SC

Service Connector

SC Message Protocol V1.0

SC_1_SCMP-V1.0_E (Version V2.3)

This document describes the SC Message Protocol V1.0 (SCMP).

This Document has been created with Microsoft Word 2003 (11) with template file C:\STABILIT\STANDARD\TEMPLATES\S_REP_E.DOT and printed at 26 May 2010 10:00.

Identification

Project:	SC
Title:	Service Connector
Subtitle:	SC Message Protocol V1.0
Version:	V2.3
Reference:	SC_1_SCMP-V1.0_E
Classification:	Confidential
Keywords:	Concepts, Communication, Protocol
Comment:	This document describes the SC Message Protocol V1.0 (SCMP).
Author(s):	STABILIT Informatik AG Jan Trnka, Daniel Schmutz, Joel Traber
Approval (Reviewed by):	Signature .... Jan Trnka Signature .... Francisco Gonzalez
Audience:	Project team, Eurex IT management, Review team
Distribution:	Project team, SIX development team
Filename	c:\stabilit\projects\eurex\sc\documents\sc_0_scmp_e.doc

Revision History

Date	Version	Author	Description
14.06.2009	D1.0	Jan Trnka	Initial draft
09.08.2009	D1.1	Jan Trnka	Separate specification and architecture documents. Fill information from Daniel into this document.
…	…	Jan Trnka	Many changes in between not tracked
24.2.2010	V2.0	Jan Trnka	Proposal for C-API
3.3.2010	V2.1	Jan Trnka	Remove SC architecture from this document
11.4.2010	V2.2	Jan Trnka	Prepare for review
14.4.2010	V2.3	Jan Trnka	Put API into its own document

Table of Contents

1 PREFACE 5
1.1 Purpose & Scope of this Document 5
1.2 Definitions & Abbreviations 5
1.3 External References 5
1.4 Typographical Conventions 5
1.5 Outstanding Issues 6

2 COMMUNICATION SCHEMA 7
2.1 Connection Topology 7
2.2 Network Security 8

3 SERVICE MODEL 10
3.1 Session Services 10
3.2 Publishing Services 11
3.3 File Services 12
3.4 HTTP Redirection Services 12

4 MESSAGE PROTOCOL 13
4.1 Headline 13
4.2 Header 14
4.3 Body 14
4.4 SCMP over TCP/IP 14
4.5 SCMP over HTTP 14
4.6 HPPT over SCMP 15

5 SEMANTIC 16
5.1 Session Service 16
5.1.1 Asynchronous Message Exchange 18
5.1.2 Large Messages 18
5.1.3 Single Threaded Server 20
5.1.4 Multi Threaded Server 20
5.1.5 Multi Connection Server 21
5.1.6 Application Server (Tomcat) 21
5.2 Publishing Service 22
5.3 Web Server (Apache) 23

6 SCMP MESSAGES 24
6.1 CONNECT 24
6.2 DISCONNECT 24
6.3 KEEPALIVE 24
6.4 INSPECT 25
6.5 ECHO_SC 25
6.6 CLN_CREATE_SESSION 25
6.7 SRV_CREATE_SESSION 26
6.8 CLN_DELETE_SESSION 26
6.9 SRV_DELETE_SESSION 27
6.10 SRV_ABORT_SESSION 27
6.11 REGISTER_SERVICE 27
6.12 DEREGISTER_SERVICE 28
6.13 CLN_ECHO 28
6.14 SRV_ECHO 29
6.15 CLN_DATA 29
6.16 SRV_DATA 30
6.17 HTTP 30

6.18 SUBSCRIBE 30
6.19 UNSUBSCRIBE 31
6.20 CHANGE_SUBSCRIPTION 31
6.21 RECEIVE_PUBLICATION 31
6.22 PUBLISH 32

7 SCMP HEADER ATTRIBUTES 34
7.1 appErrorCode 34
7.2 appErrorText 34
7.3 bodyType 34
7.4 bodyLength 34
7.5 cacheExpirationDateTime 35
7.6 cacheId 35
7.7 compression 35
7.8 ipAddressList 35
7.9 portNr 36
7.10 keepAliveInterval 36
7.11 keepAliveTimeout 36
7.12 localDateTime 36
7.13 messageID 37
7.14 mask 37
7.15 maxNodes 38
7.16 maxSessions 38
7.17 msgInfo 38
7.18 msgType 38
7.19 multiThreaded 38
7.20 noData 39
7.21 rejectSession 39
7.22 scErrorCode 39
7.23 scErrorText 39
7.24 scVersion 39
7.25 serviceName 40
7.26 sessionId 40
7.27 sessionInfo 40

8 GLOSSARY 41

APPENDIX A MESSAGE HEADER MATRIX 42

INDEX 43

Tables

Table 1 Abbreviations & Definitions 5
Table 2 External references 5
Table 3 Typographical conventions 6

Figures

Figure 1 Communication Layers 7
Figure 2 Connection Topology 8
Figure 3 Network Security 8
Figure 4 Synchronous Request/Response 10

Figure 5 Asynchronous Request/Response 11
Figure 6 Asynchronous Subscribe / Publish 11
Figure 7 Service Connector Message Protocol 13
Figure 8 Synchronous Message Exchange. 16
Figure 9 Asynchronous Session Service Message Exchange. 18
Figure 10 Large Response. 18
Figure 11 Large Request. 19
Figure 12 Multi threaded server. 20
Figure 13 Multi-Connection server. 21
Figure 14 Publishing Service 22

1 Preface

1.1 Purpose & Scope of this Document

This document describes the SCMP (**SC** **M**essage **P**rotocol).

The final and approved version of this document serves as base for the publication as Open Source.

This document is particularly important to all project team members and serves as communication medium between them.

1.2 Definitions & Abbreviations

Item / Term	Definition / Description
HTTP	Hypertext Transport Protocol
HTTPS	HTTP over SSL, encrypted and authenticated transport protocol
Java	Programming language and run-time environment from SUN
JDK	Java Development Kit
Log4j	Standard logging tool used in Java
OpenVMS	HP Operating system, platform for existing ERM application
RMI	Remote Method Invocation - RPC protocol used in Java
RPC	Remote Procedure Call
SOAP	Simple Object Access Protocol
SSL	Secure Socket Layer – secure communication protocol with encryption and authentication
TCP/IP	Transmission Control Protocol / Internet Protocol
SC	Service Connector
USP	Universal Service Processor – predecessor of SC

Table 1 Abbreviations & Definitions

1.3 External References

References	Item / Reference to other Document
[1]	SC_0_Specification_E – Requirement and Specifications for Service Connector
[2]	

Table 2 External references

1.4 Typographical Conventions

Convention	Meaning
text in italics	features not implemented in the actual release
`text in Courier font`	code example
[phrase]	In syntax diagrams, indicates that the enclosed values are optional
{ phrase1 ǀ phrase2 }	In syntax diagrams, indicates that multiple possibilities exists.
…	In syntax diagrams, indicates a repetition of the previous expression

Table 3 Typographical conventions

The terminology used in this document may be somewhat different from other sources. The chapter Glossary includes a list of often used terms with the explanation of their meaning in this document.

1.5 Outstanding Issues

Following issues are outstanding at the time of the document release:

- Body structure for INSPECT message
- How is HTTP over SCMP implemented?
- chunked transfer encoding for SCMP over HTTP
- pipelining for SCMP over HTTP

2 Communication Schema

The SC implements peer-to-peer messaging above OSI layer-7 (application) network model between client and server applications. The SC is always in between the communicating partners, controlling the entire message flow.

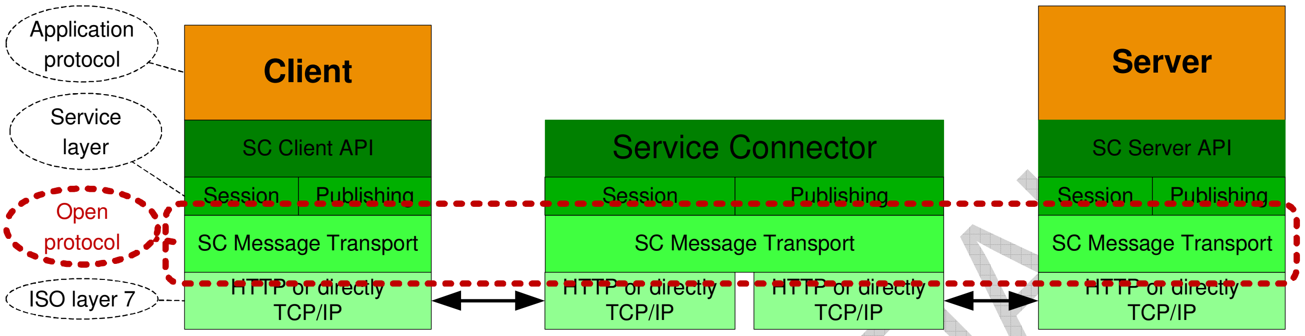


Figure 1 Communication Layers

The SC acts like a broker, passing messages between the client and the server. The communicating parties must agree on the application protocol i.e. format and content of the message payload.

2.1 Connection Topology

Client application, server application and the SC may reside on the same node or on separate nodes connected via TCP/IP network. No assumption about the physical network topology is done. Multiple firewalls can be located on the path between the communicating partners.

The SC supports following connection topology:

- Client ⇔ SC ⇔ Server = Direct connection
- Client ⇔ SC-Proxy ⇔ SC ⇔Server = Connection via SC-Proxy.
  Multiple SC-Proxies may be placed on the path between the client and service.

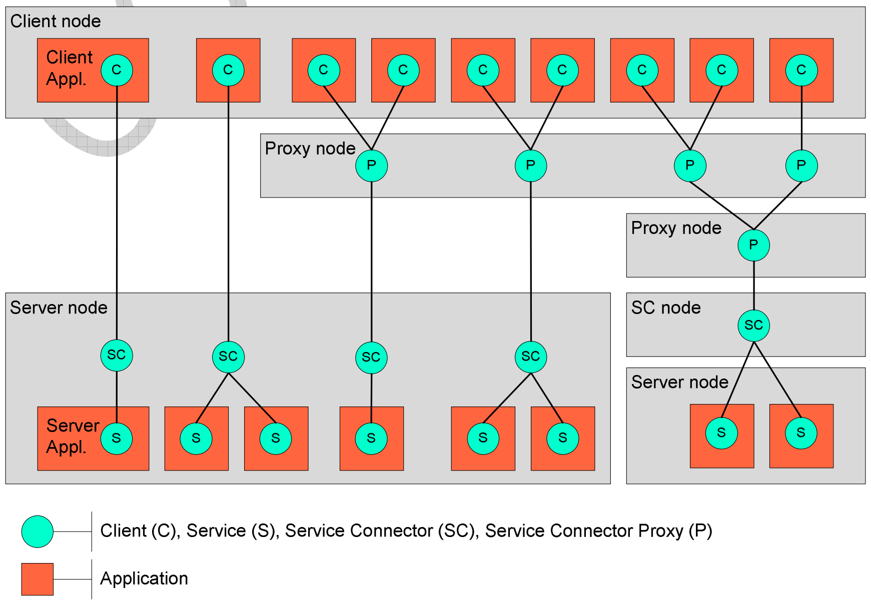


Figure 2 Connection Topology

Different connection topology types from left to right:

1. Client connected to one services
2. Client connected simultaneously to two services
3. Two clients connected to one service via proxy
4. Two clients connected to two services via proxy
5. Three clients connected to two services via cascaded proxy on two separate nodes and SC offloaded to its own node.

