# Sale to POI Transport Protocols

## Retailer Protocol Working Group

**Protocol Version 3.1**

**Document Version 3.1**

**31 July 2017**

# Revision History

| Version | Date | Object |
|---|---|---|
| 0.90 | 17 Nov. 2009 | Draft for external consultation |
| 1.0 | 10 Oct. 2010 | Version 1.0 |
| 1.9 | 15 Feb. 2013 | Draft for external consultation |
| 2.0 | 02 April 2013 | Version 2.0 |
| 2.9 | 04 May 2015 | Draft for external consultation |
| 3.0 | 3 October 2016 | Version 3.0 |
| 3.1 | 21 November 2017 | Version 3.1 – Transport protocols in a separate document |

# Table of Contents

# Figures

# 1 Introduction

## 1.1 Purpose of the Document

This document lists the transport protocols which are selected to support the Sale to POI application protocol, how to manage transport services, and defines their use in order to provide the transport services required by the application level. The transport protocols that can be used include the TCP protocol , the HTTP protocol, the HTTPS protocol with HTTP over SLL/TLS, and SLL/TLS supporting directly the application level .

Messages management in described in Sale to POI Protocol specifications itself.

## 1.2 General Organisation

The Retailer Protocol follows the traditional organisation in five layers of the model defined for the TCP/IP protocol suite.

The two layers we are interested in are:

- The Application Layer where the protocol specified here is located, and
- The Transport Layer that the Application protocol interfaces with the Transport Services.

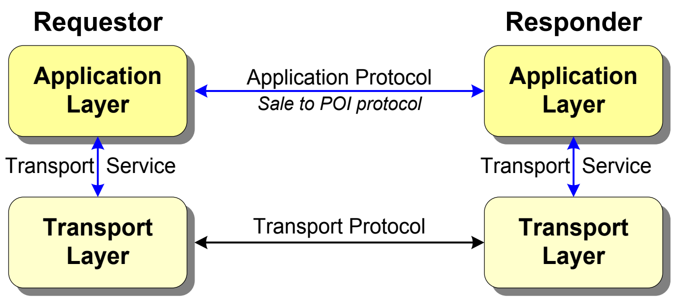

**Figure 1: Protocols Organisation**

The Sale to POI Application Protocol is independent of the Transport Protocol, and can be used without modification with various Transport Protocols, at the exception of the interface to the Transport Services, especially the transport addresses.

The Transport Protocol and therefore the Application Protocol are also independent of the communication infrastructure used below these layers, at the exception of the quality of service of the communication involving the tuning of some value of configuration parameters (e.g. value of timeout).

The Transport Protocol is a standard protocol allowed by the Application Protocol. The Application Protocol requires the Transport Services which can be used by the Application Protocol, and specifies the way to use them.

If a standard Transport Protocol authorized by the Application Protocol does not offer a particular Transport Service required by the Application Protocol, the Application Protocol specifies this Service through a Transport Adaptation Layer. This is for instance the case for the application message delimitation, which is a service not provided by the TCP Transport Protocol, but required for the decoding of a XML message before to deliver it to the Application Layer. Depending on the implementation of the application protocol, it could also be used to provide a particular flow control, for the connection management, etc...

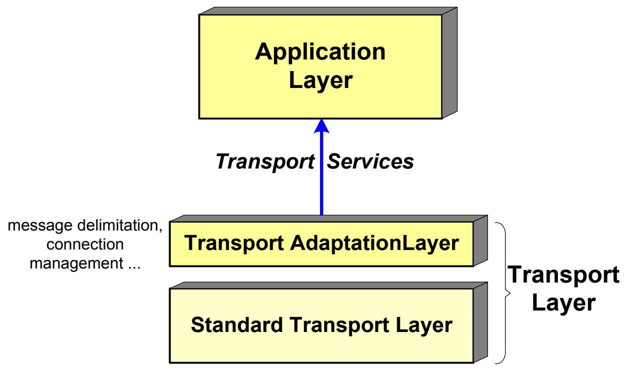

**Figure 2: Transport Adaptation Layer**

As far as the transport services and their usage are well defined, the definition in the future of a new Transport Protocol compliant to the defined Transport Service, will not impact the specifications of the Application Protocol.

# 2 Transport Services Management

This section specifies the transport services used by the Sale to POI protocol. The section is completed by the state diagrams of the two systems, and their configuration parameters.

## 2.1 Transport Connection Handling

### 2.1.1 General Rules

One or several transport connections are established between components of the Sale System and the POI System, as described in the nexo Sale to POI Architecutre document.

The number of transport connections open between a Sale Terminal and a POI Terminal or between the Sale Server and the POI Server is a configuration parameter.

The Sale system has always the initiative of the exchanges at the application level[1], i.e. the exchange of messages defined in this specification. However, transport connections can be initiated by either the Sale System or the POI System.

In addition, depending on the transport protocol (TCP, HTTP…), the POI may or may not open its own transport connections during the processing of a Sale System request, for the request of device exchanges.

Rule 1: The Sale System can open a transport connection for requesting an application message exchange with the POI. If successful, the Sale System is then responsible for initiating the application exchange.

Rule 2: The POI System can open a transport connection. If successful, the Sale System is then responsible for initiating the application exchange. If connection fails, POI system must retry indefinitely to connect to Sale System with an adaptative delay between attempts.

When transport connection are open by the Sale System, if all transport connections are closed, the POI System cannot send unsolicited notification to the Sale System.

The transport connection is closed by the transport adaptation layer only when this layer cannot continue the processing (e.g. message too big for the transport service layer, response to the transport connection request…), as defined in the section **Erreur ! Source du renvoi introuvable.** ***Erreur ! urce du renvoi introuvable.***.

[1] At the exception of the Event notification sent by the POI system.

### 2.1.2 Standard Sequence Flow

Sequence flow of a message pair exchange inside a transport connection initiated by the Sale System contains the steps presented below.

1. The Sale System component sends a transport connection request to the POI System component. The Sale System arms the timeout TC1 to wait for the transport connection response from the POI System.
2. The POI System receives the transport connection request, accepts it and sends a transport connection confirmation. At this time the transport connection is established for the POI System, which can send message, according to the dialogues and message specification.
3. After reception of the connection confirmation, the transport connection is established for the Sale System, and disarms the timeout TC1.
4. The Sale System and the POI System could exchanges messages of the Sale to POI protocol on this transport connection.
5. When there is no message pair in progress[2] and it decides to stop dialogue between these two entities, the Sale System sends a transport disconnection request. The connection is then considered by the Sale System as closed, and it cannot receive any data on this transport connection.
6. The POI System receives the transport disconnection request and sends a transport disconnection response to the Sale System[3]. The transport connection is then considered as released for the POI System.

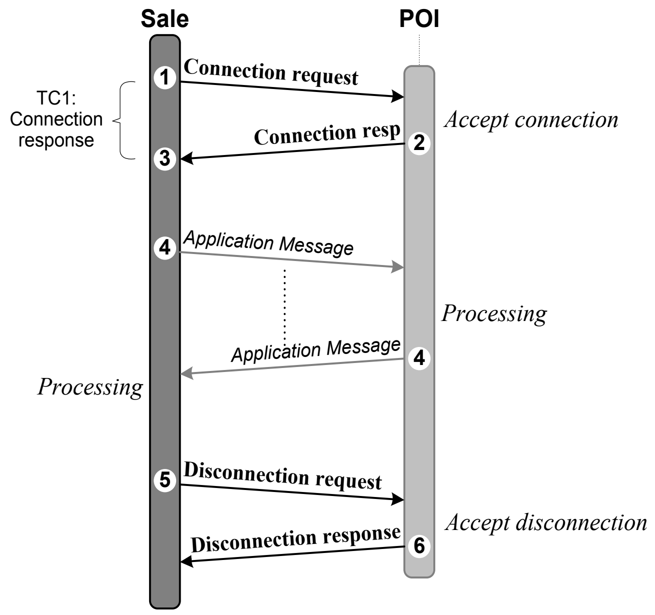

**Figure 3: Sale Initiated Transport Connection Standard Sequence Flow**

[2] A good practice is that the receptor of the last response closes the transport connection. This is normally the case because, with the exception of the notification from the POI (which are unsolicited), the Sale System always receive the last response message of a dialogue.

[3] On the TCP protocol implementations, only the protocol stack receives this request, because the Sale System has already closed the connection.

Sequence flow of a message pair exchange inside a transport connection initiated by the POI System contains the steps presented below.

1. The POI System component sends a transport connection request to the Sale System component. The POI System arms the timeout TC1 to wait for the transport connection response from the Sale System.
2. The Sale System receives the transport connection request, accepts it and sends a transport connection confirmation. At this time the transport connection is established for the Sale System, which can send message, according to the dialogues and message specification.
3. After reception of the connection confirmation, the transport connection is established for the POI System, and disarms the timeout TC1.
4. The Sale System and the POI System could exchanges messages of the Sale to POI protocol on this transport connection.
5. When there is no message pair in progress and it decides to stop dialogue between these two entities, the Sale System sends a transport disconnection request. The connection is then considered by the Sale System as closed, and it cannot receive any data on this transport connection.
6. The POI System receives the transport disconnection request and sends a transport disconnection response to the Sale System. The transport connection is then considered as released for the POI System.

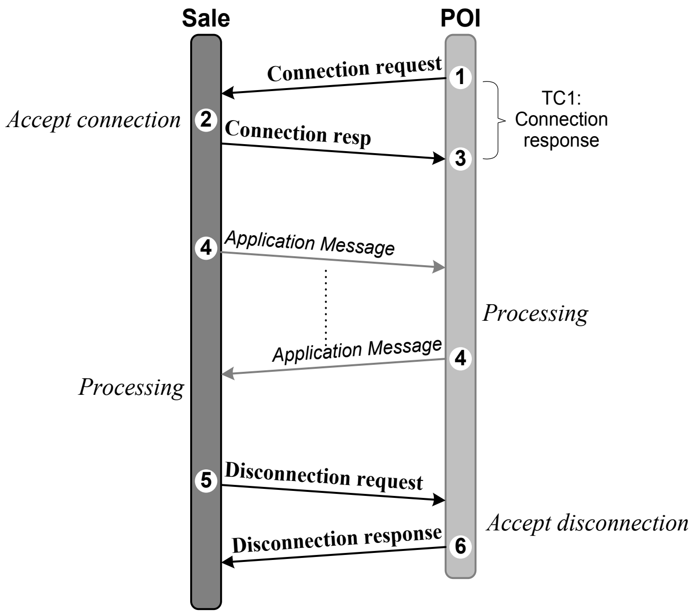

**Figure 4: POI Initiated Transport Connection Standard Sequence Flow**

## 2.2 Data Transfer

### 2.2.1 General Rules

Connections cannot be assigned to a particular pair of Sale Terminal and POI Terminal. Only messages tell which components of the Systems are the sender and the receiver of the message. So it is not recommended, except for a physical link between Terminals, for the POI to restrict an established transport connection to one Sale Terminal only.

Data are transferred inside a transport connection, at the request of the Application protocol to send a Sale to POI message which is the atomic quantity of data exchanged by the application protocol.

The dialogue of application messages might result in sending a message and receiving another message at the same time. Typically, during the reception of a message, which could be splitted in several segments due to the communication infrastructure, the application protocol could request to send a message.

In some cases the application requires sending a new message before the previous one was completely sent by the transport layer. To enable the reception and recognition of a message, the sender has to serialise the sending of consecutive messages.

### 2.2.2 Standard Processing Flow

For sending a message:

- The Application Protocol layer requests to send an application message to the interface of the transport protocol,
- The transport interface adds four bytes containing the length of the application message, and requests to the transport to send these four bytes followed by the application message.

For the reception of a message, the interface to the transport protocol:

- Waits for the reception of four bytes to get the length *L* of the message to receive, and arms the timeout TC2 to monitor the reception of the complete application message.
- Waits for the reception *L* bytes to provide a message to the Application Protocol layer.
- At the reception of the last data unit of this message, top the TC2 timer, and deliver to the application layer, the *L* bytes, content of the application message.

These processing flows are included in the general state diagrams of a transport connection management for the Sale and the POI Systems.

## 2.3 Addressing

A transport address is the identification of a peer, Requestor or Responder, which permits to establish a transport connection with this peer.

The form of the transport address depends on the type of transport protocol using this address (e.g. an IP address or a DNS name of the node, and a TCP port for the TCP transport protocol).

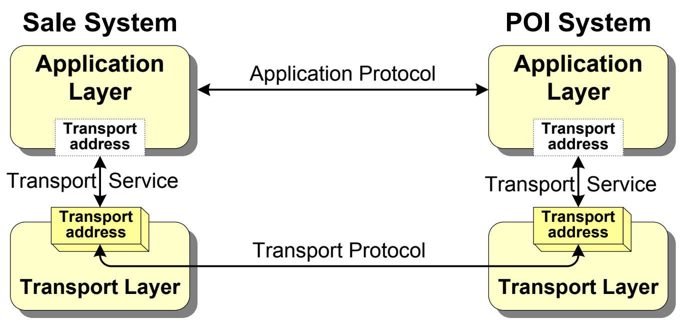

**Figure 5: Transport Address**

The transport address of the Responder is necessary for the Requestor to establish a transport connection with this Responder.

A transport connection is identified at least by the couple of transport address at each end point of the connection. In case of network connected mode above the transport protocol, other information like the sequence of nodes composing the connection path.

The transport addresses of every POI components to connect are configuration parameters used at the application level by the Sale System as a destination address to establish transport connection carrying out exchange of messages.

Each application protocol has to define assignment of the POI components transport addresses, globally for the whole application protocol, per class of message, or per service.

## 2.4 Transport Error Handling

This section defines errors detected by the transport protocol and transmitted to the application layer. The responsibility and the choice of error resolution are left to the application layer, unless the transport service layer needs to perform some action to continue to work.

### 2.4.1 Sale System Unable to Establish or receive a Transport Connection (ERTR01)

*Definition*

The transport layer is unable to establish the transport connection that the application layer of the Sale System has requested to open, because:

- The lower protocol layer has notified a "out of service" communication state as permanent or in response to the attempt to open the connection or the attempt to accept incoming connection.
- The TC1 timer has expired, and the connection in progress is considered as broken (and the connection establishment as a failure).
- The POI target component has denied the connection request, for instance because the POI component has reached the maximum number of connections it can manage in parallel.
- The transport layer has reached the maximum number of connection it can manage in parallel.

*Transport Service Behaviour*

The transport service layer considers the establishment of the transport connection as a failure, and notifies the result to the application layer of the Sale System.

The maximum number of connections open in parallel is fixed by the configuration parameter "Max Number of Connections" of both the Sale and the POI Systems.

