Organization for the Advancement of Structured Information Systems

Business Transaction Protocol

An OASIS Committee Specification

CURRENT STATUS : internal committee draft

Version 1.0 *[0.9.0.4]*

DD Mmm 2001 *[12 January 2002 18:03]*

Working draft 0.1 (pre-London)	14 June 2001
Working draft 0.2 (London)	18 June 2001
Working draft 0.3a (circulated)	12 July 2001
Working draft 0.3c (circulated)	20 July 2001
Working draft 0.4 (circulated; incorporates PRF material)	25 July 2001
Working draft 0.6 (State tables)	31 August 2001
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 solutio n to msging issues	11 December 2001
Working Draft 0.9.0.4 – issue 79 solution, revise msging issues	12 January 2002

Change marks relative to 0.9.0.2

Copyright and related notices

Acknowledgements

Employees of the following companies participated in the finalization of this specification as members of the OASIS Business Transactions Technical Committee:

BEA Systems, Inc.
Bowstreet, Inc.
Choreology Ltd.
Entrust, Inc.
Hewlett-Packard Co.
Interwoven Inc.
IONA Technologies PLC
SeeBeyond Inc.
Sun Microsystems Computer Corp.
Talking Blocks Inc.

The primary authors and editors of the main body of the specification were:

Alex Ceponkus (alex@ceponkus.org)
Peter Furniss (peter.furniss@choreology.com)
Alastair Green (alastair.green@choreology.com)

Additional contributions to its writing were made by

Sanjay Dalal (sanjay.dalal@bea.com)
Mark Little (mark_little@hp.com)

We thank Pal Takacsi-Nagy of BEA Systems Inc for his 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, on 11 September 2001.

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>
```

Illustrative fragments of code in other languages, such as Java, are given in Lucida Console:

```
int main (String[] args)
{
}
```

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

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 ten 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 uses a two-phase outcome coordination protocol to create atomic effects (results of computations). BTP also permits the composition of such atomic units of work (atoms) into cohesive business transactions (cohesions), which allow application intervention into the selection of the atoms which will be confirmed, and of those which will be cancelled.

BTP 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 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

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 provisiona l 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.

Part 2. Normative Specification of BTP

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 ~~primary~~ 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).

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.

Roles involved in the ~~Superior:Inferior~~outcome relationships

Superior

Accepts enrolments from Inferiors, 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.

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

PREPARED
CANCELLED
CONFIRMED
HAZARD
RESIGN
INFERIOR_STATE

from an enrolled Inferior.

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 a ppropriate. 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.

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.

~~An Inferior receives REQUEST_STATUS and replies with STATUS. If it is also a Superior, the STATUS concerns the Inferior as a whole.~~

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.

An Enroller sends

ENROL

to a Superior.

An Enroller receives

ENROLLED

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

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.

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.

Roles involved in the ~~Terminator:Decider~~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 decision is synonymous with the persisting of information identifying the Inferiors that are to be confirmed. An Inferior cannot be confirmed unless PREPARED has been received from it.

A Decider is instructed to cancel by receiving CANCEL_TRANSACTION~~/whole~~.

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.

All Deciders receive
~~REQUEST_CONFIRM~~CONFIRM_TRANSACTION
~~REQUEST_~~CANCEL_TRANSACTION~~/whole~~

~~REQUEST_STATUSES~~REQUEST_INFERIOR_STATUSES

All Deciders send
CONFIRM~~ED~~_COMPLETE
CANCEL~~LED~~_COMPLETE
INFERIOR_STATUSES

~~A Decider also receives REQUEST_STATUS and replies with DECIDER_STATUS, reporting its state as a whole.~~

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

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 REQUEST_PREPAREPREPARE_INFERIORS. It issues PREPARE to any of those Inferiors from which none of PREPARED, CANCELLED or RESIGN~~ED~~ have been received, and replies to the ~~REQUEST_PREPARE~~PREPARE_INFERIORS with INFERIOR_STATUSES.

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

~~In addition to the messages received by the Composer as a Decider, it receives~~
~~REQUEST_PREPARE~~
~~REQUEST_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 ~~REQUEST_CONFIRM~~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 CONFIRM~~ED~~_COMPLETE, CANCEL~~LED~~_COMPLETE 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 ~~REQUEST_PREPARE~~PREPARE_INFERIORS~~/inferiors~~. The Composer replies with INFERIOR_STATUSES.

A Terminator may send ~~REQUEST_~~CANCEL_TRANSACTION to instruct the Decider to cancel the whole business transaction.,~~ or, if it is a Cohesion Composer, some of its Inferiors~~. The Decider replies with CANCEL~~LED~~_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 ~~or for a selective cancel or in the case of problems,~~ INFERIOR_STATUSES.

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

A Terminator sends

- ~~REQUEST_CONFIRM~~CONFIRM_TRANSACTION
- ~~REQUEST_~~CANCEL_TRANSACTION
- CANCEL_INFERIORS
- ~~REQUEST_PREPARE~~PREPARE_INFERIORS~~/inferiors~~
- ~~REQUEST_STATUSES~~REQUEST_INFERIOR_STATUSES

A Terminator receives

- CONFIRM~~ED~~_COMPLETE
- CANCEL~~LED~~_COMPLETE
- INFERIOR_STATUSES
- ~~DECIDER_STATUS~~

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.

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

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 any actor that an address previously supplied for some other actor 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.)

A Redirector **may** also be used to change the address of other BTP actors.

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

Status Requestor

Requests and receives the current status of a transaction tree node – any of an Inferior, Superior or ~~a~~ 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).~~.~~

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. ~~actor concerned in its role~~ as an Inferior, even if it is also a Superior).

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.

Addresses

All of the messages except CONTEXT and CONTEXT_REPLY 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 invariably have been originally determined by the implementation that is the target or desired future 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 compound target address. This allows full flexibility for implementation choices – an implementation can:

a) Use the same binding address and additional information for multiple business transactions, using the identifier parameter to locate the relevant state information;
b) Use the same binding address for multiple business transactions and use the additional information to locate the information; or
c) Use a different binding address for each business transaction.

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). In cases b) and c), the identifier is to some extent redundant, although interoperation requires that it always be present.

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 "Redirection" 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.

For messages which are specified as sent between Superior and Inferior, a FAULT message is sent to the peer.

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 ~~In both cases~~ ~~the~~ all ~~messages will~~ 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). ~~concerns the bundling of application messages, especially where the application uses a request/response paradigm~~. The application request is sent with a related CONTEXT message. The application response is sent with a ~~related~~ relation group of CONTEXT_REPLY/related, ~~with an~~ ENROL/no-rsp-req message and a ~~bundled~~ PREPARED message ~~(assuming the operations succeeded and the Inferior has decided to be prepared)~~. 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.~~must have a binding address that is the same as the target address of the application response (i.e. the reply address for the client, as perceived by the Service) – otherwise the Service cannot determine that it should bundle the messages together. One-shot is thus a specialization of one-wire.~~

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. ~~with the application response and the CONTEXT_REPLY.~~ If an operation fails, a CANCELLED message ~~can be~~is sent ~~with the response~~ instead of a PREPARED.

If the CONTEXT has "superior-type" of "atom", then ~~If~~ 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.

~~Where does that last bit on one-shot, one-wire belong. It needs to be in somewhere. prf~~

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.

Inferior handle

Some of the messages exchanged between a Terminator and a Decider are concerned with the individual Inferiors enrolled with the Decider, and not with the business transaction as a whole. These messages distinguish the Inferiors of Decider using an “inferior handle”. This is created by the Decider and is unambiguous within the scope of the Decider .

The “inferior handle” is distinct from the “inferior identifier” passed on an ENROL message (among other places). The latter is created by the Inferior (or its enroller) and is required to be unambiguous within the scope of the address-as-inferior on the ENROL (and unambiguous within **any** of the individual addresses in that set of BTP addresses - the identifier must identify the Inferior across all the places it might migrate to or that have recovery responsibility for it).

The “inferior handle” is only used by the Terminator to refer to the inferiors of the Decider. In messages between the Decider and its Inferiors, the address-as-inferior and inferior identifier are used.

Messages

Qualifiers

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

Sub-parameter	Type
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 ~~involved in several relationships~~not restricted to outcome or control relationships.

The messages in this section are used between various roles. ~~The~~ 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.~~and used on an Application:Application relationship. Another use is when it is related to a BEGUN message on an Initiator: Factory relationship. A~~ CONTEXT_REPLY is used as the reply to a CONTEXT .~~related to an application 'message' and used on an Application:Application relationship. A~~ REQUEST_STATUS can be issued to, and STATUS returned by any of Decider, Superior or Inferior. FAULT ~~message~~ can be used on any relationship to indicate an error condition back to the sender of a message~~, except on an~~

~~Application:Application relationship where it is assumed that the application protocol will have its own means of coping with errors~~.

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").

Parameter	Type
address-as-superior	Set of BTP addresses
superior identifier	Identifier
reply-address	BTP address
superior type	cohesion/atom
qualifiers	List of qualifiers

address-as-superior 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 within the scope of the address-as-superior

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.

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.

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.

Parameter	Type
target-address	BTP address
superior-address	BTP address
superior identifier	Identifier
completion_status	complete/related/repudiated
Qualifiers	List of qualifiers

target-address the address to which the CONTEXT_REPLY is sent. This shall be the "reply-address" from the CONTEXT.

superior-address one of the addresses from the address-as-superior from the CONTEXT. (The parameter is present in CONTEXT_REPLY to disambiguate the superior identifier.)

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

Value	meaning
completed	All enrolments (if any) have succeeded already
Related	At least some enrolments are to be performed by ENROL messages related to the CONTEXT_REPLY. All other enrolments (if any) have succeeded already.
repudiated	At least one enrolment has failed. The implications of receiving the CONTEXT have **not** been honoured.

qualifiers standardised or other qualifiers.

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).

Parameter	Type
target address	BTP address
reply address	BTP address
~~inferior-~~target-identifier	Identifier
~~transaction-identifier~~	~~Identifier~~
Qualifiers	List of qualifiers

target address the address to which the REQUEST_STATUS message is sent. This can be any of address-as-decider, address-as-inferior or address-as-superior.~~If the target is an Inferior, this will be the address-as-inferior on the ENROL message. If the target is a Decider, this will be the address-as-decider on the BEGUN message.~~

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

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

~~**transaction-identifier** If the target is a Decider, the “transaction-identifier” on the BEGUN message. If the target is an Inferior, this parameter shall be absent.~~

qualifiers standardised or other qualifiers.

Types of FAULT possible (sent to reply address)

General

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 ~~Inferior or Decide~~sender~~r~~.

Parameter	**Type**
target address	BTP address
respondersaddress~~-as-inferior~~	BTP address
~~inferior-~~responders-identifier	Identifier
status	See below
qualifiers	List of qualifiers

target address the address to which the STATUS is sent. This will be the reply address on the REQUEST_STATUS message

responders-address ~~**-as-inferior**~~ the address of the sender of the STATUS message – one of address-as-inferior, address-as-decider, address-as-superior(with the responders-identifier, this determines who the message is from).. If the sender has different addresses as multiple roles (as Decider, Inferior or Superior), this shall be the address on which the REQUEST_STATUS was received. ~~If the sender is an Inferior, the address-as-inferior as on the ENROL message (with the inferior--identifier, this determines who the message is from). If the sender is a Decider, this parameter shall be absent~~

responders~~inferior~~-identifier the identifier of the state, aligned with the responders-address. If the sender has multiple roles in the transaction (as Decider, Inferior or Superior), this shall be the target-identifier on the REQUEST_STATUS ~~If the sender is an Inferior, the inferior--identifier as on the ENROL message. If the sender is a Decider, this parameter shall be absent.~~

~~**address-as-decider**~~ ~~If the sender is a Decider, the address-as-decider on the BEGUN message (with the “transaction-identifier”, this determines who the message is from). If the sender is an Inferior, this parameter shall be absent.~~

~~**transaction-identifier**~~ ~~If the sender is a Decider, the transaction identifier as on the BEGUN message. If the sender is an Inferior, this parameter shall be absent.~~

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. is 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.

status value	Meaning from ~~Decider~~Superior	Meaning from Inferior
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~~; no decision has been made.~~	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; PREPARED has not been sent
Prepared	Not applicable	PREPARED has been sent; no outcome has been received or autonomous decision made
Confirming	Confirm decision has been made or CONFIRM has been received as Inferior but responses from inferiors are pending	CONFIRM has been received; CONFIRMED/response has not bee sent
Confirmed	CONFIRMED/responses have been received from all Inferiors ~~has been sent~~	CONFIRMED/response has been sent
Cancelling	Cancel decision has been made but responses from inferiors are pending	CANCEL has been received or auto-cancel has been decided
Cancelled	CANCELLED has been ~~sent~~received from all Inferiors	CANCELLED has been sent
cancel-contradiction	Not applicable	Autonomous cancel decision was made, CONFIRM received; CONTRADICTION has not been received
confirm-	Not applicable	Autonomous confirm decision

status value	Meaning from ~~Decider~~Superior	Meaning from Inferior
contradiction		was made, CANCEL received; CONTRADICTION has not been received
Hazard	A hazard has been reported from at least one Inferior	A hazard has been discovered; CONTRADICTION has not been receive d
Contradicted	Not applicable	CONTRADICTION has been received
Unknown	No state information for the ~~transaction~~ target-identifier exists~~; no such Decider exists~~	No state information for the target-identifier exists~~; no such Inferior exists~~
Inaccessible	There may be state information for this target-identifier but it cannot be reached/existence cannot be determined	There may be state information for this target-identifier but it cannot be reached/existence cannot be determined

qualifiers standardised or other qualifiers.

Types of FAULT possible ~~(sent to address as decider)~~

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

FAULT

Sent in reply to various messages to report an error condition

Parameter	**Type**
target address	BTP address
superior identifier	Identifier
inferior identifier	Identifier
fault type	See below
fault data	See below
qualifiers	List of 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

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:

fault type	**meaning**	**fault data**
CommunicationFailure	Any fault arising from the carrier mechanism and communication infrastructure.	Determined by the carrier mechanism and binding specification
DuplicateInferior	An inferior with the same address and identifier is already enrolled with this Superior	The identifier
General	Any otherwise unspecified problem	Free text explanation
InvalidDecider	The address the message was sent to is not valid (at all or for this Terminator and transaction identifier)	The address
InvalidInferior	The Superior is known but the Inferior identified by the address-as-inferior and identifier are not enrolled in it	The Inferior Identity (address-as-inferior and identifier)
InvalidSuperior	The received identifier is not known or does not identify a known Superior	The identifier
StatusRefused	The receiver will not report the request status (or inferior statuses) to this StatusRequestor	Free text explanation
InvalidTerminator	The address the message was sent to is not valid (at all or for this Decider and transaction identifier)	The address
UnknownParameter	A BTP message has been received with an unrecognised parameter	Free text explanation
UnknownTransaction	The transaction-identifier is unknown	The transaction-identifier
UnsupportedQualifier	A qualifier has been received that is not recognised and on which “must-be-Understood” is “true”.	Qualifier group and name
WrongState	The message has arrived when the recipient is in an invalid state.	

UnknownParameter	A BTP message has been received with an unrecognised parameter	Free text explanation

q
u
Qualifiers standardised or other qualifiers.

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 ~~involved~~used in the ~~Superior:Inferior~~ 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.

Parameter	**type**
target address	BTP address
superior identifier	Identifier
reply requested	Boolean
reply address	BTP address
address-as-inferior	Set of BTP addresses
inferior identifier	Identifier
Qualifiers	List of qualifiers

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

superior identifier. The superior identifier as on the CONTEXT message

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

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

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

inferior identifier an identifier that unambiguously identifies this Inferior within the scope of any of the address-as-inferior set of BTP-addresses.

qualifiers standardised or other qualifiers. The standard qualifier "Inferior name" may be present.

Types of FAULT possible (sent to Reply address)

General
InvalidSuperior – if superior identifier is unknown
DuplicateInferior – if inferior with at least one of the set address-as-inferior 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 P REPARED 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 "reply requested" having the value "true"; ENROL/no-rsp-req refers to an ENROL message with "reply 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)

Parameter	**Type**
target address	BTP address
inferior identifier	Identifier

Parameter	Type
inferior-handle	Handle
Qualifiers	List of 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 address-as-inferiors if the reply address was empty)

inferior identifier The inferior identifier as on the ENROL message

inferior handle the inferior handle that will identify this newly enrolled Inferior in the inferiors-list parameters in messages between the Superior (acting as a Decider) and its Terminator. This parameter is optional. The value shall be different for each enrolled Inferior of the Superior.

qualifiers standardised or other qualifiers.

