btp:context-reply>
    ...<completion-status>related</completion-status> ...
  </btp:context-reply>
  <btp:enrol>...</btp:enrol>
  <btp:prepared>...</btp:prepared>
</btp:related-group>
```
<btp:messages>
  <btp:confirm>...</btp:confirm>
  <btp:cancel>...</btp:cancel>
</btp:messages>
XML Schemas

XML schema for BTP messages

<?xml version="1.0"?>
<schema
 xmlns="http://www.w3.org/2001/XMLSchema"
 targetNamespace="urn:oasis:names:tc:BTP:1.0:core"
 xmlns:btp="urn:oasis:names:tc:BTP:1.0:core"
 elementFormDefault="qualified">

<!-- Qualifiers -->
<complexType name="qualifier-type">
 <simpleContent>
  <extension base="string">
   <attribute name="must-be-understood" type="boolean"/>
   <attribute name="to-be-propagated" type="boolean"/>
  </extension>
 </simpleContent>
</complexType>

<element name="qualifier" type="btp:qualifier-type" abstract="true"/>
<element name="qualifiers">
 <complexType>
  <sequence>
   <element ref="btp:qualifier" maxOccurs="unbounded"/>
  </sequence>
 </complexType>
</element>

<!-- example qualifier:
<element name="some-qualifer" type="btp:qualifier-type" substitutionGroup="btp:qualifier"/>
-->

<!-- Message set data types -->
<simpleType name="identifier">
 <restriction base="anyURI" />
</simpleType>

<simpleType name="additional-information">
 <restriction base="string" />
</simpleType>

<complexType name="address">
<element name="binding-name" type="anyURI"/>
  <element name="binding-address" type="string"/>
  <element name="additional-information" type="btp:additional-information" minOccurs="0" />
</sequence>
</complexType>

<complexType name="superior-type">
  <restriction base="string">
    <enumeration value="cohesion"/>
    <enumeration value="atom"/>
  </restriction>
</complexType>

<complexType name="transaction-type">
  <restriction base="string">
    <enumeration value="cohesion"/>
    <enumeration value="atom"/>
  </restriction>
</complexType>

<!-- Compounding -->
<element name="messages">
  <complexType>
    <sequence>
      <element ref="btp:message" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
  </complexType>
</element>

<element name="related-group" substitutionGroup="btp:message">
  <complexType>
    <sequence>
      <element ref="btp:message" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
  </complexType>
</element>

<!-- Message set -->
<element name="message" abstract="true" />
<element name="context" substitutionGroup="btp:message">
  <complexType>
    <sequence>
    </sequence>
  </complexType>
</element>

<!-- Compounding -->
<element name="superior-address" type="btp:address"
maxOccurs="unbounded"/>
<element name="superior-identifier" type="btp:identifier"/>
<element name="superior-type" type="btp:superior-type"/>
<element ref="btp:qualifiers" minOccurs="0"/>
<element name="reply-address" type="btp:address"
minOccurs="0"/>
</sequence>
<attribute name="id" type="ID" use="optional"/>
</complexType>
</element>

<element name="context-reply" substitutionGroup="btp:message">
<complexType>
<sequence>
<element name="superior-identifier" type="btp:identifier"/>
<element name="completion-status">
<simpleType>
<restriction base="string">
<enumeration value="completed"/>
<enumeration value="incomplete"/>
<enumeration value="related"/>
<enumeration value="repudiated"/>
</restriction>
</simpleType>
</element>
<element ref="btp:qualifiers" minOccurs="0"/>
<element name="target-additional-information" type="btp:additional-information" minOccurs="0"/>
</sequence>
<attribute name="id" type="ID"/>
</complexType>
</element>

<element name="request-status" substitutionGroup="btp:message">
<complexType>
<sequence>
<element name="target-identifier" type="btp:identifier"/>
<element ref="btp:qualifiers" minOccurs="0"/>
<element name="target-additional-information" type="btp:additional-information" minOccurs="0"/>
<element name="reply-address" type="btp:address"
minOccurs="0"/>
</sequence>
<attribute name="id" type="ID"/>
</complexType>
</element>

<element name="status" substitutionGroup="btp:message">
<complexType>
<sequence>
<element name="target-identifier" type="btp:identifier"/>
<element ref="btp:qualifiers" minOccurs="0"/>
<element name="target-additional-information" type="btp:additional-information" minOccurs="0"/>
<element name="reply-address" type="btp:address"
minOccurs="0"/>
</sequence>
<attribute name="id" type="ID"/>
</complexType>
</element>

<element name="request-status" substitutionGroup="btp:message">
<complexType>
<sequence>
<element name="target-identifier" type="btp:identifier"/>
<element ref="btp:qualifiers" minOccurs="0"/>
<element name="target-additional-information" type="btp:additional-information" minOccurs="0"/>
<element name="reply-address" type="btp:address"
minOccurs="0"/>
</sequence>
<attribute name="id" type="ID"/>
</complexType>
</element>

<element name="status" substitutionGroup="btp:message">
<complexType>
<sequence>
<element name="target-identifier" type="btp:identifier"/>
<element ref="btp:qualifiers" minOccurs="0"/>
<element name="target-additional-information" type="btp:additional-information" minOccurs="0"/>
<element name="reply-address" type="btp:address"
minOccurs="0"/>
</sequence>
<attribute name="id" type="ID"/>
</complexType>
</element>

<element name="status" substitutionGroup="btp:message">
<complexType>
<sequence>
<element name="target-identifier" type="btp:identifier"/>
<element ref="btp:qualifiers" minOccurs="0"/>
<element name="target-additional-information" type="btp:additional-information" minOccurs="0"/>
<element name="reply-address" type="btp:address"
minOccurs="0"/>
</sequence>
<attribute name="id" type="ID"/>
</complexType>
</element>

<element name="status" substitutionGroup="btp:message">
<complexType>
<sequence>
<element name="target-identifier" type="btp:identifier"/>
<element ref="btp:qualifiers" minOccurs="0"/>
<element name="target-additional-information" type="btp:additional-information" minOccurs="0"/>
<element name="reply-address" type="btp:address"
minOccurs="0"/>
</sequence>
<attribute name="id" type="ID"/>
</complexType>
</element>

<element name="status" substitutionGroup="btp:message">
<complexType>
<sequence>
<element name="target-identifier" type="btp:identifier"/>
<element ref="btp:qualifiers" minOccurs="0"/>
<element name="target-additional-information" type="btp:additional-information" minOccurs="0"/>
<element name="reply-address" type="btp:address"
minOccurs="0"/>
</sequence>
<attribute name="id" type="ID"/>
</complexType>
</element>

<element name="status" substitutionGroup="btp:message">
<complexType>
<sequence>
<element name="target-identifier" type="btp:identifier"/>
<element ref="btp:qualifiers" minOccurs="0"/>
<element name="target-additional-information" type="btp:additional-information" minOccurs="0"/>
<element name="reply-address" type="btp:address"
minOccurs="0"/>
</sequence>
<attribute name="id" type="ID"/>
</complexType>
</element>

<element name="status" substitutionGroup="btp:message">
<complexType>
<sequence>
<element name="target-identifier" type="btp:identifier"/>
<element ref="btp:qualifiers" minOccurs="0"/>
<element name="target-additional-information" type="btp:additional-information" minOccurs="0"/>
<element name="reply-address" type="btp:address"
minOccurs="0"/>
</sequence>
<attribute name="id" type="ID"/>
</complexType>
</element>

<element name="status" substitutionGroup="btp:message">
<complexType>
<sequence>
<element name="target-identifier" type="btp:identifier"/>
<element ref="btp:qualifiers" minOccurs="0"/>
<element name="target-additional-information" type="btp:additional-information" minOccurs="0"/>
<element name="reply-address" type="btp:address"
minOccurs="0"/>
</sequence>
<attribute name="id" type="ID"/>
</complexType>
</element>

<element name="status" substitutionGroup="btp:message">
<complexType>
<sequence>
<element name="target-identifier" type="btp:identifier"/>
<element ref="btp:qualifiers" minOccurs="0"/>
<element name="target-additional-information" type="btp:additional-information" minOccurs="0"/>
<element name="reply-address" type="btp:address"
minOccurs="0"/>
</sequence>
<attribute name="id" type="ID"/>
</complexType>
</element>

<element name="status" substitutionGroup="btp:message">
<complexType>
<sequence>
<element name="target-identifier" type="btp:identifier"/>
<element ref="btp:qualifiers" minOccurs="0"/>
<element name="target-additional-information" type="btp:additional-information" minOccurs="0"/>
<element name="reply-address" type="btp:address"
minOccurs="0"/>
</sequence>
<attribute name="id" type="ID"/>
</complexType>
</element>

<element name="status" substitutionGroup="btp:message">
<complexType>
<sequence>
