SC

Service Connector

SC Message Protocol V1.0

SC_1_SCMP-V1.0_E (Version V2.21)

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 20 December 2010 09:51.

Identification

Project:	SC
Title:	Service Connector
Subtitle:	SC Message Protocol V1.0
Version:	V2.21
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 Signature ..... Walter Mörgeli
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
2.6.2010	V2.4	Jan Trnka	Multi-Session Server changed, ATTACH instead of CONNECT
15.6.2010	V2.5	Jan Trnka	Corrections after SIX workshop and review
30.6.2010	V2.6	Jan Trnka	Cleanup confusion detected by R. Frey Improve keepalive description Attr. *authSessionId* introduced Attr. *noDataInterval* introduced Headline format changed Chapter “Subscription monitoring” added Matrix reviewed Last header attr. also terminated with <LF> Definition of file services and http proxy.
2.7.2010	V2.7	Jan Trnka	Noi attribute changed, omi updated.
21.7.2010	V2.8	Jan Trnka	CLN_CREATE_SESSION,SRV_CREATE_SESSION,CLN_SUBSCRIBE,SRV_SUBSCRIBE,CLN_CHANGE_SUBSCRIPTION,SRV_CHANGE_SUBSCRIPTION may optionally have a body. SRV_ABORT_SESSION change

15.8.2010	V2.9	Jan Trnka	"=" or <LF> are not allowed within attribute names and/or values. Unknown header attributes will be forwarded. Large message are not possible for CLN_CREATE_SESSION or CLN_SUBSCRIBE. mid is never set in case of an error. Publishing of large messages possible from one server only. MANAGE message described. Attribute default values introduced. Chapter *Restrictions* moved to design document. Restructuring of the document. echoTimeout superseded by operationTimeout
23.8.2010	V2.10	Jan Trnka	Restrictions added again State and relationship diagrams serviceName limited to 32 chars. DATA message renamed to EXECUTE
6.9.2010	V2.11	Jan Trnka	Minor document restructuring Unknown header attributes will ignored, NOT be forwarded. Session monitoring changed. CLN_ECHO replaced by ECHO, SRV_ECHO eliminated, ECHO response sent by SC. State fallback on error added Possible error codes added
19.9.2010	V2.12	Jan Trnka	operationTimeout (oti) is now in milliseconds
22.9.2010	V2.13	Jan Trnka	REGISTER_SERVICE renamed to REGISTER_SERVER and analogous DEREGISTER_SERVICE to DEREGISTER_SERVER, MTY remains the same!
13.10.2010	V2.14	Jan Trnka	authenticatedSession (asi) removed cascadedMask (cma) introduced Errors in publishing service description and messages corrected. ipAddressList (ipl) required for INSPECT and MANAGE operationTimeout (oti) is in all client requests (sometimes optional) CacheId is unique per service
20.10.2010	V2.15	Jan Trnka	Exceeded oti in SRV_EXECUTE will cause SRV_ABORT_SESION.
25.10.2010	V2.16	Jan Trnka	Session handling in file services ipAddressList (ipl) has port numbers
02.11.2010	V2.17	Jan Trnka	*ced* sent only by server. Caching changed. Single session server is simplification of multi-session server. Optional sin (sessionInformation) in DELETE_SESSION and UNSUBSCRIBE Changes of V2.15 reverted. Exceeded *oti* in SRV_EXECUTE will NOT cause SRV_ABORT_SESION. MessageID (*mid*) replaced by messageSequenceNumber (*msn*) OriginalMessageID (*omi*) replaced by originalMessageSequenceNumber (*oms*) PAC header key introduced
29.11.2010	V2.18	Jan Trnka	Message body in CREATE_SESSION, SUBSCRIBE and CHANGE_SUBSCRIPTION reply. Invalid cache handling Port number removed from *ipl* (ipAddressList) *csi* (cascaded session id) introduced description of *msi* - messageSequenceNr changed and enhanced. *oms* – originalMessageSeqNr removed *mxc* is mandatory in REGISTER_SERVER *mxc* validation changed port number must be > 1
9.12.2010	V2.19	Jan Trnka	*set*/*sec* is also possible in server response
10.12.2010	V2.20	Jan Trnka	*rej* – reject flag can be also in SRV_SUBSCRIBE, SRV_CHANGE_SUBSCRIPTION *aec*/*aet* are possible whenever server responds with message body *set*/*sec* in server response does not make sense for regular server.
17.12.2010	V2.21	Jan Trnka	*oti* used only for pass-through or cascaded messages. CHECK_REGISTRATION message introduced *ams* actualMask attribute introduced for SRV_CHANGE_SUBSCRIPTION *crq, brq, brs, srq, srs* are not useful and have been deleted. Terminology "proxy" is confusing. Changed to "cascading".