The connection can either utilize HTTP protocol or direct TCP/IP communication.

SC Proxy is used for performance and/or for security reasons. It is transparent for the application and plays the role of the server and client at the same time.

2.2 Network Security

From the security viewpoint on the network, the number of clients or server is not relevant. Meaningful are connections between nodes and security measures taken to protect legal subjects to which the particular network segment belongs. The following schema shows some possible networks that may be configured to pass SC messages.

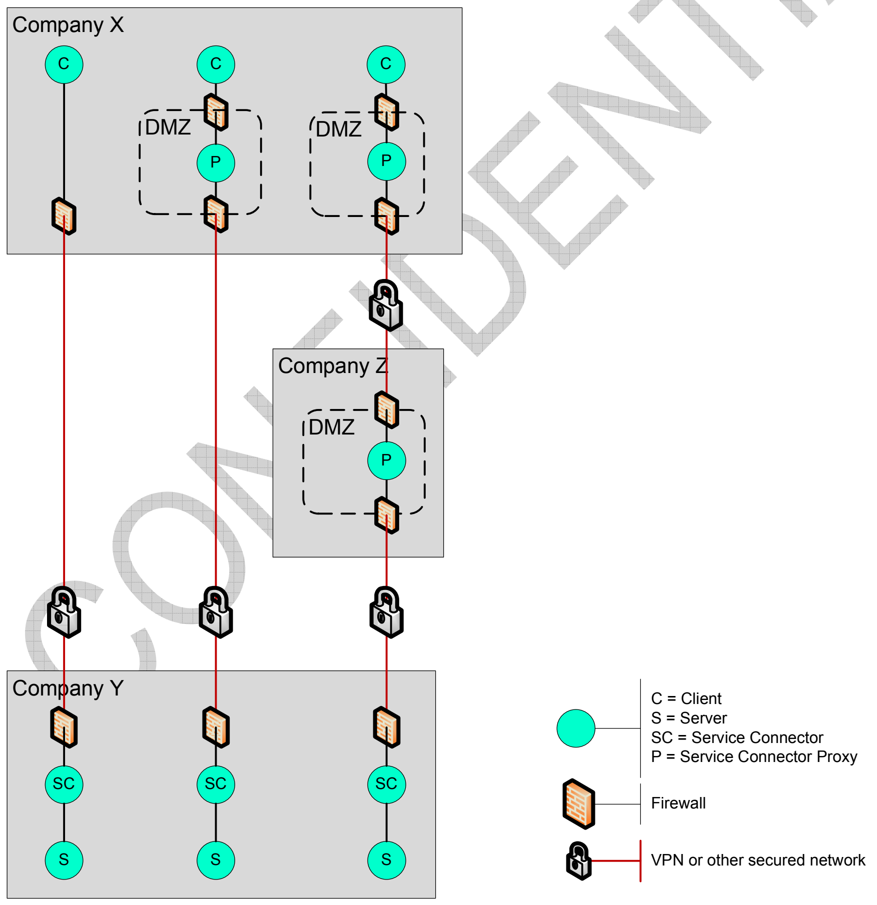


Figure 3 Network Security

Protocol

The message transport between each of the SC components (green bullet) may be configured as plain TCP/IP or HTTP. The connection is <u>always</u> initiated by the client (top-down in the picture above). The client defines which transport (TCP/IP or HTTP) it will use. The transport protocol between Proxy and the next downstream component is configured in the SC configuration. TCP/IP is strongly recommended between SC proxy and the SC though VPN tunnels as well as between SC and the Server for performance reasons. HTTP is recommended between the client and SC proxy for SC. For connection between SC and Web Server or Application Server (Tomcat) only HTTP can be used.

Firewall

Firewall with a proper configuration may be placed on any path between two SC components. Firewall between SC and the server is possible, but not recommended for performance reasons. For HTTP protocol the firewall can be configured to perform statefull packet inspection with HTTP filtering. For TCP/IP the appropriate port must be configured in the firewall.

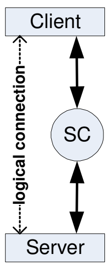
When the message traffic will pass a firewall, HTTP protocol is recommended. In such case the SC can be seen as a regular Web-Server. The firewall can be configured to perform statefull packet inspection with HTTP filtering. For connection between SC and the server and for communications though VPN tunnels direct TCP/IP is recommended for performance reasons.

When HTTP connection is used and multiple parallel requests are in progress, SC will create multiple connections for each pending request in order to keep them balanced and so satisfy firewall inspection rules. (Statefull inspection rejects two subsequent GET/POST request without response from the server)

Keepalive

Each connection between two SC components is supervised by an algorithm using keepalive messages exchanged in regular (configurable) intervals. The sender monitors the arrival of the echoed message within a timeout; the receiver monitors the interval in which keepalive messages should arrive. These two methods allow independent detection of broken connections on both sites in case the communication will be interrupted without notice. E.g. firewall swallows the traffic. This is algorithm is essential for detection of broken sessions. Keepalive messages are very important to preserve the connection state in the firewall and reset their internal timeout. Only traffic to the outside may refresh the internal firewall timeout. That’s why keepalive message is always initiated by the client. Using of keepalive messages can be disabled.

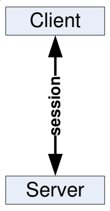
3 Service Model



The SC supports message exchange between requesting application (client) and another application providing a service (server). The client and the server are the logical communication end-points. The SC never acts as a direct executor of a service. The client can to communicate to multiple services at the same time.

Server application can provide one or multiple service. Serving multiple services within one application is possible only for multithreaded or multisession servers. Multiple server applications are running on the same server node, each providing different service. All services are independent on each other. Server application may request another service and so play the client role.

3.1 Session Services



Request/Response (client initiated communication). For session services the client and the server exchange messages in context of a logical session through the SC.

Synchronous

The client sends a request to a service that invokes an application code. Upon completion the service sends back a response message. The client <u>waits</u> for the arrival of the response message. The request and response message length is not limited in size.

The communication occurs in a scope of a logical session. SC will choose a free server and pass this and subsequent requests from this client to the same server. Session information is always passed as a part of the message header. The client may have only one outstanding request per session.

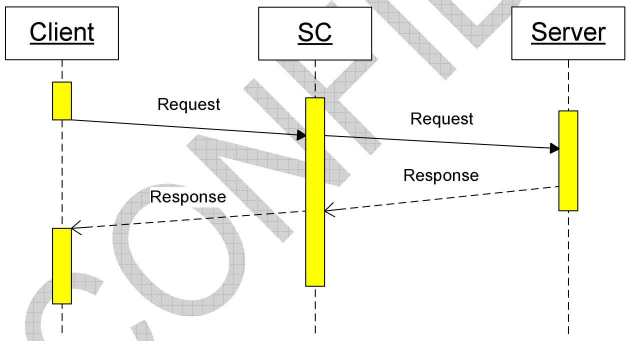


Figure 4 Synchronous Request/Response

Multiple clients may request the same service at the same time. The service execution is in parallel, each one in a separate thread or process. The server decides how many sessions it can serve.

This communication style is the most often used for getting data from the server or sending a message that triggers a transaction on the server.

Asynchronous

Asynchronous execution is functionally equal to the synchronous case with the exception that the client does not wait for the arrival of the response message. The client must declare a notification method that is invoked when the response message arrives. The client may have only one outstanding request per session. When client issues a request before the previous one was completed, the send method blocks until the previous request is satisfied.

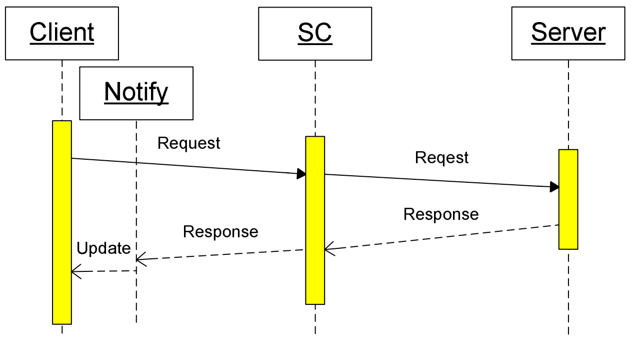
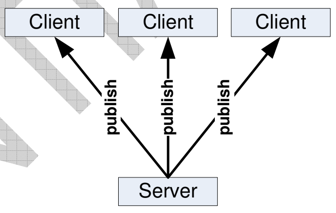


Figure 5 Asynchronous Request/Response

This communication style will be used to load data while other activities are in progress, e.g. to get large amount of static data at startup. It can be also used as fire-and-forget when the response is not meaningful.

3.2 Publishing Services



Subscribe/Publish (server initiated communication). Publishing services allows the server to send single message to many clients through the SC.

The client sends a subscription mask to the service, and so declares its interest on certain type of a message. The application service providing the message contents must designate the message with a type. When the message type matches the client subscription mask, the message will be sent to the client. Multiple clients may subscribe for the same service at the same time. In such case multiple clients can get the same copy of the message. Message that does not match any client subscription is discarded.

The client must declare a notification method that is invoked when the message arrives. The client may have only one outstanding subscription per service. The message delivery must occur in guaranteed sequence. Messages from the same service will arrive in the sequence in which they have been sent.

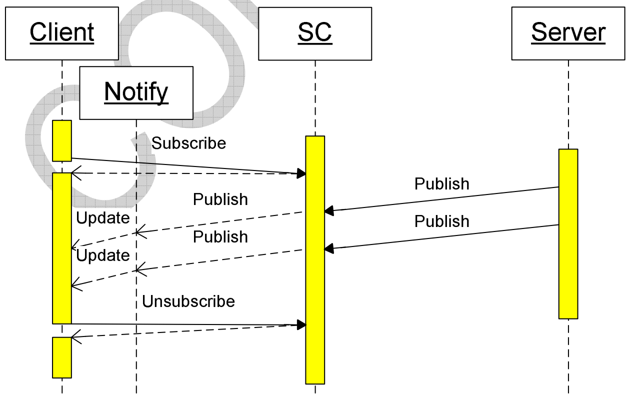


Figure 6 Asynchronous Subscribe / Publish

The client may change the subscription mask or unsubscribe. Initial subscription, subscription change or unsubscribe operation is always synchronous, even through a SC proxy.

Such communication style is used to get asynchronously events notifications or messages that are initiated on the server without an initial client action. It can be also used to distribute the same information to multiple clients.

3.3 File Services

This SC service provides API for these file operations:

- Download file from the web server to the client.
- Upload file from the client to the web server.
- List files in a file repository on the web server.