<element name="target-identifier" type="btp:identifier"/>
<element ref="btp:qualifiers" minOccurs="0"/>
<element name="target-additional-information" type="btp:additional-information" minOccurs="0"/>
<element name="reply-address" type="btp:address"
minOccurs="0"/>
</sequence>
<attribute name="id" type="ID"/>
</complexType>
</element>

<element name="status" substitutionGroup="btp:message">
<complexType>
<sequence>
<element name="target-identifier" type="btp:identifier"/>
<element ref="btp:qualifiers" minOccurs="0"/>
<element name="target-additional-information" type="btp:additional-information" minOccurs="0"/>
<element name="reply-address" type="btp:address"
minOccurs="0"/>
</sequence>
<attribute name="id" type="ID"/>
</complexType>
</element>

<element name="status" substitutionGroup="btp:message">
<complexType>
<sequence>
<element name="target-identifier" type="btp:identifier"/>
<element ref="btp:qualifiers" minOccurs="0"/>
<element name="target-additional-information" type="btp:additional-information" minOccurs="0"/>
<element name="reply-address" type="btp:address"
minOccurs="0"/>
</sequence>
<attribute name="id" type="ID"/>
<element name="responders-identifier" type="btp:identifier"/>
<element name="status-value">
  <simpleType>
    <restriction base="string">
      <enumeration value="created"/>
      <enumeration value="enrolling"/>
      <enumeration value="active"/>
      <enumeration value="resigning"/>
      <enumeration value="resigned"/>
      <enumeration value="preparing"/>
      <enumeration value="prepared"/>
      <enumeration value="confirming"/>
      <enumeration value="confirmed"/>
      <enumeration value="cancelling"/>
      <enumeration value="cancelled"/>
      <enumeration value="cancel-contradiction"/>
      <enumeration value="confirm-contradiction"/>
      <enumeration value="hazard"/>
      <enumeration value="contradicted"/>
      <enumeration value="unknown"/>
      <enumeration value="inaccessible"/>
    </restriction>
  </simpleType>
</element>
<element ref="btp:qualifiers" minOccurs="0"/>
<element name="target-additional-information" type="btp:additional-information" minOccurs="0"/>
</sequence>
<attribute name="id" type="ID"/>
</complexType>
<element name="fault" substitutionGroup="btp:message">
  <complexType>
    <sequence>
      <element name="superior-identifier" type="btp:identifier" minOccurs="0"/>
      <element name="inferior-identifier" type="btp:identifier" minOccurs="0"/>
      <element name="fault-type">
        <simpleType>
          <restriction base="string">
            <enumeration value="communication-failure"/>
            <enumeration value="duplicate-inferior"/>
            <enumeration value="general"/>
            <enumeration value="invalid-decider"/>
            <enumeration value="invalid-inferior"/>
            <enumeration value="invalid-superior"/>
            <enumeration value="status-refused"/>
            <enumeration value="invalid-terminator"/>
            <enumeration value="unknown-parameter"/>
          </restriction>
        </simpleType>
      </element>
    </sequence>
  </complexType>
</element>
<element name="enrol" substitutionGroup="btp:message">
  <complexType>
    <sequence>
      <element name="superior-identifier" type="btp:identifier" minOccurs="0" maxOccurs="unbounded"/>
      <element name="response-requested" type="boolean"/>
      <element name="reply-address" type="btp:address" minOccurs="0" maxOccurs="unbounded"/>
      <element name="inferior-address" type="btp:address" minOccurs="0" maxOccurs="unbounded"/>
      <element name="inferior-identifier" type="btp:identifier"/>
      <element ref="btp:qualifiers" minOccurs="0"/>
      <element name="target-additional-information" type="btp:additional-information" minOccurs="0"/>
    </sequence>
    <attribute name="id" type="ID"/>
  </complexType>
</element>

<element name="resign" substitutionGroup="btp:message">
  <complexType>
    <sequence>
      <element name="superior-identifier" type="btp:identifier"/>
      <element name="inferior-identifier" type="btp:identifier"/>
      <element name="response-requested" type="boolean"/>
      <element ref="btp:qualifiers" minOccurs="0"/>
      <element name="target-additional-information" type="btp:additional-information" minOccurs="0"/>
    </sequence>
    <attribute name="id" type="ID"/>
  </complexType>
</element>

<element name="enrolled" substitutionGroup="btp:message">
  <complexType>
    <sequence>
      <element name="inferior-identifier" type="btp:identifier"/>
      <element ref="btp:qualifiers" minOccurs="0"/>
      <element name="target-additional-information" type="btp:additional-information" minOccurs="0"/>
    </sequence>
    <attribute name="id" type="ID"/>
  </complexType>
</element>
<element ref="btp:qualifiers" minOccurs="0"/>
  <element name="target-additional-information" type="btp:additional-information" minOccurs="0"/>
</sequence>
</complexType>
</element>
<element name="resigned" substitutionGroup="btp:message">
  <complexType>
    <sequence>
      <element name="inferior-identifier" type="btp:identifier"/>
      <element ref="btp:qualifiers" minOccurs="0"/>
      <element name="target-additional-information" type="btp:additional-information" minOccurs="0"/>
    </sequence>
    <attribute name="id" type="ID"/>
  </complexType>
</element>
<element name="prepare" substitutionGroup="btp:message">
  <complexType>
    <sequence>
      <element name="inferior-identifier" type="btp:identifier"/>
      <element ref="btp:qualifiers" minOccurs="0"/>
      <element name="target-additional-information" type="btp:additional-information" minOccurs="0"/>
    </sequence>
    <attribute name="id" type="ID"/>
  </complexType>
</element>
<element name="prepared" substitutionGroup="btp:message">
  <complexType>
    <sequence>
      <element name="superior-identifier" type="btp:identifier"/>
      <element name="inferior-identifier" type="btp:identifier"/>
      <element name="default-is-cancel" type="boolean"/>
      <element ref="btp:qualifiers" minOccurs="0"/>
      <element name="target-additional-information" type="btp:additional-information" minOccurs="0"/>
    </sequence>
    <attribute name="id" type="ID"/>
  </complexType>
</element>
<element name="confirm" substitutionGroup="btp:message">
  <complexType>
    <sequence>
      <element name="inferior-identifier" type="btp:identifier"/>
      <element ref="btp:qualifiers" minOccurs="0"/>
      <element name="target-additional-information" type="btp:additional-information" minOccurs="0"/>
    </sequence>
    <attribute name="id" type="ID"/>
  </complexType>
</element>
<element name="target-additional-information" type="btp:additional-information" minOccurs="0"/>
</sequence>
</complexType>
</element>

<element name="confirmed" substitutionGroup="btp:message">
<complexType>
<sequence>
<element name="superior-identifier" type="btp:identifier"/>
<element name="inferior-identifier" type="btp:identifier"/>
<element name="confirmed-received" type="boolean"/>
<element ref="btp:qualifiers" minOccurs="0"/>
<element name="target-additional-information" type="btp:additional-information" minOccurs="0"/>
</sequence>
<attribute name="id" type="ID"/>
</complexType>
</element>

<element name="cancel" substitutionGroup="btp:message">
<complexType>
<sequence>
<element name="inferior-identifier" type="btp:identifier"/>
<element name="reply-address" type="btp:address" minOccurs="0"/>
<element ref="btp:qualifiers" minOccurs="0"/>
<element name="target-additional-information" type="btp:additional-information" minOccurs="0"/>
</sequence>
<attribute name="id" type="ID"/>
</complexType>
</element>

<element name="cancelled" substitutionGroup="btp:message">
<complexType>
<sequence>
<element name="superior-identifier" type="btp:identifier"/>
<element name="inferior-identifier" type="btp:identifier" minOccurs="0"/>
<element ref="btp:qualifiers" minOccurs="0"/>
<element name="target-additional-information" type="btp:additional-information" minOccurs="0"/>
</sequence>
<attribute name="id" type="ID"/>
</complexType>
</element>

<element name="confirm-one-phase" substitutionGroup="btp:message">
<complexType>
<sequence>

```xml

```
<element name="inferior-identifier" type="btp:identifier"/>
<element name="report-hazard" type="boolean"/>
<element ref="btp:qualifiers" minOccurs="0"/>
<element name="target-additional-information" type="btp:additional-information" minOccurs="0"/>
</sequence>
</complexType>
</element>
<element name="hazard" substitutionGroup="btp:message">
<complexType>
<sequence>
<element name="superior-identifier" type="btp:identifier"/>
<element name="inferior-identifier" type="btp:identifier"/>