### 2.4.2 Transport Connection Broken (ERTR02)

*Definition*

The transport layer realises that an open connection is broken. Several reasons might cause this error:

- The transport layer receives a disconnection request.
- An error occurred from lower level when the transport layer waits for the end of a message after the reception of the length prefix and incomplete data units.
- The lower level notifies the transport layer that a defect occurs (e.g. on the physical interface), and the transport connection is broken.
- The remote peer has not respected the state diagram of the connection management.
- The remote peer has released the transport connection.

*Transport Service Behaviour*

The transport service layer notifies the release of the transport connection to the application layer. Partial message are deleted, and not sent to the application layer.

### 2.4.3 Unable to Send a Message (ERTR03)

*Definition*

The transport layer cannot send an application message. Several reasons might cause this error:

- An error occurred from lower level when the transport layer sends an application message at the request of the application layer.
- The connection was broken just beforehand, but the application layer did not received the disconnection notification before the request to the transport layer to send a message.

*Transport Service Behaviour*

The transport service layer notifies the error to the application layer.

It is worth noticing that the transport is not always able to report error to the application layer. In particular when a connection is broken, the transport service has removed the connection context, and might have some difficulties to identify the context of the error.

### 2.4.4 Message Too Big (ERTR04)

*Definition*

The transport layer cannot receive an application message, because the prefix contains a message length above the limit of the transport layer memory.

*Transport Service Behaviour*

The transport service layer has an internal parameter containing the size limit of a message, or a global limit for the set of message buffers. When the limit is reached, transport layer:

- Release the connection on which the message has to be received.
- Notifies the error to the application layer.

As the transport layer cannot receive the message, it has to release the transport connection. Most of the time, errors of this class happen when there is a desynchronisation of message delimitation between the two peers.

The maximum size of an application message to be received is fixed by the configuration parameter "Max Message Size" of both the Sale and the POI Systems.

## 2.4.5 Late Arrival (ERTR05)

*Definition*

The transport service layer receives a data unit for the application after a timeout expiration and the request for a disconnection to the underneath transport connection.

The same phenomenon might arrive with a greater likelihood at the application layer.

The drawing below shows such error example, when the application timeout on the message response expires.

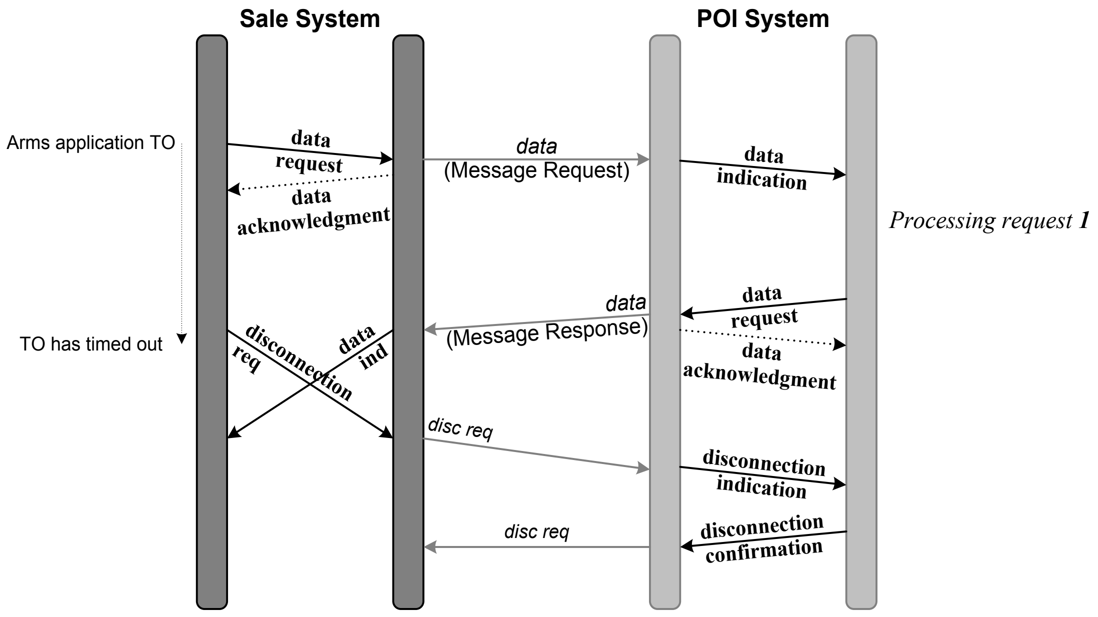

**Figure 6: Late Arrival Error Example**

*Transport Service Behaviour*

The transport service layer ignores the message or data units.

As already noticed, the transport has removed the connection context, and might have some difficulties to identify the context of the error.

Implementation has to be careful not to assign the data unit to another connection.

## 2.4.6 Max Global Number of Connections (ERTR06)

*Definition*

On the arrival of an incoming connection request, the transport service layer cannot open a new transport connection, because it has reached the maximum number of transport connections it can handle in parallel.

The same phenomenon might arrive with a greater likelihood at the application layer.

*Transport Service Behaviour*

The behaviour is implementation dependant. The transport service layer or the application layer has the configuration parameter "Max Number of Connections" containing the limit number of open connections. When the limit is reached, transport layer:

- Decline the incoming connection request.
- Notifies the error to the application layer on the connection requestor side.

### 2.4.7 Incomplete Application Message (ERTR07)

*Definition*

After the reception of an application message header, the transport layer arms the timeout TC2 to monitor the reception of the complete application message.

The expiration of the TC2 timeout terminates the reception of the application on this error case which might happen:

- When a serious error occurs on a lower communication level or on the communication infrastructure between the 2 peers.
- The remote transport or application layer encounters a serious error, or is out of order.

*Transport Service Behaviour*

The transport service layer:

1. Discards the uncompleted application message,
2. Close the transport connection to avoid problem of data synchronisation (the next data unit can be the start of the next application message, or part of the previous application message),
3. Notifies the error to the application layer.

### 2.4.8 Other Errors

All other errors are detected or resolved at the application layer, as:

- Non understandable message: all decoding of message are made by the application layer, so this error is detected and resolved at the application layer.
- Timeout: most of the time, the application layer limits the time the other peer can has to process a message and answer to it. In this case, it is recommended to close the transport connection.
- Late Arrival Message: in case of a big number of events to perform, some crossing event can be observed between the application layer and the transport layer. A disconnection request from the application layer can cross an application message from the transport layer.
- Busy situation: when the application layer has not enough resources to process al the message received, it can answer that it is busy at the application level to the other peer.

Depending on the relationship between the Sale and POI component (Terminal or Server), the application limits the number of transport connections between these two components (configuration parameter "Number of Connections").

Example
The relationship between a Sale Terminal and a POI Terminal is a Terminal to Terminal link (i.e. the parameter "Terminal Relation" has the value "Term2Term"), and the parameter "Number of Connections" for these two terminals is 1. The Sale Terminal may only open one transport connection towards this POI Terminal. Be careful to the crossing of connection and disconnection as described in the figure below.

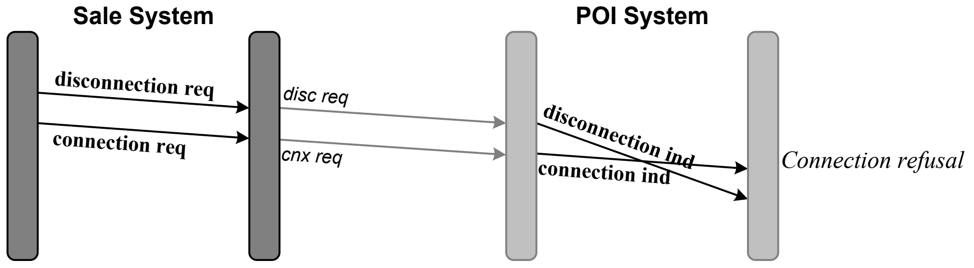

**Figure 7: Crossing of Disconnection and Connection**

## 2.5 State Diagrams

### 2.5.1 Sale System

This is the state diagram of the Sale System to manage a transport connection. The states and events of this diagram are detailed in the tables below.

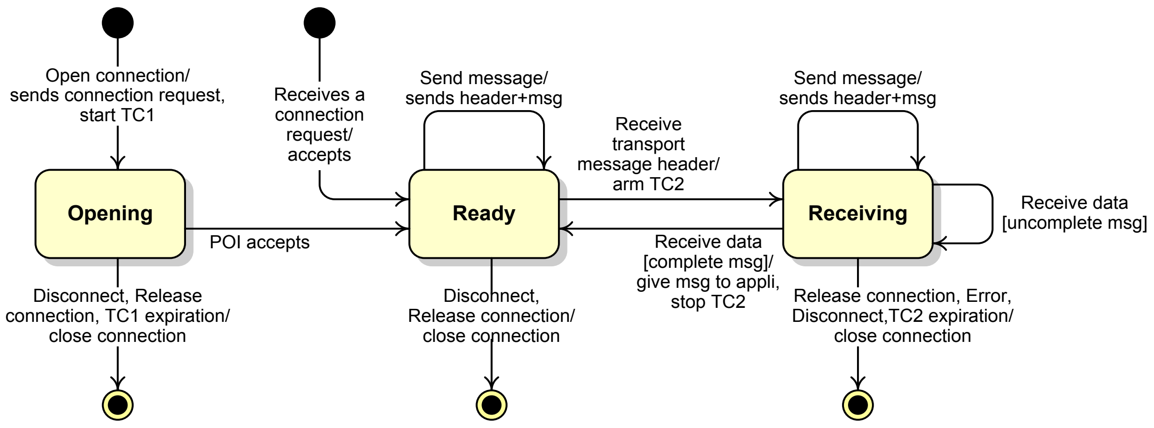
Open connection/ sends connection request, start TC1
Receives a connection request/ accepts
Send message/ sends header+msg
Receive transport message header/ arm TC2
Send message/ sends header+msg
Opening
POI accepts
Ready
Receiving
Receive data [uncomplete msg]
Receive data [complete msg]/ give msg to appli, stop TC2
Disconnect, Release connection, TC1 expiration/ close connection
Disconnect, Release connection/ close connection
Release connection, Error, Disconnect,TC2 expiration/ close connection

**Figure 8: Sale System Transport Connection Management State Diagram**

| *State* | *Definition* |
|---|---|
| **Opening** | This state exist only when the Sale System opens the connections. The transport connection is opening. The Sale System has sent a connection request to the POI component, and is waiting for the response. |
| **Ready** | The transport connection is open. The connection between the Sale System and the POI component is established and Sale System is waiting to send or receive messages. |
| **Receiving** | The transport connection is open. The Sale System has received the transport protocol header containing the length of a message from the POI component, possibly part of the message, and is waiting for the end of the message. |

**Table 1: Transport Connection Sale System States**

| *Event* | *Definition* |
|---|---|
| **Open connection** | The application protocol request to open the transport connection. |
| **Connection request** | The remote peer (POI component) sent a connection request. |
| **Release connection** | The application protocol request to release the transport connection. |
| **Disconnect** | The remote peer (POI component) or the communication infrastructure releases the transport connection. |
| **POI accepts** | The POI accepts to open the transport connection. |
| **Send message** | The application protocol request to send an application message. |
| **Receive transport message header** | The complete message length has been received in the transport message header (the four bytes of the length header for TCP, or HTTP header containing the `Content-Length` header field. |
| **Receive data** | Data are received on the transport connection. |
| **TC1/TC2 expiration** | The timer TC1 (resp. TC2) has expired. |
| **Error** | An error is detected on the transport interface: Transport Connection Broken, Unable to Send a Message, or Message Too Big. |

**Table 2: Transport Connection Sale System Events**

### 2.5.2 POI System

This is the state diagram of the POI System to manage a transport connection. The states and events of this diagram are detailed in the tables below.

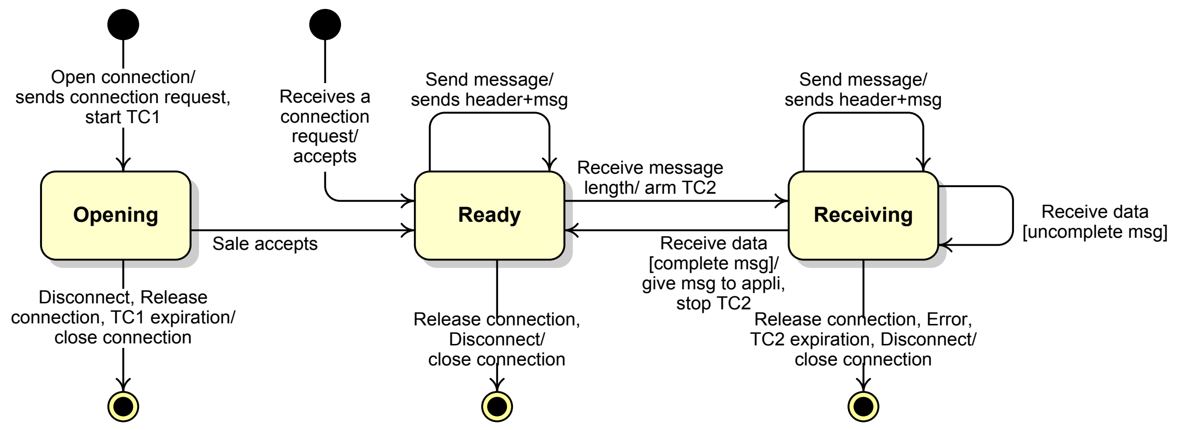
Open connection/ sends connection request, start TC1
Receives a connection request/ accepts
Send message/ sends header+msg
Send message/ sends header+msg
Opening
Ready
Receiving
Receive message length/ arm TC2
Receive data [uncomplete msg]
Sale accepts
Receive data [complete msg]/ give msg to appli, stop TC2
Disconnect, Release connection, TC1 expiration/ close connection
Release connection, Disconnect/ close connection
Release connection, Error, TC2 expiration, Disconnect/ close connection

**Figure 9: POI System Transport Connection Management State Diagram**

| *State* | *Definition* |
|---|---|
| **Opening** | This state exist only when the POI System opens the connections. The transport connection is opening. The POI System has sent a connection request to the Sale component, and is waiting for the response. |
| **Ready** | The transport connection is open. The connection between the Sale System and the POI component is established and Sale System is waiting to send or receive messages. |
| **Receiving** | The transport connection is open. The POI System has received the transport protocol header containing the length of a message from the Sale component, possibly some part of the message, and is waiting for the end of the message. |

**Table 3: Transport Connection POI System States**

| *Event* | *Definition* |
|---|---|
| **Open connection** | The application protocol request to open the transport connection. |
| **Connection request** | The remote peer (Sale component) sent a connection request. |
| **Release connection** | The application protocol request to release the transport connection. |
| **Disconnect** | The remote peer (POI component) or the communication infrastructure releases the transport connection. |
| **Send message** | The application protocol request to send an application message. |
| **Receive transport message header** | The complete message length has been received in the transport message header (the four bytes of the length header for TCP, or HTTP header containing the `Content-Length` header field. |
| **Receive data** | Data are received on the transport connection. |
| **TC1/TC2 expiration** | The timer TC1 (resp. TC2) has expired. |
| **Error** | An error is detected on the transport interface: Transport Connection Broken, Unable to Send a Message, or Message Too Big. |

**Table 4: Transport Connection POI System Events**

## 2.6 Transport Service Configuration Parameters

This section summarises the set of configuration parameters required by the transport service management. These parameters can be for instance downloaded by the EPAS TMS protocol or another way, and their values cannot be fixed by the protocol specifications as their value depends on the communication infrastructure, or the architecture of the Systems.

