HOVTP

HTTP-based OVTP

Document control:

Version	**1.0**
Author(s)	**F. George**
Status of document	**In progress / in review / approved**

OFFICIAL

Version History:

Version	Date	Author(s)	Chapters	Comments / Change description
0.1	10/10/2012	F. George	All	Preliminary draft
0.2	19/12/2012	F. George	All	Draft update after feedback
0.3	22/03/2013	F. George	4 – 5	Minor changes to header usage and/or description.
0.4	11/10/2013	F. George	5, 8	Added ODF Data Layer annexe.
0.5	01/03/2014	F. George/F. Blondeau	1,7,8	Added crossed topology, completed ODF Data Layer annexe
1.0	01/03/2015	J.Hily / A.Schneeberger	All	Approbation

Contents

1. Introduction

1.1. Context

The current status of the OVTP protocol is somewhat problematic. It is currently in a frozen state while missing core features (e.g. a built-in keep-alive mechanism). These features need to be implemented on top of it, which complicates its integration and reliability.

There is a need for a new protocol that deals with these problems and needs. The aim of this document is to define a protocol based on current experiences that can address all ranges of integration, remain flexible and with a specification still able to evolve. Furthermore, as venue results are now mainly composed of XML documents, a text-based protocol is preferred.

1.2. Requirements

The following requirements have been extracted:

- Base the protocol on standard and existing technologies
  - Implementations must be possible using various development frameworks
- The protocol must be firewall-friendly, i.e. must not use broadcast or multicast transport. In addition it should not require complex configuration, i.e. other than allowing either outbound or inbound-only connection on a single TCP or UDP port.
  - The deployment of the applications using the protocol must not be convoluted by the protocol's constraints
- It must be flexible enough to support any integration level without too much effort from the integrator
  - It must support both primary-only and primary-secondary architectures
  - It needs to be usable for all sizes of events without difficulties
- A clear separation must be made between the transport/routing and the data
  - Transport must support other formats than ODF
- It should be well defined and documented

2. Protocol description

While the protocol hereby defined, HOVTP, is inspired by OVTP and preserves some of its principles, it departs from it in some major and minor aspects.

- First and foremost, the biggest difference is that the new protocol is layered on top of HTTP. HTTP was chosen as it is standardized, widespread, text-based, flexible and doesn't require any non-trivial firewall configuration. All these points were part of the requirements. Furthermore, adopting it allows taking advantage of all the existing implementations, both on client and server sides. In addition, encryption and authentication are already part of HTTP, should it be needed in a future revision of HOVTP.
- The headers found in OVTP that belongs at the transport-level (i.e. not specific to ODF) were kept and adapted as HTTP headers. However additional headers specific to the data were not included directly. They must be defined as additional prefixed HTTP headers in the specification of the data type.
- The Keep-alive mechanism is part of the protocol; it is not a service built on top.
- The protocol was made independent of ODF. Transport of ODF messages is treated like any other format of data would.
- The environment is included in all responses. This ensures that the environment of the receiver is always known by the sender
- The error reporting was enhanced by allowing a description of the cause of the error.

3. Overall operations

3.1. Layers

This specification makes a clear distinction between the three layers composing HOVTP and its usage: the HTTP layer, the HOVTP layer and the Data layer. Here is an illustration of the layers using a representation analogous to the OSI model:

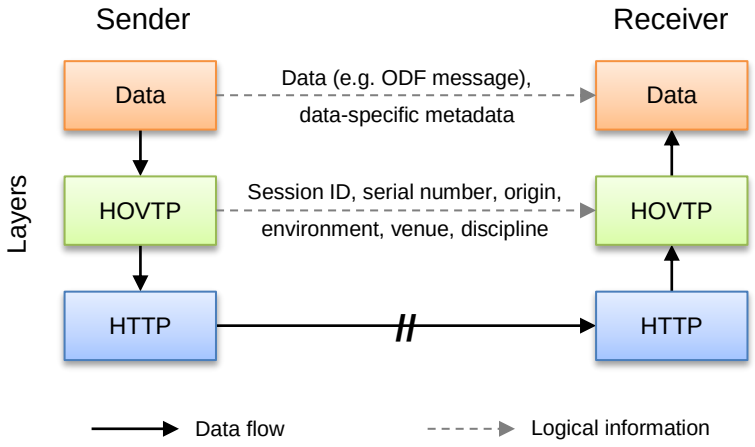


Figure 1 - Data and logical information flow through layers

HTTP: The HTTP layer is at the base, it provides the transport of the data as the body and of all metadata as headers.