<element name="level">
<simpleType>
<restriction base="string">
<enumeration value="mixed"/>
<enumeration value="possible"/>
</restriction>
</simpleType>
</element>
<element ref="btp:qualifiers" minOccurs="0"/>
<element name="target-additional-information" type="btp:additional-information" minOccurs="0"/>
</sequence>
<attribute name="id" type="ID"/>
</complexType>
</element>
<element name="contradiction" substitutionGroup="btp:message">
<complexType>
<sequence>
<element name="inferior-identifier" type="btp:identifier"/>
<element ref="btp:qualifiers" minOccurs="0"/>
<element name="target-additional-information" type="btp:additional-information" minOccurs="0"/>
</sequence>
<attribute name="id" type="ID"/>
</complexType>
</element>
<element name="superior-state" substitutionGroup="btp:message">
<complexType>
<sequence>
<element name="inferior-identifier" type="btp:identifier"/>
<element name="status">
<simpleType>
<restriction base="string">
<enumeration value="active"/>
<enumeration value="prepared-received"/>
</restriction>
</simpleType>
</element>
</sequence>
</complexType>
</element>
<element name="inferior-state" substitutionGroup="btp:message">
  <complexType>
    <sequence>
      <element name="superior-identifier" type="btp:identifier" />
      <element name="inferior-identifier" type="btp:identifier" />
      <element name="status">
        <simpleType>
          <restriction base="string">
            <enumeration value="active"/>
            <enumeration value="inaccessible"/>
            <enumeration value="unknown"/>
          </restriction>
        </simpleType>
      </element>
      <element name="response-requested" type="boolean" />
      <element ref="btp:qualifiers" minOccurs="0" type="btp:additional-information" />
      <element name="target-additional-information" type="btp:additional-information" minOccurs="0" />
    </sequence>
    <attribute name="id" type="ID" />
  </complexType>
</element>

<element name="redirect" substitutionGroup="btp:message">
  <complexType>
    <sequence>
      <element name="superior-identifier" type="btp:identifier" minOccurs="0" />
      <element name="inferior-identifier" type="btp:identifier" />
      <element name="old-address" type="btp:address" maxOccurs="unbounded" />
      <element name="new-address" type="btp:address" maxOccurs="unbounded" />
      <element ref="btp:qualifiers" minOccurs="0" type="btp:additional-information" />
      <element name="target-additional-information" type="btp:additional-information" minOccurs="0" />
    </sequence>
  </complexType>
</element>
<complexType>
  <sequence>
    <element name="transaction-type" type="btp:superior-type"/>
    <element ref="btp:qualifiers" minOccurs="0"/>
    <element name="target-additional-information" type="btp:additional-information" minOccurs="0"/>
    <element name="reply-address" type="btp:address" minOccurs="0"/>
  </sequence>
  <attribute name="id" type="ID"/>
</complexType>

[element name="begin" substitutionGroup="btp:message">
  <complexType>
    <sequence>
      <element name="transaction-type" type="btp:superior-type"/>
      <element ref="btp:qualifiers" minOccurs="0"/>
      <element name="target-additional-information" type="btp:additional-information" minOccurs="0"/>
      <element name="reply-address" type="btp:address" minOccurs="0"/>
    </sequence>
    <attribute name="id" type="ID"/>
  </complexType>
</element>

[element name="begun" substitutionGroup="btp:message">
  <complexType>
    <sequence>
      <element name="decider-address" type="btp:address" minOccurs="0" maxOccurs="unbounded"/>
      <element name="transaction-identifier" type="btp:identifier" minOccurs="0"/>
      <element name="inferior-handle" type="btp:identifier" minOccurs="0"/>
      <element name="inferior-address" type="btp:address" minOccurs="0" maxOccurs="unbounded"/>
      <element ref="btp:qualifiers" minOccurs="0"/>
      <element name="target-additional-information" type="btp:additional-information" minOccurs="0"/>
    </sequence>
    <attribute name="id" type="ID"/>
  </complexType>
</element>

[element name="prepare-inferiors" substitutionGroup="btp:message">
  <complexType>
    <sequence>
      <element name="transaction-identifier" type="btp:identifier"/>
      <element name="inferiors-list" minOccurs="0">
        <complexType>
          <sequence>
            <element name="inferior-handle" type="btp:identifier" maxOccurs="unbounded"/>
          </sequence>
        </complexType>
      </element>
      <element ref="btp:qualifiers" minOccurs="0"/>
    </sequence>
  </complexType>
</element>
<element name="target-additional-information" type="btp:additional-information" minOccurs="0"/>
<element name="reply-address" type="btp:address" minOccurs="0"/>
</sequence>
</complexType>
</element>

<element name="confirm-transaction" substitutionGroup="btp:message">
<complexType>
<sequence>
<element name="transaction-identifier" type="btp:identifier"/>
<element name="inferiors-list" minOccurs="0">
<complexType>
<sequence>
<element name="inferior-handle" type="btp:identifier" maxOccurs="unbounded"/>
</sequence>
</complexType>
</element>
<element name="report-hazard" type="boolean"/>
<element ref="btp:qualifiers" minOccurs="0"/>
<element name="target-additional-information" type="btp:additional-information" minOccurs="0"/>
<element name="reply-address" type="btp:address" minOccurs="0"/>
</sequence>
<attribute name="id" type="ID"/>
</complexType>
</element>

<element name="transaction-confirmed" substitutionGroup="btp:message">
<complexType>
<sequence>
<element name="transaction-identifier" type="btp:identifier"/>
<element ref="btp:qualifiers" minOccurs="0"/>
<element name="target-additional-information" type="btp:additional-information" minOccurs="0"/>
<element name="reply-address" type="btp:address" minOccurs="0"/>
</sequence>
<attribute name="id" type="ID"/>
</complexType>
</element>

<element name="cancel-transaction" substitutionGroup="btp:message">
<complexType>
<sequence>
<element name="transaction-identifier" type="btp:identifier"/>
<element ref="btp:qualifiers" minOccurs="0"/>
<element name="target-additional-information" type="btp:additional-information" minOccurs="0"/>
<attribute name="id" type="ID"/>
</complexType>
</element>

<element name="cancel-transaction" substitutionGroup="btp:message">
<complexType>
<sequence>
<element name="transaction-identifier" type="btp:identifier"/>
<element name="report-hazard" type="boolean"/>
</complexType>
</element>
<element ref="btp:qualifiers" minOccurs="0"/>
<element name="target-additional-information" type="btp:additional-information" minOccurs="0"/>
<element name="reply-address" type="btp:address" minOccurs="0"/>

</sequence>
</complexType>
</element>

<element name="cancel-inferiors" substitutionGroup="btp:message">
<complexType>
<sequence>
<element name="transaction-identifier" type="btp:identifier" minOccurs="0"/>
<element name="inferiors-list">
<complexType>
<sequence>
<element name="inferior-handle" type="btp:identifier" maxOccurs="unbounded"/>
</sequence>
</element>
<element ref="btp:qualifiers" minOccurs="0"/>
<element name="target-additional-information" type="btp:additional-information" minOccurs="0"/>
<element name="reply-address" type="btp:address" minOccurs="0"/>
</sequence>
</complexType>
</element>

<element name="transaction-cancelled" substitutionGroup="btp:message">
<complexType>
<sequence>
<element name="transaction-identifier" type="btp:identifier"/>
<element ref="btp:qualifiers" minOccurs="0"/>
<element name="target-additional-information" type="btp:additional-information" minOccurs="0"/>
<element name="reply-address" type="btp:address" minOccurs="0"/>
</sequence>
</complexType>
</element>

<element name="request-inferior-statuses" substitutionGroup="btp:message">
<complexType>
<sequence>
<element name="target-identifier" type="btp:identifier"/>
<element name="inferiors-list" minOccurs="0"/>
</sequence>
</complexType>
</element>

<element name="request-inferior-statuses" substitutionGroup="btp:message">
<complexType>
<sequence>
<element name="target-identifier" type="btp:identifier"/>
<element name="inferiors-list" minOccurs="0"/>
</sequence>
</complexType>
</element>
<complexType>
  <sequence>
    <element name="inferior-handle" type="btp:identifier" maxOccurs="unbounded"/>
  </sequence>
</complexType>

<element ref="btp:qualifiers" minOccurs="0"/>
<element name="target-additional-information" type="btp:additional-information" minOccurs="0"/>
<element name="reply-address" type="btp:address" minOccurs="0"/>
  <attribute name="id" type="ID"/>
</element>

<element name="inferior-statuses" substitutionGroup="btp:message">