The Sale and the POI Systems[4] have the following configuration parameters for the transport service management:

| | |
|---|---|
| *Name* | **TC2** |
| *Definition* | Application message reception timer |
| *Usage* | This timer is armed at the reception of the application message prefix to supervise the reception of the complete application message.<br>It allows the detection of an incomplete application message reception, avoiding a deadlock of the transport connection. |
| *Specification* | **2.2** Data Transfer<br>**Erreur ! Source du renvoi introuvable. Erreur ! Source du renvoi introuvable.** (Message Too g, Incomplete Application Message) |

| | |
|---|---|
| *Name* | **Max Number of Connections** |
| *Definition* | Maximum number of transport connections the Sale or The POI System component can manage simultaneously. |
| *Usage* | To avoid error on opening too many transport connections, from the application protocol layer or the remote application protocol. |
| *Specification* | **Erreur ! Source du renvoi introuvable. Erreur ! Source du renvoi introuvable.** (Unable to tablish a Transport Connection, Max Global Number of Connections) |

| | |
|---|---|
| *Name* | **Max Message Size** |
| *Definition* | Maximum size of a message the Sale or The POI System component can receive. |
| *Usage* | To detect error of messages length, in particular synchronisation of messages exchange, with the prefix length and the content of the message. |
| *Specification* | **Erreur ! Source du renvoi introuvable. Erreur ! Source du renvoi introuvable.** (Message Too g) |

**Configuration 1: Transport Service**

The Sale System has in addition the following configuration parameters for the transport service management:

| | |
|---|---|
| *Name* | **TC1** |
| *Definition* | Transport connection establishment timer |
| *Usage* | After the sending of a transport connection request, the Requestor arms the TC1 timer to watch on the response from the Responder. TC1 is reset on the transport connection confirmation reception. |
| *Specification* | 2.1 Transport Connection Handling<br>**Erreur ! Source du renvoi introuvable. Erreur ! Source du renvoi introuvable.** (Unable to tablish a Transport Connection) |

**Configuration 2: Sale Transport Service**

[4] Of course, the values for the Sale System and the POI System might be different.

For each Terminal to Terminal relationship, and the Server to Server if any:

| *Name* | **Number of Connections** |
|---|---|
| *Definition* | Number of transport connections the Sale System component may establish simultaneously toward this POI System component. |
| *Usage* | To avoid error on opening too many transport connections between two components. The number depends on the type of relation between these two components defined by the parameter "Terminal Relation" and "Server Relation" in the Architectures and Models document. |
| *Specification* | **Erreur ! Source du renvoi introuvable. Erreur ! Source du renvoi introuvable.** (Other Errors) |

| *Name* | **POI Component Address** |
|---|---|
| *Definition* | Transport address of the POI Component |
| *Usage* | This (or these) transport address is used by the Sale System to establish a transport connection with a POI System component. |
| *Specification* | **Erreur ! Source du renvoi introuvable. Erreur ! Source du renvoi introuvable.** |

**Configuration 3: Transport Connection**

# 3 Transport Protocol

## 3.1 Introduction

Given the great variety of contexts, particularly for the communication between systems and between terminals, the number of transport protocols to take into account is limited. For each transport protocol which is used by Application Protocols, the following items have to be described:

- Specification of transport services required by the application protocol, which are not provided by the transport protocol.
- Specification of the ways transport services are used by the application protocol, that are particular to this transport protocol.
- Configuration parameters to be used to manage the transport protocol: addressing, time-out, data context, limits…

For the current specification two transport protocols may be used:

- TCP (Transmission Control Protocol),
- HTTP (Hypertext Transfert Protocol)

They are specified in the two following sections.

## 3.2 TCP Protocol

The TCP transport protocol is specified in the RFC 793 (Transmission Control Protocol - DARPA Internet Program) in September 1981.

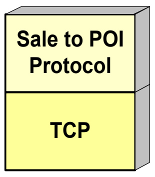

Figure 10: TCP Protocol

### 3.2.1 Typical Use

Because of its widespread availability, the TCP transport protocol remains one of the favourite transport protocols. It can be used as:

4. An end-to-end transport protocol between two entities at each extremity of the Application Protocol.

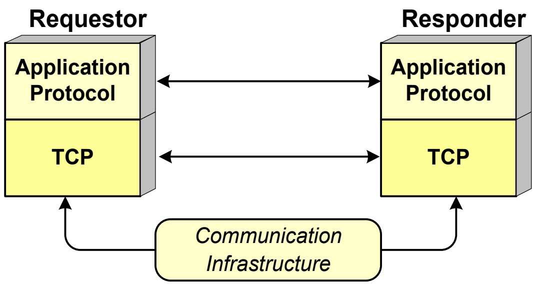

Figure 11: Peer-to-peer TCP Transport Protocol

5. An interface to a gateway or a driver to make the conversion of transport protocol with the other side of the Application Protocol. This case is only a transitory solution allowing the adaptation of a legacy system to the Application Protocol, avoiding the specification of all the existing communication protocols.

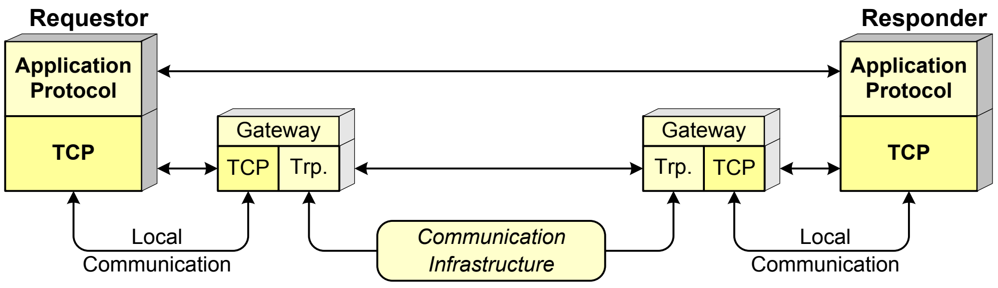

Figure 12: Gateway TCP Transport Protocol

6. An interface between two applications using Application Protocol when they are on the same platform.

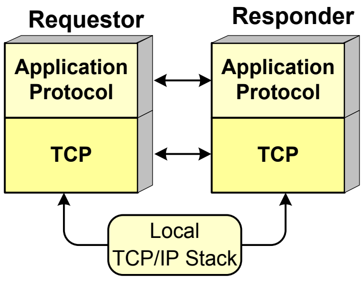

Figure 13: Local TCP Transport Protocol

### 3.2.2 Message Delimitation

The TCP protocol is a stream protocol which does not offer message delimitation or data-unit delimiting. It uses the general mechanism of message delimitation provided for all the transport protocols.

**Definition**

Message delimitation or data-unit delimiting is the service provided by the transport protocol allowing the recognition of an application data-unit by the transport protocol, in order to transfer a complete application message to the Application Protocol.

**Specifications**

Application messages are preceded by four bytes containing the length in network order[5], of the application message. The length does not include the four bytes containing the length.

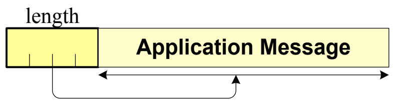

**Figure 14: Message Header Length**

Rule 1: If the four bytes of the message length are all equal to zero, it is considered as a zero length application message. These four bytes are ignored, and the next four bytes if any, are interpreted as the header length of the following application message.

**Notes**

An example of application message with the length, to be sent on the transport connection, is provided in  4.

Use of message delimitation at the transport level ensures:

- Independence of this service on the data coding used by the application, which can provide this functionality (e.g. ASN.1/BER data coding).
- Restrict message delimitation to the transport layer interface avoids partial progressive decoding of the message mixed with reception of message fragments,
- Separates decoding of the message from its reception processing.

Use of the message length to delimit a message has the advantage to be able to send any value in the application message, but prohibits resynchronisation of message after a data loss.

Use of a fixed number of bytes to contain the length of the message facilitates implementation of message reception.

Use of specific characters to delimit the message requires either a parsing of the message to treat these specific characters, either to forbid use of these characters by the application in the messages.

A number of four bytes has been chosen to carry out the message length, because of the 64 K bytes limitation for a 2-bytes length, and the effort required passing from 2-bytes to 4-bytes length.

[5] Most significant byte first (i.e.big endian).

### 3.2.3 Transport Connection Handling

To allow a maximum of flexibility, the Sale System has the entire responsibility to manage the connection at its convenience, and during the time it considers necessary to its own management.

The Sale System might decide to use permanent transport connections, opens a connection for each message, or mixes any type of management according the requirements of the sales.

Connection Opening

General rules for opening connections applies as defined in section 2.1. Transport Connection Handling.

Connection Release

The Sale System may release a transport connection whenever the application judges it appropriate. The Sale System has to ensure that all the response messages have been received before to close the transport connection.

Rule 1: The Sale System may close a transport connection at any time.
The POI System may close the transport connection only in case of error.

When the transport connection is closed, the POI System cannot send unsolicited notification to the Sale System.

Keep Alive message

In a connectionless network mode (as IP), it could be difficult to detect a broken transport connection without sending data. In particular, some protocol stacks does not provide such out of band "keep alive" service.

The *Message Delimitation* transport services defined in the section 3.2.2 provides the "Keep Alive" service:

- After a certain time without exchange on an open connection, send four null bytes (hexadecimal `00`) on the transport connection.

As the reception of those four null bytes is considered as an application message of length zero, they are ignored by the receiver.

### 3.2.4 Addressing

A transport address for the TCP protocol is composed of:

1. The *IP Address* or the *DNS Name* to resolve of the host on which the application protocol lives.
2. The *TCP Port*, to dispatch the connections to the application.

## 3.3 HTTP Protocol

### 3.3.1 Introduction

The current version of the HTTP protocol is v1.1 specified in the RFC 2616 in June 1999 (Hypertext Transfer Protocol -- HTTP/1.1).

HTTP is an application protocol above the TCP transport protocol.

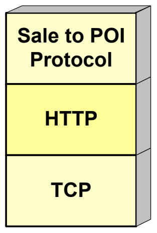

**Figure 15: HTTP Protocol**

The main advantages to use HTTP instead of TCP are:

- To traverse Firewalls without any addition for incoming connections in the communication infrastructure, as HTTP port is configured everywhere.
- Facilitate the remote location of payment services.
- Simplify the use of intermediary agent at the transport level, as very often used today in the card payment communications.
- Make possible the adoption of Web services. Web services have to be clearly studied before any modification of the specification to adopt them.

HTTP is a request/response protocol between HTTP Client and HTTP Server to exchange multimedia documents that might be used, instead of TCP, to support the Sale to POI protocol.

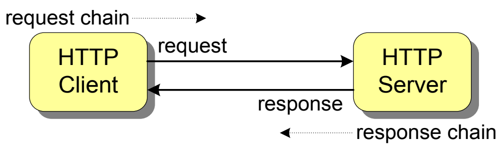

**Figure 16: Basic HTTP Communication**

HTTP has been designed to include HTTP Intermediaries between the Client and the Server, with dedicated functions.

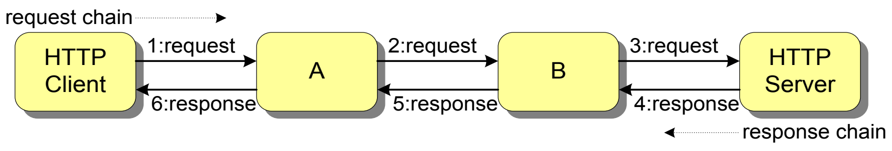

**Figure 17: HTTP Request/response Chain**

The structure of HTTP request and response messages contains three parts:

- The *Start Line*, indicating for a request the nature of the demand, and for a response the outcome of the request.
- The *Header Fields*, containing information on the management of the message or the protocol. Each *Field*, of the form *HeaderName:HeaderValue*, is located on a separate line. An empty line closes the *Header Fields* part of the HTTP message.
- The *Body*, containing the Sale to POI message, which is present if a message has to be sent.

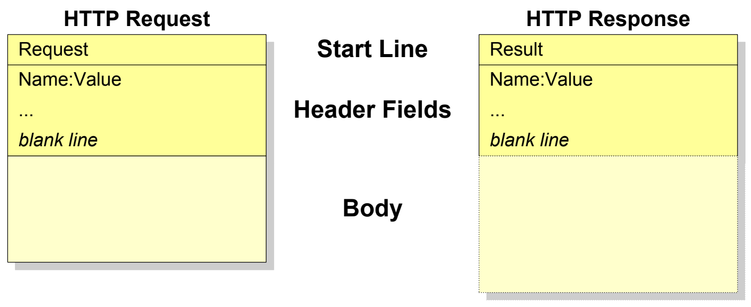

**Figure 18: HTTP Message General Format**

The header fields are separated in four classes:

- The *General HTTP Headers*, not specific to any particular kind of message (HTTP Request or HTTP Response) or message content (Body),
- The *HTTP Request Headers*, used only in HTTP Request messages,
- The *HTTP Response Headers*, used only in HTTP Reesponse messages,
- The *HTTP Entity Headers*, concerning the resource carried in HTTP Body.

It is recommended to place the general header fields first, then the request or response header fields, and the entity headers at the end of the HTTP header.

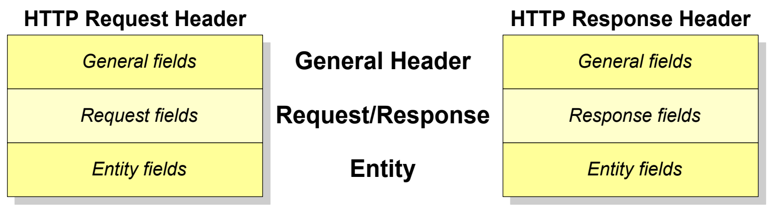

**Figure 19: HTTP Header Fields**

### 3.3.2 Transport Connection Handling

The standard sequence flow of an HTTP exchange is:

1. The HTTP Client open a TCP connection with the HTTP Server,
2. The HTTP Client sends an *HTTP Request* message,
3. The HTTP Server process the request of the client, and sends an *HTTP Response* message containing the result of the processing,
4. If the connection is persistent, the Client might send other *HTTP Request* messages, potentially in parallel,
5. The HTTP Client closes the TCP connection.