3.4 HTTP Redirection Services

The SC supports redirecting of regular HTTP traffic to another server. It is acting like normal HTTP Proxy with no caching.

4 Message Protocol

The Service Connector Message Protocol (SCMP) defines how the messages are transmitted. It uses a simple header – body pattern.

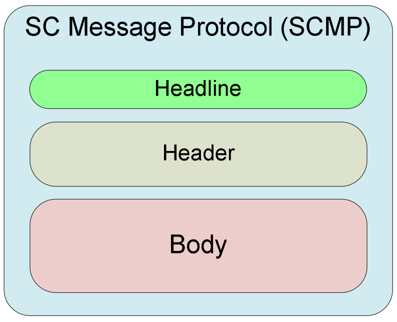
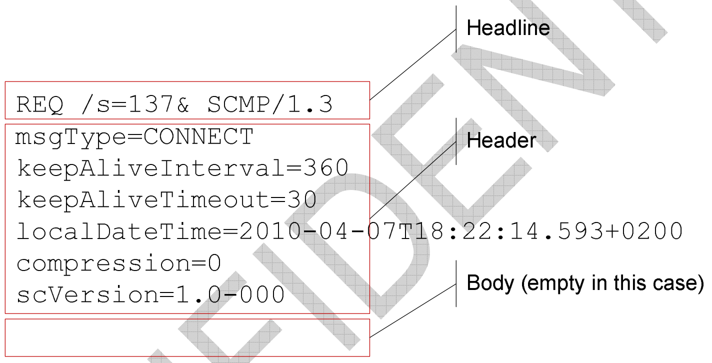


Figure 7 Service Connector Message Protocol

Wireshark example:



```
REQ /s=137& SCMP/1.3
msgType=CONNECT
keepAliveInterval=360
keepAliveTimeout=30
localDateTime=2010-04-07T18:22:14.593+0200
compression=0
scVersion=1.0-000
```

4.1 Headline

The headline defines the SCMP version, the size of the message and the header key. It is encoded in ISO-8859-1 (Latin 1) and terminated by <CR><LF>.

HeaderKey The header key of the message defines the purpose of the message and can be:
- REQ – Request from client to SC or SC to server
- RES – Response from server to SC or SC to client
- PRQ – Large request part from client to SC or SC to server
- PRS – Large response part from server to SC or SC to client
- EXC – Exception returned after REQ or PRQ in case of an error

Size The size of the message in bytes is counted from the beginning of the next line, i.e. from the first header and includes the body if present. The size is the number between “/s=” and “&”

Version The SCMP version is the version of the protocol specification to which this message adheres. It has the format 9.9 and it ensures that the receiver knows how this message is structured. The version number (e.g. 1.3) means: 1 = Release number, 3 = Version number.
The receiver may implement multiple protocol versions, thus “understand” older versions. The following matching rules applies:
- Message: 1.0 + receiver implements: 1.0 => compatible
- Message: 1.0 + receiver implements: 1.1 => compatible

- Message: 1.3 + receiver implements: 1.0 => not compatible
  (message may have new headers unknown to the receiver)
- Message: 1.4 + receiver implements: 2.0 => not compatible
  (old message structure and possibly not understood here)
- Message: 2.0+ receiver implements: 1.8 => not compatible
  (new message and surely not understood here)

4.2 Header

Message header containing attributes of variable number and length each attribute is on a separate line e.g. delimited by <CR><LF>. Attributes are encoded in ISO-8859-1 (Latin 1) character set.

4.3 Body

The message body itself is binary data or ISO-8859-1 (Latin 1) encoded text. The attribute *bodyType* defines the format. It is under control of the applications. When compression is enabled, body is ZIP-compressed during transmission.

4.4 SCMP over TCP/IP

For direct transport over TCP/IP the headline, messages header and the body is directly written to the network connection.

4.5 SCMP over HTTP

For transport over HTTP the headline and messages header have content type **text/plain** and the message body the content type **application/zip**.
The request message uses method POST, the response is regular HTTP response.

In order to distinguish regular (plain) HTTP traffic from SCMP over HTTP, the following HTTP headers are used for request and response:

```
Pragma: SCMP
Cache-Control: no-cache
```

Actually chunked transfer encoding and pipelining are not used.

Wireshark example:

```
POST / HTTP/1.1
Content-Length: 178
Content-Type: text/plain
Host: 192.234.123.33
Pragma: SCMP
Cache-Control: no-cache

REQ /s=137& SCMP/1.3
msgType=CONNECT
keepAliveInterval=360
keepAliveTimeout=30
localDateTime=2010-04-07T18:22:14.593+0200
compression=0



HTTP/1.1 200 OK
Content-Length: 74
Content-Type: text/plain
Pragma: SCMP
Cache-Control: no-cache

RES /s=59& SCMP/1.3
msgType=CONNECT
localDateTime=2010-04-07T18:22:14.593+0200
```

4.6 HPPT over SCMP

The flexible SC topology allows placement of multiple SC-Proxies on the path between the client and the server. Therefore HTTP may pass a network section configured as TCP/IP. In such section HTTP over SCMP is used. The traffic is transferred as regular messages with bodyType = http. The maximal body size limit of 64kB does not apply for HTTP over SCMP

Wireshark example:

```
REQ /s=5156& SCMP/1.3
msgType=HTTP
bodyLength=5139
bodyType=http

GET /wiki/Main_Page HTTP/1.1
Host: en.wikipedia.org



RES /s=515& SCMP/1.3
msgType=HTTP
bodyLength=467
bodyType=http

HTTP/1.0 200 OK
Content-Length: 74
Content-Type: text/html
<html xmlns="http://www.w3.org/1999/xhtml" lang="en" dir="ltr">
<head>
<title>Main Page - Wikipedia, the free encyclopedia</title>
...
```

5 Semantic

The sequence of the messages exchanged between the components is shown in the following diagrams.

5.1 Session Service

The following schema shows message exchange with single threaded session server.

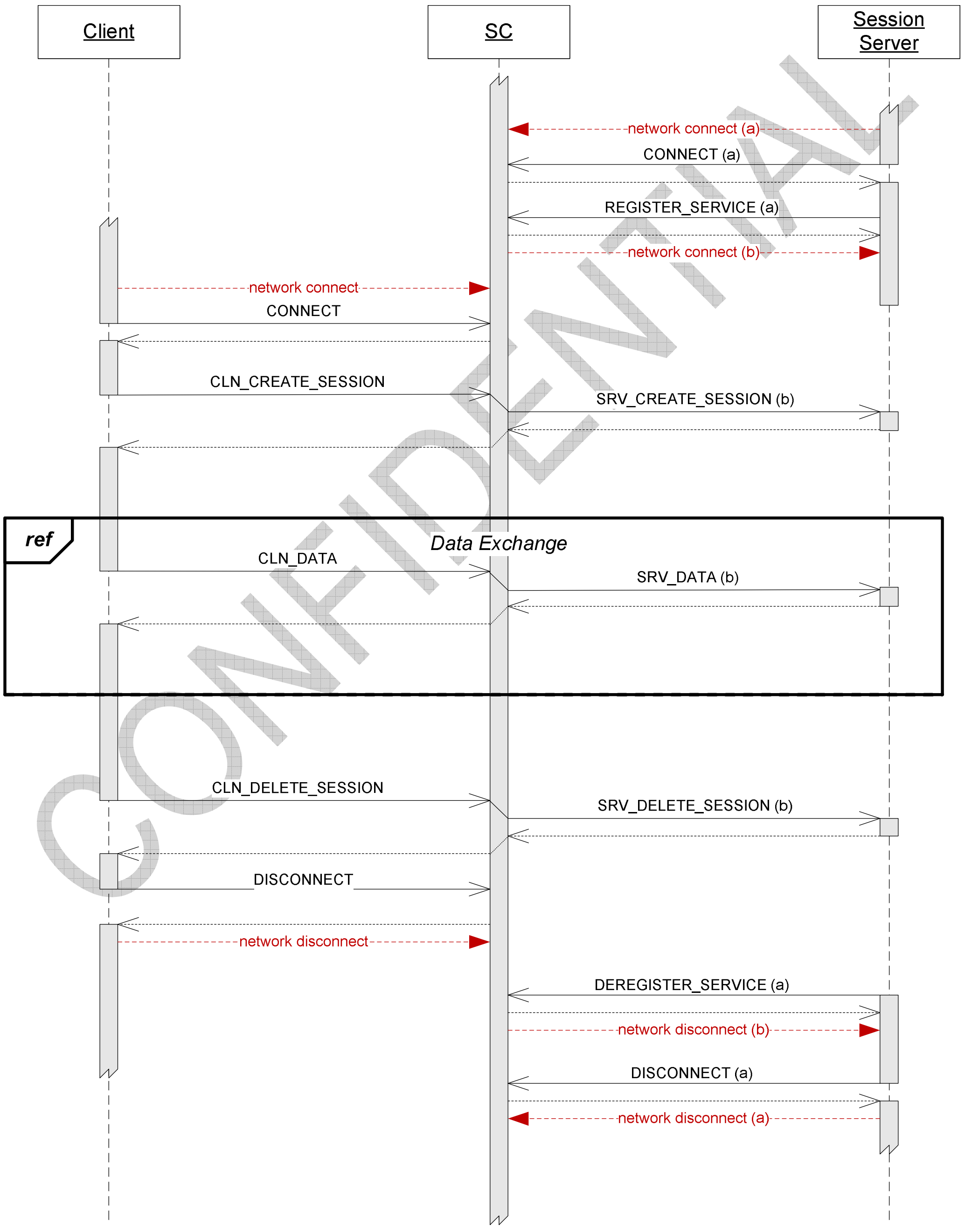


Figure 8 Synchronous Message Exchange.

Client

1. The client establishes a network connection to SC and starts communication with the CONNECT message. KEEPALIVE messages can be sent on this connection.
2. Then it starts a session with a service with the CLN_CREATE_SESSION message
3. The SC allocates a server providing this service and notifies it about the session start with the ASRV_CREATE_SESSION message. It also creates a unique sessionId for this session.
4. Then the client can exchange messages with the server via CLN_DATA messages.
5. When the session with this service is no longer needed, it will be deleted with the CLN_DELETE_SESSION message. The server is notified with the SRV_DELETE_SESSION message.
6. Before the client terminates, it should send DISCONNECT message and then terminate the network connection to SC.

When session is abnormally terminated, the client is notified and will receive and error message. The reasons for this can be:

- Server sends DEREGISTER_SERVICE or DISCONNECT message
- Unexpected server exit (e.g. keepalive timeout expiration)
- Underlying communication error (e.g. keepalive timeout expiration)

Note

The client may have multiple SCs connected at the same time. Per connected SC the client may have multiple active sessions at the same time. For each particular session only one request may be pending at any time.