  <complexType>
    <sequence>
      <element name="responders-identifier" type="btp:identifier"/>
      <element name="status-list">
        <complexType>
          <sequence>
            <element name="status-item" maxOccurs="unbounded">
              <complexType>
                <sequence>
                  <element name="inferior-handle" type="btp:identifier"/>
                  <element name="status">
                    <simpleType>
                      <restriction base="string">
                        <enumeration value="active"/>
                        <enumeration value="resigned"/>
                        <enumeration value="preparing"/>
                        <enumeration value="prepared"/>
                        <enumeration value="autonomously-confirmed"/>
                        <enumeration value="autonomously-cancelled"/>
                        <enumeration value="confirming"/>
                        <enumeration value="confirmed"/>
                        <enumeration value="cancelling"/>
                        <enumeration value="cancelled"/>
                        <enumeration value="cancel-contradiction"/>
                        <enumeration value="confirm-contradiction"/>
                        <enumeration value="hazard"/>
                        <enumeration value="invalid"/>
                      </restriction>
                    </simpleType>
                  </element>
                  <element ref="btp:qualifiers" minOccurs="0"/>
                </sequence>
              </complexType>
            </element>
          </sequence>
        </complexType>
      </element>
    </sequence>
  </complexType>
</element>
<?xml version="1.0"?>
<schema
  xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="urn:oasis:names:tc:BTP:1.0:qualifiers"
  xmlns:btpq="urn:oasis:names:tc:BTP:1.0:qualifiers"
  xmlns:btp="urn:oasis:names:tc:BTP:1.0:core"
  elementFormDefault="qualified">
  <element name="transaction-timelimit" substitutionGroup="btp:qualifier">
    <complexType>
      <complexContent>
        <extension base="btp:qualifier-type">
          <sequence>
            <element name="timelimit" type="nonNegativeInteger"/>
          </sequence>
        </extension>
      </complexContent>
    </complexType>
  </element>
  <element name="inferior-timeout" substitutionGroup="btp:qualifier">
    <complexType>
      <complexContent>
        <extension base="btp:qualifier-type">
          <sequence>
            <element name="timelimit" type="nonNegativeInteger"/>
            <element name="intended-decision">
              <simpleType>
                <restriction base="string">
                  <enumeration value="confirm"/>
                </restriction>
              </simpleType>
            </element>
          </sequence>
        </extension>
      </complexContent>
    </complexType>
  </element>
</schema>
<element name="cancel"/>
</restriction>
</simpleType>
</element>
</sequence>
</complexType>
</element>
</extension>
</complexContent>
</complexType>
</element>
<element name="minimum-inferior-timeout"
substitutionGroup="btp:qualifier">
<complexType>
<complexContent>
<extension base="btp:qualifier-type">
<sequence>
<element name="minimum-timeout" type="nonNegativeInteger"/>
</sequence>
</extension>
</complexContent>
</complexType>
</element>
<element name="inferior-name" substitutionGroup="btp:qualifier">
<complexType>
<complexContent>
<extension base="btp:qualifier-type">
<sequence>
<element name="inferior-name" type="string"/>
</sequence>
</extension>
</complexContent>
</complexType>
</element>
</schema>
Carrier Protocol Bindings

The notion of bindings is introduced to act as the glue between the BTP messages and an underlying transport. A binding specification must define various particulars of how the BTP messages are carried and some aspects of how the related application messages are carried. This document specifies two bindings: a SOAP binding and a SOAP + Attachments binding. However, other bindings could be specified by the Oasis BTP technical committee or by a third party. For example, in the future a binding might exist to put a BTP message directly on top of HTTP without the use of SOAP, or a closed community could define their own binding. To ensure that such specifications are complete, the Binding Proforma defines the information that must be included in a binding specification.

Carrier Protocol Binding Proforma

A BTP carrier binding specification should provide the following information:

Binding name: A name for the binding, as used in the “binding name” field of BTP addresses (and available for declaring the capabilities of an implementation). Binding specified in this document, and future revisions of this document have binding names that are simple strings of letters, numbers and hyphens (and, in particular, do not contain colons). Bindings specified elsewhere shall have binding names that are URIs. Bindings specified in this document use numbers to identify the version of the binding, not the version(s) of the carrier protocol.

Binding address format: This section states the format of the “binding address” field of a BTP address for this binding. For many bindings, this will be a URL of some kind; for other bindings it may be some other form.

BTP message representation: This section will define how BTP messages are represented. For many bindings, the BTP message syntax will be as specified in the XML schema defined in this document, and the normal string encoding of that XML will be used.

Mapping for BTP messages (unrelated): This section will define how BTP messages that are not related to application messages are sent in either direction between Superior and Inferior. (i.e. those messages sent directly between BTP actors). This mapping need not be symmetric (i.e. Superior to Inferior may differ to some degree to Inferior to Superior). The mapping may define particular rules for particular BTP messages, or messages with particular parameter values (e.g. the FAULT message with “fault-type” “CommunicationFailure” will typically not be sent as a BTP message). The mapping states any constraints or requirements on which BTP may or must be bundled together by compounding.

Mapping for BTP messages related to application messages: This section will define how BTP messages that are related to application messages are sent. A binding specification may...
defer details of this to a particular application (e.g. a mapping specification could just say
“the CONTEXT may be carried as a parameter of an application invocation”). Alternatively,
the binding may specify a general method that represents the relationship between application
and BTP messages.

**Implicit messages**: This section specifies which BTP messages, if any, are not sent explicitly
but are treated as implicit in carrier-protocol mechanisms, application messages or other BTP
messages. This may depend on particular parameter values of the BTP messages or the
application messages.

**Faults**: The relationship between the fault and exception reporting mechanisms of the carrier
protocol and of BTP shall be defined. This may include definition of which carrier protocol
exceptions are equivalent to a FAULT/communication-failure message.

**Relationship to other bindings**: Any relationship to other bindings is defined in this section.
If BTP addresses with different bindings are be considered to match (for purposes of
identifying the peer Superior/Inferior and redirection), this should be specified here.

**Limitations on BTP use**: Any limitations on the full range of BTP functionality that are
imposed by use of this binding should be listed. This would include limitations on which
messages can be sent, which event sequences are supported and restrictions on parameter
values. Such limitations may reduce the usefulness of an implementation, but may be
appropriate in certain environments.

**Other**: Other features of the binding, especially any that will potentially affect interoperation
should be specified here. This may include restrictions or requirements on the use or support
of optional carrier parameters or mechanisms or use of standard or other qualifiers.

**Bindings for request/response carrier protocols**

BTP does not generally follow a request/response pattern. In particular, on the outcome
relationship either side may initiate a message – this is an essential part of the presume-abort
recovery paradigm although it is not limited to recovery cases. However, there are some BTP
messages, especially in the control relationship, that do have a request/response pattern.