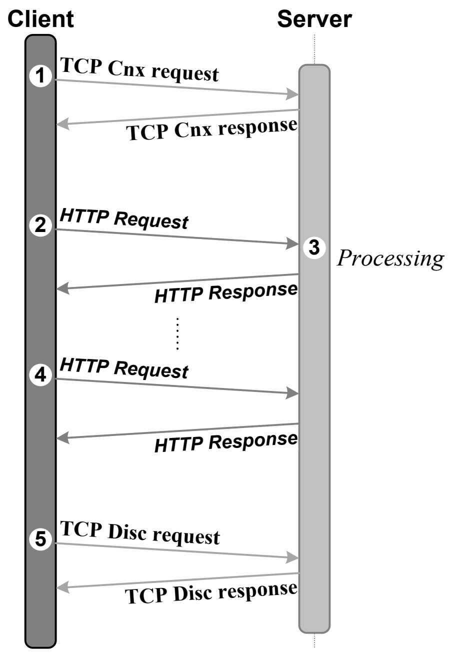

**Figure 20: Standard HTTP Sequence Flow**

Rule 1: An HTTP Request message body contains a Sale to POI request or notification message. An HTTP Response message body contains the related Sale to POI response message, and is empty if there is no response to the request message.

Rule 2: The Sale System may open a TCP connection for requesting a Sale to POI message exchange to the POI. The Sale System might decide to use persistent TCP connections, uses parallel TCP connections, opens a TCP connection for each exchange, or mixes any type of management according the requirements of the sales.

Rule 3: The POI System may open a TCP connection for requesting Sale to POI device message exchange to the Sale System:

1. During the processing of a Sale System Sale-to-POI request,
2. To send a notification.

The POI System might decide to use persistent TCP connections or not, uses parallel TCP connections but the TCP connection have to be close before sending the response to the initial Sale System request message.

### 3.3.3 Application Dialogues

To validate the rules and specify how they apply, this section presents the way HTTP is used for the application dialogues defined in the protocol specifications,in the message management section.

#### Service Dialogue

The service dialogue respects the following process flow:

1. If there is no open TCP connection between the Sale and the POI, the Sale opens a TCP connection.
   The Sale sends to the POI an HTTP Request containing the Service Request application message.
   The POI process the requested Service.
2. If a Device Request must be sent during the processing the POI open a TCP connection with the Sale.
3. The POI sends to the Sale an HTTP Request containing the Device Request application message.
4. The Sale performs the device request, and sends to the POI the Device Response application message,
5. The POI disconnects the TCP connection, when no more Device application request must be exchanged with the Sale.
6. At the end of the service processing, the POI sends an HTTP Response to the Sale containing the Service

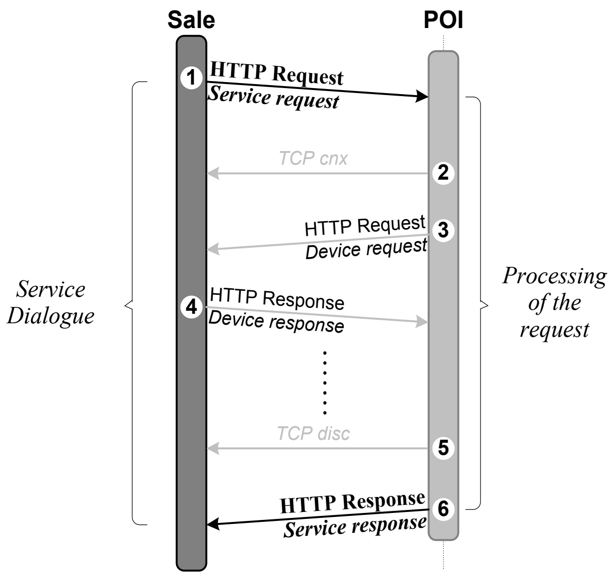

**Figure 21: HTTP Service Dialogue**

If there are no Device requests to send to the Sale when performing the service request, the steps 2 to 5 are skipped.

To be able for the POI system to be an HTTP client, it must know the TCP address of the Sale system requiring the Service.

During the processing of a Service dialogue, the POI Terminal can send several Device request messages in parallel, and is not required to wait the Device response message of the preceding request before sending the next one.

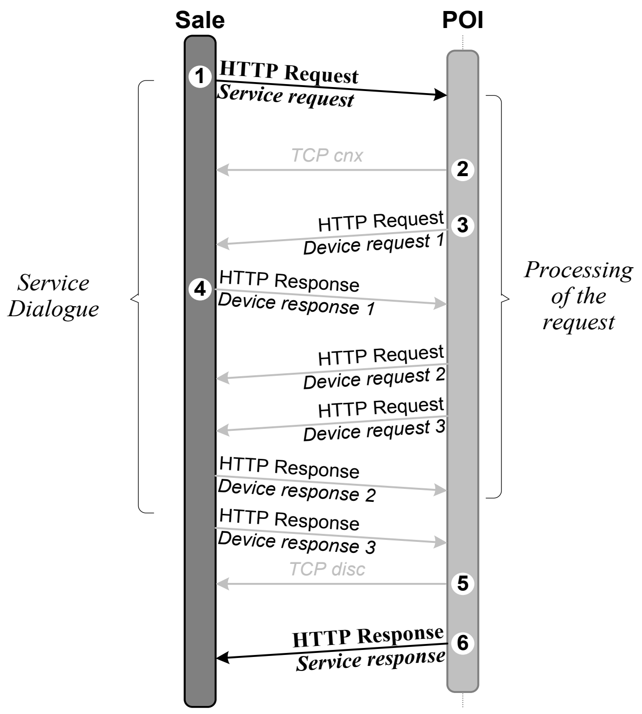
Sale
POI
1 HTTP Request Service request
TCP cnx 2
HTTP Request Device request 1 3
4 HTTP Response Device response 1
HTTP Request Device request 2
HTTP Request Device request 3
HTTP Response Device response 2
HTTP Response Device response 3
TCP disc 5
HTTP Response Service response 6
Service Dialogue
Processing of the request

**Figure 22: HTTP Service with Device Requests in Parallel**

When several Device requests are sent in parallel:

- The same TCP connection might be used, as HTTP allows on a persistent connection sending a request before having received the response of the previous request. The constraint is that the (HTTP) responses are sent in the order of the requests. So if the POI starts a Print then a Display, the Display response is received after the Print response.
- To avoid this constraint, several TCP connections might be open by the POI, and the Display response is independent from the Print response received in another TCP connection.

Some Device request application messages do not require response message, only the Device Request message is exchanged.

In this case, the Device request is sent in the body of an HTTP request, a related HTTP response is sent without body and the status code 204 (No Content).

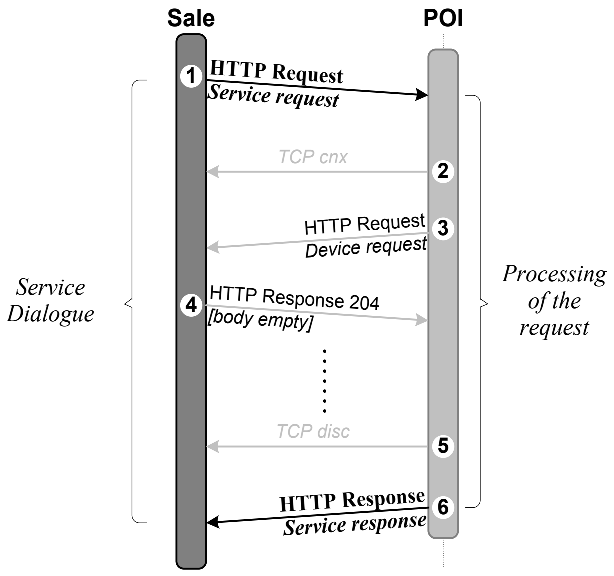

**Figure 23: HTTP Service and Device Request without Response**

## Error Resolution Dialogue

Error resolution dialogues are specified in the nexo Sale to POI protocol specification, in the error mamangement section.

The dialogue uses the Abort message. This message has no applicative response.

The standard abort processing with the HTTP protocol is the following:

1. The Service starts with the sending of a Service (or Device) application message request, inside an HTTP Request, by the Sale to the POI.
2. After a timeout, or an operator action, the Sale aborts the service, sending to the POI an HTTP Request containing an Abort application message.
3. An HTTP response associated with the Abort request without body (status code 204), is then sent by the POI.
4. An HTTP Response containing the Service response associated to the HTTP Request containing the Service Request.

The Abort Request has to be sent inside a different TCP connection if the Sale system wants to receive it before the Service response (both are allowed).

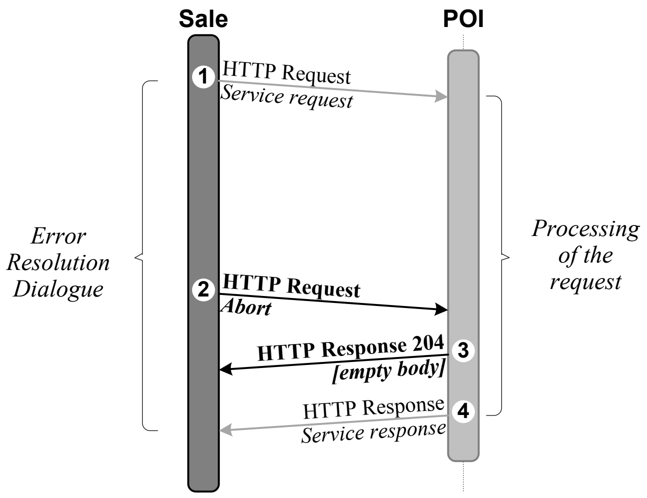
Sale
POI
1 HTTP Request Service request
Error Resolution Dialogue
Processing of the request
2 HTTP Request Abort
HTTP Response 204 [empty body] 3
HTTP Response Service response 4

**Figure 24: HTTP Error Dialogue**

If the Service request to abort is not found, an Event notification (Reject) has to be sent in the body of the associated HTTP Response.

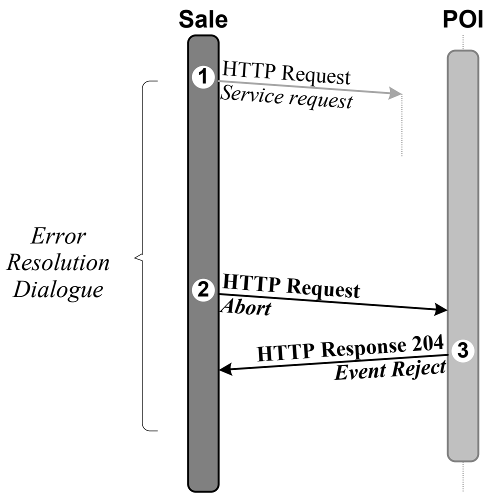
Sale
POI
1 HTTP Request Service request
Error Resolution Dialogue
2 HTTP Request Abort
3 HTTP Response 204 Event Reject

**Figure 25: HTTP Abort of an Unknown Request**

When the Service response is not received (e.g. disconnection), but the Service response was sent by the POI, an Event notification "Completed" has to be sent in the related HTTP Response.

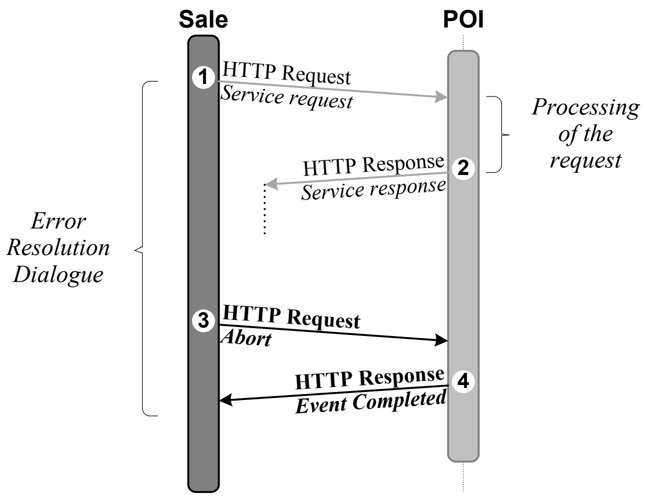
Sale
POI
1 HTTP Request Service request
Processing of the request
2 HTTP Response Service response
Error Resolution Dialogue
3 HTTP Request Abort
4 HTTP Response Event Completed

**Figure 26: HTTP Error Dialogue**

As a reminder, a TransactionStatus request has to be sent before any Abort as defined in the protocol specification..

The TransactionStatus request/response messages have to be sent in an HTTP Request/Response exchange.

A different connection has to be used to receive the TransactionStatus response before the Service response.

### Device Dialogue

The dialogue follows the standard http Request/Response exchange containing the Device request/response messages.

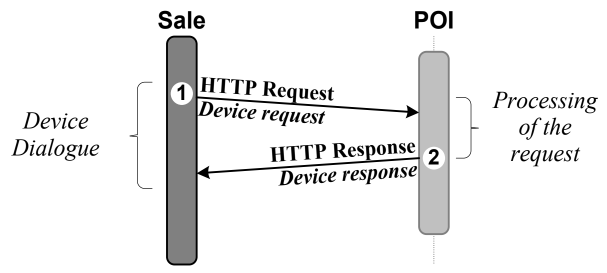

**Figure 27: HTTP Device Dialogue**

### Notification Dialogue

.

The dialogue is based on the Event Notification message. This message has no applicative response.

If the Event is directly related to the processing of a Service or a Device, that cannot be processed as a “Reject” or “Completed”, and without response (as for Abort or a reject), the event is sent directly in the HTTP Response of the Service or Device requests.

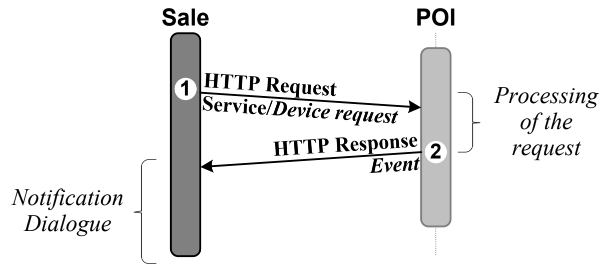

**Figure 28: HTTP Exchange for a Linked Notification Dialogue**

For other cases, the POI sends the Event notification in an HTTP Request, and the related HTTP Response is sent by the POI with the status code 204 and no message body.