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.

Parameter	type
target address	BTP address
superior identifier	identifier
address-as-inferior	Set of BTP addresses
inferior identifier	identifier
response requested	Boolean
Qualifiers	List of qualifiers

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

superior-identifier The superior identifier as on the ENROL message

address-as-inferior The address-as-inferior as on the earlier ENROL message (with the inferior identifier, this determines who the message is from)

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

response-requested is set to "true" if a RESIGNED response is required.

qualifiers standardised or other qualifiers.

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

Types of FAULT possible (sent to address-as-inferior)

General
InvalidSuperior – if superior identifier is unknown
InvalidInferior – if no ENROL had been received for this address-as-inferior and identifier (Inferior Identity)
WrongState – if a PREPARED or CANCELLED has already been received by the Superior from this Inferior

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

RESIGNED

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

Parameter	**Type**
target address	BTP address
inferior identifier	Identifier
qualifiers	List of qualifiers

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

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

qualifiers standardised or other qualifiers.

After receiving this message the Inferior will not receive any more messages with this address-as-inferior and identifier.

No FAULT messages are issued on receiving RESIGNED.

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.

~~Sent from a Terminator to a 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 Composer. 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 Composer.~~

Parameter	Type
target address	BTP address
inferior identifier	Identifier
~~reply address~~	~~BTP address~~
~~transaction-identifier~~	~~Identifier~~
~~inferiors-list~~	~~List of inferior handles~~
qualifiers	List of qualifiers

target address the address to which the PREPARE message is sent. When sent from Superior to Inferior, this will be the address-as-inferior from the ENROL message. When sent from Terminator to Composer, this will be the decider-~~address from the BEGUN message .~~

inferior identifier When sent from Superior to Inferior, the inferior identifier as on the earlier ENROL message. ~~This parameter shall be absent when sent from Terminator to Composer.~~

~~**reply address** When sent from Terminator to Composer, the address of the Terminator sending the PREPARE message. This parameter shall be absent when sent from Superior to Inferior.~~

transaction identifier When sent from Terminator to Composer, identifies the ~~Composer and will be the transaction-identifier from the BEGUN message.. This parameter shall be absent when sent from Superior to Inferior.~~

~~**inferiors-list** When sent from Terminator to Composer, defines which of the Inferiors of this Composer preparation is requested for. If this parameter is absent~~

~~when sent to a Composer, the PREPARE applies to all Inferiors. This parameter shall be absent when sent from Superior to Inferior.~~

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

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

~~When sent to a Composer, 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 Composer shall issue PREPARE. It will reply to the Terminator, using the reply address on the PREPARE 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).~~

Types of FAULT possible (sent to Superior address)

General
~~***UnknownTransaction*** – if the transaction-identifier is unknown~~
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~~; if a REQUEST_CONFIRM or CANCEL/whole has already been received by this Composer~~.

~~The form PREPARE/whole refers to a PREPARE message sent to a Composer where the "inferiors-list" parameter is absent. The form PREPARE/inferiors refers to a PREPARE message sent to a Composer where the "inferiors-list" parameter is present. The unqualified form PREPARE is used for a PREPARE message sent to an 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.

Parameter	Type
target address	BTP address
superior identifier	Identifier
address-as-inferior	Set of BTP addresses

inferior identifier	Identifier
default is cancel	Boolean
qualifiers	List of qualifiers

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

superior identifier When the message is sent from an Inferior to the Superior, the superior identifier as on the ENROL message

address-as-inferior When the message is sent from an Inferior to the Superior, the address-as-inferior as on the earlier ENROL message (with the inferior identifier, this determines who the message is from)

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.

On sending a PREPARED, the Inferior undertakes to maintain its ability to confirm or cancel the effects of the associa ted 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 address-as-inferior)

General
InvalidSuperior – if Superior identifier is unknown
InvalidInferior – if no ENROL has been received for this address-as-inferior and 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.

Parameter	Type
target address	BTP address
inferior identifier	Identifier
qualifiers	List of qualifiers

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

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

qualifiers standardised or other qualifiers.

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 Superior 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.

~~CONFIRMED is also sent by Decider to a Terminator in reply to REQUEST_CONFIRM if all of the confirm-set confirms (and, for a Cohesion, all other Inferiors cancel) without reporting hazards.~~

Parameter	Type
target address	BTP address
superior identifier	Identifier

Parameter	**Type**
address-as-inferior	Set of BTP addresses
inferior identifier	Identifier
~~address-as-decider~~	~~BTP address~~
~~transaction-identifier~~	~~identifier~~
confirm received	Boolean
qualifiers	List of qualifiers

target address the address to which the CONFIRMED is sent. When sent by an Inferior to a Superior, this will be the Superior address as on the CONTEXT message. When sent from a Decider to a Terminator it will be the reply address ~~from the REQUEST_CONFIRM message.~~

superior identifier When the message is sent from an Inferior to the Superior, this shall be the superior identifier as on the CONTEXT message. ~~This parameter shall be absent when CONFIRMED is sent from Decider to Terminator.~~

address-as-inferior When the message is sent from an Inferior to the Superior, this shall be the address-as-inferior as on the earlier ENROL message (with the inferior identifier, this determines who the message is from). ~~This parameter shall be absent when CONFIRMED is sent from Decider to Terminator.~~

inferior identifier When the message is sent from an Inferior to the Superior, this shall be the inferior identifier as on the earlier ENROL message. This parameter ~~shall be absent when CONFIRMED is sent from Decider to Terminator.~~

~~**address-as-decider** When the message is sent from a Decider to the Terminator, this shall be the address-as-decider of the Decider as on the BEGUN message (with the transaction-identifier, this determines who the message is from). This parameter shall be absent when CONFIRMED is sent from an Inferior to Superior.~~

~~**transaction identifier** When the message is sent from a Decider to the Terminator, this shall be the transaction identifier as on the BEGUN message (i.e. the identifier of the Decider as a whole). This parameter shall be absent when CONFIRMED is sent from an Inferior to Superior~~

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). ~~This parameter shall be absent when CONFIRMED is sent from Decider to Terminator.~~

qualifiers standardised or other qualifiers.

Types of FAULT possible (sent to address-as-inferior)

General
InvalidSuperior – if Superior identifier is unknown
InvalidInferior – if no ENROL has been received for this address-as-inferior and 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”. ~~The unqualified form CONFIRMED refers to the message without an confirm received parameter, as used between Decider and Terminator.~~

CANCEL

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

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

Parameter	Type
target address	BTP address
inferior identifier	Identifier
~~reply address~~	~~BTP address~~
~~transaction identifier~~	~~Identifier~~
~~inferiors-list~~	~~List of inferior handles~~
qualifiers	List of qualifiers

target address the address to which the CANCEL message is sent. When sent from Superior to Inferior, this will be the address-as-inferior from the ENROL message. ~~When sent from Terminator to Composer, this will be the decider-address from the BEGUN message.~~

inferior identifier When sent from Superior to Inferior, the inferior identifier as on the earlier ENROL message. ~~This parameter shall be absent when sent from Terminator to Decider.~~

~~**reply address** When sent from Terminator to Decider, the address of the Terminator sending the CANCEL message. This parameter shall be absent when sent from Superior to Inferior.~~

~~**transaction identifier** When sent from Terminator to Decider, identifies the Decider and will be the transaction-identifier from the BEGUN message.. This parameter shall be absent when sent from Superior to Inferior.~~

~~**inferiors-list** When sent from Terminator to Composer, defines which of the Inferiors of this Composer are to be cancelled. This parameter shall be absent when sent from a Superior to an Inferior and when sent from a Terminator to a Coordinator.~~

qualifiers standardised or other qualifiers.

When sent to 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.

~~When sent to a Decider with the inferiors-list parameter is absent, 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.~~

~~When sent to a Composer, with the inferiors-list parameter present, 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 issued to a Cohesion Composer identifying all of its currently enrolled Inferiors will leave the Cohesion 'empty', but permitted to continue with new Inferiors, if any enrol.~~

Types of FAULT possible (sent to Superior address)

General
~~***UnknownTransaction*** – if the transaction-identifier is unknown~~
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~~; if a REQUEST_CONFIRM has been received by this Composer~~.

~~The form CANCEL/whole refers to a CANCEL message sent to a Decider where the "inferiors-list" parameter is absent. The form CANCEL/inferiors refers to a CANCEL~~ message sent to a Composer where the "inferiors-list" parameter is present. The unqualified ~~form CANCEL is used to refer to a CANCEL message sent from a Superior to an 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.

~~CANCELLED is also sent by Decider to a Terminator in reply to REQUEST_CONFIRM if all Inferiors cancel without reporting hazards.~~

Parameter	
target address	BTP address
superior identifier	Identifier
address-as-inferior	Set of BTP address
inferior identifier	Identifier
~~address-as-decider~~	~~BTP address~~
~~transaction-identifier~~	~~identifier~~
qualifiers	List of qualifiers

target address the address to which the CANCELLED is sent. When sent by an Inferior to a Superior, this will be the Superior address as on the CONTEXT message. ~~When sent from a Decider to a Terminator it will be the reply address from the REQUEST_CONFIRM message.~~

superior identifier When the message is sent from an Inferior to the Superior, this shall be the superior identifier as on the CONTEXT message.~~ This parameter shall be absent when CANCELLED is sent from Decider to Terminator.~~

address-as-inferior When the message is sent from an Inferior to the Superior, this shall be the address-as-inferior as on the earlier ENROL message (with the inferior identifier, this determines who the message is from).~~ This parameter shall be absent when CANCELLED is sent from Decider to Terminator.~~

inferior identifier When the message is sent from an Inferior to the Superior, this shall be the inferior identifier as on the earlier ENROL message.~~ This parameter shall be absent when CANCELLED is sent from Decider to Terminator.~~

address-as-decider When the message is sent from a Decider to the ~~Terminator, this shall be the address-as-decider of the Decider as on the BEGUN message (with the transaction identifier, this determines who the message is from). This parameter shall be absent when CANCELLED is sent from an Inferior to Superior.~~

~~**transaction identifier** When the message is sent from a Decider to the Terminator, this shall be the transaction identifier as on the BEGUN message (i.e. the identifier of the Decider as a whole). This parameter shall be absent when CANCELLED is sent from an Inferior to Superior~~

qualifiers standardised or other qualifiers.

Types of FAULT possible (sent to address-as-inferior)

General
InvalidSuperior – if Superior identifier is unknown
InvalidInferior – if no ENROL has been received for this address-as-inferior and 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.

Parameter	**Type**
target address	BTP address
inferior identifier	Identifier
report-hazard	boolean
qualifiers	List of qualifiers

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

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.

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 Superior address)

General
InvalidInferior – if inferior identifier is unknown
WrongState – if a PREPARE has already been received from 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 (either the received decision of the superior or its own autonomous 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.

Parameter	**Type**
target address	BTP address
superior identifier	Identifier
address-as-inferior	Set of BTP addresses
inferior identifier	Identifier
level	mixed/possible
Qualifiers	List of qualifiers

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

superior identifier The superior identifier as used on the ENROL message

address-as-inferior The address-as-inferior as on the earlier ENROL message (with the inferior identifier, this determines who the message is from)

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.

Types of FAULT possible (sent to address-as-inferior)

General
InvalidSuperior – if Superior identifier is unknown
InvalidInferior – if no ENROL has been received for this address-as-inferior and 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	**Type**

target address	BTP address
inferior identifier	Identifier
Qualifiers	List of qualifiers

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

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

qualifiers standardised or other qualifiers.

Types of FAULT possible (sent to Superior 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	**Type**
target address	BTP address
inferior identifier	Identifier
Status	*see below*
reply requested	Boolean
Qualifiers	List of qualifiers

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

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.

status value	**Meaning**
active	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)
prepared-received	PREPARED has been received from the Inferior, but no outcome is yet available
inaccessible	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
unknown	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

Reply 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.

qualifiers standardised or other qualifiers.

The Inferior, on receiving SUPERIOR_STATE with reply 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 "reply requested" = "false". SUPERIOR_STATE/abcd/y refers to a similar message, but with “reply requested” = “true”. The form SUPERIOR_STATE/*/y refers to a SUPERIOR_STATE message with “reply 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.

Parameter	Type
target address	BTP address
superior identifier	Identifier
address-as-inferior	BTP address
inferior identifier	Identifier
Status	*see below*
reply requested	Boolean
Qualifiers	List of qualifiers

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

superior identifier The superior identifier as used on the ENROL message

address-as-inferior The address-as-inferior as on the ENROL message (with the inferior identifier, this determines who the message is from)

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

status value	meaning/previous message sent
active	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.
inaccessible	The state information for the relationship with the Superior, if it exists, cannot be accessed at the moment. This should be a transient condition

unknown	The Inferior is not known – it does not exist from the perspective of the Superior. The Inferior can be treated as cancelled

reply 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". Can only be "true" if "status" is "active".

qualifiers standardised or other qualifiers.

The Superior, on receiving INFERIOR_STATE with "reply 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 targetted 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 application elements may be continued, with new application 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 "reply requested" = "false". INFERIOR_STATE/abcd/y refers to a similar message, but with "reply requested" = "true". The form INFERIOR_STATE/*/y refers to a INFERIOR_STATE message with "reply requested" = "true" and any value for status.

~~CHECK_STATUS~~

~~Sent to an Inferior to ask it to reply with STATUS.~~

~~Parameter~~	~~Type~~
~~target address~~	~~BTP address~~

~~reply address~~	~~BTP address~~
~~inferior identifier~~	~~Identifier~~
~~Qualifiers~~	~~List of qualifiers~~

~~**target address** the address to which the CHECK_STATUS message is sent. This will be the address-as-inferior on the ENROL message.~~

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

~~**inferior identifier** This will be the "inferior-identifier" on the ENROL message.~~

~~**qualifiers** standardised or other qualifiers.~~

~~Types of FAULT possible (sent to reply address)~~

~~***General***~~

~~**STATUS**~~

~~Sent by an Inferior in reply to a CHECK_STATUS, reporting the overall state of the transaction tree node represented by the Inferior.~~

~~**Parameter**~~	~~**Type**~~
~~target address~~	~~BTP address~~
~~address-as-inferior~~	~~BTP address~~
~~inferior identifier~~	~~Identifier~~
~~Status~~	~~See below~~
~~Qualifiers~~	~~List of qualifiers~~

~~**target address** the address to which the STATUS is sent. This will be the reply address on the CHECK_STATUS message~~

~~**address-as-inferior** This will be the address-as-inferior as on the ENROL message (with the inferior-identifier, this determines who the message is from).~~

~~**inferior-identifier** This will be the inferior-identifier as on the ENROL message.~~

~~**status** states the current status of the transaction tree node represented by the sender.~~

~~**status value**~~	~~**Meaning from Inferior**~~

~~**status value**~~	~~**Meaning from Inferior**~~
~~*Created*~~	~~The Inferior exists (and is addressable) but it has not been enrolled with a Superior~~
~~*Enrolling*~~	~~ENROL has been sent, but ENROLLED is awaited~~
Active	The Inferior is enrolled
~~*Resigning*~~	~~RESIGN has been sent; RESIGNED is awaited~~
~~*Resigned*~~	~~RESIGNED has been received~~
~~*Preparing*~~	~~PREPARE has been received; PREPARED has not been sent~~
~~*Prepared*~~	~~PREPARED has been sent; no outcome has been received or autonomous decision made~~
~~*Confirming*~~	~~CONFIRM has been received; CONFIRMED/response has not bee sent~~
~~*Confirmed*~~	~~CONFIRMED/response has been sent~~
~~*Cancelling*~~	~~CANCEL has been received or auto-cancel has been decided~~