Many (potential) carrier protocols (e.g. HTTP) do have a request/response pattern. The
specification of a binding specification to a request/response carrier protocol needs to state
what rules apply – which messages can be carried by requests, which by responses. The
simplest rule is to send all BTP messages on requests, and let the carrier responses travel back
empty. This would be inefficient in use of network resources, and possibly inconvenient
when used for the BTP request/response pairs.

This section defines a set of rules that allow more efficient use of the carrier, while allowing
the initiator of a BTP request/response pair to ensure the BTP response is sent back on the
carrier response. These rules are specified in this section to enable binding specifications to
reference them, without requiring each binding specification to repeat similar information.
These rules also allow the receiver of a message between Superior and Inferior (in either
direction) on a carrier protocol request to send any reply message on the carrier response –
the “sender-address” field is implicitly considered to be that of the sender of the carrier
request.

A binding to a request/response carrier is not required to use these rules. It may define other
rules.

**Request/response exploitation rules**

These rules allow implementations to use the request and response of the carrier protocol
efficiently, and, when a BTP request/response exchange occurs, to either treat the
request/response exchanges of the carrier protocol and of BTP independently, if both sides
wish, or allow either side to map them closely.

Under these rules, an implementation sending a BTP request (i.e. a message, other than
CONTEXT, which has “reply-address” as a parameter in the abstract message definition), can
ensure that it and the reply map to a carrier request/response by supplying no value for the
“reply-address”. An implementation receiving such a request is required to send the BTP
response on the carrier response.

Conversely, if an implementation does supply a “reply-address” value on the request, the
receiver has the option of sending the BTP response back on the carrier response, or sending
it on a new carrier request.

Within the outcome relationship, apart from ENROL, there is no “reply-address”, and the
parties normally know each other’s “superior-address” and “inferior-address”. However,
these messages have a “sender-address”, which is used when the receiver does not have
knowledge of the peer. In this case, the “sender-address” is treated as the “reply-address” of
the other messages – if the field is absent in a message on a carrier request, the “sender-
address” is implicitly that of the request sender. Any message for the peer (including the three
messages mentioned, FAULT but also any other valid message in the Superior:Inferior
relationship) may be sent on the carrier response. Apart from this, both sides are permitted to
treat the carrier request/response exchanges as opportunities for sending messages to the
appropriate destination.

The rules:

a) A BTP actor **may** bundle one or more BTP messages and related groups that
have the same binding address for their target in a single btp:messages and
transmit this btp:messages element on a carrier protocol request. There is no
restriction on which combinations of messages and groups may be so bundled,
other than that they have the same binding address, and that this binding address
is usable as the destination of a carrier protocol request.

b) A BTP actor that has received a carrier protocol request to which it has not yet
responded, and which has one or more BTP messages and groups whose binding
address for the target matches the origin of the carrier request may bundle such
BTP messages in a single btp:messages element and transmit that on the carrier
protocol response.

   c) A BTP actor that has received, on a carrier protocol request, one or more BTP
messages or related groups that require a BTP response and for which no “reply-
address” was supplied, must bundle the responding BTP message and groups in a
btp:messages element and transmit this element on the carrier protocol response
to the request that carried the BTP request.

   d) A BTP actor that has received, on a carrier protocol request, one or more BTP
messages or related groups that, as abstract messages, have a “sender-address”
parameter but no “reply-address” was supplied and does not have knowledge of
the peer address, must bundle the responding BTP message and groups in a
btp:messages element and transmit this element on the carrier protocol response
to the request that carried the BTP request. If the actor does have knowledge of
the peer address it may send one or messages for the peer in the carrier protocol
response, regardless of whether the binding address of the peer matches the
address of the carrier protocol requestor.

   e) Where only one message or group is to be sent, it shall be contained within a
btp:messages element, as a bundle of one element.

   f) A BTP actor that receives a carrier protocol request carrying BTP messages that
do have a “reply-address”, or which initiate processing that produces BTP
messages whose target binding address matches the origin of the request, may
freely choose whether to use the carrier protocol response for the replies, or to
send back an “empty carrier protocol response”, and send the BTP replies in a
separately initiated carrier protocol request. The characteristics of an “empty
carrier protocol response” shall be stated in the particular binding specification.

   g) A BTP actor that sends BTP messages on a carrier protocol request must be able
to accept returning BTP messages on the corresponding carrier protocol response
and, if the actor has offered an address on which it will receive carrier requests,
must be able to accept “replying” BTP messages on a separate carrier protocol
request.

SOAP Binding

   This binding describes how BTP messages will be carried using SOAP as in the [SOAP 1.1]
specification, using the SOAP literal messaging style conventions. If no application message
is sent at the same time, the BTP messages are contained within the SOAP Body element. If
application messages are sent, the BTP messages are contained in the SOAP Header element.

   Binding name: soap-http-1
Binding address format: shall be a URL, of type HTTP.

BTP message representation: The string representation of the XML, as specified in the XML schema defined in this document shall be used. The BTP XML messages are embedded in the SOAP message without the use of any specific encoding rules (literal style SOAP message); hence the encodingStyle attribute need not be set or can be set to an empty string.

Mapping for BTP messages (unrelated): The “request/response exploitation” rules shall be used.

BTP messages sent on an HTTP request or HTTP response which is not carrying an application message, the messages are contained in a single btp:messages element which is the immediate child element of the SOAP Body element.

An “empty carrier protocol response” sent after receiving an HTTP request containing a btp:messages element in the SOAP Body and the implementation BTP actor chooses just to reply at the lower level (and when the request/response exploitation rules allow an empty carrier protocol response), shall be any of:

a) an empty HTTP response
b) an HTTP response containing an empty SOAP Envelope
c) an HTTP response containing a SOAP Envelope containing a single, empty btp:messages element.

The receiver (the initial sender of the HTTP request) shall treat these in the same way – they have no effect on the BTP sequence (other than indicating that the earlier sending did not cause a communication failure.)

If an application message is being sent at the same time, the mapping for related messages shall be used, as if the BTP messages were related to the application message. (There is no ambiguity in whether the BTP messages are related, because only CONTEXT and ENROL can be related to an application message.)

Mapping for BTP messages related to application messages: All BTP messages sent with an application message, whether related to the application message or not, shall be sent in a single btp:messages element in the SOAP Header. There shall be precisely one btp:messages element in the SOAP Header.

The “request/response exploitation” rules shall apply to the BTP messages carried in the SOAP Header, as if they had been carried in a SOAP Body, unrelated to an application message, sent to the same binding address.

Note – The application protocol itself (which is using the SOAP Body) may use the SOAP RPC or document approach – this is determined by the application.
Only CONTEXT and ENROL messages are related (&) to application messages. If there is only one CONTEXT or one ENROL message present in the SOAP Header, it is assumed to be related to the whole of the application message in the SOAP Body. If there are multiple CONTEXT or ENROL messages, any relation of these BTP messages shall be indicated by application specific means.

Note 1 – An application protocol could use references to the ID values of the BTP messages to indicate relation between BTP CONTEXT or ENROL messages and the application message.

Note 2 -- However indicated, what the relatedness means, or even whether it has any significance at all, is a matter for the application.

Implicit messages: A SOAP FAULT, or other communication failure received in response to a SOAP request that had a CONTEXT in the SOAP Header shall be treated as if a CONTEXT_REPLY/repudiated had been received. See also the discussion under “other” about the SOAP mustUnderstand attribute.

Faults: A SOAP FAULT or other communication failure shall be treated as FAULT/communication-failure.

Relationship to other bindings: A BTP address for Superior or Inferior that has the binding string “soap-http-1” is considered to match one that has the binding string “soap-attachments-http-1” if the binding address and additional information fields match.

Limitations on BTP use: None

Other: The SOAP BTP binding does not make use of SOAPAction HTTP header or actor attribute. The SOAPAction HTTP header is left to be application specific when there are application messages in the SOAP Body, as an already existing web service that is being upgraded to use BTP might have already made use of SOAPAction. The SOAPAction HTTP header shall be omitted contain no value when the SOAP message carries only BTP messages in the SOAP Body.