Server

1. Like a client, the server establishes a network connection (a) to SC and starts communication with the CONNECT message. KEEPALIVE messages can be sent on this connection.
2. Then it registers itself as a instance of a service with the REGISTER_SERVICE message. At that time it should have a listener that will accept the connection (b) initiated by the SC to this server.
3. The SC registers the server instance and will create network connection (b) to it. **No** KEEPALIVE messages can be sent on this connection!
4. When a client creates a session the server is notified with the SRV_CREATE_SESSION message. The message contains additional information about the client and the sessionId. The server can accept or reject the session.
5. The server receives messages from the client as SRV_DATA and sends them back after execution of the service that it implements.
6. When the client deletes the session, the appropriate server is notified with the SRV_DELETE_SESSION message.
7. When the server is no longer needed it sends the message DEREGISTER_SERVICE. From this point the SC will not allocate it for a session and terminates the connection (b) to it.
8. Before the server terminates, it should send DISCONNECT message and then terminate the network connection (a) to SC.

When the session is abnormally terminated before, the server is notified with the SRV_ABORT_SESSION message. The reasons for this can be:

- Client sends DISCONNECT message
- Unexpected client exit (e.g. keepalive timeout expiration)
- Underlying communication error (e.g. keepalive timeout expiration)

The network connection (a) must not be dropped until DISCONNECT message is sent! Otherwise SC will treat this as server termination and clean-up its sessions and its registration.

Note

The server must be active before the client will create a session. Otherwise the client will receive a error message. The server may have only one SCs connected at the same time. The server instance may serve single or multiple sessions at the same time as described later. For each particular session only one request may be pending at any time.

5.1.1 Asynchronous Message Exchange

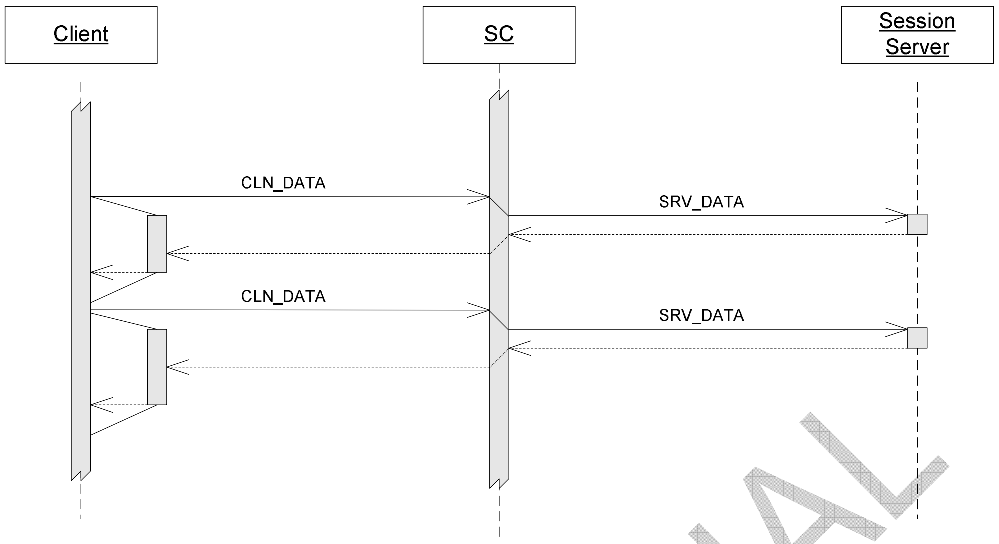


Figure 9 Asynchronous Session Service Message Exchange.

Fully asynchronous message is not possible because the server execution is always synchronous. For this reasons only the receipt of the server response can be asynchronous.

1. The client sends a CLN_DATA message and declares a notification method that will receive the response. It then can continue processing
2. When the response arrives, the notification method is invoked and can process it.

Note! **Only one request may be pending at any time. Subsequent request will block until the response for the previous message arrives.**

5.1.2 Large Messages

Regular request / response messages have a REQ / RES headerKey in the head line. Large messages are broken into parts with its own headerKey PRQ (part request) and PRS part response) see 4.1.

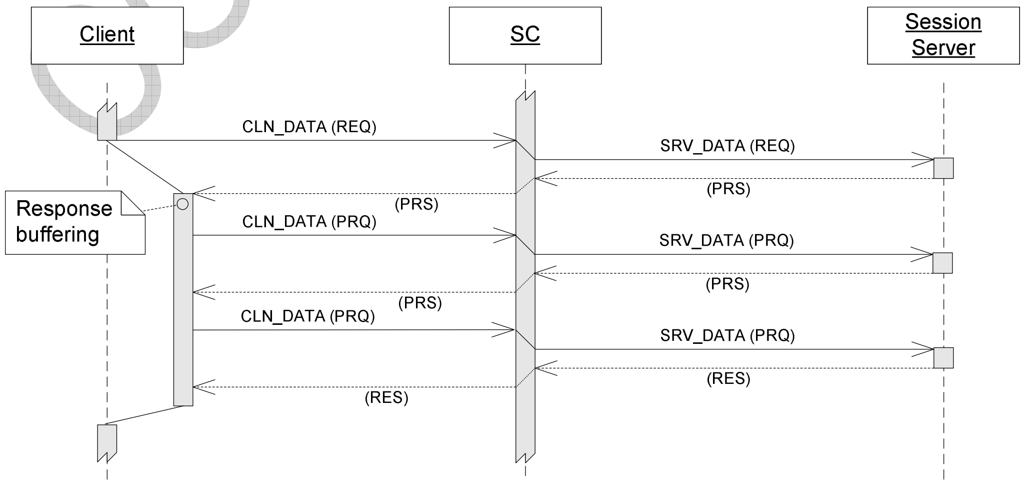


Figure 10 Large Response.

1. The client sends a regular request to the service.
2. The server receives the request messages and start producing the response.
3. When it reaches the max allowed length (60kB) it send the message part (PRS) and waits for the next part (PRQ)
4. At the end the server sends the last message part as a regular response (RES)
5. The message parts are buffered on the client side
6. When the final response arrives, the message is made available to the client.

In order to put together all message parts the server must allocate a unique partId and use it for all parts of one message.

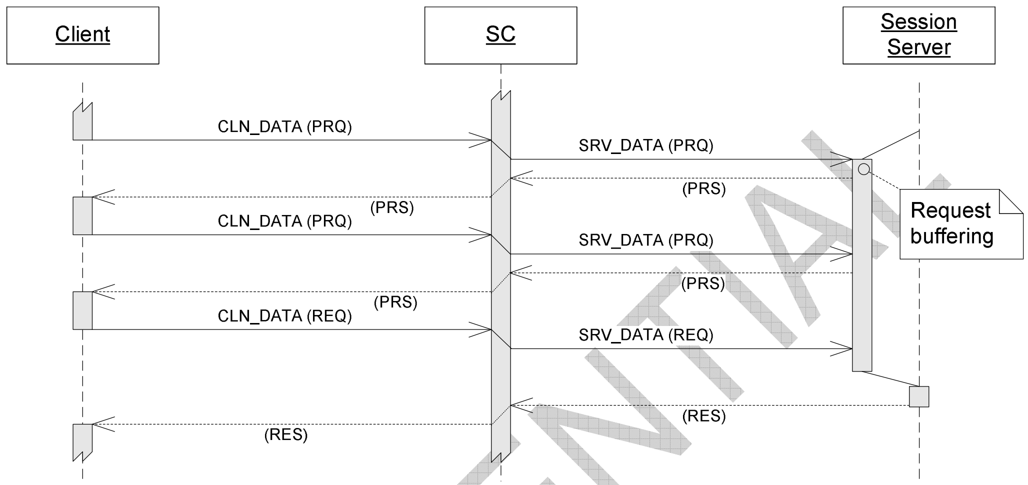


Figure 11 Large Request.

1. The client collects the request data and when it reaches the max allowed length (60kB) it sends the message part (PRQ) to the service.
2. The message part transported to the server and buffered here
3. When the final message part (REQ) arrives, the complete is made available to the server.
4. This will process the message and send back the response message.

In order to put together all message parts the client must allocate a unique partId and use it for all parts of one message.

Combination of large request and large response is possible.

For large messages the messageId contains additional counter called part sequence number. Message traffic with large request followed by a large response looks like (simplified):

```
REQ..   messageId=3
RES..   messageId=64
...
PRQ ..   messageId=4/1
PRS ..   messageId=65/1
PRQ ..   messageId=4/2
PRS ..   messageId=65/2
PRQ ..   messageId=4/3
PRS ..   messageId=65/3
REQ ..   messageId=4/4
PRS ..   messageId=66/1
PRQ ..   messageId=5/1
PRS ..   messageId=66/2
PRQ ..   messageId=5/2
RES ..   messageId=66/3
...
REQ..   messageId=6
RES..   messageId=67
```

5.1.3 Single Threaded Server

Single threaded server registers itself for one service and may serve only one session at the same time. It must define *maxSession= 1* and *multiThreaded* = 0. The required parallelism is reached by starting multiple instances of the same server. The SC keeps track of the registered servers and assigns / de-assigns sessions to them.

5.1.4 Multi Threaded Server

In opposite to single-threaded server, multi threaded server may serve multiple sessions at the same time. It must registers itself for one or more service and must define reasonable high *maxSession* > 1 and *multiThreaded* = 1. It uses the same network connection for all sessions.

Multi-threaded server is not supported together with HTTP transport. A firewall between the SC and the server may block the traffic.

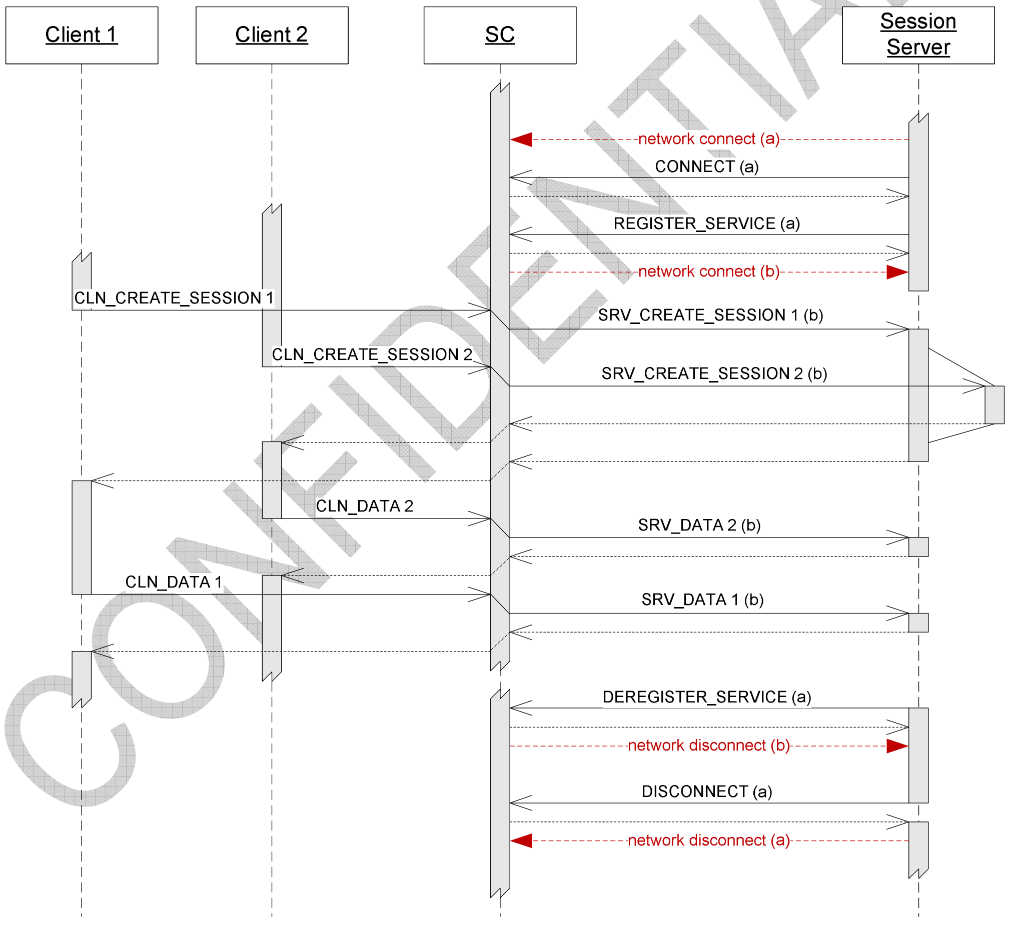


Figure 12 Multi threaded server.