HOVTP: The layer at which HOVTP resides in itself. It extends HTTP, using headers, by adding its own metadata specific to on-venue environment or necessary for the additional features of the protocol. It provides gap-free and in-order transport between a sender and a receiver using a session ID and serial numbers.
This layer is data-agnostic, i.e. is not specific to any type of data being transported.

Data: The layer at which the data and the data-specific metadata resides. It is important to understand that this layer is not part of HOVTP but layered on top of it. For each type of data being transported, there must be a definition of the data format and associated metadata; e.g. a separate specification.

3.2. Conventions

3.2.1. Connections

A connection consists of 1) a *sender*; the source of the data, 2) a *receiver*; the destination of the data. At the level of HTTP, the sender acts as a *client* and establishes the connection; the receiver acts as a *server* and listens for incoming connections.

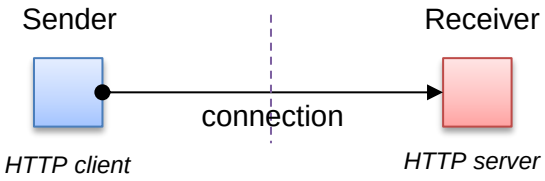


Figure 2 - Sender and receiver convention in relation to HTTP

The sender generates the *requests* and receiver processes them and generates the *responses*; the HTTP request body being the transported data.

An HOVTP connection is not directly tied a single underlying TCP/IP connection. All requests must be treated independently whether they were received on different TCP/IP connections or on the same (using HTTP/1.1 persistent connections). It is however recommended to use persistent connections to optimize the throughput.

3.2.2. Sessions

As said above, the data exchanged between the sender and receiver is not necessarily transferred using the same connection for all the requests. Each data transfer requests can be made over different connections. Therefore, to ensure gap-free and in-order transmission, they must be logically tagged as being related to each other; this is the role of the sessions. The sender informs the receiver of this relation by specifying the session to which the data transfer belong. Combined with request's serial number, which is per session, the receiver can detect transmission problems and respond accordingly.

4. Request format

Two types of request are defined: *data transfer* and *status* requests. The first provides the transfer of the data from the sender to the receiver, the latter is a lightweight request used to get the status of the receiver and on which the keep-alive mechanism is based.

4.1. Data transfer request

4.1.1. Overview

Data transfer requests are the central message of the protocol. They provide the transfer of that data from the sender to the receiver. Here is the structure of a data transfer request:

```
POST <HOVTP endpoint> HTTP/1.1
Host: <host>
Connection: Keep-Alive
Cache-Control: no-cache
X-HOVTP-Session-Id: <session ID>
X-HOVTP-Serial-Number: <serial number>
X-HOVTP-Origin: <origin>
X-HOVTP-Environment: <environment>
X-HOVTP-Venue: <venue>
X-HOVTP-Discipline: <discipline>
X-HOVTP-Data-Type: <data type>
<additional data-specific headers>: <any>
Content-Type: <data type>
Content-Length: <data length>

<data>
```

Legend

- HTTP layer
  - Standard HTTP request line and headers. Headers are un-prefixed and concern the connection between the sender and receiver.
- HOVTP layer
  - These headers contain the HOVTP information. The fields found in the original OVTP, with exception of those directly related to ODF, are found there. They are present in the form of HOVTP-prefixed HTTP headers.
- Data layer
  - Optionally, the headers that are specific to the data being transported, e.g. ODF message type.
  - The data being transported as the HTTP request body, e.g. the ODF message.

4.1.2. Headers

Note: While standard HTTP headers usually appear first, this is not required. Implementations MUST NOT depend on the order of any of the headers.

HTTP layer headers

The following headers are part of HTTP itself and are all required. For further details about the meaning of each of them, see the HTTP/1.1 specification (RFC 2616).
HTTP client implementations may add additional headers (e.g. User-Agent, Accept, etc.); they MUST be accepted (i.e. their presence in itself should not cause an error) and they SHOULD be processed as defined in the HTTP specification if applicable.