The SOAP mustUnderstand attribute, when used on the btp:messages containing a BTP CONTEXT, ensures that the receiver (server, as a whole) supports BTP sufficiently to determine whether any enrolments are necessary and replies with CONTEXT_REPLY as appropriate. The sender of the CONTEXT (and related application message) can use this to ensure that the application work is performed as part of the business transaction, assuming the receiver’s SOAP implementation supports the mustUnderstand attribute. If mustUnderstand is false, a receiver can ignore the CONTEXT (if BTP is not supported there), and no CONTEXT_REPLY will be returned. It is a local option on the sender (client) side whether the absence of a CONTEXT_REPLY is assumed to be equivalent to aCONTEXT_REPLY/ok (and the business transaction allowed to proceed to confirmation).
Note – some SOAP implementations may not support the mustUnderstand attribute sufficiently to enforce these requirements.

**Example scenario using SOAP binding**

The example below shows an application request with CONTEXT message sent from client.example.com (which includes the Superior) to services.example.com (Service).

```xml
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  soap:encodingStyle="">
  <soap:Header>
    <btp:messages xmlns:btp="urn:oasis:names:tc:BTP:1.0:core">
      <btp:context superior-type="atom">
        <btp:superior-address>
          <btp:binding>soap-http-1</btp:binding>
          <btp:binding-address>http://client.example.com/soaphandler</btp:binding-address>
          <btp:additional-information>btpengine</btp:additional-information>
        </btp:superior-address>
        <btp:superior-identifier>http://example.com/1001</btp:superior-identifier>
        <btp:qualifiers>
          <btpq:transaction-timelimit
        </btp:qualifiers>
      </btp:context>
    </btp:messages>
  </soap:Header>
  <soap:Body>
    <ns1:orderGoods
      xmlns:ns1="http://example.com/2001/Services/xyzgoods">
      <custID>ABC8329045</custID>
      <itemID>224352</itemID>
      <quantity>5</quantity>
    </ns1:orderGoods>
  </soap:Body>
</soap:Envelope>
```
The example below shows CONTEXT_REPLY and a related ENROL message sent from services.example.com to client.example.com, in reply to the previous message. There is no application response, so the BTP messages are in the SOAP Body. The ENROL message does not contain the target-additional-information, since the grouping rules for CONTEXT_REPLY & ENROL omit the “target-address” (the receiver of this example remembers the superior address from the original CONTEXT)

```xml
<soap:Envelope
 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
 soap:encodingStyle="">
  <soap:Header>
    </soap:Header>
  <soap:Body>
    <btp:messages xmlns:btp="urn:oasis:names:tc:BTP:1.0:core">
      <btp:related-group>
        <btp:context-reply>
          <btp:target-additional-information>btpengine</btp:target-additional-information>
          <btp:superior-identifier>http://example.com/1001</btp:superior-identifier>
          <completion-status>related</completion-status>
        </btp:context-reply>
        <btp:enrol response-requested="false">
          <btp:target-additional-information>btpengine</btp:target-additional-information>
          <btp:superior-identifier>http://example.com/1001</btp:superior-identifier>
          <btp:inferior-address>
            <btp:binding>soap-http-1</btp:binding>
            <btp:binding-address>
              http://services.example.com/soaphandler
            </btp:binding-address>
          </btp:inferior-address>
          <btp:inferior-identifier>
            http://example.com/AAAB
          </btp:inferior-identifier>
        </btp:enrol>
      </btp:related-group>
    </btp:messages>
  </soap:Body>
</soap:Envelope>
```
SOAP + Attachments Binding

This binding describes how BTP messages will be carried using SOAP as in the SOAP Messages with Attachments specification. It is a superset of the Basic SOAP binding, soap-http-1. The two bindings only differ when application messages are sent.

**Binding name:** soap-attachments-http-1

**Binding address format:** as for soap-http-1

**BTP message representation:** As for soap-http-1

**Mapping for BTP messages (unrelated):** As for “soap-http-1”, except the SOAP Envelope containing the SOAP Body containing the BTP messages shall be in a MIME body part, as specified in SOAP Messages with Attachments specification. If an application message is being sent at the same time, the mapping for related messages for this binding shall be used, as if the BTP messages were related to the application message(s).

**Mapping for BTP messages related to application messages:** MIME packaging shall be used. One of the MIME multipart/related parts shall contain a SOAP Envelope, whose SOAP Headers element shall contain precisely one btp:messages element, containing any BTP messages. Any BTP CONTEXT in the btp:messages is considered to be related to the application message(s) in the SOAP Body, and to also any of the MIME parts referenced from the SOAP Body (using the “href” attribute).

**Implicit messages:** As for soap-http-1.

**Faults:** As for soap-http-1.

**Relationship to other bindings:** A BTP address for Superior or Inferior that has the binding string “soap-http-1” is considered to match one that has the binding string “soap-attachments-http-1” if the binding address and additional information fields match.

**Limitations on BTP use:** None

**Other:** As for soap-http-1

**Example using SOAP + Attachments binding**

```plaintext
Content-Type: Multipart/Related; boundary=MIME_boundary;
type=text/xml;
start="someID"

--MIME_boundary
Content-Type: text/xml; charset=UTF-8
Content-ID: someID
```
Conformance

A BTP implementation need not implement all aspects of the protocol to be useful. The level of conformance of an implementation is defined by which roles it can support using the specified messages and carrier protocol bindings for interoperation with other implementations.
An **partially conformant** implementation may implement the **functionality of** some roles in a non-interoperable way – usually combining pairs of roles, such as Terminator and Decider, giving that implementation’s users comparable proprietary functionality. Such an implementation is conformant in respect of the roles it does implement in accordance with this specification.

An implementation can state which aspects of the BTP specification it conforms to in terms of which Roles it supports. Since most Roles cannot usefully be supported in isolation, the following **Roles and Role Groups** are **can be used to define** describe implementation capabilities: conformance.

<table>
<thead>
<tr>
<th>Role Group</th>
<th>Roles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initiator/Terminator</td>
<td>Initiator  &lt;br&gt; Terminator</td>
</tr>
<tr>
<td>Cohesive Hub</td>
<td>Factory  &lt;br&gt; Composer (as Decider and Superior)  &lt;br&gt; Coordinator (as Decider and Superior)  &lt;br&gt; Sub-composer  &lt;br&gt; Sub-coordinator</td>
</tr>
<tr>
<td>Atomic Hub</td>
<td>Factory  &lt;br&gt; Coordinator  &lt;br&gt; Sub-coordinator</td>
</tr>
<tr>
<td>Cohesive Superior</td>
<td>Composer (as Superior only)  &lt;br&gt; Sub-Composer  &lt;br&gt; Coordinator (as Superior only)  &lt;br&gt; Sub-coordinator</td>
</tr>
<tr>
<td>Atomic Superior</td>
<td>Coordinator (as Superior only)  &lt;br&gt; Sub-coordinator</td>
</tr>
<tr>
<td>Participant</td>
<td>Inferior  &lt;br&gt; Enroller</td>
</tr>
</tbody>
</table>
The Role Groups occupy different positions within a business transaction tree and thus require presence of implementations supporting other Role Groups:

Initiator/Terminator uses control relationship to Atomic Hub or Cohesive Hub to initiate and control Atoms or Cohesions. Initiator/Terminator would typically be a library linked with application software.

Atomic Hub and Cohesive Hub would often be standalone servers.

Cohesive Superior and Atomic Superior would provide the equivalent of Initiator/Terminator functionality by internal or proprietary means.

Cohesive Hubs, Atomic Hubs, Cohesive Superior and Atomic Superior use outcome relationships to Participants and to each other.

Participants will establish outcome relationships to implementations of any of the other Role Groups except Initiator/Terminator. A Participant “covers” a resource or application work of some kind. It should be noted that a Participant is unaffected by whether it is enrolled in an Atom or Cohesion – it gets only a single outcome.

An implementation may support one or more Role Groups. The following combinations are defined as commonly expected conformance profiles, although other combinations or selections are equally possible.