Multi threaded server receives all SC requests on the same network connection and handles them appropriately. It may use any available technique (e.g. multithreading), but must ensure that all requests are processed in parallel.

5.1.5 Multi Connection Server

Multi Connection server can also serve multiple sessions at the same time but uses individual connection for each session. The connection created when the session begins and terminated at the end, i.e. the session is assigned to the connection for its life time. The server registers itself for one or more service and defines reasonable high *maxSession* > 1 and *multiThreaded* = 0.

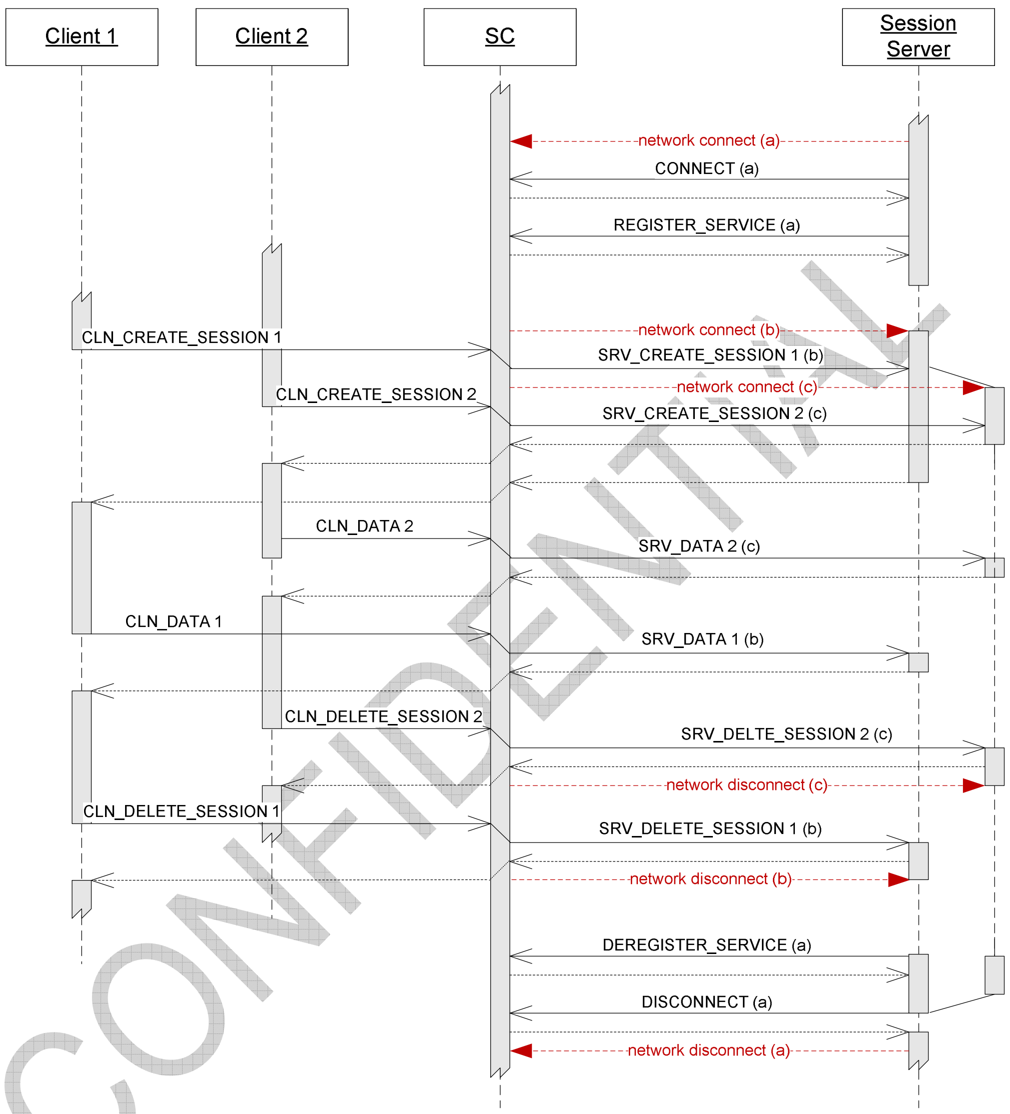


Figure 13 Multi-Connection server.

Multi-Connection server receives SC requests on the network connection that is allocated to the particular session. It may use any available technique (e.g. multithreading), but must ensure that all requests are processed in parallel.

The network connection (a) must not be dropped until DISCONNECT message is sent! Otherwise SC will treat this as server termination and clean-up all its sessions and registrations.

5.1.6 Application Server (Tomcat)

SCMP Messaging with an application server (e.g. Tomcat) works exactly like a multi-connection server with the only difference that it utilizes the HTTP protocol.

The application server must register itself for at least one service. It can register for multiple services! It must define reasonable high *maxSession* and *multiThreaded* = 0.

5.2 Publishing Service

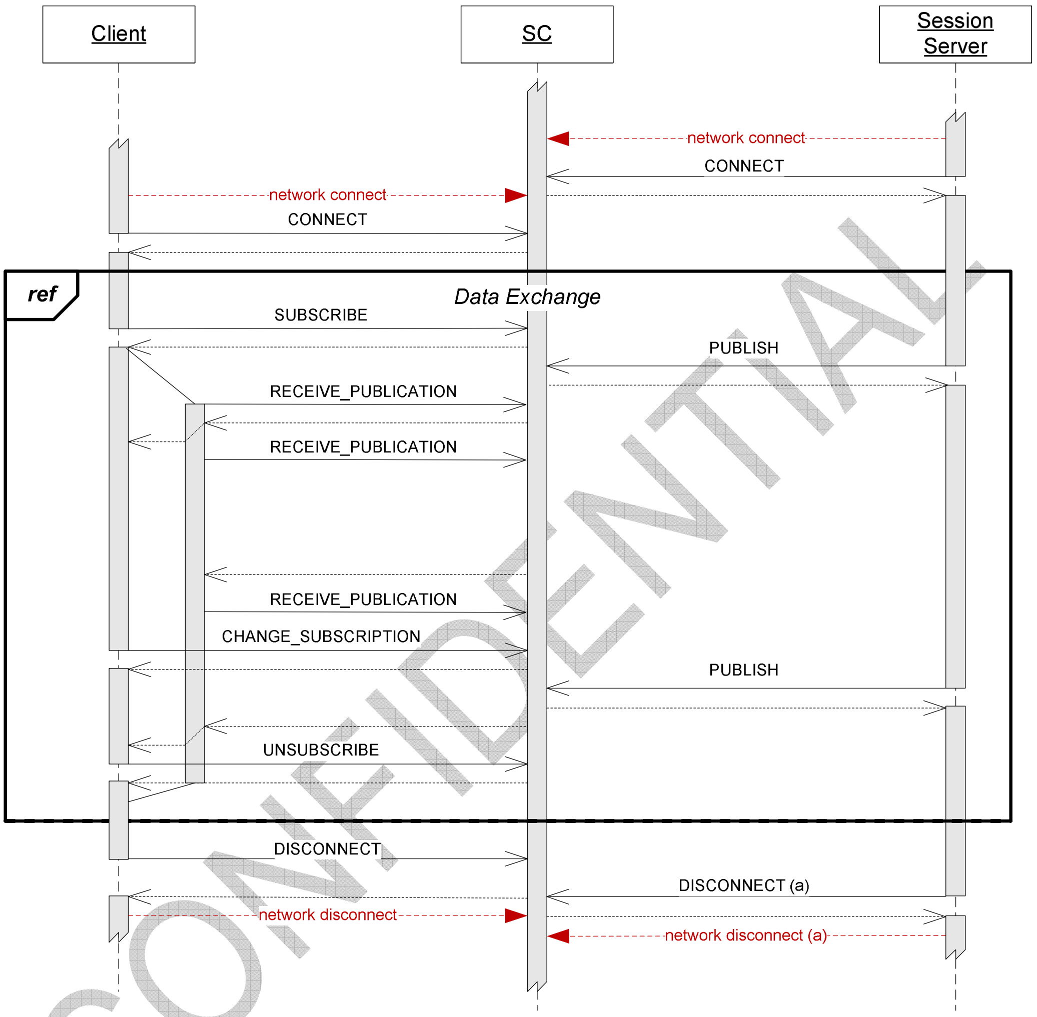


Figure 14 Publishing Service

Client

1. The client establishes a network connection to SC and starts communication with the CONNECT message. On this connection KEEPALIVE message can be sent.
2. Then it subscribes to a service with the SUBSCRIBE message and starts a listener that will receive the incoming messages.
3. The SC remembers the subscription and creates a unique sessionId for it.
4. When a message is published, SC compares the message mask with the subscription mask and based on the matching result delivers the message to the client. The client receives and processes the message and initiates the next receipt with the RECEIVE_PUBLICATION message.
5. When no message is published within a period of time (define by *keepaliveTimeout*) then SC sends an empty message to the client and this initiates the next receipt with the RECEIVE_PUBLICATION message.
6. The client can change the publication mask with the CHANGE_SUBSCRIPTION message or terminate the subscription with UNSUBSCRIBE message.

7. Before the client terminates, it should send DISCONNECT message and then terminate the network connection to SC.

When the client terminates abnormally terminated, the SC will clear its subscription and discard all messages not delivered yet. The reasons for this can be:

- Client sends DISCONNECT message
- Unexpected client exit (e.g. keepalive timeout expiration)
- Underlying communication error (e.g. keepalive timeout expiration)

Note **The client may have multiple SCs connected at the same time. Per connected SC the client may have multiple subscriptions to different services at the same time. For each subscription only one receipt request may be pending at any time.**

Server

1. Like a client, the server establishes a network connection to SC and starts communication with the CONNECT message. On this connection KEEPALIVE message can be sent.
2. From then it can publish messages to the service. It does not have to register to the service. SC immediately responds when the PUBLISH message has been queued. The server does not wait for message delivery to the clients.
3. The published message must have a mask designating its contents. The SC compares this with the subscription mask of the clients and delivers the message to them. Messages that do not match any subscription are discarded. The server does not know how many clients did get the message and if at all.
4. Messages are delivered in the order of their publishing. E.g. in order SC receives them. For this reason they are queued in SC.
5. Before the server terminates, it should send DISCONNECT message and then terminate the network connection (a) to SC.