~~*Cancelled*~~	~~CANCELLED has been sent~~
~~*cancel-contradiction*~~	~~Autonomous cancel decision was made, CONFIRM received; CONTRADICTION has not been received~~
~~*confirm-contradiction*~~	~~Autonomous confirm decision was made, CANCEL received; CONTRADICTION has not been received~~
~~*Hazard*~~	~~A hazard has been discovered; CONTRADICTION has not been received~~
~~*Contradicted*~~	~~CONTRADICTION has been received~~
~~*Unknown*~~	~~No state information for the identifier exists; no such Inferior exists~~

~~status value~~	~~Meaning from Inferior~~
~~*Inaccessible*~~	~~There may be state information for this identifier but it cannot be reached/existence cannot be determined~~

qualifiers ~~standardised or other qualifiers.~~

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).

Parameter	Type
target address	BTP address
superior identifier	Identifier
inferior identifier	Identifier
old address	Set of BTP addresses
new address	Set of BTP addresses
qualifiers	List of qualifiers

target address the address to which the REDIRECT 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

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 addresses match one that is already known.

new address The (set of alternatives) new addresses to be used for messages sent to this entity.

qualifiers standardised or other qualifiers.

If the actor whose address is changed is an Inferior, the new address value replaces the address-as-inferior 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 ~~involved~~used in control ~~in the Initiator:Factory~~ 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).

Parameter	Type
target address	BTP address
reply address	BTP address
transaction type	cohesion/atom
qualifiers	List of qualifiers

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.

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.

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

BEGUN

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

Parameter	**Type**
target address	BTP address
address-as-decider	Set of BTP addresses
transaction-identifier	Identifier
inferior-handle	Handle
address-as-inferior	Set of BTP addresses
qualifiers	List of qualifiers

target address the address to which the BEGUN is sent. This will be the reply address from the BEGIN.

address-as-decider for a top-level transaction (no CONTEXT related to the BEGIN), this is the address to which ~~REQUEST_PREPARE~~PREPARE_INFERIORS, ~~REQUEST_CONFIRM~~CONFIRM_TRANSACTION, ~~REQUEST_~~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

transaction-identifier identifies the new Decider (Composer or Coordinator) within the scope of the address-as-decider. If this is not a top-level transaction, the transaction-identifier is optional, but if present shall be the inferior-identifier used in the enrolment with the Superior identified by the CONTEXT related to the BEGIN.

inferior handle Shall be absent if this is a top-level transaction and may or may not be present otherwise. (Presence or absence will be determined by the nature of the Superior identified in the CONTEXT related to the BEGIN). If present, the inferior handle will identify this new business transaction as in the inferiors-list parameters in messages between the Superior identified in the CONTEXT related to the BEGIN (acting as a Decider) and its Terminator. The value shall be different for each enrolled Inferior of that Superior.

address-as-inferior This parameter shall be absent if this is a top-level transaction and may be present, at implementation option otherwise. If present, it shall be the address-as-inferior used in the enrolment with the Superior identified by the CONTEXT related to the BEGIN. If this is a top-level transaction

qualifiers standardised or other qualifiers.

At implementation option, the "address-as-decider" and/or "address-as-inferior" and the "address-as-superior" 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 inferior identifier on messages will ensure they are applied to the appropriate Composer or Coordinator).

No FAULT messages are issued on receiving BEGUN.

~~REQUEST_PREPARE~~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).

Parameter	Type
target address	BTP address
reply address	BTP address
transaction-identifier	Identifier
inferiors-list	List of inferior handles
qualifiers	List of qualifiers

target address the address to which the ~~REQUEST_PREPARE~~PREPARE_INFERIORS message is sent. ~~When sent from Terminator to Decider, t~~This will be the decider-address from the BEGUN message.

reply address ~~When sent from Terminator to Decider,~~ the address of the Terminator sending the ~~PREPARE~~PREPARE_INFERIORS message.

transaction identifier ~~When sent from Terminator to Decider,~~ identifies the Decider and will be the transaction-identifier from the BEGUN message.

inferiors-list ~~When sent from Terminator to Decider,~~ defines which of the Inferiors of this Decider preparation is requested for. If this parameter is absent ~~when sent to a Decider~~, the PREPARE applies to all Inferiors.

qualifiers standardised or other qualifiers.

~~When sent to a Decider, f~~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 ~~REQUEST_PREPARE~~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).

Types of FAULT possible (sent to Superior address)

General
InvalidDecider – if Decider address is unknown
UnknownTransaction – if the transaction-identifier is unknown
InvalidInferior – if an inferior-handle on the inferiors-list is unknown
WrongState – if a ~~REQUEST_CONFIRM~~CONFIRM_TRANSACTION or CANCEL_TRANSACTION~~/whole~~ has already been received by this Composer.

The form ~~REQUEST_PREPARE~~PREPARE_INFERIORS/~~whole~~all refers to a ~~REQUEST_PREPARE~~PREPARE_INFERIORS message ~~sent to a Decider (Composer)~~ where the "inferiors-list" parameter is absent. The form ~~REQUEST_PREPARE~~PREPARE_INFERIORS/~~inferiors~~specific refers to a ~~REQUEST_PREPARE~~PREPARE_INFERIORS message ~~sent to a Decider (Composer)~~ where the "inferiors-list" parameter is present.

~~REQUEST_CONFIRM~~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.

Parameter	**Type**
target address	BTP address

reply address	BTP address
transaction identifier	Identifier
inferiors-list	List of inferior handles
~~Report~~ report-hazard	Boolean
Qualifiers	List of qualifiers

target address the address to which the ~~REQUEST_CONFIRM~~CONFIRM_TRANSACTION message is sent. This will be the address-as-decider on the BEGUN message.

reply address the address of the Terminator sending the ~~REQUEST_CONFIRM~~CONFIRM_TRANSACTION message.

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. 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 CONFIRM~~ED~~_COMPLETE or CANCEL~~LED~~_COMPLETE as soon as the decision for the transaction is known.

qualifiers standardised or other qualifiers.

If the "inferiors-list" parameter is present, the Inferiors identified shall be the "confirm-set" of the Cohesion. It 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 CONFIRM~~ED~~ COMPLETE message shall be sent to the “reply-address”.

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

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

Types of FAULT possible (sent to reply address)

General
InvalidDecider – if Decider address is unknown
UnknownTransaction – if the transaction-identifier is unknown
InvalidInferior – if an inferior handle in the inferiors-list is unknown
WrongState – if a ~~REQUEST_~~CANCEL_TRANSACTION~~/whole~~ has already been received .

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

~~CONFIRM_COMPLETE~~TRANSACTION_CONFIRMED

A Decider sends ~~CONFIRM_COMPLETE~~TRANSACTION_CONFIRMED to a Terminator in reply to ~~REQUEST_CONFIRM~~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”.~~.~~

Parameter	Type

Parameter	**Type**
target address	BTP address
address-as-decider	BTP address
transaction-identifier	identifier
qualifiers	List of qualifiers

target address the address to which the ~~CONFIRM_COMPLETE~~TRANSACTION_CONFIRMED is sent., ~~When sent from a Decider to a Terminator it~~this will be the reply address from the ~~REQUEST_CONFIRM~~CONFIRM_TRANSACTION message.

address-as-decider ~~When the message is sent from a Decider to the Terminator, this shall be~~ the address-as-decider of the Decider as on the BEGUN message (with the transaction identifier, this determines who the message is from).

transaction identifier ~~When the message is sent from a Decider to the Terminator, this shall be~~ the transaction identifier as on the BEGUN message (i.e. the identifier of the Decider as a whole).

qualifiers standardised or other qualifiers.

Types of FAULT possible (sent to address-as-decider)

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

~~Note – A CONFIRM_COMPLETE message arriving before a REQUEST_CONFIRM message is sent, or after a REQUEST_CANCEL has been sent will occur when the Decider has taken an autonomous decision and is not regarded as occurring in the wrong state.~~

CANCEL_TRANSACTION

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

Parameter	**Type**
target address	BTP address
reply address	BTP address

transaction identifier	Identifier
report-hazard	Boolean
qualifiers	List of 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.

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.

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.

Types of FAULT possible (sent to Superior address)

General
InvalidDecider – if Decider address is unknown
UnknownTransaction – if the transaction-identifier is unknown
WrongState – if a CONFIRM_TRANSACTION has been received by this Composer.

~~REQUEST_CANCEL~~CANCEL_INFERIORS

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

Parameter	**Type**
target address	BTP address
reply address	BTP address

transaction identifier	Identifier
inferiors-list	List of inferior handles
qualifiers	List of qualifiers

target address the address to which the ~~REQUEST_CANCEL~~CANCEL_TRANSACTION message is sent. ~~When sent from Terminator to Decider, t~~This will be the decider-address from the BEGUN message.

reply address ~~When sent from Terminator to Decider,~~ the address of the Terminator sending the ~~REQUEST_CANCEL~~CANCEL_TRANSACTION message.

transaction identifier ~~When sent from Terminator to Decider,~~ identifies the Decider and will be the transaction-identifier from the BEGUN message.

inferiors-list ~~When sent from Terminator to Decider (Composer),~~ defines which of the Inferiors of this Decider are to be cancelled. ~~This parameter shall be absent when sent from a Terminator to a Decider (Coordinator).~~

qualifiers standardised or other qualifiers.

~~When sent to a Decider with the inferiors-list parameter is absent, 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.~~

~~When sent to a Decider (Composer), with the inferiors-list parameter present, only~~ Only the Inferiors identified in the inferiors-list are to be cancelled. Any other inferiors are unaffected by a ~~REQUEST_CANCEL~~CANCEL_INFERIORS~~/inferiors~~. Further Inferiors may be enrolled.

Note – A ~~REQUEST_CANCEL~~CANCEL_INFERIORS~~/inferiors issued to a Decider (Cohesion Composer) identifying~~ all of ~~its~~the currently enrolled Inferiors will leave the ~~C~~cohesion 'empty', but permitted to continue with new Inferiors, if any enrol.

Types of FAULT possible (sent to Superior address)

General
InvalidDecider – if Decider address is unknown
UnknownTransaction – if the transaction-identifier is unknown
InvalidInferior – if an inferior-handle on the inferiors-list is unknown

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

~~The form REQUEST_CANCEL/whole refers to a REQUEST_CANCEL message sent to a Decider where the "inferiors-list" parameter is absent. The form REQUEST_CANCEL/inferiors refers to a REQUEST_CANCEL message sent to a Decider (Composer) where the "inferiors-list" parameter is present.~~

~~CANCEL_COMPLETE~~TRANSACTION_CANCELLED

A Decider sends ~~CANCEL_COMPLETE~~TRANSACTION_CANCELLED to a Terminator in reply to REQUEST_~~CONFIRM~~CANCEL 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	
target address	BTP address
address-as-decider	BTP address
transaction-identifier	identifier
qualifiers	List of qualifiers

target address the address to which the ~~CANCEL_COMPLETE~~TRANSACTION_CANCELLED is sent. ~~When sent from a Decider to a Terminator it~~This will be the reply address from the CANCEL_TRANSACTION or ~~REQUEST_CONFIRM~~CONFIRM_TRANSACTION message.

address-as-decider ~~When the message is sent from a Decider to the Terminator, this shall be~~ the address-as-decider of the Decider as on the BEGUN message (with the transaction identifier, this determines who the message is from).

transaction identifier ~~When the message is sent from a Decider to the Terminator, this shall be~~ the transaction identifier as on the BEGUN message (i.e. the identifier of the Decider as a whole).

qualifiers standardised or other qualifiers.

Types of FAULT possible (sent to address-as-decider)

General
InvalidTerminator – if Terminator address is unknown

UnknownTransaction – if the transaction-identifier is unknown

---

~~Note – A CANCEL_COMPLETE message arriving before a REQUEST_CANCEL message is sent, or after a REQUEST_CONFIRM has been sent will occur when the Decider has taken an autonomous decision and is not regarded as occurring in the wrong state.~~

~~REQUEST_STATUSES~~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 an address-as-superior or address-as-inferior, 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.

Parameter	**Type**
target address	BTP address
reply address	BTP address
~~transaction~~ target-identifier	Identifier
inferiors-list	List of inferior handles
Qualifiers	List of qualifiers

target address the address to which the REQUEST_ STATUS message is sent. When used to a Decider, t~~T~~his will be the address-as-decider from the BEGUN message. Otherwise it may be an address-as-superior from a CONTEXT or address-as-inferior from an ENROL message.

reply address the address to which the replying INFERIOR_STATUSES is to be sent

~~transaction~~ target-identifier identifies the transaction (or transaction tree node) within the scope of the target address. ~~Decider.~~ When the message is used to a Decider, t~~T~~his 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 ~~Composer or Coordinator~~target are to be included in the INFERIOR_STATUSES. If the list is absent, the status of all enrolled inferiors will be reported.

qualifiers standardised or other qualifiers.

Types of FAULT possible (sent to reply-address)

General

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 REQUES_STATUSES from the Terminator.*

~~***InvalidDecider*** – if Decider address is unknown~~

UnknownTransaction – if the transaction-identifier is unknown

The form ~~REQUEST_STATUSES~~REQUEST_INFERIOR_STATUSES/all~~whole~~ refers to a REQUEST_STATUS with the inferiors-list absent. The form REQUEST_INFERIOR_STATUS/specific ~~inferiors~~ refers to a REQUEST_INFERIOR_STATUS with the inferiors-list present.

INFERIOR_STATUSES

Sent by a Decider to report the status of all or some of its inferiors in response to a ~~REQUEST_STATUSES~~REQUEST_INFERIOR_STATUSES, ~~REQUEST_PREPARE~~PREPARE_INFERIORS, ~~REQUEST_CANCEL~~CANCEL_INFERIORS, CANCEL_TRANSACTION with “report-hazard” value of “true”~~/inferiors~~ and ~~REQUEST_CONFIRM~~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.

Parameter	**Type**
target address	BTP address
responders-address~~-as-decider~~	BTP address
responders~~transaction~~-identifier	Identifier
status-list	Set of Status items - see below
general-qualifiers	List of qualifiers

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

responders-address~~**-as-decider**~~ If the sender is a Decider, t~~T~~he address-as-decider ~~of the Decider~~ as on the BEGUN message. Otherwise the address of the sender of this message – one of address-as-inferior, address-as-superior. With the responders-identifier, this determines who the message is from. ~~(with the transaction-identifier, this determines who the message is from)~~

transaction ~~responders-~~identifier If the sender is a Decider, ~~t~~The transaction identifier as on the BEGUN message ~~(i.e. the identifier of the Decider as a whole)~~. Otherwise, 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

Field	Type
Inferior-handle	Inferior handle, identifying which inferior this Status-item contains information for.
Status	One of the status values below (these are a subset of those for STATUS)
Qualifiers	A list of qualifiers as received from the particular inferior or associated with the inferior in earlier messages (e.g. an Inferior name qualifier).

The status value reports the current status of the particular inferior, as known to the Decider (Composer or Coordinator). Values are:

status value	Meaning
active	The Inferior is enrolled
resigned	RESIGNED has been received from the Inferior
preparing	PREPARE has been sent to the inferior, none of PREPARED, RESIGNED, CANCELLED, HAZARD have been received
prepared	PREPARED has been received
autonomously confirmed	CONFIRMED/auto has been received, no completion message has been sent
autonomously cancelled	PREPARED had been received, and since then CANCELLED has been received but no completion message has been sent
confirming	CONFIRM has been sent, no outcome reply has been received
confirmed	CONFIRMED/response has been received
cancelling	CANCEL has been sent, no outcome reply has been received
cancelled	CANCELLED has been received, and PREPARED was not received previously

status value	**Meaning**
cancel-contradiction	Confirm had been ordered (and may have been sent), but CANCELLED was received
confirm-contradiction	Cancel had been ordered (and may have been sent) but CONFIRM/auto was received
hazard	A HAZARD message has been received
invalid	No such inferior is enrolled (used only in reply to a REQUEST_INFERIOR_STATUSES/specific)

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.

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 address-as-decider)

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

~~**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).~~

~~**Parameter**~~	~~**Type**~~
~~target address~~	~~BTP address~~
~~superior identifier~~	~~Identifier~~
~~inferior identifier~~	~~Identifier~~
~~old address~~	~~Set of BTP addresses~~
~~new address~~	~~Set of BTP addresses~~
~~Qualifiers~~	~~List of qualifiers~~

target address the address to which the REDIRECT 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~~

~~**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 addresses match one that is already known.~~

~~**new address** The (set of alternatives) new addresses to be used for messages sent to this entity.~~

~~**qualifiers** standardised or other qualifiers.~~

~~If the actor whose address is changed is an Inferior, the new address value replaces the address-as-inferior 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).~~

FAULT

~~Sent in reply to various messages to report an error condition~~

~~**Parameter**~~	~~**Type**~~
target address	BTP address
~~superior identifier~~	~~Identifier~~
~~inferior identifier~~	~~Identifier~~
~~fault type~~	~~See below~~
~~fault data~~	~~See below~~
~~Qualifiers~~	~~List of 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~~

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:~~

~~**fault type**~~	~~**meaning**~~	~~**fault data**~~
~~*General*~~	~~Any otherwise unspecified problem~~	~~Free text explanation~~
~~*UnknownParameter*~~	~~A BTP message has been received with an unrecognised parameter~~	~~Free text explanation~~
~~*WrongState*~~	~~The message has arrived when the recipient is in an invalid state.~~	
~~*CommunicationFailure*~~	~~Any fault arising from the carrier mechanism and communication infrastructure.~~	~~Determined by the carrier mechanism and binding specification~~
~~*InvalidSuperior*~~	~~The received identifier is not known or does not identify a known Superior~~	~~The identiifier~~