<table>
<thead>
<tr>
<th>Conformance Profile</th>
<th>Role Groups</th>
</tr>
</thead>
<tbody>
<tr>
<td>Participant Only</td>
<td>Participant</td>
</tr>
<tr>
<td>Atomic</td>
<td>Atomic Superior</td>
</tr>
<tr>
<td></td>
<td>Participant</td>
</tr>
<tr>
<td>Cohesive</td>
<td>Cohesive Superior</td>
</tr>
<tr>
<td></td>
<td>Participant</td>
</tr>
<tr>
<td>Atomic Coordination Hub</td>
<td>Initiator/Terminator</td>
</tr>
<tr>
<td></td>
<td>Atomic Coordination Hub</td>
</tr>
<tr>
<td></td>
<td>Participant</td>
</tr>
<tr>
<td>Cohesive Coordination Hub</td>
<td>Initiator/Terminator</td>
</tr>
<tr>
<td></td>
<td>Cohesive Coordination Hub</td>
</tr>
</tbody>
</table>
BTP has several features, such as optional parameters, that allow alternative implementation architectures. Implementations should pay particular attention to avoid assuming their peers have made the same implementation options as they have (e.g., an implementation that always sends ENROL with the same inferior address and with the “reply-address” absent (because the Inferior in all transactions are dealt with by the same addressable entity), must not assume that the same is true of received ENROLs).
## Part 3. Glossary

### A. Glossary

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Actor</strong></td>
<td>An entity that executes procedures, a software agent. (See also BTP Actor)</td>
</tr>
<tr>
<td><strong>Address</strong></td>
<td>An identifier for an endpoint.</td>
</tr>
</tbody>
</table>
| **Application**  | An actor, which uses the Business Transaction Protocol (in the context of this specification). Also, a group of such actors, which may be distributed, that perform a common purpose.  
(When used in phrases such as “determined by the Application”, it is not relevant to BTP whether this is determined by the owner of a single system or is explicitly part of the contract that defines the distributed collaborative application. When it is necessary to distinguish the responsibilities of a single party, the term “Application element” is used.) |
<p>| <strong>Application element</strong> | An actor that communicates, using application protocols, with other application elements, as part of an overall distributed application. A single system may contain more than one application element. |
| <strong>Application Endpoint</strong> | An endpoint of an application message.                                                                                                       |
| <strong>Application Message</strong> | A message produced by an application element and consumed by an application element.                                                           |
| <strong>Application Operation</strong> | An operation, which is started when an application message arrives.                                                                           |
| <strong>Appropriate</strong>  | In accordance with a pertinent contract or specification.                                                                                   |</p>
<table>
<thead>
<tr>
<th><strong>Atom</strong></th>
<th>A set of participants, which are the direct inferiors of a node (which may have only one member), all of which will receive instructions that will result in a homogeneous outcome. That is they will be issued instructions to all confirm or all cancel. (Transitively, a set of operations whose effect is capable of counter effect.)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Atomic Business Transaction</strong></td>
<td>A complete business transaction that follows the atom rules for every node in the transaction tree over space and time, so that all the participants in the transaction will receive instructions that will result in a homogeneous outcome. That is they will be issued instructions to all confirm or all cancel. (Transitively, a set of operations whose effect is capable of counter effect.)</td>
</tr>
<tr>
<td><strong>Become prepared</strong></td>
<td>Ensure that of a set of procedures is capable of being successfully instructed to cancel or to confirm.</td>
</tr>
<tr>
<td><strong>BTP Actor</strong></td>
<td>A software entity, or agent, that is able to take part in Business Transaction Protocol exchanges i.e. that sends or receives BTP messages. A BTP Actor may be capable of only playing a single role, or of playing several different roles concurrently and / or sequentially. A BTP Actor may be involved in one, or more, transactions, concurrently and / or sequentially.</td>
</tr>
<tr>
<td><strong>BTP element</strong></td>
<td>A BTP actor that supports an application element (or elements) but is not itself concerned with application messages or semantics.</td>
</tr>
<tr>
<td><strong>(Business) Application Protocol</strong></td>
<td>The messages, their meanings and their permitted sequences used to effect a change in the state of a business relationship.</td>
</tr>
<tr>
<td><strong>(Business) application system</strong></td>
<td>A system that contains one, or more, business applications, and resources such as volatile and persistent storage for business state information. It may also contain other things such as an operating system and BTP elements.</td>
</tr>
<tr>
<td><strong>Business relationship agreement</strong></td>
<td>The contract and / or set of agreements that govern and constrain a business relationship between two, or more, parties.</td>
</tr>
<tr>
<td><strong>Business relationship</strong></td>
<td>A <em>business relationship</em> is any distributed state held by the parties, which is subject to contractual constraints agreed by those parties.</td>
</tr>
<tr>
<td><strong>Business Transaction Protocol (BTP)</strong></td>
<td>The messages, their meanings and their permitted sequences defined in this specification. Its purpose is to provide the interactions (or signalling) required to coordinate the effects of application protocol to achieve a business transaction.</td>
</tr>
<tr>
<td><strong>BTP-Address</strong></td>
<td>A compound address consisting of three parts. The first part, the “binding name”, identifies the binding to a particular carrier protocol – some bindings are specified in this document, others can be specified elsewhere. The second part of the address, the “binding address”, is meaningful to the carrier protocol itself, which will use it for the communication (i.e. it will permit a message to be delivered to a receiver). The third part, “additional information”, is not used or understood by the carrier protocol. The “additional information” may be a structured value.</td>
</tr>
<tr>
<td><strong>Business transaction</strong></td>
<td>A set of state changes that occur, or are desired, in computer systems controlled by some set of parties, and these changes are related in some application defined manner. A business transaction is subject to, and a part of, a business relationship. (BTP assumes that the parties involved in a business transaction have distinct and autonomous application systems, which do not require knowledge of each others’ implementation or internal state representations in volatile or persistent storage. Access to such loosely coupled systems is assumed to occur only through service interfaces.)</td>
</tr>
<tr>
<td><strong>Cancel</strong></td>
<td>Process a counter effect for the current effect of a set of procedures. There are a number of different ways that this may be achieved in practice.</td>
</tr>
<tr>
<td><strong>Carrier Protocol</strong></td>
<td>A protocol, which defines how the transmission of BTP messages occur.</td>
</tr>
<tr>
<td><strong>Carrier Protocol Address (CPA)</strong></td>
<td>The address of an endpoint for a particular carrier protocol.</td>
</tr>
<tr>
<td><strong>Client</strong></td>
<td>An actor, which sends application messages to services.</td>
</tr>
</tbody>
</table>
**Cohesion**

A set of participants, which are the direct inferiors of a node that may receive instructions that may result in different outcomes for each participant. That is they will be issued instructions to confirm or cancel according to the application logic. Participants may resign or be instructed to cancel until the confirm set is fixed. Once the confirm set for a cohesion is fixed, then all participants in the confirm set are treated atomically. That is they will all be instructed to confirm unless one, or more, cancel in which case all will be instructed to cancel. All participants not in the confirm set will be instructed to cancel.

**Cohesive Business Transaction**

A complete business transaction for which at least one node over space and time follows the cohesion rules. The other nodes in the transaction tree of a cohesive business transaction may follow either the cohesion rules or the atom rules.

**Confirm**

Ensure that the effect of a set of procedures is completed. There are a number of different ways that this may be achieved in practice.

**Context**

Information pertinent to a single transaction, or branch of a transaction.

**Contract**

Any rule, agreement or promise which constrains an actor’s behaviour and is known to any other actor, and upon which any other knowing actor may rely.

**Control relationship**

The application element:BTP element relationships that create the nodes of the transaction tree (Initiator:Factory) and drive the completion (Terminator:Decider).

**Coordinator**

A BTP actor, which is the top ‘node’ of a transaction and decides the outcome of its immediate branches according to the atom rules defined in this specification. It has a lifetime, which is coincident with that of the atom. A coordinator can issue instructions to prepare, cancel and confirm. These instructions take the form of BTP messages. A coordinator is identified by its transaction-identifier. A coordinator must also have a BTP Address to which participants can send BTP messages.

**Counter effect**

An appropriate effect intended to counteract a prior effect.