5.3 Web Server (Apache)

Web Server does not register to service. Instead SC has configuration items that allows redirection of plain HTTP traffic to the configured node and port. The network connections are dynamically created form SC to the server when they are needed. Multiple HTTP requests can be pending at the same time, each one using one network connection.

The limitation of 64kB for SCMP messages does not apply for traffic to and from Web Server

6 SCMP Messages

6.1 CONNECT

This message is sent from the client to SC or from server to SC in order to initiate the communication. The message has no body and contains these attributes:

```
msgType=CONNECT
scVersion=1.0-023
compression=0
localDateTime=1997-07-16T19:20:30.064+0100
keepAliveTimeout=10
keepAliveInterval=60
```

SC receives the message, starts monitoring of the connection based on keep alive values and sends back the response:

```
msgType=CONNECT
localDateTime=1997-07-16T19:20:34.044+0200
```

When an error occurs the response message contains the attributes:

```
msgType=CONNECT
localDateTime=1997-07-16T19:20:30.453+0100
scErrorCode=3000
scErrorText=Service Connector Message Protocol mismatch
```

6.2 DISCONNECT

This message is sent from the client to SC or from server to SC in order to terminate the communication. The message has no body and contains these attributes:

```
msgType=DISCONNECT
```

SC receives the message, starts monitoring of the connection based on keep alive values and sends back the response:

```
msgType=DISCONNECT
```

When an error occurs the response message contains the attributes:

```
msgType=DISCONNECT
scErrorCode=3000
scErrorText=protocol mismatch
```

6.3 KEEPALIVE

This message is sent from the client to SC or from server to SC in order to verify the communication. The message has no body and contains these attributes:

```
msgType=KEEPALIVE
localDateTime=1997-07-16T19:20:30.343+0100
```

The sender of the keep alive message starts a timer that monitors the arrival of the response message. In case the timeout expires the connection must be aborted and cleaned up. The receiver of the keep alive message monitors the interval in which these messages arrive. In case the message does not arrive in the pre-defined interval, the connection must be aborted and cleaned up.

SC receives the message, and sends back the response:

```
msgType=KEEPALIVE
localDateTime=1997-07-16T19:20:34.237+0200
```

6.4 INSPECT

This message is sent from the client to SC in order to get internal information from the SC. The message has body of type *message* and contains these attributes:

```
msgType=INSPECT
bodyLength=253
bodyType=message
```

The body content and its processing will be described at later project stage.

The message returned by SC has a body of type *message* and contains these attributes:

```
msgType= INSPECT
bodyLength=127
bodyType=message
ipAddressList=10.0.4.32/10.2.54.12/192.243.43.1
```

6.5 ECHO_SC

This message is sent from client to SC in order to verify the proper functioning of the SC transport. The message is sent on the path specified by the *serviceName* attribute but does not require a session. The message has no body and contains these attributes:

```
msgType=ECHO_SC
serviceName=P01_RTXS_RPRWS1
maxNodes=9
```

The message returned by SC has no body and contains these attributes:

```
msgType=ECHO_SC
serviceName=P01_RTXS_RPRWS1
ipAddressList=10.0.4.32/10.2.54.12/192.243.43.1
```

6.6 CLN_CREATE_SESSION

This message is sent from the client to SC in order to start a new session for a service. The message has no body and contains these attributes:

```
msgType=CLN_CREATE_SESSION
serviceName=P01_RTXS_RPRWS1
ipAddressList=10.0.4.32/10.2.54.12
sessionInfo=SNBZHP - TradingClientGUI 10.2.7
```

SC receives the message and does these actions:

1. Generates a unique session id
2. Chooses a free server instance from the list of available servers serving the requested service.
3. Allocates the server instance to this session
4. Sends the message SRV_CREATE_SESSION to the allocated server and awaits the server response.
5. If the response message has attribute rejectFlag = 0, the SC keeps the session and sends back to client the message with the following attributes:

```
msgType=CLN_CREATE_SESSION
serviceName=P01_RTXS_RPRWS1
sessionId=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d
```

rejectFlag=0

6. If the response message contains the attribute rejectFlag = 1, the SC deletes the session, de-allocates the server and sends back to client the message with the following attributes:
   msgType=CLN_CREATE_SESSION
   serviceName=P01_RTXS_RPRWS1
   rejectSession=1
   appErrorCode=4334591
   appErrorText=%RTXS-E-NOPARTICIPANT, Authorization error – Unknown participant

When an error occurs the response message contains the attributes:
msgType=CLN_CREATE_SESSION
scErrorCode=3000
scErrorText=Unkown Service = P01_RTXS_RPRWS3

6.7 SRV_CREATE_SESSION

This message is sent from the SC to the server when the server instance has been be allocated to a session. The message has no body and contains these attributes:
msgType=SRV_CREATE_SESSION
serviceName=P01_RTXS_RPRWS1
sessionId=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d
sessionInfo=SNBZHP - TradingClientGUI 10.2.7
ipAddressList=10.0.4.32/10.2.54.12/192.243.43.1

The server receives the message and must decide to accept or reject this request. If it accepts, then it must return a message with the following attributes:
msgType=SRV_CREATE_SESSION
serviceName=P01_RTXS_RPRWS1
sessionId=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d
rejectFlag=0

If it rejects the session, then it must return a message with the following attributes:
msgType=SRV_CREATE_SESSION
serviceName=P01_RTXS_RPRWS1
sessionId=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d
rejectFlag=1
appErrorCode=4334591
appErrorText=%RTXS-E-NOPARTICIPANT, Authorization error – Unknown participant

6.8 CLN_DELETE_SESSION

This message CLN_DELETE_SESSION is sent from the client to SC in order to close an existing session. The message has no body and contains these attributes:
msgType=CLN_DELETE_SESSION
serviceName=P01_RTXS_RPRWS1
sessionId= cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d

SC receives the message and does these actions:
1. Finds the server allocated to this session
2. Sends the message SRV_DELETE_SESSION to the allocated sever and awaits its response.
3. De-allocates the server instance from this session
4. Sends back the message CLN_DELETE_SESSION with the following attributes:

msgType=CLN_DELETE_SESSION
serviceName=P01_RTXS_RPRWS1
sessionId= cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d

Due to timing issues the client may send a delete session request to a non existing session or to session that has no allocated server. The SC must handle such situation and do all appropriate clean-up actions.

When an error occurs the response message contains the attributes:
msgType=CLN_DELETE_SESSION
scErrorCode=3000
scErrorText=Session does not exist

6.9 SRV_DELETE_SESSION

This message is sent from the SC to the server when the session will be deleted by the client and the server instance will no longer be bound to it. The message has no body and contains these attributes:
msgType=SRV_DELETE_SESSION
serviceName=P01_RTXS_RPRWS1
sessionId=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d

The server must return a message with the following attributes:
msgType=SRV_DELETE_SESSION
serviceName=P01_RTXS_RPRWS1
sessionId=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d

After this message the server will not receive any data requests until the next session is started.

6.10 SRV_ABORT_SESSION

This message is sent from the SC to the server when the session is aborted due to errors or other unexpected events. The message has no body and contains these attributes:
msgType=SRV_ABORT_SESSION
serviceName=P01_RTXS_RPRWS1
sessionId=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d

The server must return a message with the following attributes:
msgType= SRV_ABORT_SESSION
serviceName=P01_RTXS_RPRWS1
sessionId=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d

After this message the server will not receive any data requests until the next session is started.

6.11 REGISTER_SERVICE

This message is sent from the server instance to SC in order to tell the SC which service it serves. The message has no body and contains these attributes:
msgType=REGISTER_SERVICE
serviceName=P01_RTXS_RPRWS1
maxSessions=10
multiThreaded=0
portNr=9100

SC receives the message and does these actions:
1. Registers the server for this service.
2. Creates the requested number of connection to the server on port that was specified.

3. Sends back a message with the following attributes:
   msgType=REGISTER_SERVICE
   serviceName=P01_RTXS_RPRWS1

When an error occurs the response message contains the attributes:
msgType=REGISTER_SERVICE
scErrorCode=3000
scErrorText=Service name=P01_RTXS_RPRWS5 not found

6.12 DEREGISTER_SERVICE

This message is sent from the server instance to SC in order to tell the SC that the server will no longer provide the service and will perform a conscious shutdown. The message has no body and contains these attributes:
msgType=DEREGISTER_SERVICE
serviceName=P01_RTXS_RPRWS1

SC receives the message and does these actions:
1. Finds the server and performs a cleanup by:
2. Aborting and de-allocation all sessions of this server. If the server has allocated sessions the SC will first send the SRV_ABORT_SESSION to it.
3. Sends back message with the following attributes:
   msgType=DEREGISTER_SERVICE
   serviceName=P01_RTXS_RPRWS1

When an error occurs the response message contains the attributes:
msgType=DEREGISTER_SERVICE
scErrorCode=3000
scErrorText=Server is not registered

After this message the server may close connection to the SC with the DISCONNECT

6.13 CLN_ECHO

This message is sent from the client to SC and passed to the server in order to verify the complete message exchange through all SC components. The client may send this message only in scope of a session. The message has body of any type and contains these attributes:
msgType=CLN_ECHO
sessionId=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d
serviceName=P01_RTXS_RPRWS1
messageId=974833
bodyLength=2534
bodyType=text

SC receives the message and finds the server allocated to this session.
It passes the message SRV_ECHO to the server and awaits the response. Then it sends back a message with a body and the following attributes:
msgType=CLN_ECHO
sessionId=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d
serviceName=P01_RTXS_RPRWS1
messageId=974834
bodyLength=2534
bodyType=text

Large messages are supported in this context.