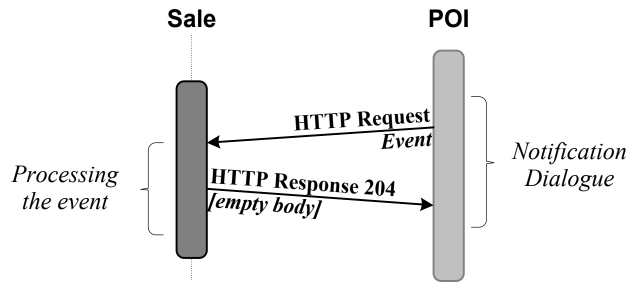

**Figure 29: HTTP Exchange for an Isolated Notification Dialogue**

### 3.3.4 Request Start Line

*POST* is the HTTP method used for the transport of Sale to POI request and response messages[6].

The URI never includes the host. The path in the URI contains:

- At the first level the name of the protocol "EPASSaleToPOI", and
- At the second level the *ProtocolVersion* "3.1".

To be able to make a first switch of the request, the query contains the following data elements of the message header:

- the *MessageClass*, with the name "Class"
- the *MessageCategory*, , with the name "Category"
- the *SaleID*, with the name "SaleID", and the value if the content is acceptable in a URI,
- the *POIID*, with the name "POIID", and the value if the content is acceptable in a URI.

The version of HTTP to use is 1.1

The format of the start line of an HTTP Request is summarised in the figure below.

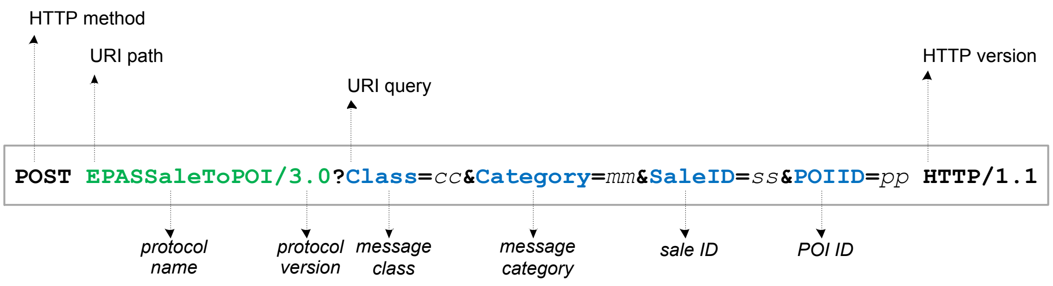

**Figure 30: HTTP Request Start Line**

Rule 1: All the non US-ASCII characters of the *SaleID* value or the *POIID* value must be encoded with their UTF-8 value using the percentage encoding, i.e. each byte is encoded as a characters triplet: the character "`%`" following by the two hexadecimal digits representing their octet's numeric value (the digits A-F and their lower case corresponding values `a-f` are equivalents).

For example the character 'é', which has the ASCII value `E9`, the utf-8 value `C3 A9`, will be encoded in `%C3%A9`.

[6] RESTFUL services will be added in a following version.

### 3.3.5 Error Handling and Response Start Line

The format of the start line of an HTTP Response is summarised in the figure below.

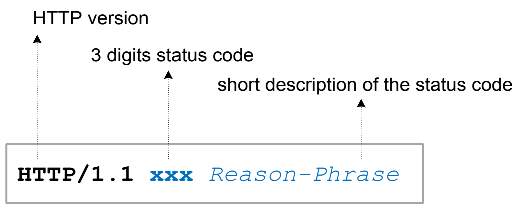
HTTP version
3 digits status code
short description of the status code
HTTP/1.1 xxx Reason-Phrase

**Figure 31: HTTP Response Start Line**

Rule 1: Informal status codes (`1xx`) are not allowed.

Rule 2: Two successful status codes (`2xx`) are allowed:

1) `200 OK`: when the Sale to POI message request included in the HTTP Request has been processed and a Sale to POI message response is present in the HTTP Response, whatever the response code of the Sale to POI message response.
2) `204 No content`: when the Sale to POI message request included in the HTTP Request has been validated and processed and the HTTP Response does not contain any Sale to POI message.

Rule 3: Redirection (`3xx`) and Server Error (`5xx`) status codes are allowed without any relationship to the Sale to POI protocol.

Rule 4: Client error status codes (`4xx`):

1) `408 Request Timeout`: is sent at the expiration of the TC2 timeout (see section 2.4.7 Incomplete Application Message (ERTR07) and section 2.5 State Diagrams).
2) `413 Request Entity Too Large`: is sent when the `Content-length` of the HTTP body is greater than the maximum size of the message (see section 2.4.4 Message Too Big (ERTR04)).
3) All other client error status codes are related to the HTTP Request message only.

### 3.3.6 HTTP General Header Fields

Exchange of application do not use any cache functionalities of HTTP, and the storage of Sale To POI messages must be forbidden.

Rule 1: The `Cache-Control` general header field is mandatory in both HTTP Request and HTTP Responses with the value `no-store`.
The `Pragma` general header field is mandatory in HTTP Request with the value `no-cache`, for compatibility with version 1.0 of HTTP.

Rule 2: Other HTTP request header fields:

1) The following fields should be absent, they have to be ignored:
   `Date`
   `Transfer-Encoding`
   `Pragma` (other than `no-cache`)
   `Upgrade`
   `Via`
   `Warning`
2) If present, the following fields have to be perform according to the HTTP specifications:
   `Connection`

### 3.3.7 HTTP Request Header Fields

Rule 1: If the `Accept` HTTP request header field is present, it must have the value `text/xml`, `application/asn1` or `application/json` according to the coding of the application message.

Rule 2: If the `Accept-Charset` HTTP request header field is present, it must have the value `utf-8`. The HTTP server has to ignore other value, and answer with the `utf-8` charset of the application response message.

Rule 3: The `Host` HTTP request header field is mandatory. It must contain the Internet address of the Sale component or the Internet address of the POI component the Sale to POI message request is addressed.
The format of the `Host` value is:

```
    host [: port]
```

where `host` is the IP or the dns address of the taget and,
The TCP port number `port` might be absent if the value is `80`, default TCP port number of the HTTP protocol.

Rule 4: If the `User-Agent` HTTP request header field is present, it should have the product name and version of the products managing the Sale to POI protocol and the HTTP protocol.

Rule 5: If the `Accept-Encoding` HTTP request header field is present, it must have the value `Identity`. The HTTP server has to ignore other value.

Rule 6: Other HTTP request header fields:

1) The following fields should be absent, they have to be ignored:
```
Accept-Language
Expect
From
If-Match
If-Modified-Since
If-None-Match
If-Range
If-Unmodified-Since
Range
Referer
TE
```
2) If present, the following fields have to be perform according to the HTTP specifications:
```
Authorization
Max-Forwards
Proxy-Authorization
```

### 3.3.8 HTTP Response Header Fields

Rule 1: The `Location` HTTP response header field might be used in an HTTP Response (status `3xx`), to redirect an application request message to a backup component able to perform the request.
The value of the field must be an HTTP “absolute” URI, i.e. in front of the path and the query, the host address and port number if not the default.

An example is:

```
Location: http://test.epasorg.eu:8080/EPASSaleToPOI/3.0?Class=cc&Category=mm&SaleID=ss&POIID=pp
```

Rule 2: If the `Server` HTTP request header field is present, it should have the product name and version of the products managing the Sale to POI protocol and the HTTP protocol.

Rule 3: Other HTTP response header fields:

1) The following fields should be absent, they have to be ignored:
   `Accept-Ranges`
   `Age`
   `ETag`
   `Vary`
   `Retry-After`
2) If present, the following fields have to be perform according to the HTTP specifications:
   `Proxy-Authenticate`
   `WWW-Authenticate`

### 3.3.9 HTTP Entity Header Fields

Rule 1: If the `Allow` entity header field is present in an HTTP request, the value must contain `POST`. If an HTTP request does not use the POST method, the `Allow` entity header must be present in an HTTP response (status `405`), with the value `POST`.

The HTTP protocol is a document oriented protocol which offers message delimitation or data-unit delimiting. It uses the *`Content-Length`* header field to provide the number of bytes of the HTTP request or response body where is located the Sale to POI message.

Rule 2: If the HTTP Request or Reponse contains an application message, the `Content-Length` HTTP entity header field must be present. The value is the number of byte of the application message.
If the HTTP Reponse does not contain any application message, the `Content-Length` HTTP entity header field must be absent or present with the value `0`. In this case the status code of the response stat line must be: `204 No content`.

Rule 3: In any HTTP Request or Reponse containing an application message, the `Content-Type` HTTP entity header field must be present. It must have the value `text/xml ; charset=utf-8`, `application/asn1 ; charset=utf-8` or `application/json ; charset=utf-8` according to the coding of the application message. In case of unappropriate `Content-Type`, the application error handling has to be performed.

Rule 4: Other HTTP entity header fields:

1) If present, the following fields have to be perform according to the HTTP specifications:
   `Content-MD5`
2) The following fields should be absent, they have to be ignored:
   `Content-Encoding`
   `Content-Location` (initial location of the “resource”)
   `Content-Range`
   `Expires`
   `Last-Modified`
   and any other non HTTP 1.1 extension header.

### 3.3.10 Addressing

To establish a TCP connection, HTTP needs a TCP protocol transport address. This TCP address is composed of:

1. The *IP Address* or the *DNS Name* to resolve of the host on which the application protocol lives.
2. The *TCP Port*, to dispatch the connections to the application, the default TCP port of HTTP is `80`.

## 3.4 HTTPS Protocol

### 3.4.1 Introduction

HTTPS, or HTTP over SSL/TLS is defined in the RFC 2818 (HTTP over TLS) issued in May 2000.

HTTPS protocol is defined as the HTTP protocol, end to end protected by the SSL or TLS protocol, the application security protocol over TCP, as presented in the figure below.

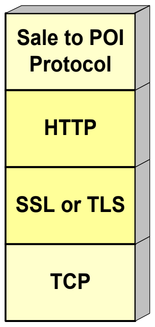

**Figure 32: HTTPS Protocol**

In the following of the chapter, we will use the name TLS instead of SSL/TLS. SSL is now unrecommanded and use of SSL and early TLS (TLS 1.0) should be avoid.

The protocol HTTPS provides the same interface than the protocol HTTP, however, the usage of TLS which has been designed to be managed by the application requires some additional process.

The section 2 presents the characteristics of the application dialogues over HTTPS, which are quite similar to the application dialogues over HTTP, replacing TCP by TLS.

The section 3 lists the constraints of TLS on HTTP.

The section 4 presents the transport and security services which are offered by TLS. It provides rules and recommendations for the usage of these services.

The last section presents the X.509 certificate requirements, their validation and the PKI management.

### 3.4.2 Application Dialogues

As presented in the previous chapter (section *3.3.1*), an HTTP client initiates connections and sends HTTP request messages.

As presented in the following section (*3.4.4 TLS Services*), a TLS client initiates connections, initiates and process the TLS handshakes[7], potentially reusing an existing TLS session.

After a successful handshake, the data flow is not qualified for TLS, and the application protocol may send any type of message or data, using whatever type of dialogue.

The standard HTTPS sequence flow is presented in the figure below:

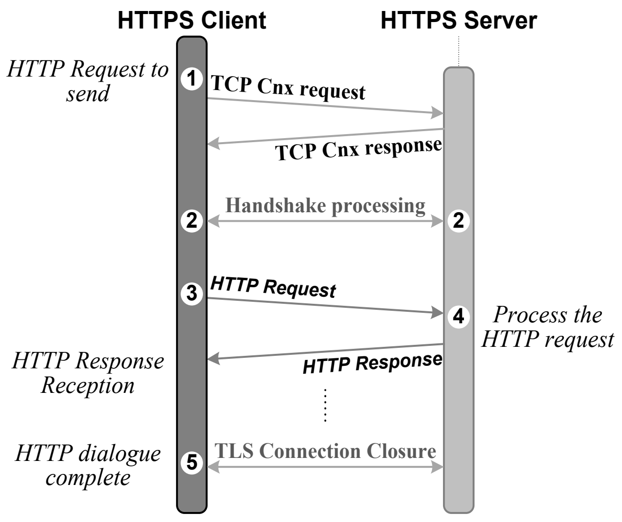

**Figure 33: Standard HTTPS Sequence Flow**

1. The application protocol requests to send an *HTTP request* message. Then the HTTPS Client open a TCP connection with the HTTPS Server,
2. The HTTPS Client initiates and processes with the HTTPS server a TLS handshake, potentially reusing an existing TLS session,
3. The HTTPS Client sends the *HTTP Request* message in a TLS data record,
4. The HTTPS Server process the *HTTP request* of the client, and sends the *HTTP Response* message containing the result of the processing in a TLS data record. If the connection is persistent, the HTTPS Client might send other *HTTP Request* messages,
5. The HTTPS client closes the connection, resulting in a TLS Close Notify alert and a TCP disconnection.

As the HTTPS client initiates a TLS connection, sends HTTP request, and accepts only HTTP responses, the TLS session cannot be used by the HTTPS server to send HTTP requests to the HTTP client.

Rule 1: The HTTPS client must be the TLS client. The TLS session established by an HTTPS client has to be used by the HTTPS server to answer to HTTP requests.

[7] TLS handshake negociates the security algorithms, exchanges the shared secret between the TLS client and the TLS server.

As a consequence, using HTTPS transport services, the Sale to POI protocol dialogue must follow the dialogue defined by the HTTP transport services (section *3.3.3 Application Dialogues*).

## Service Dialogue

The general process flow of the service dialogue follows the *Figure 21: HTTP Service Dialogue*, replacing TCP by TLS:

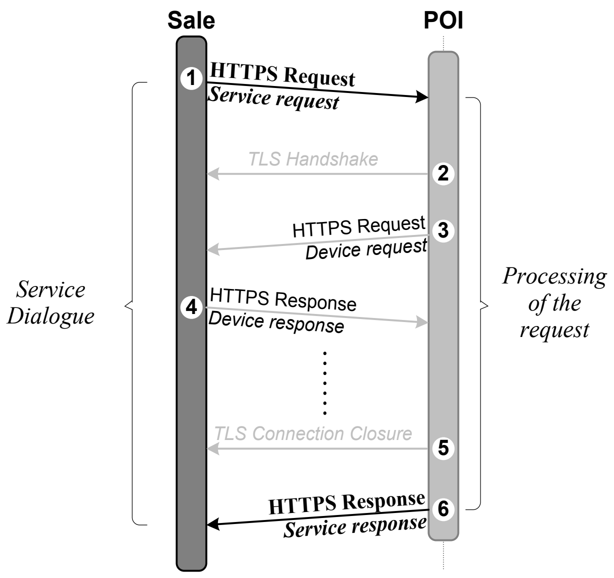

**Figure 34: HTTPS Service Dialogue**

If there are Device requests to send to the Sale during the performing of the service request (steps 2 to 5), the POI must establish a TLS session to send the Device request message to the Sale which plays then the role of a TSL server.