DuplicateInferior	An inferior with the same ~~address and identifier is already enrolled with this Superior~~	The identiifier
~~*InvalidInferior*~~	~~The Superior is known but the Inferior identified by the address-as-inferior and identifier are not enrolled in it~~	~~The Inferior Identi ty (address-as-inferior and identifier)~~
~~*UnsupportedQualifier*~~	~~A qualifier has been received that is not recognised and on which "must-be-Understood" is "true".~~	~~Qualifier group and name~~

qualifiers standardised or other qualifiers.

> ~~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.~~

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 “completed” or “related” (as required by the application). If the value is “completed”, the actor receiving the group shall forward the ENROLs, but is not required to (though it may) prevent confirmation.

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 CANCLLED 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 addresses.

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-handle” received on the ENROLLED with the application message in a manner that is visible to the application receiving the message.

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: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:

Content field	**Type**
Timelimit	Integer

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:

Content field	Type
Timeout	Integer
IntendedDecision	"confirm" or "cancel"

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:

Content field	Type
MinimumTimeout	Integer

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 handle" 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 "inferior-handle" on ENROLLED and BEGUN, which is required to be unambiguous within the scope of the Decider). 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:

Content field	Type
inferior-name	String

Inferior name the name assigned to the enrolling Inferior.

State Tables

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.

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 "reply requested" equal to "false" are only sent when the other message with "reply requested" equal to "true" has been received and no other message has been sent.

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 2~~Table 2~~ and Table 3~~Table 3~~ 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 itself an Inferior (to another Superior entity), in a hierarchic 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.

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~~Table 1~~ defines the events corresponding to sending or receiving BTP messages and the disruption events. Table 2~~Table 2~~ describes the decision events for an Inferior, Table 3~~Table 3~~ those for a Superior.

The decision events for a Superior, defined in Table 3~~Table 3~~ 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 1 : send, receive and disruption events

Event name	Meaning
send/receive ENROL/rsp-req	send/receive ENROL with reply-requested = true
send/receive ENROL/no-rsp-req	send/receive ENROL with reply-requested = false
send/receive RESIGN/rsp-req	send/receive RESIGN with reply-requested = true
send/receive RESIGN/no-rsp-req	send/receive RESIGN with reply-requested = false
send/receive PREPARED	send/receive PREPARED, with default-cancel = false
send/receive PREPARED/cancel	send/receive PREPARED, with default-cancel = true
send/receive CONFIRMED/auto	send/receive CONFIRMED, with confirm-received = true
send/receive CONFIRMED/response	send/receive CONFIRMED, with confirm-received = false
send/receive HAZARD	send/receive HAZARD
send/receive INF_STATE/***/y	send/receive INFERIOR_STATE with status *** and reply-requested = true
send/receive INF_STATE/***	send/receive INFERIOR_STATE with status *** and reply-requested = false

Event name	Meaning
send/receive SUP_STATE/***/y	send/receive SUPERIOR_STATE with status *** and reply-requested = true ("prepared-rcvd" represents "prepared-received")
send/receive SUP_STATE/***	send/receive SUPERIOR_STATE with status *** and reply-requested = false ("prepared-rcvd" represents "prepared-received")
disruption ***	Loss of state – new state is state applying after any local recovery processes complete

Table 2 : Decision events for Inferior

Event name	Meaning
decide to resign	• Any associated operations have had no effect (data state is unchanged)).
decide to be prepared	• Effects of all associated operations can be confirmed or cancelled; • information to retain confirm/cancel ability has been made persistent
decide to be prepared/cancel	• As "decide to be prepared"; • the persistent information specifies that the default action will be to cancel
decide to confirm autonomously	• Decision to confirm autonomously has been made persistent; • the effects of associated operations will be confirmed regardless of failures
decide to cancel autonomously	• Decision to cancel autonomously has been made persistent • the effects of associated operations will be cancelled regardless of failures
apply ordered confirmation	• Effects of all associated operations have been confirm ed; • Persistent information is effectively removed
remove persistent information	• Persistent information is effectively removed;

Event name	Meaning
detect problem	• For at least some of the associated operations, EITHER o they cannot be consistently cancelled or consistently confirmed; OR o it cannot be determined whether they will be cancelled or confirmed • AND, information about this is not persistent
detect and record problem	• As for the first condition of "detect problem" • 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)

3223

3224 **Table 3: Decision events for a Superior**

Event name	Meaning
decide to ~~request~~ confirm one-phase	• All associated application messages to be sent to the service have been sent; • There are no other remaining Inferiors • If an atom, a~~A~~ll enrolments that would create other Inferiors have completed (no outstanding CONTEXT_REPLYs) • The Superior has been requested to confirm
decide to prepare	• All associated application messages to be sent to the service have been sent; • The Superior has been requested to prepare this Inferior
decide to confirm	• Either o PREPARED or PREPARED/cancel has been received from all other remaining Inferiors; AND o Superior has been requested to confirm; AND o persistent information records the confirm decision and identifies all remaining Inferiors; • Or o persistent information records an offer of confirmation and has been instructed to confirm
decide to cancel	• Superior has not offered confirmation; OR • Superior has offered confirmation and has been instructed to cancel; OR

Event name	Meaning
	• Superior has offered confirmation but has made an autonomous cancellation decision
remove confirm information	• Persistent information has been effectively removed;
record contradiction	• Information recording the contradiction has been persisted (to the degree considered appropriate)

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 configuratio n 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 4 : Superior states

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

3256 **Table 5 : Inferior states**

State	summary
i1	aware of CONTEXT
a1	enrolling
b1	enrolled
c1	resigning
d1	preparing
e1	prepared
e2	prepared,default to cancel
f1	confirming
f2	confirming after default cancel
g1	CANCEL received in prepared state
g2	CANCEL received in prepared/cancel state
h1	Autonomously confirmed
h2	autonomously confirmed, superior confirmed
j1	autonomously cancelled
j2	autonomously cancelled, superior cancelled
k1	autonomously cancelled, contradicted
k2	autonomously cancelled, CONTRADICTION received
l1	autonomously confirmed, contradicted
l2	autonomously confirmed, CONTRADICTION received
m1	confirmation applied
n1	cancelling
p1	hazard detected, not recorded
p2	hazard detected in prepared state, not recorded
q1	hazard recorded
s1	~~REQUEST~~ CONFIRM_ONE_PHASE received after prepared state
s2	~~REQUEST~~ CONFIRM_ONE_PHASE received
s3	~~REQUEST~~ CONFIRM_ONE_PHASE received, confirming
s4	~~REQUEST~~ CONFIRM_ONE_PHASE received, cancelling
s5	~~REQUEST~~ CONFIRM_ONE_PHASE received, hazard detected
s6	~~REQUEST~~ CONFIRM_ONE_PHASE received, hazard recorded
x1	completed, presuming abort
x2	completed, presuming abort after prepared/cancel

State	summary
y1	completed, queried
y2	completed, default cancel, a message received
z	completed
z1	completed with default cancel

Table 6: Superior state table – normal forward progression

	I1	A1	B1	C1	D1	E1	E2	F1	F2
receive ENROL/rsp-req	A1								
receive ENROL/no-rsp-req	B1								
receive RESIGN/rsp-req	Y1		C1	C1	C1				
receive RESIGN/no-rsp-req	Z		Z	Z	Z				
receive PREPARED	Y1		E1		E1	E1		F1	
receive PREPARED/cancel	Y1		E2		E2		E2	F1	
receive CONFIRMED/auto	Q1		H1		H1	H1		F1	
receive CONFIRMED/response								F2	F2
receive CANCELLED	Y1		Z		Z	J1	J1	K1	
receive HAZARD	P1	P1	P1		P1	P1	P1	P3	
receive INF_STATE/active/y	Y1	A1	B1		D1				
receive INF_STATE/active			B1		D1				
receive INF_STATE/unknown			Z	Z	Z				
send ENROLLED		B1							
send RESIGNED				Z					
send PREPARE					D1	E1	E2		
send CONFIRM_ONE_PHASE									
send CONFIRM								F1	
send CANCEL									
send CONTRADICTION									
send SUP_STATE/active/y			B1						
send SUP_STATE/active			B1						
send SUP_STATE/prepared-rcvd/y						E1	E2		
send SUP_STATE/prepared-rcvd						E1	E2		
send SUP_STATE/unknown									
decide to ~~request~~ confirm one-phase			S1			S1	S1		
decide to prepare			D1						
decide to confirm						F1	F1		
decide to cancel			G1		G1	G1	Z		
remove persistent information									Z
record contradiction									
disruption I	Z	Z	Z	Z	Z	Z	Z		F1
disruption II						D1	D1		
disruption III						B1	B1		
disruption IV									

Table 7: Superior state table – cancellation and contradiction

	G1	G2	G3	G4	H1	J1	K1	L1
receive ENROL/rsp-req								
receive ENROL/no-rsp-req								
receive RESIGN/rsp-req	G3	Z	G3					
receive RESIGN/no-rsp-req	Z	Z	Z					
receive PREPARED	G1	G2						
receive PREPARED/cancel	G1	G2						
receive CONFIRMED/auto	L1	L1			H1			L1
receive CONFIRMED/response								
receive CANCELLED	G4	Z		G4		J1	K1	
receive HAZARD	P4	P4						
receive INF_STATE/active/y	G1	G2						
receive INF_STATE/active	G1	G2						
receive INF_STATE/unknown	Z	Z	Z	Z				
send ENROLLED								
send RESIGNED								
send PREPARE								
send CONFIRM_ONE_PHASE								
send CONFIRM								
send CANCEL	G2	G2	Z	Z				
send CONTRADICTION								
send SUP_STATE/active/y								
send SUP_STATE/active								
send SUP_STATE/prepared-rcvd/y								
send SUP_STATE/prepared-rcvd								
send SUP_STATE/unknown								
decide to ~~request~~ confirm one-phase								
decide to prepare								
decide to confirm					F1	K1		
decide to cancel					L1	G4		
remove persistent information								
record contradiction							R1	R1
disruption I	Z	Z	Z	Z	Z	Z	F1	Z
disruption II			G2	G2	E1	E1		G2
disruption III					D1	D1		
disruption IV					B1	B1		

Table 8: Superior state table – hazard and request confirm

	P1	P2	P3	P4	Q1	R1	R2	S1
`receive ENROL/rsp-req`								
`receive ENROL/no-rsp-req`								
`receive RESIGN/rsp-req`								C1
`receive RESIGN/no-rsp-req`								Z
`receive PREPARED`								S1
`receive PREPARED/cancel`								S1
`receive CONFIRMED/auto`					Q1	R1	R1	S1
`receive CONFIRMED/response`					Z	R2		Z
`receive CANCELLED`						R1	R1	Z
`receive HAZARD`	P1	P2	P3	P4		R1	R1	Z
`receive INF_STATE/active/y`								S1
`receive INF_STATE/active`								S1
`receive INF_STATE/unknown`	P1	P2		P4		R2	R2	Z
`send ENROLLED`								
`send RESIGNED`								
`send PREPARE`								
`send CONFIRM_ONE_PHASE`								S1
`send CONFIRM`								
`send CANCEL`								
`send CONTRADICTION`						R2		
`send SUP_STATE/active/y`								
`send SUP_STATE/active`								
`send SUP_STATE/prepared-rcvd/y`								
`send SUP_STATE/prepared-rcvd`								
`send SUP_STATE/unknown`								
`decide to ~~request~~ confirm one-phase`								
`decide to prepare`								
`decide to confirm`								
`decide to cancel`								
`remove persistent information`							Z	
`record contradiction`	R1	R1	R1	R1	R1			
`disruption I`	Z	Z	Z	Z	Z		R1	Z
`disruption II`	D1		F1	G2				
`disruption III`	B1							
`disruption IV`								

Table 9: Superior state table – query after completion and completed states

	Y1	Z
receive ENROL/rsp-req		Y1
receive ENROL/no-rsp-req		Y1
receive RESIGN/rsp-req	Y1	Y1
receive RESIGN/no-rsp-req	Z	Z
receive PREPARED	Y1	Y1
receive PREPARED/cancel	Y1	Y1
receive CONFIRMED/auto	Q1	Q1
receive CONFIRMED/response	Z	Z
receive CANCELLED	Y1	Y1
receive HAZARD	P2	P2
receive INF_STATE/active/y	Y1	Y1
receive INF_STATE/active	Y1	Z
receive INF_STATE/unknown	Z	Z
send ENROLLED		
send RESIGNED		
send PREPARE		
send CONFIRM_ONE_PHASE		
send CONFIRM		
send CANCEL		
send CONTRADICTION		
send SUP_STATE/active/y		
send SUP_STATE/active		
send SUP_STATE/prepared-rcvd/y		
send SUP_STATE/prepared-rcvd		
send SUP_STATE/unknown	Z	
decide to ~~request~~ confirm one-phase		
decide to prepare		
decide to confirm		
decide to cancel		
remove persistent information		
record contradiction		
disruption I	Z	
disruption II		
disruption III		
disruption IV		

Table 10: Inferior state table – normal forward progression

	i1	a1	b1	c1	d1	e1	e2	f1	f2
send ENROL/rsp-req	a1								
send ENROL/no-rsp-req	b1								
send RESIGN/rsp-req				c1					
send RESIGN/no-rsp-req				z					
send PREPARED						e1			
send PREPARED/cancel							e2		
send CONFIRMED/auto									
send CONFIRMED/response									
send CANCELLED			z		z				
send HAZARD									
send INF_STATE/active/y		a1	b1		d1				
send INF_STATE/active			b1		d1				
send INF_STATE/unknown									
receive ENROLLED		b1							
receive RESIGNED				z					
receive PREPARE		d1	d1	c1	d1	e1	e2		
receive CONFIRM_ONE_PHASE		s2	s2	c1		s1	s1		
receive CONFIRM						f1	f2	f1	f2
receive CANCEL		n1	n1	z	n1	g1	g2		
receive CONTRADICTION									
receive SUP_STATE/active/y		b1	b1	c1		e1	e2		
receive SUP_STATE/active		b1	b1	c1		e1	e2		
receive SUP_STATE/prepared-rcvd/y						e1	e2		
receive SUP_STATE/prepared-rcvd						e1	e2		
receive SUP_STATE/unknown		z	z	z	z	x1	x2		
decide to resign			c1		c1				
decide to be prepared			e1		e1				
decide to be prepared/cancel			e2		e2				
decide to confirm autonomously						h1			
decide to cancel autonomously						j1	z1		
apply ordered confirmation								m1	m1
remove persistent information									
detect problem		p1	p1		p1	p2	p2	p2	p2
detect and record problem									
disruption I		z	z	z	z			e1	e2
disruption II					b1				
disruption III									

Table 11: Inferior state table – cancellation and contradiction

	g1	g2	h1	h2	j1	j2	k1	k2	l1	l2
send ENROL/rsp-req										
send ENROL/no-rsp-req										
send RESIGN/rsp-req										
send RESIGN/no-rsp-req										
send PREPARED										
send PREPARED/cancel										
send CONFIRMED/auto			h1						l1	
send CONFIRMED/response										
send CANCELLED					j1		k1			
send HAZARD										
send INF_STATE/active/y										
send INF_STATE/active										
send INF_STATE/unknown										
receive ENROLLED										
receive RESIGNED										
receive PREPARE			h1		j1					
receive CONFIRM_ONE_PHASE			s3		s4					
receive CONFIRM			h2	h2	k1		k1			
receive CANCEL	g1	g2	l1		j2	j2			l1	
receive CONTRADICTION			l2		k2		k2	k2	l2	l2
receive SUP_STATE/active/y			h1		j1					
receive SUP_STATE/active			h1		j1					
receive SUP_STATE/prepared-rcvd/y			h1		j1					
receive SUP_STATE/prepared-rcvd			h1		j1					
receive SUP_STATE/unknown	x1	x2	l1		j2	j2	k2	k2	l1	
decide to resign										
decide to be prepared										
decide to be prepared/cancel										
decide to confirm autonomously										
decide to cancel autonomously										
apply ordered confirmation										
remove persistent information	n1	n1		m1		z		z		z
detect problem	p2	p2								
detect and record problem										
disruption I	e1	e2		h1		j1	j1	k1	h1	l1
disruption II								j1		h1
disruption III										

Table 12: Inferior state table– confirm, cancel ordered and hazard recording

	m1	n1	p1	p2	q1
send ENROL/rsp-req					
send ENROL/no-rsp-req					
send RESIGN/rsp-req					
send RESIGN/no-rsp-req					
send PREPARED					
send PREPARED/cancel					
send CONFIRMED/auto					
send CONFIRMED/response	z				
send CANCELLED		z			
send HAZARD			p1	p2	q1
send INF_STATE/active/y					
send INF_STATE/active					
send INF_STATE/unknown					
receive ENROLLED			p1		q1
receive RESIGNED					
receive PREPARE			p1	p2	q1
receive CONFIRM_ONE_PHASE			s5	s5	s6
receive CONFIRM	m1			p2	q1
receive CANCEL		n1	p1	p2	q1
receive CONTRADICTION			z	z	z
receive SUP_STATE/active/y			p1	p2	q1
receive SUP_STATE/active			p1	p2	q1
receive SUP_STATE/prepared-rcvd/y				p2	q1
receive SUP_STATE/prepared-rcvd				p2	q1
receive SUP_STATE/unknown		z	p1	p2	q1
decide to resign					
decide to be prepared					
decide to be prepared/cancel					
decide to confirm autonomously					
decide to cancel autonomously					
apply ordered confirmation					
remove persistent information					
detect problem					
detect and record problem			q1	q1	
disruption I	z	z	z		
disruption II		d1			
disruption III		b1			

Table 13: Inferior state table – request confirm states

	s1	s2	s3	s4	s5	s6
send ENROL/rsp-req						
send ENROL/no-rsp-req						
send RESIGN/rsp-req						
send RESIGN/no-rsp-req						
send PREPARED						
send PREPARED/cancel						
send CONFIRMED/auto						
send CONFIRMED/response			z			
send CANCELLED				z		
send HAZARD					z	z
send INF_STATE/active/y						
send INF_STATE/active						
send INF_STATE/unknown						
receive ENROLLED						
receive RESIGNED						
receive PREPARE						
receive CONFIRM_ONE_PHASE	s1	s2	s3	s4	s5	s6
receive CONFIRM						
receive CANCEL						
receive CONTRADICTION			s3		z	s6
receive SUP_STATE/active/y						
receive SUP_STATE/active						
receive SUP_STATE/prepared-rcvd/y						
receive SUP_STATE/prepared-rcvd						
receive SUP_STATE/unknown	x1	z	z	z	z	z
decide to resign						
decide to be prepared						
decide to be prepared/cancel						
decide to confirm autonomously		s3				
decide to cancel autonomously		s4				
apply ordered confirmation						
remove persistent information	s2					
detect problem						
detect and record problem		s6				
disruption I	e1	z		z	z	
disruption II						
disruption III						

Table 14: Inferior state table– completed states (including presume -abort and queried)

	x1	x2	y1	y2	z	z1
send ENROL/rsp-req						
send ENROL/no-rsp-req						
send RESIGN/rsp-req						
send RESIGN/no-rsp-req						
send PREPARED						
send PREPARED/cancel						
send CONFIRMED/auto						
send CONFIRMED/response						
send CANCELLED				z1		
send HAZARD						
send INF_STATE/active/y						
send INF_STATE/active						
send INF_STATE/unknown			z			
receive ENROLLED					z	
receive RESIGNED			y1		z	
receive PREPARE			y1	y2	y1	z1
receive CONFIRM_ONE_PHASE			y1	y2	y1	y1
receive CONFIRM				y2	m1	y2
receive CANCEL			y1	z	y1	y1
receive CONTRADICTION			z	z	z	z
receive SUP_STATE/active/y			y1	y2	y1	y2
receive SUP_STATE/active			y1	y2	z	z1
receive SUP_STATE/prepared-rcvd/y				y2		y2
receive SUP_STATE/prepared-rcvd				y2		y2
receive SUP_STATE/unknown	x1	x2	y1	y2	z	z
decide to resign						
decide to be prepared						
decide to be prepared/cancel						
decide to confirm autonomously						
decide to cancel autonomously						
apply ordered confirmation						
remove persistent information	z	z				
detect problem						
detect and record problem						
disruption I	e1	e2				
disruption II						
disruption III						

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).

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)
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:

Inferior address (as on ENROL)
Inferior identifier (as on ENROL)

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 "reply-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 ~~REQUEST_CONFIRM~~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 ~~REQUEST_CONFIRM~~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:xml

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."

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:

```
<btp:some-address>
  <btp:binding-name>...carrier binding URI...</btp:binding-name>
  <btp:binding-address>...carrier specific
address...</btp:binding-address>
  <btp:additional-information>...optional additional addressing
information...</btp:additional-information> ?
</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:

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

<auth:username
       xmlns:auth="http://www.example.com/ns/auth"
       xmlns:btp="urn:oasis:names:tc:BTP:xml"
       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

Unspecified length strings made of up hexadecimal digits (0->9, A->F). Note: lower case a->f are not valid.

Examples: "01", "FAB224234CCCC2"

Note – Use of hexadecimal digits avoids problems with character-code representations. 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

```

<btp:context id? superior-type="cohesion|atom">
  <btp:superior-address>  +
    ...address...
  </btp:superior-address>
  <btp:superior-identifier>...hexstring...</btp:superior-
identifier>
  <btp:reply-address> ?
    ...address...
  </btp:reply-address>
  <btp:qualifiers> ?
    ...qualifiers...
  </btp:qualifiers>
</btp:context>

```

CONTEXT_-REPLY

```
<btp:context-reply id? superior-type="cohesion|atom">
  <btp:target-additional-information> ?
    ...additional address information...
  </btp:target-additional-information>
  <btp:superior-address>  +
           ...address...
  </btp:superior-address>
  <btp:superior-identifier>...hexstring...</btp:superior-
identifier>
```

```
  <completion-status>completed|related|repudiated</completion-
status>
  <btp:qualifiers> ?
    ...qualifiers...
  </btp:qualifiers>
</btp:context>
```

BEGIN

```
<btp:begin id? transaction-type=”cohesion|atom”>
  <btp:target-additional-information>
    ...additional address information...
  </btp:target-additional-information>
  <btp:reply-address>
    ...address...
  </btp:reply-address>
  <btp:qualifiers> ?
    ...qualifiers...
  </btp:qualifiers>
</btp:begin>
```

BEGUN

```
<btp:begun id? transaction-type=”cohesion|atom”>
  <btp:target-additional-information>
    ...additional address information...
  </btp:target-additional-information>
  <btp:decider-address> ?
    ...address...
  </btp:decider-address>
  <btp:transaction-identifier>...hexstring...</btp:transaction-
identifier> ?
  <btp:inferior-handle>...hexstring...</btp:inferior:handle> ?
  <btp:inferior-address> ?
    ...address...
  </btp:inferior-address>
  <btp:qualifiers> ?
    ...qualifiers...
  </btp:qualifiers>
</btp:begun>
```

ENROL

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

ENROLLED

```
<btp:enrolled id?>
<btp:target-additional-information>
    ...additional address information...
  </btp:target-additional-information>
  <btp:inferior-identifier>...hexstring...</btp:inferior-
identifier>
  <btp:inferior-handle>...hexstring...</btp:inferior:handle> ?
  <btp:qualifiers> ?
    ...qualifiers...
  </btp:qualifiers>
</btp:enrolled>
```

RESIGN

```
<btp:resign response-requested="true|false" id?>
<btp:target-additional-information>
    ...additional address information...
  </btp:target-additional-information>
  <btp:superior-identifier>...hexstring...</btp:superior-
identifier>
  <btp:inferior-address>  +
    ...address...
  </btp:inferior-address>
  <btp:inferior-identifier>...hexstring...</btp:inferior-
identifier>
  <btp:qualifiers> ?
    ...qualifiers...
  </btp:qualifiers>
</btp:resign>
```

RESIGNED

```
<btp:resigned id?>
  <btp:target-additional-information>
    ...additional address information...
  </btp:target-additional-information>
  <btp:inferior-identifier>...hexstring...</btp:inferior-
identifier>
  <btp:qualifiers> ?
    ...qualifiers...
  </btp:qualifiers>
</btp:resigned>
```

PREPARE

```
<btp:prepare id?>
  <btp:target-additional-information>
    ...additional address information...
  </btp:target-additional-information>
  <btp:inferior-identifier>...hexstring...</btp:inferior-
identifier> ?
```
~~<btp:reply-address> ?~~
~~...address...~~
~~</btp:reply-address>~~
~~<btp:transaction-identifier>...hexstring...</btp:transaction-identifier> ?~~
~~<btp:inferiors-list> ?~~
~~<btp:inferior-handle>...hexstring...</btp:inferior-handle> +~~
~~</btp:inferiors-list>~~
```
  <btp:qualifiers> ?
    ...qualifiers...
  </btp:qualifiers>
</btp:prepare>
```

PREPARED

```
<btp:prepared default-is-cancel="false|true" id?>
  <btp:target-additional-information>
    ...additional address information...
  </btp:target-additional-information>
  <btp:superior-identifier>...hexstring...</btp:superior-
identifier>
  <btp:inferior-address> +
    ...address...
  </btp:inferior-address>
  <btp:inferior-identifier>...hexstring...</btp:inferior-
identifier>
  <btp:qualifiers> ?
    ...qualifiers...
  </btp:qualifiers>
</btp:prepared>
```

CONFIRM

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

CONFIRMED

```
<btp:confirmed confirmed-received="true|false" id?>
  <btp:target-additional-information>
    ...additional address information...
  </btp:target-additional-information>
  <btp:superior-identifier>...hexstring...</btp:superior-
identifier>
  <btp:inferior-address> ?
    ...address...
  </btp:inferior-address>
  <btp:inferior-identifier>...hexstring...</btp:inferior-
identifier> ?
~~  <btp:decider-address> ?~~
~~    ...address...~~
~~  </btp:decider-address>~~
~~  <btp:transaction-identifier>...hexstring...</btp:transaction-~~
identifier> ?
  <btp:qualifiers> ?
    ...qualifiers...
  </btp:qualifiers>
</btp:confirmed>
```

CANCEL

```
<btp:cancel id?>
  <btp:target-additional-information>
    ...additional address information...
  </btp:target-additional-information>
  <btp:inferior-identifier>...hexstring...</btp:inferior-
identifier> ?
  <btp:reply-address>  ?
```

```
    ...address...
  </btp:reply-address>
~~  <btp:transaction-identifier>...hexstring...</btp:transaction-~~
~~identifier> ?~~
~~  <btp:inferiors-list> ?~~
~~       <btp:inferior-handle>...hexstring...</btp:inferior-handle>~~
~~  </btp:inferiors-list>~~
  <btp:qualifiers> ?
    ...qualifiers...
  </btp:qualifiers>
</btp:cancel>
```

CANCELLED

```
<btp:cancelled id?>
  <btp:target-additional-information>
    ...additional address information...
  </btp:target-additional-information>
  <btp:superior-identifier>...hexstring...</btp:superior-
identifier>
  <btp:inferior-address> +
    ...address...
  </btp:inferior-address> ?
  <btp:inferior-identifier>...hexstring...</btp:inferior-
identifier> ?
~~  <btp:decider-address> ?~~
~~    ...address...~~
~~  </btp:decider-address>~~
~~  <btp:transaction-identifier>...hexstring...</btp:transaction-~~
~~identifier> ?~~
  <btp:qualifiers> ?
    ...qualifiers...
  </btp:qualifiers>
</btp:cancelled>
```

CONFIRM_ONE_PHASE

```
<btp:confirm-one-phase report-hazard="true|false" id?>
  <btp:target-additional-information>
    ...additional address information...
  </btp:target-additional-information>
  <btp:inferior-identifier>...hexstring...</btp:inferior-
identifier>
  <btp:qualifiers> ?
    ...qualifiers...
  </btp:qualifiers>
</btp:confirm-one-phase>
```

HAZARD

```
<btp:hazard level=”mixed|possible” id?>
  <btp:target-additional-information>
    ...additional address information...
  </btp:target-additional-information>
  <btp:superior-identifier>...hexstring...</btp:superior-
identifier>
  <btp:inferior-address> +
    ...address...
  </btp:inferior-address>
  <btp:inferior-identifier>...hexstring...</btp:inferior-
identifier>
  <btp:qualifiers> ?
    ...qualifiers...
  </btp:qualifiers>
</btp:hazard>
```

CONTRADICTION

```
<btp:contradiction id?>
  <btp:target-additional-information>
    ...additional address information...
  </btp:target-additional-information>
  <btp:inferior-identifier>...hexstring...</btp:inferior-
identifier>
  <btp:qualifiers> ?
    ...qualifiers...
  </btp:qualifiers>
</btp:contradiction>
```

SUPERIOR_STATE

```
<btp:superior-state reply-requested="true|false" id?>
  <btp:target-additional-information>
    ...additional address information...
  </btp:target-additional-information>
  <btp:inferior-identifier>...hexstring...</btp:inferior-
identifier>
  <btp:status>active|prepared-
received|inaccessible|unknown</btp:status>
  <btp:qualifiers> ?
    ...qualifiers...
  </btp:qualifiers>
</btp:superior-state>
```

INFERIOR_STATE

```
<btp:inferior-state reply-requested="true|false" id?>
  <btp:target-additional-information>
    ...additional address information...
  </btp:target-additional-information>
  <btp:superior-identifier>...hexstring...</btp:superior-
identifier>
  <btp:inferior-address> +
    ...address...
  </btp:inferior-address>
  <btp:inferior-identifier>...hexstring...</btp:inferior-
identifier>
  <btp:status> active|_~~prepared 
received|~~inaccessible|unknown</btp:status>
  <btp:qualifiers> ?
    ...qualifiers...
  </btp:qualifiers>
</btp:inferior-state>
```

~~CHECK_STATUS~~

```
~~<btp:check_status id?>~~
~~  <btp:target-additional-information>~~
~~    ...additional address information...~~
~~  </btp:target-additional-information>~~
  <btp:reply-address>
~~    ...address...~~
~~  </btp:reply-address>~~
~~  <btp:inferior-identifier>...hexstring...</btp:inferior-~~
~~identifier> ?~~
~~  <btp:qualifiers> ?~~
~~    ...qualifiers...~~
~~  </btp:qualifiers>~~
~~</btp:check_status>~~
```

~~STATUS~~

```
~~<btp:status id?>~~
~~  <btp:target-additional-information>~~
~~    ...additional address information...~~
~~  </btp:target-additional-information>~~
~~  <btp:inferior-address> ?~~
~~    ...address...~~
~~  </btp:inferior-address>~~
~~  <btp:inferior-identifier>...hexstring...</btp:inferior-~~
~~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>~~
```

REDIRECT

```
<btp:redirect id?>
  <btp:target-additional-information>
     ...additional address information...
  </btp:target-additional-information>
  <btp:superior-identifier>...hexstring...</btp:superior-
identifier> ?
  <btp:inferior-identifier>...hexstring...</btp:inferior-
identifier>
  <btp:old-address>  +
     ...address...
  </btp:old-address>
  <btp:new-address>  +
     ...address...
  </btp:new-address>
  <btp:qualifiers> ?
     ...qualifiers...
  </btp:qualifiers>
</btp:redirect>
```

~~CONFIRM_ONE_PHASE~~

```
~~<btp:confirm-one-phase report-hazard="true|false" id?>~~
  <btp:target-additional-information>
~~     ...additional address information...~~
~~  </btp:target-additional-information>~~
~~  <btp:inferior-identifier>...hexstring...</btp:inferior-~~
~~identifier>~~
~~  <btp:qualifiers> ?~~
~~     ...qualifiers...~~
~~  </btp:qualifiers>~~
~~</btp:confirm-one-phase>~~
```

~~REQUEST_PREPARE~~PREPARE_INFERIORS

```
<btp: ~~request-~~prepare-inferiors id?>
  <btp:target-additional-information>
     ...additional address information...
  </btp:target-additional-information>
  <btp:reply-address>  ?
     ...address...
  </btp:reply-address>
  <btp:transaction-identifier>...hexstring...</btp:transaction-
identifier> ?
  <btp:inferiors-list> ?
       <btp:inferior-handle>...hexstring...</btp:inferior-handle>
+
  </btp:inferiors-list>
  <btp:qualifiers> ?
    ...qualifiers...
  </btp:qualifiers>
</btp:~~request-~~prepare-inferiors>
```

~~REQUEST_CONFIRM~~CONFIRM_TRANSACTION

```
<btp:~~request-~~confirm-transaction report-hazard="true|false" id?>
  <btp:target-additional-information>
    ...additional address information...
  </btp:target-additional-information>
  <btp:reply-address>
    ...address...
  </btp:reply-address>
  <btp:transaction-identifier>...hexstring...</btp:transaction-
identifier>
  <btp:inferiors-list> ?
       <btp:inferior-handle>...hexstring...</btp:inferior-handle>
+
  </btp:inferiors-list>
  <btp:qualifiers> ?
    ...qualifiers...
  </btp:qualifiers>
</btp:_~~request-~~confirm_transaction>
```

~~CONFIRM_COMPLETE~~TRANSACTION_CONFIRMED

```
<btp:transaction-confirmed-complete confirmed-
~~received="true|false"~~ id?>
  <btp:target-additional-information>
    ...additional address information...
  </btp:target-additional-information>
  <btp:decider-address> ?
    ...address...
  </btp:decider-address>
  <btp:transaction-identifier>...hexstring...</btp:transaction-
identifier> ?
  <btp:qualifiers> ?
    ...qualifiers...
  </btp:qualifiers>
</btp:transaction-confirmed~~-complete~~>
```

CANCEL_TRANSACTION

```
<btp:cancel_transaction id?>
  <btp:target-additional-information>
    ...additional address information...
  </btp:target-additional-information>
  <btp:reply-address>  ?
    ...address...
  </btp:reply-address>
  <btp:transaction-identifier>...hexstring...</btp:transaction-
identifier> ?
  <btp:qualifiers> ?
    ...qualifiers...
  </btp:qualifiers>
</btp:cancel_transaction>
```

~~REQUEST_CANCEL~~CANCEL_INFERIORS

```
<btp: ~~request~~-cancel-inferiors id?>
  <btp:target-additional-information>
    ...additional address information...
  </btp:target-additional-information>
  <btp:reply-address>  ?
    ...address...
  </btp:reply-address>
  <btp:transaction-identifier>...hexstring...</btp:transaction-
identifier> ?
  <btp:inferiors-list>~~ ?~~
        <btp:inferior-handle>...hexstring...</btp:inferior-handle>
  </btp:inferiors-list>
  <btp:qualifiers> ?
    ...qualifiers...
  </btp:qualifiers>
</btp:~~request-~~cancel-inferiors>
```

~~CANCEL_COMPLETE~~TRANSACTION_CANCELLED

```
<btp:cancel-complete id?>
  <btp:target-additional-information>
    ...additional address information...
  </btp:target-additional-information>
  <btp:decider-address> ?
    ...address...
  </btp:decider-address>
  <btp:transaction-identifier>...hexstring...</btp:transaction-
identifier> ?
  <btp:qualifiers> ?
    ...qualifiers...
  </btp:qualifiers>
</btp: cancel-complete>
```

~~REQUEST_STATUSES~~REQUEST_INFERIOR_STATUSES

```
<btp:request_statuses id?>
  <btp:target-additional-information>
    ...additional address information...
  </btp:target-additional-information>
  <btp:reply-address>
    ...address...
  </btp:reply-address>
  <btp:~~transaction~~target-
identifier>...hexstring...</btp:~~transaction~~target-identifier>
  <btp:inferiors-list> ?
       <btp:inferior-handle>...hexstring...</btp:inferior-handle>
+
  </btp:inferiors-list>
  <btp:qualifiers> ?
    ...qualifiers...
  </btp:qualifiers>
</btp:request_statuses>
```

INFERIOR_STATUSES

```
<btp:inferior_statuses id?>
  <btp:target-additional-information>
    ...additional address information...
  </btp:target-additional-information>
  <btp:~~decider~~responders-address>
    ...address...
  </btp:responders~~decider~~-address>
  <btp:~~transaction~~responders-
identifier>...hexstring...</btp:~~transaction~~responders-identifier>
  <btp:status-list>
       <btp:status-item> +
          <btp:inferior-handle>...hexstring...</btp:inferior-
handle>
          <btp:status>active|resigned|preparing|prepared|
               autonomously-confirmed|autonomously-cancelled|
               confirming|confirmed|cancelling|cancelled|
               cancel-contradiction|confirm-contradiction|
               hazard</btp:status>
          <btp:qualifiers> ?
               ...qualifiers...
         </btp:qualifiers>
       </btp:status-item>
  </btp:status-list>
  <btp:qualifiers> ?
    ...qualifiers...
  </btp:qualifiers>
</btp:inferior_statuses>
```