Header Name	Required	Values	Remarks
Host	yes	Server host or IP address	Required by HTTP/1.1. Used for the routing of the request to the application. No specific usage by HOVTP.
Connection	yes	‘keep-alive’, ‘close’	It is recommended that persistent connections be used. However receivers MUST be able to handle connection closing after any request and reopening for next. Requests MUST NOT be treated differently in that case.
Cache-Control	yes	‘no-cache’	Caching of resources must be disabled so that responses always come from the receiver, never from an intermediate node performing caching.

OVTP layer headers

These headers contain information belonging to the HOVTP layer. In accordance to HTTP usage conventions, they are all prefixed by '*X*-' to indicate that they are non-standard in regard to HTTP.

Header Name	Required?	Values	Remarks
X-HOVTP-Session-Id	yes	A universally unique identifier (UUID)[1] generated for the session.	The session ID is generated by the sender at the first data transfer request to establish a session with receiver. It is used to keep track of the serial number.
X-HOVTP-Serial-Number	yes	Current serial number of the session	See dedicated description after this table.
X-HOVTP-Environment	yes	*Production* or *Test*	The current environment of the sender.
X-HOVTP-Origin	yes	A string identifying the sender, for example the fully-qualified hostname or IP address of the sender system.	The origin is used to identify the sender. Note that in HOVTP, it is NOT used for the serial numbering.
X-HOVTP-Discipline	yes	To be agreed in advance between the source data producer and the integrator.	Name of the discipline.
X-HOVTP-Venue	yes		Name of the venue.
X-HOVTP-Data-Type	yes	The identification of the type of data being transported (e.g. ODF, PDF, etc.).	Each data format specification MUST define value of this field. This value MUST be the prefix used in the data headers (see *Data layer headers*).

Serial Number

The serial number is a strictly-incrementing 64-bit unsigned integer counter starting 1, it must not have gaps (except for the highly improbable rollover to 1 at 2^{64}-1). The receiver must keep track of the serial number separately for each session (as identified by the *X-Session-Id* header). If the serial number breaks sequence, the request MUST be refused and a response with status code of *450 Out Of Synchro* MUST be sent back. It is up to the sender to decide whether it can recover from this situation or has start again at 1. This means that sending the same request twice will only result on the second one being refused, transmission can continue normally after that. A sender that keeps messages after their transmission COULD try sending them again in case of *450 Out Of Synchro* by using the *X-HOVTP-Last-Serial-Number* header to know how far it has to go back (if the value is not 0).

A message with serial number 1 MUST always be accepted by a receiver, independently of what it received before. The receiver MUST start keeping track of the serial number again from this value (i.e. next expected value is 2).

[1] UUID: http://en.wikipedia.org/wiki/Universally_unique_identifier

Data layer headers

Although two headers described here are part of HTTP/1.1, they are used to describe the data being transported. Hence they are logically part of the data layer.

Header Name	Required?	Values	Remarks
Content-Type	Only if request contains data	The type of the data being transported	The value is specific the data being transported and should be part of its specification. It should follow the standard MIME type usage (e.g. text/xml for xml content or application/pdf for PDF files).
Content-Length	Only if request contains data	The length of the data being transported	

Additional headers containing information about the transported data can be added to the requests if necessary; e.g. ODF headers. They MUST be prefixed with "X-*DataType*-" where *DataType* is the same value as specified in the *X-HOVTP-Data-Type* header; for example for ODF: X-ODF-Message-Type.

4.1.3. Example

Here is an example of a data transfer request with an ODF payload with fictional ODF headers. The actual headers to use when transporting ODF data should be part of a separate specification dedicated on the usage of ODF on top of this transport protocol.

```
POST /HOVTP/ HTTP/1.1
Host: 172.24.44.85
Connection: Keep-Alive
Cache-Control: no-cache
X-HOVTP-Session-Id: B7F40576-D3DA-4417-8492-8DBDEACE5E13
X-HOVTP-Serial-Number: 12345
X-HOVTP-Origin: OVR-02
X-HOVTP-Environment: Production
X-HOVTP-Venue: STA
X-HOVTP-Discipline: AT
X-HOVTP-Data-Type: ODF
X-ODF-Type: DT_PARTIC_UPDATE
Content-Type: text/xml
Content-Length: 1402

<?xml version="1.0" encoding="utf-8"?>
<OdfBody DocumentCode="AT0000000" DocumentSubcode="GENERAL"
DocumentType="DT_PARTIC_UPDATE" FeedFlag="P" Date="20120811" Time="121537867"
LogicalDate="20120811" Version="1" Serial="1">
  <Competition Code="LOCOG">
...
```