Conforming to the *Figure 22: HTTP Service with Device Requests in Parallel*, when several Device request are sent in parallel:

- The same TLS session might be used, as HTTP allows on a persistent connection sending a request before having received the response of the previous request. The constraint is that the (HTTP) responses are sent in the order of the requests. So if the POI starts a Print then a Display, the Display response is received after the Print response.
- To avoid this constraint, several TLS sessions might be open by the POI, and the Display response is independent from the Print response received in another TLS session.

For Service or Device request that not require response message (*see Figure 23: HTTP Service and Device Request without Response*), the Device request is sent in the body of an HTTP request, a related HTTP response is sent without body and the status code 204 (No Content).

### Error Resolution Dialogue

Error resolution dialogues are conforming to the HTTP error resolution dialogues specified in *Error Resolution Dialogue* at page 33.

### Device Dialogue

Device dialogues are conforming to the HTTP device dialogues specified in *Device Dialogue* at page 35.

### Notification Dialogue

Notification dialogues are conforming to the HTTP notification dialogues specified in *Notification Dialogue* at page 35.

## 3.4.3 HTTPS Constraints

HTTPS is conforming to the start line rule defined in the section *3.3.4 Request Start Line*.

HTTPS is conforming to the error handling rules and to the response start line rules defined in the section *3.3.5 Error Handling and Response Start Line*.

HTTPS is conforming to the general header fields rules defined in the section *3.3.6 HTTP General Header Fields*.

HTTPS is conforming to the request header fields rules defined in the section *3.3.7 HTTP Request Header Fields*.

HTTPS is conforming to the response header fields rules defined in the section *3.3.8 HTTP Response Header Fields*.

HTTPS is conforming to the entity header fields rules defined in the section *3.3.9 HTTP Entity Header Fields*.

To establish a TCP connection, HTTPS needs a TCP protocol transport address. This TCP address is composed of:

1. The *IP Address* or the *DNS Name* to resolve of the host on which the application protocol lives.
2. The *TCP Port*, to dispatch the connections to the application, the default TCP port of HTTPS is `443`.

.

### 3.4.4 TLS Services

The SSL (Secure Socket Layer) protocol has been specified by Netscape, the last version being SSL v3.0.

The TLS (Transport Layer Security) protocol has been specified by IETF:

- RFC 2246 - The TLS Protocol version 1.0 – January 1999
- RFC 4346 - The Transport Layer Security (TLS) Protocol version 1.1 – April 2006
- RFC 5246 - The Transport Layer Security (TLS) Protocol version 1.2 – August 2008

The SSL version 2.0 has known deficiencies and does not provide a sufficient level of security, the RFC 6176 describes these deficiencies.

Rule 1: When TLS clients and servers establish a connection, they must never negociate the use of SSL version 2.0. In particular the version 2.0 of the handshake initial message (Client Hello) must never be sent by a TLS Client.

TLS is designed to provide a secure end-to-end service with TCP. TLS is composed of two layers of protocols, as presented in the figure below.

1. The top level with the following protocols:
   a. The *TLS Handshake Protocol* which allows the negotiation of cryptographic algorithms and keys, and client and server authentication,
   b. The *TLS Change Cipher Spec Protocol* processing the update of cryptographic algorithms and keys, just negotiated,
   c. The *TLS Alert Protocol* which convey warnings or fatal errors.
2. The *TLS Record Protocol* which provides the format of the message, including application data, on which the security services are performed.

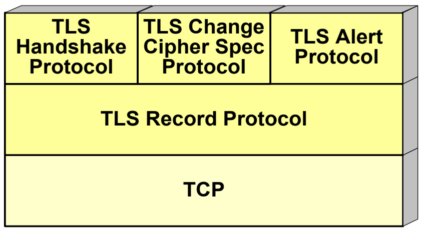

**Figure 35: TLS Protocol Layers**

The *TLS Record Protocol* performs the operations of fragmentation, compression, MACing and encryption

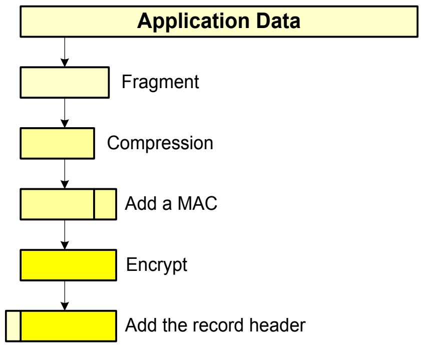

**Figure 36: TLS Record Protocol**

## TLS Session

When a TLS client and server first start communicating, they agree on a protocol version, select cryptographic algorithms, optionally authenticate each other, and use public-key encryption techniques to generate shared secrets.

These processes are performed in the handshake protocol and the result is the establishment of a *TLS session*.

1. After connecting to the TLS Server, the TLS Client sends a *ClientHello* message containing the sequence of cryptographic algorithms it can manage.
2. Then the TLS Server sends a *ServerHello* message containing the selected algorithm.
   It sends a *Certificate* message, either for key exchange, or to sign a public key for encryption with the *ServerKeyExchange* message.
   It sends a *CertificateRequest* message to request a client authentication with a certificate.
   The *ServerHelloDone* message terminates the Server initiatialisation phase.
3. To be authenticated, the TLS Client sends the *Certificate* message containing its authentication certificate, and the *CertificateVerify* message containing the signature of a Server challenge.
   The Client sends the *ClientKeyExchange* message containing the pre-master secret, encrypted by the server public key, to generate the keys of the session.
   Then the Client sends the *ChangeCipherSpec* message to switch to the new set of keys and cryptographic algoritms, and the *Finished* message which is the 1st message protected by these news keys, and which terminates the handshake for the Client.
4. In turn, the TLS Server sends the *ChangeCipherSpec* message, the *Finished* message, and which terminates the handshake.
5. Application data may then be exchanged between the TLS Client and the TLS Server without any constraint on the type of dialogue.

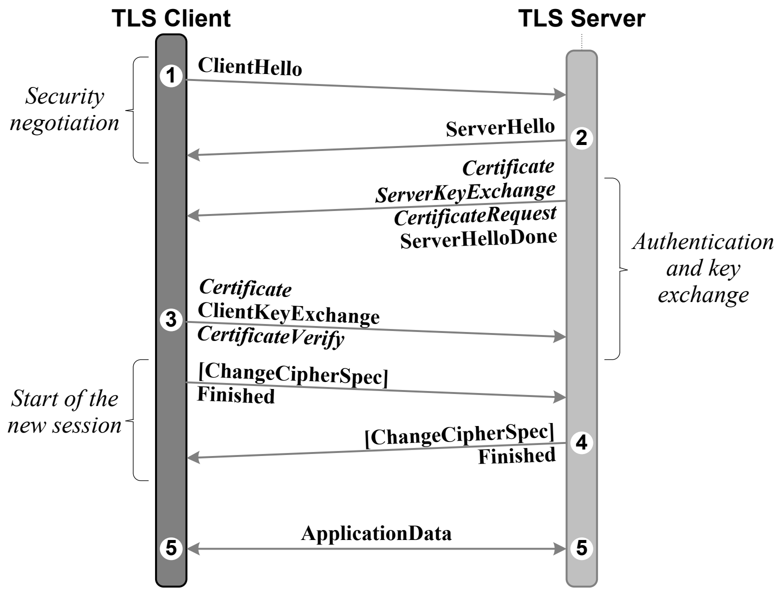

**Figure 37: New TLS Session (Full Handshake)**

Rule 2: It is recommended to use the same TLS Server certificate for authentication and encryption of the pre-master secret.

In the *ServerKeyExchange* message, the public encryption key cannot be presented as a certificate, and is protected by the signature of the public key of the authentication certificate.

When the Client and Server decide to resume a previous session or duplicate an existing session (instead of negotiating new security parameters), the handshake process is simplified.

1. After connecting to the TLS Server, the TLS Client sends a *ClientHello* message containing the session identification he wants to resume.
2. Then the TLS Server sends a *ServerHello* message containing the selected algorithm. It sends the *ChangeCipherSpec* message to switch to the new set of keys and cryptographic algoritms, and it sends the *Finished* message which terminates the handshake resume.
3. Then the Client sends the *ChangeCipherSpec* message and the *Finished* message which terminates the handshake resume.
4. Application data may then be exchanged between the TLS Client and the TLS Server without any constraint on the type of dialogue.

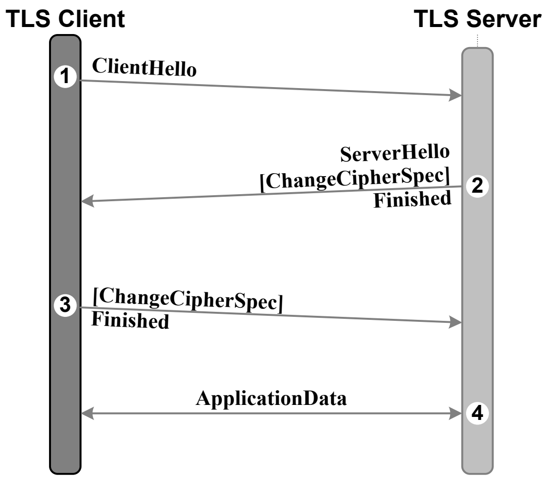

**Figure 38: Resume Previous TLS Session**

Rule 3: TLS Server shall manage for each session the time limit of validity of the session, typically 8 hours.
After this time limit, the TLS Server must refuse any resuming of the session, and send a *HelloRequest* message if the session is active to force the TLS Client sending a *Hello* message.

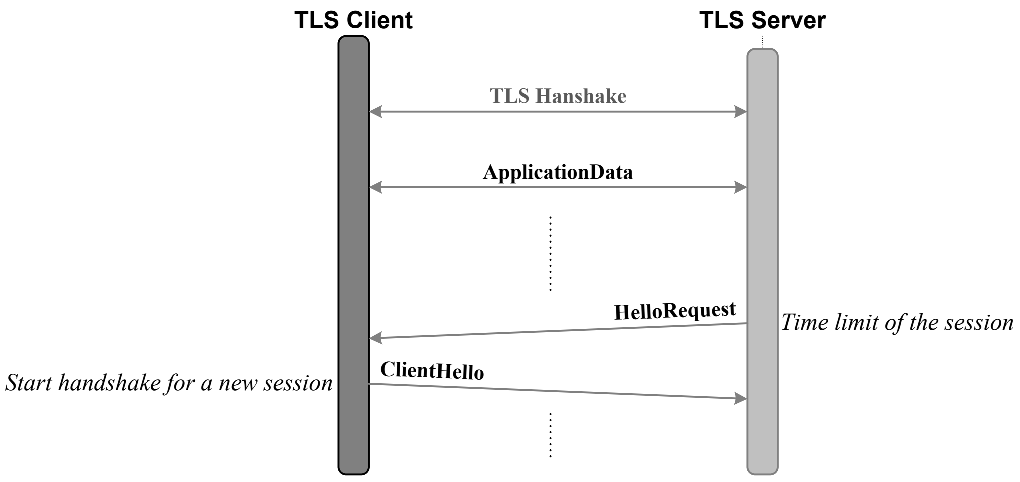

**Figure 39: Force the End of a TLS Session**

## Client and Server Authentication

Rule 4: TLS Client and TLS Server must implement TLS Server authentication. TLS Client authentication is optional, depending on the context.

In the section *3.4.2*, when the POI implementation sends Device request messages or Event messages or initiates connection, the POI must play the role of a TLS Client. As TLS Server authentication is mandatory, the Sale System must manage security for its own authentication.

Rule 5: When POI Device request messages or Event messages is accepted by the Sale system, mutual authentication is required by TLS Client and Server.

## Cryptographic algorithms

Rule 6: The recommended type of key exchange is RSA.

It requires a TLS Server certificate for authentication, and the encryption of a pre-master-secret by the key of the certificate or a temporary RSA key to generate the keys.

Rule 7: The minimum recommended of cryptographic algorithms are Triple DES with a triple key length, CBC encryption mode, and CBC with SHA1 MAC algorithm.

### 3.4.5 TLS Certificates

The X.509 certificates management has been specified by IETF in April 2002 by the RFC 3280 "Internet X.509 Public Key Infrastructure Certificate (PKI) and Certificate Revocation List (CRL) Profile". This specification is often referred as PKIX (Internet PKI).

An X.509 certificate is coded in ASN.1 and is composed of:

- The certificate's data in the structure *CertificateTBS* (certificate to be signed),
- The identification of the signature algorithm,
- The signature of *CertificateTBS*, with a specific format, different from PKCS#7.

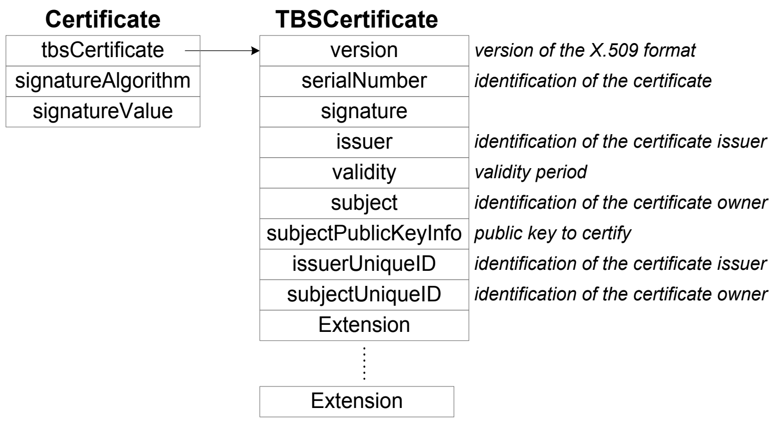

**Figure 40: X.509 Certificate Format**

Rule 1: X.509 version 3 of the certificate is mandatory.

## Certificate Path Validation

The certificate path is the sequence of certificate starting with the certificate of the root Certificate Authority and ending with the certificate of the public key to validate. Each certificate of the path signs the following certificate, except the first one (root certificate) which is auto-signed.