REQUEST_STATUS

```
<btp:request_status id?>
  <btp:target-additional-information>
    ...additional address information...
  </btp:target-additional-information>
  <btp:reply-address>
    ...address...
  </btp:reply-address>
  <btp:~~inferior~~target-
identifier>...hexstring...</btp:~~inferior~~target-identifier> ~~?~~
  ~~<btp:transaction-identifier>...hexstring...</btp:transaction-
identifier> ?~~
  <btp:qualifiers> ?
    ...qualifiers...
  </btp:qualifiers>
</btp:request_status>
```

STATUS

```
<btp:status id?>
  <btp:target-additional-information>
    ...additional address information...
  </btp:target-additional-information>
  <btp:~~inferior~~responder-address>~~ ?~~
    ...address...
  </btp:~~inferior~~responder-address>
  <btp:~~inferior~~responder-
identifier>...hexstring...</btp:~~inferior~~responder-identifier>~~ ?~~
  ~~<btp:decider-address> ?~~
~~    ...address...~~
  </btp:decider-address>
~~  <btp:transaction-identifier>...hexstring...</btp:transaction-
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>
```

~~REDIRECT~~

```
~~<btp:redirect id?>~~
```

```
~~  <btp:target-additional-information>~~
~~    ...additional address information...~~
~~  </btp:target-additional-information>~~
~~  <btp:superior-identifier>...hexstring...</btp:superior-~~
~~identifier> ?~~
~~  <btp:inferior-identifier>...hexstring...</btp:inferior-~~
~~identifier>~~
~~  <btp:old-address>  +~~
~~    ...address...~~
~~  </btp:old-address>~~
~~  <btp:new-address>  +~~
~~    ...address...~~
~~  </btp:new-address>~~
~~  <btp:qualifiers> ?~~
~~    ...qualifiers...~~
~~  </btp:qualifiers>~~
~~</btp:redirect>~~
```

FAULT

```
<btp:fault id?>
  <btp:target-additional-information>
    ...additional address information...
  </btp:target-additional-information>
  <btp:superior-identifier>...hexstring...</btp:superior-
identifier> ?
  <btp:inferior-identifier>...hexstring...</btp:inferior-
identifier> ?
  <btp:fault-type>...fault type name...</btp:fault-type>
  <btp:fault-data>...fault 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:

- general
- unknown-parameter
- wrong-state
- communication-failure
- invalid-superior
- duplicate-inferior
- unknown-inferior

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:

Free text:

```
<btp:fault-data>...string data...</btp:fault-data>
```

Identifier:

```
<btp:fault-data>...hexstring...</btp:fault-data>
```

Inferior Identity:

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

Standard qualifiers

The informal syntax for these messages assumes the namespace prefix “btpq” is associated with the URI “`urn:oasis:names:tc:BTP:qualifiers`”.

Transaction timelimit

```
<btpq:transaction-timelimit>
  <btpq:timelimit>
    ...time in seconds...
  </btpq:timelimit>
</btpq:transaction-timelimit>
```

Inferior timeout

```
       <btpq:inferior-timeout>
  <btpq:timeout>
    ...time in seconds...
  </btpq:timeout>
  <btpq:intended-decision>confirm|cancel</btpq:intended-decision>
</btpq:inferior-timeout>
```

Minimum inferior timeout

```
       <btpq:minimum-inferior-timeout>
  <btpq:minimum-timeout>
```

```
    ...time in seconds...
  </btpq:minimum-timeout>
</btpq:minimum-inferior-timeout>
```

Compounding of Messages

Relating BTP to one another, in a “group” is represented by containing them within the btp:relatedgroup 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

```
  <btp:relatedgroup>
       <btp:context-reply>
         ...<completion-status>related</completion-status> ...
       </btp:context-reply>
      <btp:enrol>...</btp:enrol>
       <btp:prepared>...</btp:prepared>
  </btp:relatedgroup>
```

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 relatedgroup. 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):

```
  <btp:messages>
    <btp:~~enrol~~confirm>...</btp:~~enrol~~confirm>
    <btp:~~prepared~~cancel>...</btp:~~prepared~~cancel>
  </btp:messages>
```

~~Relating BTP messages to one another is achieved through containment. For example:~~

```
~~  <btp:context-reply>~~
~~      ...<completion-status>related</completion-status> ...~~
~~      <btp:enrol>...</btp:enrol>~~
~~  </btp:context-reply>~~
```

~~The carrier protocol binding specifies how a relation between application and BTP messages is represented.~~

Carrier Protocol Bindings

The notion of bindings is introduced to act as the glue between the BTP ~~XML~~ 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 re~~r~~presented. For many bindings, the BTP message syntax will be ~~this will be the normal string encoding of the XML, in accordance~~as specified in ~~with~~ 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 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~~>~~.

Bindings for request/response carrier protocols

BTP does not generally follow 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.

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/ENROLLED, there is no "reply-address", and the parties know each other's "address-as-superior" and "address-as-inferior". Both sides are permitted to treat the carrier request/response exchanges as just 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) 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.

e) 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.

f) 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. ~~conform to the rules of the Section 5 (of the SOAP 1.1 specification) SOAP Encoding as specified by the URI: "http://schemas.xmlsoap.org/soap/encoding/".~~

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 no application message is being sent at the same time, BTP messages shall be contained in a btp:messages element which shall be an immediate child element of the SOAP-Body. There shall be precisely one btp:messages element. Any number of BTP messages with the same binding address in their target address may be carried in the same btp:messages element.~~

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~~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~~ 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 when the SOAP message carries only BTP messages in the SOAP ~~:~~Body.

The SOAP mustUnderstand attribute, when used on the btp:messages containing a ~~the~~ BTP CONTEXT, ensures that the receiver (server, ~~(~~as a whole) supports BTP sufficiently to determine~~s~~ whether any enrolments are necessary and replies~~y~~ 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 if false, a ~~server~~ receiver can ignore the CONTEXT (if BTP is not supported there), and no CONTEXT_REPLY will be returned. It is a local~~n implementation or configuration~~ option on the sender (client) side whether the absence of a CONTEXT_REPLY is assumed to be equivalent to a~~-~~CONTEXT_REPLY/ok (and the business transaction allowed to proceed to confirmation).~~ is assumed to be implicit in the HTTP response in such a case. (If no CONTEXT_REPLY/ok is assumed, it will be impossible for the business transaction to confirm) .~~

> Note – some SOAP implementations may not support the mustUnderstand attribute sufficiently to enforce these requirements. ~~If using such an implementation on the service side, it may be necessary to assume an CONTEXT_REPLY/ok.~~

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).

```
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    soap~~-env:~~encodingStyle="
~~http://schemas.xmlsoap.org/soap/encoding/~~">
```

```
  <soap:Header>

    <btp:messages xmlns:btp="urn:oasis:names:tc:BTP:xml">
      <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>1001</btp:superior-identifier>
        <btp:qualifiers>
          <btpq:transaction-timelimit
xmlns:btpq="urn:oasis:names:tc:BTP:qualifiers">1800</btpq:transact
ion-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~~ (and therefore contained)~~ 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)

```
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    soap~~-~~
~~env~~:encodingStyle="~~http://schemas.xmlsoap.org/soap/encoding/~~">

  <soap:Header>
  </soap:Header>

  <soap:Body>

    <btp:messages xmlns:btp="urn:oasis:names:tc:BTP:xml">
       <btp:relatedgroup>
       <btp:context-reply>
         <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>1001</btp:superior-identifier>
        <completion-status>related</completion-status>
        </btp:context-reply>

        <btp:enrol reply-requested="false">
          ~~<btp:target-additional-information>~~
               ~~btpengine~~
          ~~</btp:target-additional-information>~~
          <btp:superior-identifier>
               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>
               AAAB
          </btp:inferior-identifier>
         </btp:enrol>

       </btp:~~context-reply~~relatedgroup>

    </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-attachements-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

```
MIME-Version: 1.0
Content-Type: Multipart/Related; boundary=MIME_boundary;
type=text/xml;
        start="someID"

--MIME_boundary
Content-Type: text/xml; charset=UTF-8
Content-ID: someID

<?xml version='1.0' ?>
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    soap-
env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
```

```
  <soap:Header>

    <btp:messages xmlns:btp="urn:oasis:names:tc:BTP:xml">
      <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:superior-address>
        <btp:superior-identifier>1001</btp:superior-identifier>
      </btp:context>
    </btp:messages>

  </soap:Header>

  <soap:Body>
    <orderGoods href="cid:anotherID"/>
  </soap:Body>

</soap:Envelope>

--MIME_boundary
Content-Type: text/xml
Content-ID: anotherID

    <ns1:orderGoods
xmlns:ns1="http://example.com/2001/Services/xyzgoods">
      <custID>ABC8329045</custID>
      <itemID>224352</itemID>
      <quantity>5</quantity>
    </ns1:orderGoods>


--MIME_boundary--
```

XML Schema ~~for SOAP Bindings~~

```
<?xml version="1.0"?>
<schema targetNamespace="urn:oasis:names:tc:BTP:xml"
        xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:tns="urn:oasis:names:tc:BTP:xml">

    <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="tns:qualifier_type"/>
    <element name="qualifiers">
        <complexType>
            <sequence>
                <element ref="tns:qualifier" maxOccurs="unbounded"/>
            </sequence>
        </complexType>
    </element>

    <complexType name="address">
        <sequence>
            <element name="binding-name" type="string"/>
            <element name="binding-address" type="string"/>
            <element name="additional-information" type="string"
minOccurs="0"/>
        </sequence>
    </complexType>

    <simpleType name="identifier">
      <restriction base="string">
       <pattern value="([0-9,A-Z])*"/>
      </restriction>
    </simpleType>

    <simpleType name="superior-type">
        <restriction base="string">
            <enumeration value="cohesion"/>
            <enumeration value="atom"/>
        </restriction>
    </simpleType>

    <simpleType name="transaction-type">
        <restriction base="string">
            <enumeration value="cohesion"/>
            <enumeration value="atom"/>
        </restriction>
    </simpleType>


    <element name="context">
        <complexType>
            <sequence>
                <element name="superior-address" type="tns:address"
maxOccurs="unbounded"/>
                <element name="superior-identifier" type="tns:identifier"/>
                <element ref="tns:qualifiers" minOccurs="0"/>
            </sequence>
            <attribute name="id" type="ID" use="optional"/>
            <attribute name="superior-type" type="tns:superior-type"
use="required"/>
        </complexType>
    </element>

    <element name="context-reply">
        <complexType>
            <sequence>
                <element name="superior-address" type="tns:address"
maxOccurs="unbounded"/>
                <element name="superior-identifier" type="tns:identifier"/>
                <element name="completion-status">
                    <simpleType>
                        <restriction base="string">
                            <enumeration value="completed"/>
                            <enumeration value="related"/>
                            <enumeration value="repudiated"/>
                        </restriction>
                    </simpleType>
                </element>
                <element ref="tns:qualifiers" minOccurs="0"/>
            </sequence>
            <attribute name="id" type="ID"/>
            <attribute name="superior-type" type="tns:superior-type"
use="required"/>
        </complexType>
    </element>

    <element name="begin">
        <complexType>
            <sequence>
                <element name="target-additional-information"
type="string"/>
                <element name="reply-address" type="tns:address"/>
                <element ref="tns:qualifiers" minOccurs="0"/>
            </sequence>
            <attribute name="id" type="ID"/>
            <attribute name="transaction-type" type="tns:superior-type"
use="required"/>
        </complexType>
    </element>

    <element name="begun">
        <complexType>
            <sequence>
                <element name="target-additional-information"
type="string"/>
                <element name="decider-address" type="tns:address"
minOccurs="0"/>
                <element name="transaction-identifier"
type="tns:identifier" minOccurs="0"/>
                <element name="inferior-handle" type="tns:identifier"
minOccurs="0"/>
                <element name="inferior-address" type="tns:address"
minOccurs="0"/>
                <element ref="tns:qualifiers" minOccurs="0"/>
            </sequence>
            <attribute name="id" type="ID"/>
            <attribute name="transaction-type" type="tns:superior-type"
use="required"/>
        </complexType>
    </element>

    <element name="enrol">
        <complexType>
            <sequence>
                <element name="target-additional-information"
type="string"/>
                <element name="superior-identifier" type="tns:identifier"/>
                <element name="reply-address" type="tns:address"
minOccurs="0"/>
                <element name="inferior-address" type="tns:address"
minOccurs="1" maxOccurs="unbounded"/>
                <element name="inferior-identifier" type="tns:identifier"/>
                <element ref="tns:qualifiers" minOccurs="0"/>
            </sequence>
            <attribute name="id" type="ID"/>
            <attribute name="reply-requested" type="boolean"/>
        </complexType>
    </element>


    <element name="enrolled">
        <complexType>
            <sequence>
                <element name="target-additional-information"
type="string"/>
                <element name="inferior-identifier" type="tns:identifier"/>
                <element name="inferior-handle" type="tns:identifier"
minOccurs="0"/>
                <element ref="tns:qualifiers" minOccurs="0"/>
            </sequence>
            <attribute name="id" type="ID"/>
        </complexType>
    </element>

    <element name="resign">
        <complexType>
            <sequence>
                <element name="target-additional-information"
type="string"/>
                <element name="superior-identifier" type="tns:identifier"/>
                <element name="inferior-address" type="tns:address"
minOccurs="1" maxOccurs="unbounded"/>
                <element name="inferior-identifier" type="tns:identifier"/>
                <element ref="tns:qualifiers" minOccurs="0"/>
            </sequence>
            <attribute name="id" type="ID"/>
            <attribute name="response-requested" type="boolean"/>
        </complexType>
    </element>

    <element name="resigned">
        <complexType>
            <sequence>
                <element name="target-additional-information"
type="string"/>
                <element name="inferior-identifier" type="tns:identifier"/>
                <element ref="tns:qualifiers" minOccurs="0"/>
            </sequence>
            <attribute name="id" type="ID"/>
        </complexType>
    </element>

    <element name="prepare">
        <complexType>
            <sequence>
                <element name="target-additional-information"
type="string"/>
                <element name="inferior-identifier" type="tns:identifier"
minOccurs="0"/>
                <element name="reply-address" type="tns:address"
minOccurs="0"/>
                <element name="transaction-identifier"
type="tns:identifier" minOccurs="0"/>
                <element name="inferiors-list" minOccurs="0">
                    <complexType>
                        <sequence>
                            <element name="inferior-handle"
type="tns:identifier" maxOccurs="unbounded"/>
                        </sequence>
                    </complexType>
                </element>
                <element ref="tns:qualifiers" minOccurs="0"/>
            </sequence>
            <attribute name="id" type="ID"/>
        </complexType>
    </element>