4.2. Status request

4.2.1. Overview

A status request is a body-less HTTP/1.1 request using the *OPTIONS* method. It allows a sender to query the receiver about its current status and configuration.
A status request MUST NOT have any side-effect on both the sender and the receiver. Particularly, they MUST NOT increment the current serial number.
The lightweight nature in addition to these constraints allows status request to be used as the base of the keep-alive mechanism and to detect the presence of a receiver (see chapter 6 *Keep-alive mechanism*).
The response to a status request is identical to a successful response to a data transfer request.

```
OPTIONS <OVTP endpoint> HTTP/1.1
Host: <host>
Connection: Keep-Alive
Cache-Control: no-cache
X-HOVTP-Session-Id: <session ID>
X-HOVTP-Origin: <origin>
X-HOVTP-Environment: <environment>
X-HOVTP-Venue: <venue>
X-HOVTP-Discipline: <discipline>
```

4.2.2. Headers

Headers present in status requests have the same meaning as in the data transfer requests. See chapter 4.1.2 for their details.

4.2.3. Example

```
OPTIONS /HOVTP/ HTTP/1.1
Host: 172.24.44.85
Connection: Keep-Alive
Cache-Control: no-cache
X-HOVTP-Session-Id: B7F40576-D3DA-4417-8492-8DBDEACE5E13
X-HOVTP-Origin: OVR-02
X-HOVTP-Environment: Production
X-HOVTP-Venue: STA
X-HOVTP-Discipline: AT
```

5. Response format

5.1. Overview

All responses transmitted by the receiver, including to status requests and indicating success or failure, must respect the following format.

```
HTTP/1.1 <status code> <reason phrase>
Cache-Control: no-cache
X-HOVTP-Environment: <environment>
X-HOVTP-Last-Serial-Number: <serial number>
[X-HOVTP-Keep-Alive-Interval: <keep-alive interval in secs>]
[X-HOVTP-Error-Reason: <host>]
[<additional data-specific headers>: <any>]
```

5.1.1. Legend

- HTTP layer
  - Standard HTTP status line and headers
    - Status codes and reason phrase in detailed in chapter *5.2 Status codes*
- OVTP layer
  - The environment of the server is always specified
  - All responses indicate the expected serial number of the next data transfer request.
  - Failure response can optionally include a description of the cause of the error
- Data layer
  - Optionally, headers that are specific to the data being transported
  - Only in response to data transfer requests

5.2. Status codes

The following list contains the important standard HTTP status codes and the supplementary status codes defined by the protocol.
Additional standard HTTP status codes may be returned by implementations in specific conditions and should be treated as unexpected errors by the senders. Those are not listed here (e.g. *408 Request Timeout*, *413 Request Entity Too Large*, etc.).

- **Successful 2xx**
  - 200 OK
- **Client Error 4xx**
  - 400 Bad Request
    - The request is invalid; e.g. missing a required header, a header has an incorrect value, etc. The *X-HOVTP-Error-Reason* should be used to specify the cause of the error. This error can be issued by the HTTP layer as well in case of protocol violation not related to HOVTP.
  - 450 Out Of Synchro
    - The serial number of the request does not have the value expected by the receiver for the request's session.
    - The *X-HOVTP-Last-Serial-Number* header contains the serial of the last received message; the expected serial number is therefore this value + 1. The sender can try resending the message starting from this expected serial number if it still has it available or resend only this message starting back at 1.
    - In any case, if the receiver issues 10 (?TBD) *Out Of Synchro* in a row, the sender MUST stop its transmission.
  - 451 Data Layer Error
    - The upper data-specific layer refused or could not process the request data. The *X-HOVTP-Error-Reason* should be used to specify the cause of the error reported by the data processing layer.
  - 453 Bad Environment
    - There is a mismatched between the environments of the sender and receiver and the request was therefore refused. This error MUST only be returned if the receiver DOES NOT process the request.
    - The *X-HOVTP-Environment* header contains the environment expected by the receiver.
    - The receiver is not required to fail when there is a mismatched between the environments, it could decide to route the data anyway to its environment. In this case it MUST indicate a successful status (i.e. 200 OK) and not generate this error. However the *X-HOVTP-Environment* MUST contain the environment used, i.e. not lie about it.
- **Server Error 5xx**
  - 500 Internal Server Error
    - The receiver encountered an unexpected condition which prevented it from fulfilling the request.
  - 503 Service Unavailable
    - The receiver is unable to handle requests. It may indicate temporary overloading, maintenance or a wrong endpoint used in the request.