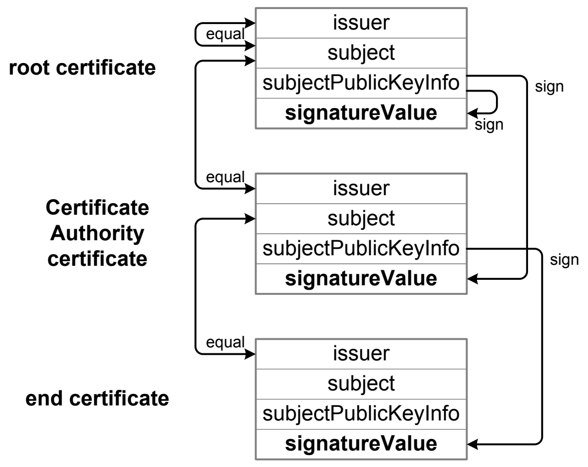

**Figure 41: Certificate Path**

The PKI (Public Key Infrastructure) of the Sale system and the POI system could be:

- Independent, where the root of the Sale certificate and the root of the POI certificate are not the same.
- Shared, where the root of the Sale certificate and the root of the POI certificate are the same, or
- With cross-certificates (network topology), a CA certificate of the Sale PKI is signed by a CA of the POI, and the opposite.

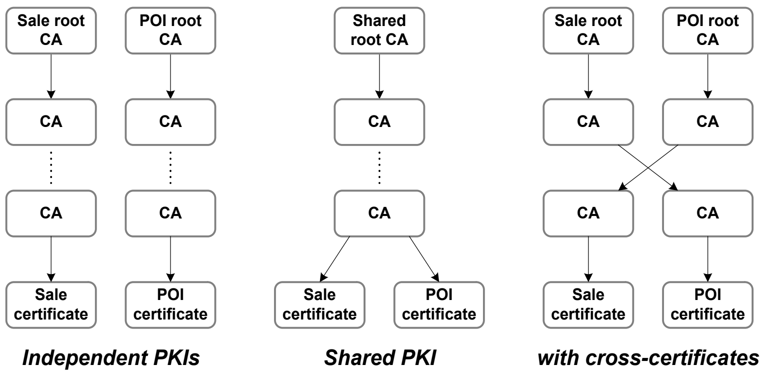

**Figure 42: PKI Topologies**

Rule 2: No topology of the Sale and PKI is recommended, independent, shared, or with cross-certificates.

The complete certificate path validation has to be verified by the TLS Client and the TLS Server. In other words, the validator of a certificate path needs to have stored securely the root of the certificate path.

Rule 3: The validator of a certificate path must store securely the root of the certificate path.

### Certificate Information

Information contained in the extensions of the certificate contains information and property of the certificate.

Rule 4: The public key usage must appear in one of the following certificates extensions:
The Key Usage certificate extension (`id-ce-keyUsage ::= {id-ce 15}`) with the digitalSignature or keyEncipherment value.
The Netscape certificate type extension (`netscape-cert-type ::= { netscape-cert-extension 1 }`) with theTLS client or TLS server value.

Rule 5: The public key owner authentication must be performed, for instance with the Subject Alternative Name certificate extension (`id-ce-subjectAltName ::= {id-ce 17}`) with the dNSName or iPAddress value.

### Certificate Revocation List (CRL)

RSA key length and certificates validity period must be consistent with the security rules of the POI system.

Rule 6: Management of CRL or real-time validation of certificates is optional.

## 3.5 Sale to POI Protocol over TLS

### 3.5.1 Introduction

This section specifies the interface between the Sale to POI and the SSL/TLS security protocol, i.e. when no HTTP protocol is used above SSL/TLS.

For security reason, the Sale to POI protocol between a Sale Server and a POI Server could use SSL/TLS without the HTTP protocol for simplification of the application dialogues.

SSL/TLS is an application protocol between the Sale to POI protocol and the TCP transport protocol. This protocol was presented in a previous section (3.4.4 *TLS Services*).

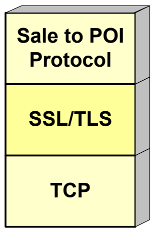

**Figure 43: Sale to POI Protocol over SSL/TLS**

As in the previous section, we will use in this section the name TLS instead of SSL/TLS.

### 3.5.2 Transport Service Constraints

#### Message Delimitation

The TLS protocol, as TCP, does not provide any message delimitation or data-unit delimiting, and the Sale to POI protocol uses the message delimitation as specified in 3.2.2 *Message Delimitation*.

#### TLS Session

To allow a maximum of flexibility, the Sale System has the entire responsibility to manage the TLS session at its convenience, and during the time it considers necessary to its own management.

The Sale System might decide to use permanent transport connections, to open a TLS session for each message, or to mix any type of management according to the requirements of the sales.

<u>TLS Session Creation or Resume</u>

Rule 1: The Sale System can initiate or resume a TLS session for requesting an application message exchange with the POI. If successful, the Sale System is then responsible for initiating the application exchange.

Rule 2: The POI System can initiate or resume a TLS session for requesting an application message exchange with the Sale System. If successful, the Sale System is then responsible for initiating the application exchange. If connection fails, POI system must retry indefinitely to connect to Sale System with an adaptative period between connection attemps.

<u>TLS Session Release</u>

The Sale System may release a TLS session whenever the application judges it appropriate. The Sale System has to estimate the need for receiving the last response message before to close the TLS session.

Rule 3: The Sale System may close a TLS session at any time.
The POI System may close the TLS session only in case of error.

When the TLS session is closed, the POI System cannot send unsolicited notification to the Sale System.

#### Addressing

To establish TLS sessions, TLS needs a TCP protocol transport address. This TCP address is composed of:

1. The *IP Address* or the *DNS Name* to resolve of the host on which the application protocol lives.
2. The *TCP Port*, to dispatch the connections to the application.

### 3.5.3 Application Dialogues

The standard sequence flow is presented in the figure below:

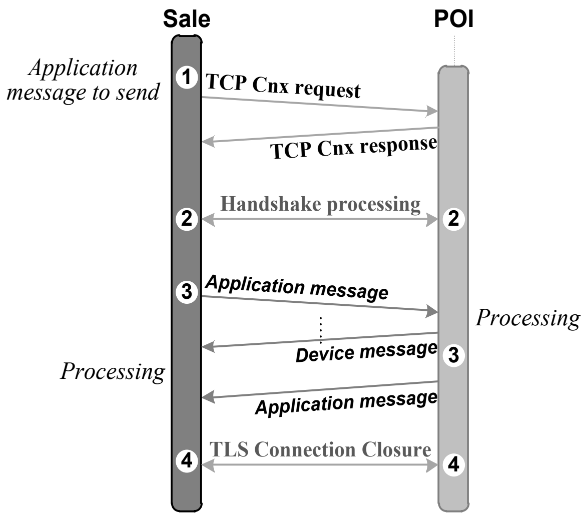

**Figure 44: Standard Sequence Flow**

1. The Sale System requests to send an application message, and open a TCP connection with the POI System,
2. The Sale System as a TLS Client initiates and processes a TLS handshake, potentially reusing an existing TLS session,
3. The Sale System and the POI System exchanges messages of the Sale to POI protocol on this TLS session, Any sequence of request, devices command or admin message may be exchance.
4. The Sale System or the POI System closes the connection, resulting in a TLS Close Notify alert and a TCP disconnection.

All the application dialogues use a TLS session initiated or resumed by the Sale System as TLS Client.

### 3.5.4 TLS Services and Certificates

The Sale to POI protocol over TLS uses the TLS services as specified in the section 3.4.4 *TLS Services*, except the Rule 5 which must be removed.

The Sale to POI protocol over TLS uses the TLS certificates as specified in the section 3.4.5 *TLS Certificates*.

# 4 Transported Message Examples

We consider the following Logout Request message:

| | | |
|---|---|---|
| **SaleToPOIRequest** | | |
| **MessageHeader** | | |
| | MessageClass | Service |
| | MessageCategory | Logout |
| | MessageType | Request |
| | ServiceID | 613 |
| | SaleID | SaleTermA |
| | POIID | POITerm1 |
| **LogoutRequest** | | |

## 4.1 XML and JSON Coding of the Message

### XML Coding

For the message presented above, the XML coding is:

```
<?xml version="1.0" encoding="UTF-8"?>
<SaleToPOIRequest xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="EpasSaleToPOIMessages.xsd">
    <MessageHeader MessageClass="Service" MessageCategory="Logout"
     MessageType="Request" ServiceID="613" SaleID="SaleTermA" POIID="POITerm1"/>
    <LogoutRequest/>
</SaleToPOIRequest>
```

The XML canonical form is presented below, all the useless spaces, tabulations and ends of line are stripped, the length of the message is 341 bytes, or `0155` in hexadecimal.

```
0000  3C 3F 78 6D 6C 20 76 65 72 73 69 6F 6E 3D 22 31  |<?xml version="1|
0010  2E 30 22 20 65 6E 63 6F 64 69 6E 67 3D 22 55 54  |.0" encoding="UT|
0020  46 2D 38 22 3F 3E 3C 53 61 6C 65 54 6F 50 4F 49  |F-8"?><SaleToPOI|
0030  52 65 71 75 65 73 74 20 78 6D 6C 6E 73 3A 78 73  |Request xmlns:xs|
0040  69 3D 22 68 74 74 70 3A 2F 2F 77 77 77 2E 77 33  |i="http://www.w3|
0050  2E 6F 72 67 2F 32 30 30 31 2F 58 4D 4C 53 63 68  |.org/2001/XMLSch|
0060  65 6D 61 2D 69 6E 73 74 61 6E 63 65 22 20 78 73  |ema-instance" xs|
0070  69 3A 6E 6F 4E 61 6D 65 73 70 61 63 65 53 63 68  |i:noNamespaceSch|
0080  65 6D 61 4C 6F 63 61 74 69 6F 6E 3D 22 45 70 61  |emaLocation="Epa|
0090  73 53 61 6C 65 54 6F 50 4F 49 4D 65 73 73 61 67  |sSaleToPOIMessag|
00A0  65 73 2E 78 73 64 22 3E 3C 4D 65 73 73 61 67 65  |es.xsd"><Message|
00B0  48 65 61 64 65 72 20 4D 65 73 73 61 67 65 43 6C  |Header MessageCl|
00C0  61 73 73 3D 22 53 65 72 76 69 63 65 22 20 4D 65  |ass="Service" Me|
00D0  73 73 61 67 65 43 61 74 65 67 6F 72 79 3D 22 4C  |ssageCategory="L|
00E0  6F 67 6F 75 74 22 20 4D 65 73 73 61 67 65 54 79  |ogout" MessageTy|
00F0  70 65 3D 22 52 65 71 75 65 73 74 22 20 53 65 72  |pe="Request" Ser|
0100  76 69 63 65 49 44 3D 22 36 31 33 22 20 53 61 6C  |viceID="613" Sal|
0110  65 49 44 3D 22 53 61 6C 65 54 65 72 6D 41 22 20  |eID="SaleTermA" |
0120  50 4F 49 49 44 3D 22 50 4F 49 54 65 72 6D 31 22  |POIID="POITerm1"|
0130  2F 3E 3C 4C 6F 67 6F 75 74 52 65 71 75 65 73 74  |/><LogoutRequest|
0140  2F 3E 3C 2F 53 61 6C 65 54 6F 50 4F 49 52 65 71  |/></SaleToPOIReq|
0150  75 65 73 74 3E                                   |uest>           |
```

**JSON Coding**

For the message presented above, the JSON coding is:

```
{
   "SaleToPOIRequest":{
      "MessageHeader":{
         "ProtocolVersion":"2.1",
         "MessageClass":"Service",
         "MessageCategory":"Login",
         "MessageType":"Request",
         "ServiceID":"498",
         "SaleID":"SaleTermA",
         "POIID":"POITerm1"
      },
      "LogoutRequest":{
      }
   }
}
```

The JSON canonical form is presented below, all the useless spaces, tabulations and ends of line are stripped, the length of the message is 216 bytes, or `00D8` in hexadecimal.

```
0000  7B 22 53 61 6C 65 54 6F 50 4F 49 52 65 71 75 65   |{"SaleToPOIReque|
0010  73 74 22 3A 7B 22 4D 65 73 73 61 67 65 48 65 61   |st":{"MessageHea|
0020  64 65 72 22 3A 7B 22 50 72 6F 74 6F 63 6F 6C 56   |der":{"ProtocolV|
0030  65 72 73 69 6F 6E 22 3A 22 32 2E 31 22 2C 22 4D   |ersion":"2.1","M|
0040  65 73 73 61 67 65 43 6C 61 73 73 22 3A 22 53 65   |essageClass":"Se|
0050  72 76 69 63 65 22 2C 22 4D 65 73 73 61 67 65 43   |rvice","MessageC|
0060  61 74 65 67 6F 72 79 22 3A 22 4C 6F 67 69 6E 22   |ategory":"Login"|
0070  2C 22 4D 65 73 73 61 67 65 54 79 70 65 22 3A 22   |,"MessageType":"|
0080  52 65 71 75 65 73 74 22 2C 22 53 65 72 76 69 63   |Request","Servic|
0090  65 49 44 22 3A 22 34 39 38 22 2C 22 53 61 6C 65   |eID":"498","Sale|
00A0  49 44 22 3A 22 53 61 6C 65 54 65 72 6D 41 22 2C   |ID":"SaleTermA",|
00B0  22 50 4F 49 49 44 22 3A 22 50 4F 49 54 65 72 6D   |"POIID":"POITerm|
00C0  31 22 7D 2C 22 4C 6F 67 6F 75 74 52 65 71 75 65   |1"},"LogoutReque|
00D0  73 74 22 3A 7B 7D 7D 7D                           |st":{}}}        |
```

## 4.2 TCP Transport of the Message

### XML Coding

The prefix to add for the message length is:

```
        00 00 01 55
```

So the message sent by the TCP transport protocol start with this prefix length, a dump of the message sent by the transport protocol is presented below.

```
0000  00 00 01 55 3C 3F 78 6D 6C 20 76 65 72 73 69 6F  |...U<?xml versio|
0010  6E 3D 22 31 2E 30 22 20 65 6E 63 6F 64 69 6E 67  |n="1.0" encoding|
0020  3D 22 55 54 46 2D 38 22 3F 3E 3C 53 61 6C 65 54  |="UTF-8"?><SaleT|
0030  6F 50 4F 49 52 65 71 75 65 73 74 20 78 6D 6C 6E  |oPOIRequest xmln|
0040  73 3A 78 73 69 3D 22 68 74 74 70 3A 2F 2F 77 77  |s:xsi="http://ww|
0050  77 2E 77 33 2E 6F 72 67 2F 32 30 30 31 2F 58 4D  |w.w3.org/2001/XM|
0060  4C 53 63 68 65 6D 61 2D 69 6E 73 74 61 6E 63 65  |LSchema-instance|
0070  22 20 78 73 69 3A 6E 6F 4E 61 6D 65 73 70 61 63  |" xsi:noNamespac|
0080  65 53 63 68 65 6D 61 4C 6F 63 61 74 69 6F 6E 3D  |eSchemaLocation=|
0090  22 45 70 61 73 53 61 6C 65 54 6F 50 4F 49 4D 65  |"EpasSaleToPOIMe|
00A0  73 73 61 67 65 73 2E 78 73 64 22 3E 3C 4D 65 73  |ssages.xsd"><Mes|
00B0  73 61 67 65 48 65 61 64 65 72 20 4D 65 73 73 61  |sageHeader Messa|
00C0  67 65 43 6C 61 73 73 3D 22 53 65 72 76 69 63 65  |geClass="Service|
00D0  22 20 4D 65 73 73 61 67 65 43 61 74 65 67 6F 72  |" MessageCategor|
00E0  79 3D 22 4C 6F 67 6F 75 74 22 20 4D 65 73 73 61  |y="Logout" Messa|
00F0  67 65 54 79 70 65 3D 22 52 65 71 75 65 73 74 22  |geType="Request"|
0100  20 53 65 72 76 69 63 65 49 44 3D 22 36 31 33 22  | ServiceID="613"|
0110  20 53 61 6C 65 49 44 3D 22 53 61 6C 65 54 65 72  | SaleID="SaleTer|
0120  6D 41 22 20 50 4F 49 49 44 3D 22 50 4F 49 54 65  |mA" POIID="POITe|
0130  72 6D 31 22 2F 3E 3C 4C 6F 67 6F 75 74 52 65 71  |rm1"/><LogoutReq|
0140  75 65 73 74 2F 3E 3C 2F 53 61 6C 65 54 6F 50 4F  |uest/></SaleToPO|
0150  49 52 65 71 75 65 73 74 3E                       |IRequest>       |
```