6.14 SRV_ECHO

This message is sent from SC to the server as result of the CLN_ECHO request. The message has body of any type and contains these attributes:

```
msgType=SRV_ECHO
sessionId=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d
serviceName=P01_RTXS_RPRWS1
messageId=974833
bodyLength=2534
bodyType=text
```

The server receives the message and sends back a message with the same body and the following attributes:

```
msgType=SRV_ECHO
sessionId=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d
serviceName=P01_RTXS_RPRWS1
messageId=974834
bodyLength=2534
bodyType=text
```

6.15 CLN_DATA

This message is sent from the client to SC in order exchange information with the allocated server. The client may send this message only in scope of a session. The message has a body and contains these attributes:

```
msgType=CLN_DATA
sessionId=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d
serviceName=P01_RTXS_RPRWS1
bodyLength=2534
messageId=974833
compression=0
msgInfo=SECURITY_MARKET_QUERY
```

Optionally these attributes can also be set when the client wants to fetch the message from the SC Proxy cache:

```
cacheId=CBCD_SECURITY_MARKET
cacheExpirationDateTime=1997-08-16T19:20:34.237+0200
```

SC receives the message and finds the server allocated to this session.
It sends the message SRV_DATA to the server it and awaits the response. Then it sends back a message with a body and the following attributes:

```
msgType=CLN_DATA
sessionId=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d
serviceName=P01_RTXS_RPRWS1
bodyLength=62454
messageId=974834
compression=0
msgInfo=SECURITY_MARKET_RESULT
```

Optionally these attributes can also be set when the server wants to store the message in the SC Proxy cache:

```
cacheId=CBCD_SECURITY_MARKET
cacheExpirationDateTime=1997-08-16T17:00:00.000+0100
```

In case of an application error these attributes can also be set.

```
appErrorCode=4334591
appErrorText=%RDB-F-NOTXT, no transaction open
```

When an error occurs the response message contains the attributes:

```
msgType=CLN_DATA
scErrorCode=3000
scErrorText=Session does not exist
```

Large messages are supported in this context.

6.16 SRV_DATA

This message is sent from the SC to the server allocated to this session in order to execute the request. The SC will send this message only in scope of a session. The message has a body and contains these attributes:

```
msgType=SRV_DATA
sessionId=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d
serviceName=P01_RTXS_RPRWS1
bodyLength=2534
messageId=974833
compression=0
msgInfo=SECURITY_MARKET_QUERY
```

The server receives the message extracts the body and executes the application code. It must send back a message with a body and the following attributes:

```
msgType=SRV_DATA
sessionId=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d
serviceName=P01_RTXS_RPRWS1
bodyLength=62454
messageId=974834
compression=0
msgInfo=SECURITY_MARKET_RESULT
```

Optionally these attributes can also be set when the server wants to insert the message into the SC Proxy cache:

```
cacheId=CBCD_SECURITY_MARKET
cacheExpirationDateTime=1997-08-16T17:00:00.000+0100
```

In case of an application error these attributes can also be set.

```
appErrorCode=4334591
appErrorText=%RDB-F-NOTXT, no transaction open
```

6.17 HTTP

This message is used to transport HTTP protocol over SCMP. This is section necessary when a SC network segment is configured to use plain TCP/IP. The message has a regular body and contains these attributes:

```
msgType=HTTP
bodyLength=2534
bodyType=http
compression=0
```

SC passes this message to the next node as defined in its configuration.

6.18 SUBSCRIBE

This message is sent from the client to SC in order to subscribe for a publishing service. The message has no body and contains these attributes:

```
msgType=SUBSCRIBE
serviceName= P01_BCST_CH_RPRWS2
mask= 000012100012832102FADF-------------------------
```

SC receives the message and does these actions:

1. Generates a unique session id
2. Registers the client subscription for the service
3. Sends back a message with the following attributes:
   msgType=SUBSCRIBE
   sessionId=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d
   serviceName=P01_RTXS_RPRWS1

When an error occurs the response message contains the attributes:
msgType=SUBSCRIBE
scErrorCode=3000
scErrorText=Unkown Service = P01_BCST_CH_RPRWS2

6.19 UNSUBSCRIBE

This message is sent from the client to SC in order to delete the subscription for a publishing service. The message has no body and contains these attributes:
msgType=UNSUBSCRIBE
sessionId=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d
serviceName=P01_BCST_CH_RPRWS2

SC receives the message and does these actions:

1. Deletes the client subscription for the services and deletes all pending messages for this client.
2. Sends back a message with the following attributes:
   msgType=UNSUBSCRIBE
   serviceName=P01_RTXS_RPRWS1

When an error occurs the response message contains the attributes:
msgType=UNSUBSCRIBE
scErrorCode=3000
scErrorText=Client is not subscribed

6.20 CHANGE_SUBSCRIPTION

This message is sent from the client to SC in order to change the subscription for a publishing service. The message has no body and contains these attributes:
msgType=CHANGE_SUBSCRIPTION
sessionId=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d
serviceName=P01_BCST_CH_RPRWS2
mask= 000012100012832102FADF-------------------------

SC receives the message and does these actions:

1. Changes the subscription mask of the client for the service
2. Sends back a message with the following attributes:
   msgType=CHANGE_SUBSCRIPTION
   serviceName=P01_RTXS_RPRWS1

When an error occurs the response message contains the attributes:
msgType=CHANGE_SUBSCRIPTION
scErrorCode=3000
scErrorText=Client is not subscribed

6.21 RECEIVE_PUBLICATION

This message is sent from the client to SC in order to get data published by a server. The client may send this message only in scope of a subscription session. The message has no body and contains these attributes:

msgType=RECEIVE_PUBLICATION
sessionId=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d
serviceName=P01_BCST_CH_RPRWS2
messageId=974833

SC receives the message and does these actions:

1. Finds the client subscription
2. Creates a timer monitoring the response delivery
3. Waits until one of these two events occurs:
   a. A message that matches the client subscription arrives. Then it sends back a message with the body and the following attributes:
      msgType=RECEIVE_PUBLICATION
      sessionId=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d
      serviceName= P01_BCST_CH_RPRWS2
      bodyLength=2534
      messageId=974834
      compression=1
      msgInfo=CH_AUCTION
      mask=%%%%%%%%%%%%%%%%%%%%%%%%%%%%X%%
      %%%%%%%%%%%%%%%%%%

   b. The timeout expires. Then it sends back a message with the no body and the following attributes:
      msgType=RECEIVE_PUBLICATION
      sessionId=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d
      serviceName=P01_BCST_CH_RPRWS2
      messageId=974834
      noData=1

When an error occurs the response message contains the attributes:
msgType=RECEIVE_PUBLICATION
scErrorCode=3000
scErrorText=Client is not subscribed

6.22 PUBLISH

This message is sent from the publishing server to SC in order to send this message to the subscribed clients. The message has a body and contains these attributes:
msgType=PUBLISH
serviceName= P01_BCST_CH_RPRWS2
bodyLength=2534
compression=0
messageId=65411
msgInfo=CH_AUCTION_IOI
mask=%%%%%%%%%%%%%%%%%%%%%%%%%%%%X%%%%%%%%%
%%%%%%%%%%

SC receives the message and does the following steps:

1. It inserts the message on top of the message queue for this service
2. Sends back to the server a message with the following attributes:
   msgType=PUBLISH
   serviceName=P01_BCST_CH_RPRWS2
   messageId=65412
3. Starts distribution of the message to the subscribed clients based on their subscription mask and the mask of the message.

When an error occurs the response message contains the attributes:
msgType=PUBLISH
scErrorCode=3000

scErrorText=Service does not exist

7 SCMP Header Attributes

The following is a list of all possible attributes in a SC message header in alphabetical order. All attributes are ASCII, encoded as ISO 8859-1 (Latin-1).

You can find the matrix describing which attribute is used in which message in **Error! Reference source not found.**.

7.1 appErrorCode

Name	appErrorCode
Description	Numeric value passed between server and the client used to implement error protocol on the application level.
Validation	Numeric value ≥ 0
Comment	This can be used by the client to check a specific server error.
Example	appErrorCode=4334591

7.2 appErrorText

Name	appErrorText
Description	Textual value passed between server and the client used to implement error protocol on the application level. It can be the textual interpretation of the *appErrorCode*.
Validation	Any printable character, length > 0 and < 256Byte
Comment	This can be used by the client to display or log an error that occurred on the server and so get the user better understanding what happened.
Example	appErrorText=%RDB-F-NOTXT, no transaction open

7.3 bodyType

Name	bodyType
Description	Type of the message body.
Validation	Enumeration: • text – message body is ISO-8859-1 (Latin 1) encoded text • message – internal SC message • binary – binary data (default if attribute is missing) • http – message is part of HTTP over SCMP transport • *xml – XML data (not implemented yet)*
Comment	When http transport is used, the content-type header is set according to this attribute.
Example	bodyType=text

7.4 bodyLength

Name	bodyLength
Description	Number of bytes in the message body.
Validation	Positive numeric value ≥ 0 and < 65534
Comment	If missing, the message has no body. If the bodyLength = 0 then body is present but empty.

Example	bodyLength=2534

7.5 cacheExpirationDateTime

Name	cacheExpirationDateTime
Description	Absolute expiration date and time of a cached message. It must be set together with *cacheId* attribute.
Validation	YYYY-MM-DDThh:mm:ss.fff+hhmm It is UTC plus zone information. The fff are seconds fractions and time zone offset is at the end.
Comment	The client uses *cacheExpirationDateTime* to tell how old the message could be. The server uses *cacheExpirationDateTime* to designate how long the message should be cached.
Example	cacheExpirationDateTime=1997-08-16T19:20:34.237+0200

7.6 cacheId

Name	cacheId
Description	Identification agreed by the communicating applications to uniquely identify the cached content. When *cacheID* is used the attribute *cacheExpirationDateTime* must also be present.
Validation	Any printable character, length > 0 and < 256Byte
Comment	The client uses *cacheId* to identify which message should be retrieved from the cache. The server uses *cacheId* to designate message that should be cached. The client will get a cached message when: 1. the *cacheId* matches a message in the cache and 2. *cacheExpirationDateTime* requested by the client is timely before the *cacheExpirationDateTime* of the cached message.
Example	cacheId=CBCD_SECURITY_MARKET

7.7 compression

Name	compression
Description	Flag true or false describing if the message body is compressed or not. If missing the default value is true.
Validation	0 (false) or 1 (true)
Comment	The compression is enabled or disabled on the connection level, not for an individual message or service.
Example	compression=0

7.8 ipAddressList

Name	ipAddressList
Description	List of IP addresses on the network path between the client and the session server. The first is the IP address of the client as appears on the SC component to which the client connects; the last is the IP address of the SC passing the request to the server. In between there can be any number of addresses of the intermediate SC proxies
Validation	List in format 999.999.999.999{/999.999.999.999}...
Comment	When VPN is used, the tunnel IP is part of this list. This information is used for client authentication.
Example	ipAddressList=10.0.4.32/10.2.54.12/192.243.43.1

7.9 portNr

Name	portNr
Description	Number of the TCP/IP port the session server accepts the connection(s).
Validation	Number > 0 and < 99999
Comment	When a session server registers, SC will create a connection to this server on the IP address of the server and the given port number. Multiple connections are created to the same port. The value must be > 1.
Example	portNr=9100

7.10 keepAliveInterval

Name	keepAliveInterval
Description	Interval in which the sender sends keep alive messages. The value = 0 means no keep alive messages will be used.
Validation	Number ≥ 0 and < 3600 If *keepAliveTimeout* is 0 then *keepAliveInterval* must also be 0. *keepAliveInterval* must be greater then *keepAliveTimeout*
Comment	This is used by the receiver to detect a broken connection. The receiver calculates the maximal time between two subsequent KEEPALIVE messages as *maxTime= keepAliveInterval +(keepAliveTimeout/2)* If this time expires the connection is treated as dead and a cleanup is done.
Example	keepAliveInterval =60

7.11 keepAliveTimeout

Name	keepAliveTimeout
Description	Maximal time in seconds allowed between KEEPALIVE request and response (measures by the sender). The value = 0 means no keep alive messages will be sent.
Validation	Number ≥ 0 and < 3600 If *keepAliveTimeout* is 0 then *keepAliveInterval* must also be 0.
Comment	This is used by the sender to detect a broken connection. When this timeout expires, the connection is treated as dead and a cleanup is done.
Example	keepAliveTimeout =10

7.12 localDateTime

Name	localDateTime
Description	String value describing the actual local date and time.
Validation	YYYY-MM-DDThh:mm:ss.fff+hhmm It is UTC plus zone information. The fff are seconds fractions and time zone offset is at the end.
Comment	The local date time is exchanged at the beginning of each connection and in the keep alive messages. It is used to calculate the time difference between the communicating parties and to harmonize the log for troubleshooting purposes.
Example	localDateTime=1997-07-16T19:20:30.064+0100

7.13 messageID

Name	messageId
Description	Identification generated by the sender of a message in order to identify and track it during transmission. The sessionId + the messageId uniquely identify the message at any point of its transmission. Request and response messages are treated as independent.
Validation	Composite Id in format 9[/9] First is a message sequence number optionally followed by delimiter "/" and a part sequence number to count parts of large messages. Both numbers > 0, steadily increasing.
Comment	For large messages the message sequence number is extended with a part sequence number. The message sequence number and the part sequence number are steadily incremented by the sender. This can be the client, the server or SC.