<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Counter effect contract</td>
<td>The contract, which governs the relationship between the effect and the counter effect of a procedure. In the absence of any other overriding contracts the counter effect contract is the promise that the Counter effect will attempt so far as is possible to reverse or cancel the Effect such that an observer (on completion of the Counter effect) is unaware that the Effect ever occurred, but this attempt cannot be guaranteed to succeed.</td>
</tr>
<tr>
<td>Decider</td>
<td>The top node of a transaction tree, a composer or a coordinator (so called because the Terminator can only request confirmation – the Decider makes the final determination). The term can always be interpreted as “Composer or Coordinator”. It is the role at the other end of a control relationship to a Terminator.</td>
</tr>
<tr>
<td>Delivery parameter</td>
<td>A parameter of an abstract message that is concerned with the transmission of the message to its target or the transmission of an immediate reply. Distinguished from Payload parameter.</td>
</tr>
<tr>
<td>Effect</td>
<td>The changes induced by the incomplete or complete processing of a set of procedures by an actor, which are observable by another contemporary or future actor, and which are made in conformance with a contract known to any such observer. This contract must state the counter effect of the effect, and this is known as a counter effect contract. An effect is Completed when the change inducing processing of the set of procedures is finished.</td>
</tr>
<tr>
<td>Endpoint</td>
<td>A sender or receiver.</td>
</tr>
<tr>
<td>Enroller</td>
<td>The BTP Actor role that informs a superior of the existence of an inferior.</td>
</tr>
<tr>
<td>Factory</td>
<td>The BTP Actor role that creates transaction contexts and deciders.</td>
</tr>
<tr>
<td>Inappropriate</td>
<td>In violation of a pertinent contract or specification.</td>
</tr>
<tr>
<td>Ineffectual</td>
<td>Describes a set of procedures, which has no effect.</td>
</tr>
<tr>
<td>Inferior</td>
<td>The end of end of a BTP node to BTP node relationship governed by the outcome protocol that is topologically further from the top of the transaction tree.</td>
</tr>
</tbody>
</table>
**Inferior-Address**  
The address used to communicate with an actor playing the role of an Inferior.

**Inferior-identifier**  
A globally unambiguous identification of a particular Inferior within a single transaction (represented as an URI or equivalent).

**Initiator**  
The BTP Actor role (an application element) that starts a transaction.

**Intermediate**  
A node that is a sub-composer or a sub-coordinator. An alternative term to interposed.

**Interposed**  
A node that is a sub-composer or a sub-coordinator. An alternative term to intermediate.

**Message**  
A datum, which is produced and then consumed.

**Node**  
A logical entity that is associated with a single transaction. A node is a composer, a coordinator, a sub-coordinator, a sub-composer, or a participant.

**Operation**  
A procedure, which is started by a receiver when a message arrives at it.

**Outcome**  
A decision to either cancel or confirm.

**Outcome relationship**  
The Superior:Inferior relationship (i.e. between BTP actors within the transaction tree) and the Enroller:Superior relationship used in establishing it.

**Participant**  
A participant is part of an application system that also contains one, or more, applications, which manipulate resources. It is a role of a BTP Actor that is (or is equivalent to) a set of procedures, which is capable of receiving instructions from another BTP Actor to prepare, cancel and confirm. These signals are used by the application(s) to determine whether to effect (confirm) or counter effect (cancel) the results of application operations. A participant must also have a BTP Address, to which these instructions will be delivered, in the form of BTP messages. A participant is identified by an inferior-identifier.
<table>
<thead>
<tr>
<th><strong>Payload parameter</strong></th>
<th>A parameter of an abstract message that is will be received and processed or retained by the receiving BTP actor. The various identifier parameters are considered Payload parameters. Distinguished from Delivery parameter.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Peer</strong></td>
<td>The other party in a two-party relationship, as in Superior to Inferior, or Sender to Receiver.</td>
</tr>
<tr>
<td><strong>Provisional Effect</strong></td>
<td>The changes induced by the incomplete or complete processing of a set of procedures by an actor, which are subject to later completion or counter-effecting. The provisional effect may or may not be observable by other actors.</td>
</tr>
<tr>
<td><strong>Receiver</strong></td>
<td>The consumer of a message.</td>
</tr>
<tr>
<td><strong>Relationship parties</strong></td>
<td>The legal entities that enter into an agreement that forms the basis of the relationship.</td>
</tr>
<tr>
<td><strong>Responders-identifier</strong></td>
<td>An identifier carried in a BTP message that can be interpreted as transaction-identifier, a superior-identifier, or an inferior-identifier according to the nature of the role in a BTP actor that is responding to a received message.</td>
</tr>
<tr>
<td><strong>Role</strong></td>
<td>The participation of a software agent in a particular relationship in a particular business transaction. The software agent performing a role is termed an <strong>Actor</strong>.</td>
</tr>
<tr>
<td><strong>Sender</strong></td>
<td>The producer of a message.</td>
</tr>
<tr>
<td><strong>Service</strong></td>
<td>An actor (an application element), which on receipt of application messages, may start an appropriate application operation. For example, a process that advertises an interface allowing defined RPCs (remote procedure calls) to be invoked by a remote client.</td>
</tr>
<tr>
<td><strong>Status requestor</strong></td>
<td>The BTP Actor role that requests the status of another BTP actor.</td>
</tr>
</tbody>
</table>
**Sub-composer**

An actor, which is not the top ‘node’ of a transaction. It receives an outcome from its superior and decides the outcome of its immediate branches according to the cohesive rules defined in this specification. It has a lifetime, which is coincident with that of the cohesion. A sub-composer can issue instructions to prepare, cancel and confirm on individual branches. These instructions take the form of BTP messages. A sub-composer must also have at least one BTP Address to which lower nodes can send BTP messages.

**Sub-coordinator**

An actor, which is not the top ‘node’ of a transaction. It receives an outcome from its superior and propagates the outcome to its immediate branches according to the atom rules defined in this specification. It has a lifetime, which is coincident with that of this atom. A sub-coordinator can issue instructions to prepare, cancel and confirm. These instructions take the form of BTP messages. A sub-coordinator must also have at least one BTP Address to which lower nodes can send BTP messages.

**Superior**

The BTP role that will accept enrolments of Inferiors and subsequently inform the Inferior of the Outcome applicable to it.

A Superior will be one of Composer, Coordinator, Sub-composer, or Sub-coordinator.

A Superior is considered to be a Superior even if it currently has no enrolled Inferiors.

**Superior-address**

The set of BTP-addresses used to communicate with an actor playing the role of a Superior.

**Superior-identifier**

A globally unambiguous identifier of a particular Superior within a particular transaction (represented as an URI or equivalent).

**Target-identifier**

An identifier carried in a BTP message that can be interpreted as transaction-identifier, a superior-identifier, or an inferior identifier according to the nature of the role in a BTP actor that receives this identifier.
**Terminator**
A BTP role performed by an Application element communicating with a Decider to control the completion of the Business Transaction. Frequently will be identical to the Initiator, but distinguished because the control of the Business Transaction can be passed between Application elements.

**Transaction**
A complete unit of work as defined by an application. A transaction starts when a part of the distributed transaction first initiates some work that is to be a part of a new transaction. The transaction tree may grow and shrink over time and (logical) space. A transaction completes when all the participants in a transaction have completed (that is have replied to their confirm or cancel instruction).

**Transaction tree**
A pattern of BTP nodes that provides the coordination of a distributed application transaction. There is single top node (a Decider) that interacts with the initiating application (which is a part of a distributed application). The Decider node has one, or more outcome relationships with other BTP nodes (sub-composer, sub-coordinator, or participant nodes). Any intermediate nodes (Sub-composer or Sub-coordinator nodes) have exactly one relationship up the tree in which they act as Inferior, and one, or more, relationships down the tree in which they act as Superior. Participants are leaves of the tree. That is they have exactly one relationship up the tree in which they act as Inferior and no down tree relationships.

**Transaction-identifier**
A globally unambiguous identifier for a particular a Decider(represented as an URI or equivalent). A Decider is the top `node` of the transaction and thus this identifier also unambiguously identifies the transaction. Often identical to the Superior-identifier of the Decider in its role as Superior, though the protocol does not require this.

**Transmission**
The passage of a message from a sender to a receiver.