5.3. Headers

Note: While standard HTTP headers usually appear first, this is not required. Implementations MUST NOT depend on the order of any of the headers.

5.3.1. HTTP layer headers

The following headers are part of the HTTP itself. For further details about the meaning of each of them, see the specification of HTTP/1.1 (RFC 2616).
HTTP server implementations may add additional headers (e.g. Server, Date, Expires, etc.); they MUST be accepted (i.e. their presence in itself should not cause an error) and they SHOULD be processed as defined in the HTTP specification if applicable.

Header Name	Required?	Values	Remarks
Connection	no	'keep-alive', 'close' When not present, 'keep-alive' is assumed.	It is recommended that persistent connections be used. However senders MUST be able to handle connection closing after any request and having to open a new one for the next.
Cache-Control	yes	'no-cache'	Caching of resources must be disabled. Response must always come from the receiver, never from an intermediate node performing caching.

5.3.2. OVTP layer headers

These headers contain information belonging to the OVTP layer. In accordance to HTTP usage conventions, they are all prefixed by '*X*-' to indicate that they are non-standard.

Header Name	Required?	Values	Remarks
X-HOVTP-Environment	yes	*Production* or *Test*	The current environment of the receiver.
X-HOVTP-Last-Serial-Number	yes	The serial number of the last data transfer request received	If no data transfer request was received yet for the session, this value is 0.
X-HOVTP-Keep-Alive-Interval	no	The recommended number of seconds between keep-alive requests	See chapter 6.1 *Keep-alive interval advertisement by receiver*.
X-HOVTP-Error-Reason	no	In case of failure, a short message indicating the reason	The header should use to provide the most accurate information as possible to facilitated problem solving.

5.3.3. Data layer headers

Additional headers containing information about the request result or error condition can be added to the response if necessary; e.g. ODF headers. They MUST follow the same naming restriction as in the request. The data type of the response is implicitly the same as the request. They MUST NOT be different.

5.4. Examples

5.4.1. Successful data transfer or status response

```
HTTP/1.1 200 OK
Cache-Control: no-cache
X-HOVTP-Environment: Production
X-HOVTP-Last-Serial-Number: 12345
X-HOVTP-Keep-Alive-Interval: 20
```

5.4.2. Failure due to mismatched environment

```
HTTP/1.1 453 Bad Environment
X-HOVTP-Environment: Test
X-HOVTP-Last-Serial-Number: 12345
```

5.4.3. Failure due to desynchronization of serial number

```
HTTP/1.1 450 Out Of Synchro
X-HOVTP-Environment: Production
X-HOVTP-Last-Serial-Number: 12345
X-HOVTP-Error-Reason: Expected serial number 12346
```

5.4.4. Failure due to invalid data

```
HTTP/1.1 451 Data Error
X-HOVTP-Environment: Production
X-HOVTP-Last-Serial-Number: 12345
X-HOVTP-Keep-Alive-Interval: 20
X-HOVTP-Error-Reason: The XML of the ODF message is not valid.
```

5.4.5. Failure due receiver not running or not ready to receive

```
HTTP/1.1 503 Service Unavailable
```

- or -

Timeout while trying to establish the connection

6. Keep-alive mechanism

In contrast to OVTP, the keep-alive mechanism has been integrated into the protocol.
Keep-alive messages are performed using status requests described in chapter *4.2 Status request*.
As those status requests do not impact serial numbering, there is no interference between the data transfers and the keep-alive mechanism. However, to avoid using unnecessary bandwidth while data transfers are occurring, they SHOULD only be sent while the connection is idle, i.e. when no data transfer has occurred for a certain period of time. In addition, as data transfer responses include the same information, it would be redundant to have both at the same time. Figure 3 below shows an example of when keep-alive should be performed.