Example	**`REQ..   messageId=3`**
	`RES..   messageId=64`
	`...`
	`PRQ ..   messageId=4/1`
	`PRS ..   messageId=64/1`
	`PRQ ..   messageId=4/2`
	`PRS ..   messageId=64/2`
	`PRQ ..   messageId=4/3`
	`PRS ..   messageId=64/3`
	`REQ ..   messageId=4/4`
	`PRS ..   messageId=65/1`
	`PRQ ..   messageId=5/1`
	`PRS ..   messageId=65/2`
	`PRQ ..   messageId=5/2`
	`RES ..   messageId=65/3`
	`...`
	`REQ..   messageId=6`
	`RES..   messageId=66`

7.14 mask

Name	mask
Description	The mask is used in SUBSCRIBE or CHANGE_SUBSCRIPTION to express the client interest and in PUBLISH to designate the message contents. Only printable characters are allowed.
Validation	Any printable character, length < 256Byte
Comment	If the message mask matches the subscription mask, the client will get this message. The matching rules: • masks of unequal length <u>do not</u> match • % - matches any single character at this position • All other characters must exactly match (case sensitive)
Example	Subscription: mask= `000012100012832102FADF-----------X-----------` Matching examples: Message: mask= `000012100012832102FADF-----------X-----------` Message: mask= `0000121%%%%%%%%%%%%%%%-----------X-----------` <u>Not</u> matching examples: Message: mask= `000012100012832102FADF-----------------------` Message: mask= `0000121%%%%%%%%%%%%%%%-----------------------`

7.15 maxNodes

Name	maxNodes
Description	Number of nodes the ECHO_SC message may pass before it is echoed.
Validation	Number > 1 < 9
Comment	This is used to limit the depth of the ECHO_SC message in the SC network path. The value 1 means message to the next available SC component.
Example	maxNodes=9

7.16 maxSessions

Name	maxSessions
Description	Number of sessions this server instance can serve.
Validation	Number > 0
Comment	When a session server registers to a service it must tell the SC how many sessions it can serve. This is necessary to know in order to maintain the count of available and free session servers in SC. The value 1 means single session server. Value > 1 means multi-connection or multi-threaded server, depending on the parameter *multiThreaded*.
Example	maxSessions=10

7.17 msgInfo

Name	msgInfo
Description	Optional information passed together with the message body that helps to identify the message content without investigating the body.
Validation	Any printable character, length > 0 and < 256Byte
Comment	This can be used by the receiver of the message to simplify decision how the message should be processed. I can also be used for troubleshooting to identify the message during the message transmission.
Example	msgInfo=SECURITY_MARKET_QUERY

7.18 msgType

Name	msgType
Description	Unique message type
Validation	List of known message types
Comment	Message type that represents a certain command. The direction of the message is visible in the headline.
Example	msgType=CONNECT

7.19 multiThreaded

Name	multiThreaded
Description	Flag true or false to designate a multithreaded server. If missing the default value is false.
Validation	0 (false) or 1 (true)
Comment	When a session server registers and *maxSession* > 1, SC must know if this is a multi-connection or multi-threaded server. For multi-connection server SC will create the defined number of connections and use one connection per session, for multi-threaded servers SC will create only one connection and use it for all sessions.

Example	multiThreaded=1

7.20 noData

Name	noData
Description	NoData flag is used in RECEIVE_PUBLICATION to tell the subscribed client, that no data for publishing exists. The client will immediately send another RECEIVE_PUBLICATION to renew the interest.
Validation	0 (false) or 1 (true)
Comment	noData=0 is never sent.
Example	noData=1

7.21 rejectSession

Name	rejectSession
Description	Flag in CREATE_SESSION message set by the server when it rejects the session. In such case the server can also set the *appErrorCode* and *appErrorText* to explain the rejection reason.
Validation	0 (false) or 1 (true)
Comment	
Example	rejectSession=1

7.22 scErrorCode

Name	scErrorCode
Description	Numeric error code set by SC in other to inform the communication partner about an error.
Validation	Number > 1
Comment	This is used to handle communication error. The message must have EXC key in the headline.
Example	scErrorCode=4453

7.23 scErrorText

Name	scErrorText
Description	English text set by the SC in other to describing the error signalled as *scErrorCode*.
Validation	Any printable character, length > 0 and < 256Byte
Comment	This is used to log the communication error. The message must have EXC key in the headline.
Example	scErrorText=Unknow service name = P01_RTXR_RPRWS4

7.24 scVersion

Name	scVersion
Description	Software version number of the producer of this message.
Validation	String format 9.9-999
Comment	This version number is sent in CONNECT and checked by the receiver against its own SC version number. This ensures that only compatible components can communicate to each other. The value is hard coded in the communication components like API, SC or SC-Proxy. The version number looks like 1.0-023: • 1 = Release number

	• 0 = Version number • 023 = Revision number The matching rules are: • Request: 1.0-023 + own: 1.0-023 => compatible • Request: 1.0-023 + own: 1.0-025 => compatible • Request: 1.0-025 + own: 1.0-023 => not compatible (requestor may utilize new features unknown here) • Request: 1.0-023 + own: 1.2-005 => compatible • Request: 1.2-004 + own: 1.0-023 => not compatible (requestor uses new functions unknown here) • Request: 1.0-023 + own: 2.0-007 => not compatible (possibly other incompatible interface) • Request: 2.0-001 + own: 1.0-023 => not compatible (possibly other incompatible interface)
Example	scVersion=1.0-023

7.25 serviceName

Name	serviceName
Description	Name of the service
Validation	Any printable character, length > 0 and < 256Byte
Comment	The service name is an abstract name and represents the logical address of the service. It order to allow message routing the name must be unique in scope of the entire SC network. Service names are defined at application level and are stored in the SC configuration.
Example	serviceName=P01_RTXS_RPRWS1

7.26 sessionId

Name	sessionId
Description	Unique identification of the session
Validation	Known session
Comment	The sessionId is allocated by SC to which the client is connected when it sends the request CLN_CREATE_SESSION. The sessionID is universally unique because multiple SC may exist in the same network. The client must set the sessionId in each message during the session. For publishing services the sessionId is allocated by SC to the client when it sends the request SUBSCRIBE message. Subscription is internally treated as a session.
Example	sessionId=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d

7.27 sessionInfo

Name	sessionInfo
Description	Additional information passed by the client to the session server when the session starts.
Validation	Any printable character, length > 0 and < 256Byte
Comment	This is used to pass additional authentication or authorization data to the server.
Example	sessionInfo=SNBZHP - TradingClientGUI 10.2.7

8 Glossary

Glossary item

glossary text

Appendix A Message Header Matrix

		appErrorCode	appErrorText	bodyType	bodyLength	cacheId	cacheExpoirationDateTim	compression	ipAddressList	portNr	keepAliveInterval	keepAliveTimeout	localDateTime	messageId	mask	maxNodes	maxSessions	msgInfo	msgType	multithreaded	noData	rejectSession	scErrorCode	scErrorText	scVersion	serviceName	sessionId	sessionInfo
CONNECT	REQ							X			X	X	X						X						X			
	RES												X						X				E	E				
DISCONNECT	REQ																		X									
	RES																		X				E	E				
KEEPALIVE	REQ												X						X									
	RES												X						X				E	E				
INSPECT	REQ			X	X											X			X									
	RES			X	X				X										X				E	E				
ECHO_SC	REQ															X			X							X		
	RES								X										X				E	E				
CLN_CREATE_SESSION	REQ								X										X							X		X
	RES	O	O																X			X	E	E		I	X	
SRV_CREATE_SESSION	REQ								X										X							X	X	X
	RES	O	O																X			X				I	X	
CLN_DELETE_SESSION	REQ																		X							I	X	
	RES																		X				E	E		I	X	
SRV_DELETE_SESSION	REQ																		X							I	X	
	RES																		X							I	X	
SRV_ABORT_SESION	REQ																		X							I	X	
	RES																		X							I	X	
REGISTER_SERVICE	REQ									X							X		X	X						X		
	RES																		X				E	E		I		
DEREGISTER_SERVICE	REQ																		X							X		
	RES																		X				E	E		I		
CLN_ECHO	REQ/PRQ			X	X			X						X					X								X	
	RES/PRS			X	X			X						X					X				E	E			X	
SRV_ECHO	REQ/PRQ			X	X			X						X					X								X	
	RES/PRS			X	X			X						X					X								X	
CLN_DATA	REQ/PRQ			X	X	O	O	X						X				X	X								X	
	RES/PRS	O	O	X	X	O	O	X						X				X	X				E	E			X	
SRV_DATA	REQ/PRQ			X	X	O	O	X						X				X	X								X	
	RES/PRS	O	O					X						X				X	X								X	
HTTP	REQ			X	X			X											X									
	RES			X	X			X											X				E	E				
SUBSCRIBE	REQ														X				X							X		
	RES																		X				E	E		I	X	
UNSUBSCRIBE	REQ																		X							I	X	
	RES																		X				E	E		I	X	
CHANGE_SUBSCRIPTION	REQ														X				X							I	X	
	RES																		X				E	E		I	X	
RECEIVE_PUBLICATION	REQ													X				X	X							I	X	
	RES			X	X			X						X	X				X		O		E	E		I	X	
PUBLISH	REQ			X	X			X						X	X			X	X							X		
	RES													X					X				E	E		I		

Legend:
X => required attribute
O => optional attribute
I => informational only
E => EXC exception message only

Index

No index entries found.