    <element name="prepared">
        <complexType>
            <sequence>
                <element name="target-additional-information"
type="string"/>
                <element name="superior-identifier" type="tns:identifier"/>
                <element name="inferior-address" type="tns:address"
maxOccurs="unbounded"/>
                <element name="inferior-identifier" type="tns:identifier"/>
                <element ref="tns:qualifiers" minOccurs="0"/>
            </sequence>
            <attribute name="id" type="ID"/>
            <attribute name="default-is-cancel" type="boolean"/>
        </complexType>
    </element>

    <element name="confirm">
        <complexType>
            <sequence>
                <element name="target-additional-information"
type="string"/>
                <element name="inferior-identifier" type="tns:identifier"/>
                <element ref="tns:qualifiers" minOccurs="0"/>
            </sequence>
            <attribute name="id" type="ID"/>
        </complexType>
    </element>

    <element name="confirmed">
        <complexType>
            <sequence>
                <element name="target-additional-information"
type="string"/>
                <element name="superior-identifier" type="tns:identifier"/>
                <element name="inferior-address" type="tns:address"
minOccurs="0"/>
                <element name="inferior-identifier" type="tns:identifier"
minOccurs="0"/>
                <element name="decider-address" type="tns:address"
minOccurs="0"/>
                <element name="transaction-identifier"
type="tns:identifier" minOccurs="0"/>
                <element ref="tns:qualifiers" minOccurs="0"/>
            </sequence>
            <attribute name="id" type="ID"/>
            <attribute name="confirmed-received" type="boolean"/>
        </complexType>
    </element>

    <element name="cancel">
        <complexType>
            <sequence>
                <element name="target-additional-information"
type="string"/>
                <element name="inferior-identifier" type="tns:identifier"
minOccurs="0"/>
                <element name="reply-address" type="tns:address"
minOccurs="0"/>
                <element name="transaction-identifier"
type="tns:identifier" minOccurs="0"/>
                <element name="decider-address" type="tns:address"
minOccurs="0"/>
                <element name="transaction-identifier"
type="tns:identifier" minOccurs="0"/>
                <element name="inferiors-list" minOccurs="0">
                    <complexType>
                        <sequence>
```

```
                            <element name="inferior-handle"
type="tns:identifier" maxOccurs="unbounded"/>
                        </sequence>
                    </complexType>
                </element>
                <element ref="tns:qualifiers" minOccurs="0"/>
            </sequence>
            <attribute name="id" type="ID"/>
        </complexType>
    </element>

    <element name="cancelled">
        <complexType>
            <sequence>
                <element name="target-additional-information"
type="string"/>
                <element name="superior-identifier" type="tns:identifier"/>
                <element name="inferior-address" type="tns:address"
maxOccurs="unbounded"/>
                <element name="inferior-identifier" type="tns:identifier"
minOccurs="0"/>
                <element name="decider-address" type="tns:address"
minOccurs="0"/>
                <element name="transaction-identifier"
type="tns:identifier" minOccurs="0"/>
                <element ref="tns:qualifiers" minOccurs="0"/>
            </sequence>
            <attribute name="id" type="ID"/>
        </complexType>
    </element>

    <element name="hazard">
        <complexType>
            <sequence>
                <element name="target-additional-information"
type="string"/>
                <element name="superior-identifier" type="tns:identifier"/>
                <element name="inferior-address" type="tns:address"
maxOccurs="unbounded"/>
                <element name="inferior-identifier" type="tns:identifier"/>
                <element ref="tns:qualifiers" minOccurs="0"/>
            </sequence>
            <attribute name="id" type="ID"/>
        </complexType>
    </element>

    <element name="contradiction">
        <complexType>
            <sequence>
                <element name="target-additional-information"
type="string"/>
                <element name="inferior-identifier" type="tns:identifier"/>
                <element ref="tns:qualifiers" minOccurs="0"/>
            </sequence>
            <attribute name="id" type="ID"/>
        </complexType>
    </element>

    <element name="superior-state">
        <complexType>
            <sequence>
                <element name="target-additional-information"
type="string"/>
                <element name="inferior-identifier" type="tns:identifier"/>
                <element name="status">
                    <simpleType>
                        <restriction base="string">
                            <enumeration value="active"/>
                            <enumeration value="prepared-received"/>
                            <enumeration value="inaccessible"/>
                            <enumeration value="unknown"/>
                        </restriction>
                    </simpleType>
                </element>
                <element ref="tns:qualifiers" minOccurs="0"/>
            </sequence>
            <attribute name="id" type="ID"/>
            <attribute name="reply-requested" type="boolean"/>
        </complexType>
    </element>

    <element name="inferior-state">
        <complexType>
            <sequence>
                <element name="target-additional-information"
type="string"/>
                <element name="superior-identifier" type="tns:identifier"/>
                <element name="inferior-address" type="tns:address"
maxOccurs="unbounded"/>
                <element name="inferior-identifier" type="tns:identifier"/>
                <element name="status">
                    <simpleType>
                        <restriction base="string">
                            <enumeration value="active"/>
                            <enumeration value="prepared-received"/>
                            <enumeration value="inaccessible"/>
                            <enumeration value="unknown"/>
                        </restriction>
                    </simpleType>
                </element>
                <element ref="tns:qualifiers" minOccurs="0"/>
            </sequence>
            <attribute name="id" type="ID"/>
            <attribute name="reply-requested" type="boolean"/>
        </complexType>
    </element>

    <element name="confirm-one-phase">
        <complexType>
            <sequence>
                <element name="target-additional-information"
type="string"/>
                <element name="inferior-identifier" type="tns:identifier"/>
                <element ref="tns:qualifiers" minOccurs="0"/>
            </sequence>
            <attribute name="id" type="ID"/>
            <attribute name="report-hazard" type="boolean"/>
        </complexType>
    </element>

    <element name="request-confirm">
        <complexType>
            <sequence>
                <element name="target-additional-information"
type="string"/>
                <element name="reply-address" type="tns:address"/>
                <element name="transaction-identifier"
type="tns:identifier"/>
                <element name="inferiors-list" minOccurs="0">
                    <complexType>
                        <sequence>
                            <element name="inferior-handle"
type="tns:identifier" maxOccurs="unbounded"/>
                        </sequence>
                    </complexType>
                </element>
                <element ref="tns:qualifiers" minOccurs="0"/>
            </sequence>
            <attribute name="id" type="ID"/>
            <attribute name="report-hazard" type="boolean"/>
        </complexType>
    </element>

    <element name="request-statuses">
        <complexType>
            <sequence>
                <element name="target-additional-information"
type="string"/>
                <element name="reply-address" type="tns:address"/>
                <element name="transaction-identifier"
type="tns:identifier"/>
                <element name="inferiors-list" minOccurs="0">
                    <complexType>
                        <sequence>
                            <element name="inferior-handle"
type="tns:identifier" maxOccurs="unbounded"/>
                        </sequence>
                    </complexType>
                </element>
                <element ref="tns:qualifiers" minOccurs="0"/>
            </sequence>
            <attribute name="id" type="ID"/>
        </complexType>
    </element>

    <element name="inferior-statuses">
        <complexType>
            <sequence>
                <element name="target-additional-information" 
type="string"/>
                <element name="decider-address" type="tns:address"/>
                <element name="transaction-identifier" 
type="tns:identifier"/>
                <element name="status-list">
                  <complexType>
                    <sequence>
                      <element name="status-item" maxOccurs="unbounded">
                    <complexType>
                      <sequence>
                        <element name="inferior-handle" 
type="tns: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"/>
                        </restriction>
                          </simpleType>
                        </element>
                        <element ref="tns:qualifiers" minOccurs="0"/>
                      </sequence>
                    </complexType>
                      </element>
                    </sequence>
                  </complexType>
                </element>
                <element ref="tns:qualifiers" minOccurs="0"/>
            </sequence>
            <attribute name="id" type="ID"/>
        </complexType>
    </element>

    <element name="request-status">
        <complexType>
            <sequence>
                <element name="target-additional-information"
type="string"/>
                <element name="reply-address" type="tns:address"/>
                <element name="inferior-identifier" type="tns:identifier"
minOccurs="0"/>
                <element name="transaction-identifier"
type="tns:identifier" minOccurs="0"/>
                <element ref="tns:qualifiers" minOccurs="0"/>
            </sequence>
            <attribute name="id" type="ID"/>
        </complexType>
    </element>

    <element name="status">
        <complexType>
            <sequence>
                <element name="target-additional-information"
type="string"/>
                <element name="inferior-address" type="tns:address"
minOccurs="0"/>
                <element name="inferior-identifier" type="tns:identifier"
minOccurs="0"/>
                <element name="decider-address" type="tns:address"
minOccurs="0"/>
                <element name="transaction-identifier"
type="tns:identifier" minOccurs="0"/>
                <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="tns:qualifiers" minOccurs="0"/>
            </sequence>
            <attribute name="id" type="ID"/>
        </complexType>
    </element>

    <element name="redirect">
        <complexType>
            <sequence>
                <element name="target-additional-information"
type="string"/>
                <element name="superior-identifier" type="tns:identifier"
minOccurs="0"/>
                <element name="inferior-identifier" type="tns:identifier"/>
                <element name="old-address" type="tns:address"
maxOccurs="unbounded"/>
                <element name="new-address" type="tns:address"
maxOccurs="unbounded"/>
                <element ref="tns:qualifiers" minOccurs="0"/>
            </sequence>
            <attribute name="id" type="ID"/>
        </complexType>
    </element>

    <element name="fault">
        <complexType>
            <sequence>
                <element name="target-additional-information"
type="string"/>
                <element name="superior-identifier" type="tns:identifier"
minOccurs="0"/>
                <element name="inferior-identifier" type="tns:identifier"
minOccurs="0"/>
                <element name="fault-type" type="string"/>
                <element name="fault-data" type="anyType" minOccurs="0"/>
                <element ref="tns:qualifiers" minOccurs="0"/>
            </sequence>
            <attribute name="id" type="ID"/>
        </complexType>
    </element>

</schema>
```

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.

A partially conformant implementation may implement some roles in a non-interoperable way, giving that implementation's users comparable proprietary functionality.

The following Roles and Role Groups are used to define conformance:

Role Group	Role
Initiator/Terminator	Initiator Terminator
Cohesive Hub	Factory Composer (as Decider and Superior) Coordinator (as Decider and Superior) Sub-composer Sub-coordinator
Atomic Hub	Factory Coordinator Sub-coordinator
Cohesive Superior	Composer (as Superior only) Sub-Composer Coordinator (as Superior only) Sub-coordinator
Atomic Superior	Coordinator (as Superior only)) Sub-coordinator
Participant	Inferior

Enroller

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.

Conformance Profile	**Role Groups**
Participant Only	Participant
Atomic	Atomic Superior Participant
Cohesive	Full Superior Participant
Atomic Coordination Hub	Initiator/Terminator Atomic Coordination Hub Participant
Cohesive Coordination Hub	Initiator/Terminator Cohesive Coordination Hub Participant

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. Appendices

These terms seem to be all either not used, or effectively defined elsewhere

A. Glossary

Message	A datum which is produced and then consumed.
Sender	The producer of a message.
Receiver	The consumer of a message.
Transmission	The passage of a message from a sender to a receiver.
Endpoint	A sender or receiver.
Address	An identifier for an endpoint.
Carrier Protocol	A protocol which defines how transmissions occur.
Carrier Protocol Address (CPA)	The address of an endpoint for a particular carrier protocol.
Business Transaction Protocol Address (BTPA)	A compound address consisting of a mandatory *carrier protocol address* and an optional opaque suffix. *PRF - suffix ? I've used "additional information"*
Actor	An entity which executes procedures, a software agent.
Application	An actor which uses the Business Transaction Protocol.
Application Message	A message produced by an application and consumed by an application.
Application Endpoint	An endpoint of an application message.

Operation	A procedure which is started by a receiver when a message arrives at it.
Application Operation	An operation which is started when an application message arrives.
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.
Appropriate	In accordance with a pertinent contract.
Inappropriate	In violation of a pertinent contract.
Service	An actor, which on receipt of an application messages, may start an appropriate application operation. For example, a process which advertises an interface allowing defined RPCs to be invoked by a remote client.
Client	An actor which sends application messages to services.
Effect	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 countereffect of the effect, and is known as the countereffect contract. An effect is **Completed** when the change-inducing processing of the set of procedures is finished. [Need an indirect or consequential damage exclusion clause] *PRF - Sentence about countereffect contract doesn't fit well*
Ineffectual	Describes a set of procedures which has no effect.
Countereffect	An appropriate effect intended to counteract a prior effect.

Countereffect Contract	The contract which governs the relationship between the effect and the countereffect of a procedure. In the absence of any other overriding contracts the countereffect contract is the promise that “The **Countereffect** will attempt so far as is possible to reverse or cancel the **Effect** such that an observer (on completion of the **Countereffect**) is unaware that the **Effect** ever occurred, but this attempt cannot be guaranteed to succeed”.
Cancel	Process a countereffect for the current effect of a set of procedures.
Confirm	Ensure that the effect of a set of procedures is completed.
Prepare	Ensure that of a set of procedures is capable of being successfully instructed to cancel or to confirm.
Outcome	A decision to either cancel or confirm.
Participant	A set of procedures which is capable of receiving instructions from a coordinator to prepare, cancel and confirm. A participant must also have a BTPA to which these instructions will be delivered, in the form of BTP messages. A participant is identified by a participant identifier.
Inferior Identifier	An identifier assigned to an Inferior which is unique within the scope of an Address-as-Inferior.
Atomic Business Transaction *or* **Atom**	A set of participants (which may have only one member), all of which will receive instructions that will result in a homogeneous outcome. (Transitively, a set of operations, whose effect is capable of countereffect.) An atom is identified by an atom identifier.
Atom Identifier	A globally unique identifier assigned to an atom. *PRF – abs msgs define as unambiguous in scope of its address-as-superior, I think.*

Coordinator	An actor which decides the outcome of a single atom, and has a lifetime which is coincident with that of the atom. A coordinator can issue instructions to a participant to prepare, cancel and confirm. These instructions take the form of BTP messages. A coordinator is identified by its atom's atom identifier. A coordinator must also have a BTPA to which participants can send BTP messages.
Address-as-Superior	The address used to communicate with an actor playing the role of an Superior
Address-as-Composer	The address used to communicate with a Composer by an application actor that controls its resolution. The messages that might be sent to or received from this endpoint are undefined.
Address-as-Inferior	The address used to communicate with an actor playing the role of an Inferior.
Identity-as-Superior	The combination of Superior Identifier and Address-as-Superior of a given Superior.
Identity-as-Inferior	The combination of Inferior Identifier and Address-as-Inferior of a given Inferior.