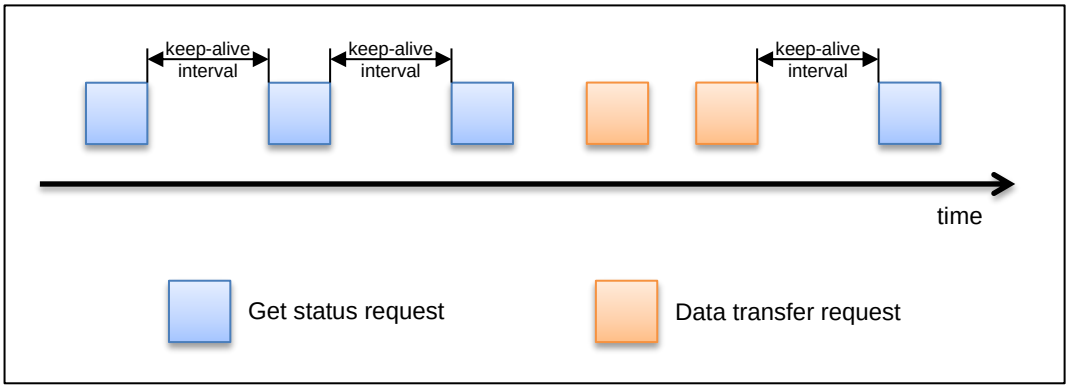


Figure 3 – Data transfers and keep-alive example

6.1. Keep-alive interval advertisement by receiver

The receiver can advertise a recommended value for the keep-alive interval in the *X-HOVTP-Keep-Alive-Interval* response header. The senders SHOULD respect this value, but are not required to do so. This way, senders having multiple distinct receivers can respect the keep-alive interval recommended by the each receiver without having to pre-configure them in advance.
If used, the advertised interval MUST be greater than 0.

7. Annexe A – Failover

7.1. Topology scenarios

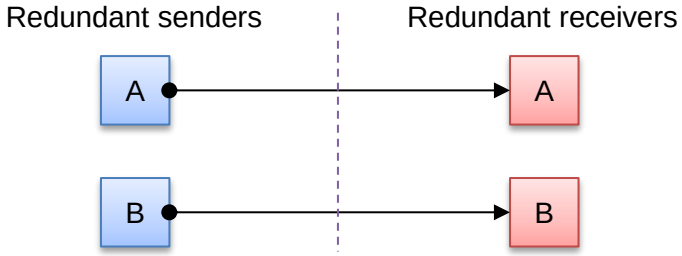
This chapter describes the different topology scenarios this protocol must support. To allow any type of integration, the scenarios range from the simple topology with no redundancy or failover to a fully redundant crossed topology.

7.1.1. Primary-only

No redundancy/failover

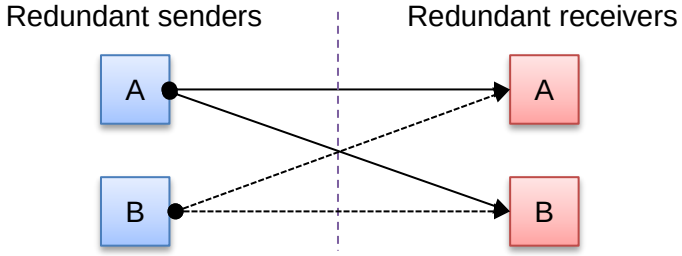
This scenario represents the simplest use-case without redundancy or failover. It allows a straightforward integration with limited development and configuration required from the integrator.

7.1.2. Primary/secondary



This scenario represents a topology with a simple redundancy by duplicating the sender/receiver pair. Each receiver has only knowledge of one receiver, and both receivers get the data, requiring multiplexing on the receiving side.
Alternatively, the *B* pair of sender and receiver could be brought up only in case of failure of A, simplifying integration by having data received only by A or B.

7.1.3. Crossed



This scenario represents a topology with a component redundancy by duplicating the connexion and ensuring hot switch of the senders. Only one of the senders is active at the same time. However as a clear improvement from current existing solution, B connexions will be checked and monitored through the keep-alive to ensure the link is available before doing the switch.

8. Annexe B – ODF Data Layer

Data Type: ODF

8.1. Request headers

To facilitate early-processing or routing of an ODF message without parsing the message itself, additional data headers are defined. They MUST contain the same value as their corresponding XML attribute in the *OdfBody* element of the message.

Header Name	Required?	Values	Remarks
X-ODF-DocumentCode	yes	As per ODF specification	
X-ODF-DocumentSubcode	no		Optional
X-ODF-DocumentType	yes		
X-ODF-DocumentSubtype	no		Optional
X-ODF-Version	yes		
X-ODF-Serial	yes		

8.2. Response headers

None