### JSON Coding

The prefix to add for the message length is:

```
        00 00 00 D8
```

So the message sent by the TCP transport protocol start with this prefix length, a dump of the message sent by the transport protocol is presented below.

```
0000  00 00 00 D8 7B 22 53 61 6C 65 54 6F 50 4F 49 52  |....{"SaleToPOIR|
0010  65 71 75 65 73 74 22 3A 7B 22 4D 65 73 73 61 67  |equest":{"Messag|
0020  65 48 65 61 64 65 72 22 3A 7B 22 50 72 6F 74 6F  |eHeader":{"Proto|
0030  63 6F 6C 56 65 72 73 69 6F 6E 22 3A 22 32 2E 30  |colVersion":"2.0|
0040  22 2C 22 4D 65 73 73 61 67 65 43 6C 61 73 73 22  |","MessageClass"|
0050  3A 22 53 65 72 76 69 63 65 22 2C 22 4D 65 73 73  |:"Service","Mess|
0060  61 67 65 43 61 74 65 67 6F 72 79 22 3A 22 4C 6F  |ageCategory":"Lo|
0070  67 69 6E 22 2C 22 4D 65 73 73 61 67 65 54 79 70  |gin","MessageTyp|
0080  65 22 3A 22 52 65 71 75 65 73 74 22 2C 22 53 65  |e":"Request","Se|
0090  72 76 69 63 65 49 44 22 3A 22 34 39 38 22 2C 22  |rviceID":"498","|
00A0  53 61 6C 65 49 44 22 3A 22 53 61 6C 65 54 65 72  |SaleID":"SaleTer|
00B0  6D 41 22 2C 22 50 4F 49 49 44 22 3A 22 50 4F 49  |mA","POIID":"POI|
00C0  54 65 72 6D 31 22 7D 2C 22 4C 6F 67 6F 75 74 52  |Term1"},"LogoutR|
00D0  65 71 75 65 73 74 22 3A 7B 7D 7D 7D              |equest":{}}}    |
```

### 4.2.1 HTTP Transport of the Message

This section presents the XML and JSON Logout request examples of HTTP message.

**XML Coding**

The prefix to add for the message length is:

```
POST /EPASSaleToPOI/3.0/LogoutRequest?
 Class=Service&Category=Logout&SaleID=SaleTermA&POIID=POITerm1 HTTP/1.1
Host: test.epasorg.eu:8080
User-Agent: EPASTest/2.1 (Windows NT 6.1)
Accept: text/xml
Accept-charset: utf-8
Content-type: text/xml; charset=utf-8
Content-length: 341

<?xml version="1.0" encoding="UTF-8"?>
<SaleToPOIRequest xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="EpasSaleToPOIMessages.xsd">
    <MessageHeader MessageClass="Service" MessageCategory="Logout"
     MessageType="Request" ServiceID="613" SaleID="SaleTermA" POIID="POITerm1"/>
    <LogoutRequest/>
</SaleToPOIRequest>
```

So the message sent by the TCP transport protocol start with this prefix length, a dump of the message sent by the transport protocol is presented below.

```
0000   50 4F 53 54 20 2F 45 50 41 53 53 61 6C 65 54 6F   |POST /EPASSaleTo|
0010   50 4F 49 2F 32 2E 31 2F 4C 6F 67 6F 75 74 52 65   |POI/2.1/LogoutRe|
0020   71 75 65 73 74 3F 20 43 6C 61 73 73 3D 53 65 72   |quest? Class=Ser|
0030   76 69 63 65 26 43 61 74 65 67 6F 72 79 3D 4C 6F   |vice&Category=Lo|
0040   67 6F 75 74 26 53 61 6C 65 49 44 3D 53 61 6C 65   |gout&SaleID=Sale|
0050   54 65 72 6D 41 26 50 4F 49 49 44 3D 50 4F 49 54   |TermA&POIID=POIT|
0060   65 72 6D 31 20 48 54 54 50 2F 31 2E 31 0D 0A 48   |erm1 HTTP/1.1..H|
0070   6F 73 74 3A 20 74 65 73 74 2E 65 70 61 73 6F 72   |ost: test.epasor|
0080   67 2E 65 75 3A 38 30 38 30 0D 0A 55 73 65 72 2D   |g.eu:8080..User-|
0090   41 67 65 6E 74 3A 20 45 50 41 53 54 65 73 74 2F   |Agent: EPASTest/|
00A0   32 2E 31 20 28 57 69 6E 64 6F 77 73 20 4E 54 20   |2.1 (Windows NT |
00B0   36 2E 31 29 0D 0A 41 63 63 65 70 74 3A 20 74 65   |6.1)..Accept: te|
00C0   78 74 2F 78 6D 6C 0D 0A 41 63 63 65 70 74 2D 63   |xt/xml..Accept-c|
00D0   68 61 72 73 65 74 3A 20 75 74 66 2D 38 0D 0A 43   |harset: utf-8..C|
00E0   6F 6E 74 65 6E 74 2D 74 79 70 65 3A 20 74 65 78   |ontent-type: tex|
00F0   74 2F 78 6D 6C 3B 20 63 68 61 72 73 65 74 3D 75   |t/xml; charset=u|
0100   74 66 2D 38 0D 0A 43 6F 6E 74 65 6E 74 2D 6C 65   |tf-8..Content-le|
0110   6E 67 74 68 3A 20 33 34 31 0D 0A 0D 0A 3C 3F 78   |ngth: 341....<?x|
0120   6D 6C 20 76 65 72 73 69 6F 6E 3D 22 31 2E 30 22   |ml version="1.0"|
0130   20 65 6E 63 6F 64 69 6E 67 3D 22 55 54 46 2D 38   | encoding="UTF-8|
0140   22 3F 3E 3C 53 61 6C 65 54 6F 50 4F 49 52 65 71   |"?><SaleToPOIReq|
0150   75 65 73 74 20 78 6D 6C 6E 73 3A 78 73 69 3D 22   |uest xmlns:xsi="|
0160   68 74 74 70 3A 2F 2F 77 77 77 2E 77 33 2E 6F 72   |http://www.w3.or|
0170   67 2F 32 30 30 31 2F 58 4D 4C 53 63 68 65 6D 61   |g/2001/XMLSchema|
0180   2D 69 6E 73 74 61 6E 63 65 22 20 78 73 69 3A 6E   |-instance" xsi:n|
0190   6F 4E 61 6D 65 73 70 61 63 65 53 63 68 65 6D 61   |oNamespaceSchema|
01A0   4C 6F 63 61 74 69 6F 6E 3D 22 45 70 61 73 53 61   |Location="EpasSa|
01B0   6C 65 54 6F 50 4F 49 4D 65 73 73 61 67 65 73 2E   |leToPOIMessages.|
01C0   78 73 64 22 3E 3C 4D 65 73 73 61 67 65 48 65 61   |xsd"><MessageHea|
01D0   64 65 72 20 4D 65 73 73 61 67 65 43 6C 61 73 73   |der MessageClass|
01E0   3D 22 53 65 72 76 69 63 65 22 20 4D 65 73 73 61   |="Service" Messa|
01F0   67 65 43 61 74 65 67 6F 72 79 3D 22 4C 6F 67 6F   |geCategory="Logo|
0200   75 74 22 20 4D 65 73 73 61 67 65 54 79 70 65 3D   |ut" MessageType=|
0210   22 52 65 71 75 65 73 74 22 20 53 65 72 76 69 63   |"Request" Servic|
0220   65 49 44 3D 22 36 31 33 22 20 53 61 6C 65 49 44   |eID="613" SaleID|
0230   3D 22 53 61 6C 65 54 65 72 6D 41 22 20 50 4F 49   |="SaleTermA" POI|
0240   49 44 3D 22 50 4F 49 54 65 72 6D 31 22 2F 3E 3C   |ID="POITerm1"/><|
0250   4C 6F 67 6F 75 74 52 65 71 75 65 73 74 2F 3E 3C   |LogoutRequest/><|
0260   2F 53 61 6C 65 54 6F 50 4F 49 52 65 71 75 65 73   |/SaleToPOIReques|
0270   74 3E                                             |t>              |
```

### JSON Coding

The prefix to add for the message length is:

```
POST
  /EPASSaleToPOI/2.1/LogoutRequest?MessageClass=Service&MessageCategory=Logout&SaleI
  D=SaleTermA&POIID=POITerm1 HTTP/1.1
Cache-Control: no-cache
Host: test.epasorg.eu:8080
User-Agent: EPASTest/2.1 (Windows NT 6.1)
Accept: application/json
Content-type: application/json; charset=utf-8
Content-length: 216

{
   "SaleToPOIRequest":{
      "MessageHeader":{
         "ProtocolVersion":"2.1",
         "MessageClass":"Service",
         "MessageCategory":"Logout",
         "MessageType":"Request",
         "ServiceID":"498",
         "SaleID":"SaleTermA",
         "POIID":"POITerm1"
      },
      "LogoutRequest":{
      }
   }
}
```

So the message sent by the TCP transport protocol start with this prefix length, a dump of the message sent by the transport protocol is presented below.

```
0000   50 4F 53 54 20 2F 45 50 41 53 53 61 6C 65 54 6F   |POST /EPASSaleTo|
0010   50 4F 49 2F 32 2E 31 2F 4C 6F 67 6F 75 74 52 65   |POI/2.1/LogoutRe|
0020   71 75 65 73 74 3F 4D 65 73 73 61 67 65 43 6C 61   |quest?MessageCla|
0030   73 73 3D 53 65 72 76 69 63 65 26 4D 65 73 73 61   |ss=Service&Messa|
0040   67 65 43 61 74 65 67 6F 72 79 3D 4C 6F 67 6F 75   |geCategory=Logou|
0050   74 26 53 61 6C 65 49 44 3D 53 61 6C 65 54 65 72   |t&SaleID=SaleTer|
0060   6D 41 26 50 4F 49 49 44 3D 50 4F 49 54 65 72 6D   |mA&POIID=POITerm|
0070   31 20 48 54 54 50 2F 31 2E 31 0D 0A 43 61 63 68   |1 HTTP/1.1..Cach|
0080   65 2D 43 6F 6E 74 72 6F 6C 3A 20 6E 6F 2D 63 61   |e-Control: no-ca|
0090   63 68 65 0D 0A 48 6F 73 74 3A 20 74 65 73 74 2E   |che..Host: test.|
00A0   65 70 61 73 6F 72 67 2E 65 75 3A 38 30 38 30 0D   |epasorg.eu:8080.|
00B0   0A 55 73 65 72 2D 41 67 65 6E 74 3A 20 45 50 41   |.User-Agent: EPA|
00C0   53 54 65 73 74 2F 32 2E 31 20 28 57 69 6E 64 6F   |STest/2.1 (Windo|
00D0   77 73 20 4E 54 20 36 2E 31 29 0D 0A 41 63 63 65   |ws NT 6.1)..Acce|
00E0   70 74 3A 20 61 70 70 6C 69 63 61 74 69 6F 6E 2F   |pt: application/|
00F0   6A 73 6F 6E 0D 0A 43 6F 6E 74 65 6E 74 2D 74 79   |json..Content-ty|
0100   70 65 3A 20 61 70 70 6C 69 63 61 74 69 6F 6E 2F   |pe: application/|
0110   6A 73 6F 6E 3B 20 63 68 61 72 73 65 74 3D 75 74   |json; charset=ut|
0120   66 2D 38 0D 0A 43 6F 6E 74 65 6E 74 2D 6C 65 6E   |f-8..Content-len|
0130   67 74 68 3A 20 32 31 36 0D 0A 0D 0A 7B 22 53 61   |gth: 216....{"Sa|
0140   6C 65 54 6F 50 4F 49 52 65 71 75 65 73 74 22 3A   |leToPOIRequest":|
0150   7B 22 4D 65 73 73 61 67 65 48 65 61 64 65 72 22   |{"MessageHeader"|
0160   3A 7B 22 50 72 6F 74 6F 63 6F 6C 56 65 72 73 69   |:{"ProtocolVersi|
0170   6F 6E 22 3A 22 32 2E 31 22 2C 22 4D 65 73 73 61   |on":"2.1","Messa|
0180   67 65 43 6C 61 73 73 22 3A 22 53 65 72 76 69 63   |geClass":"Servic|
0190   65 22 2C 22 4D 65 73 73 61 67 65 43 61 74 65 67   |e","MessageCateg|
01A0   6F 72 79 22 3A 22 4C 6F 67 69 6E 22 2C 22 4D 65   |ory":"Login","Me|
01B0   73 73 61 67 65 54 79 70 65 22 3A 22 52 65 71 75   |ssageType":"Requ|
01C0   65 73 74 22 2C 22 53 65 72 76 69 63 65 49 44 22   |est","ServiceID"|
01D0   3A 22 34 39 38 22 2C 22 53 61 6C 65 49 44 22 3A   |:"498","SaleID":|
01E0   22 53 61 6C 65 54 65 72 6D 41 22 2C 22 50 4F 49   |"SaleTermA","POI|
01F0   49 44 22 3A 22 50 4F 49 54 65 72 6D 31 22 7D 2C   |ID":"POITerm1"},|
0200   22 4C 6F 67 6F 75 74 52 65 71 75 65 73 74 22 3A   |"LogoutRequest":|
0210   7B 7D 7D 7D                                       |{}}}            |
```