Table of Contents

Tables

Figures

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 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
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
Wireshark	Open source product to capture and analyze network traffic

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 | phrase2 }	In syntax diagrams, indicates that multiple possibilities exists.
…	In syntax diagrams, indicates a repetition of the previous expression

Table 3 Typographical conventions

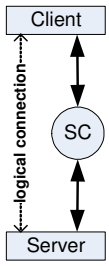
The terminology used in this document may be somewhat different from other sources. The chapter Glossary includes a list of often used terms with the explanation of their meaning in this document.

1.5 Outstanding Issues

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

1. Describe Cascaded Services and routing
2. Describe errors returned by SC

2 Communication Schema



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.

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.

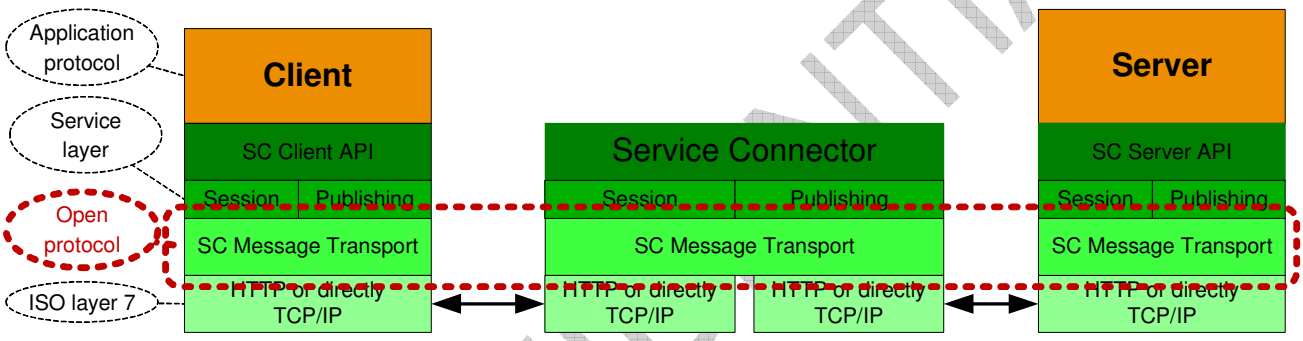


Figure 1 Communication Layers

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. The connection can either utilize HTTP protocol or direct TCP/IP communication. 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 ⇔ SC ⇔Server = Connection via cascaded SC.
  Multiple SC may be placed on the path between the client and service.

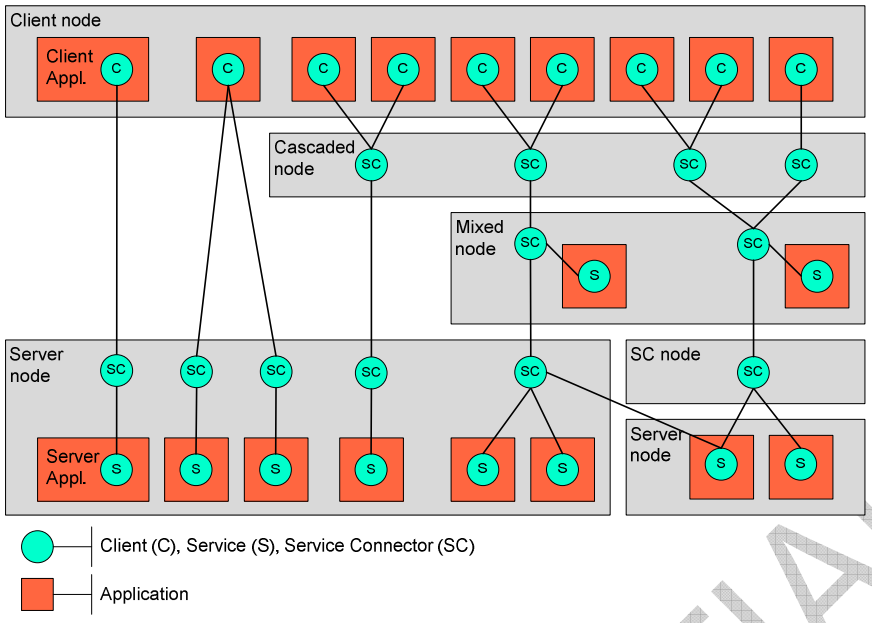


Figure 2 Connection Topology

Different connection topology types from left to right:

1. Client connected to one service
2. Client connected simultaneously to SCs and two services
3. Two clients connected to one service via cascaded service and cascaded SC
4. Two clients connected to three services via cascaded service on different server nodes
5. Complex configuration with three clients connected to three services on different server nodes via cascaded SCs and SC offloaded to its own node. One server can be registered to multiple services and different SCs. However this is possible for multisession servers only.

Limitation: The same service can be accessed by one SC only. When the same service should be used on different nodes, it must have a different name (e.g. node suffix).

Two different transport types can be individually configured for each network segment i.e. between the Client ⇔ SC, SC ⇔ SC or SC ⇔ Server.

a. HTTP
   Such connection may pass screening firewalls and is appropriate for communication within the customer organization e.g. Client ⇔ SC or SC ⇔ SC.
b. TCP/IP
   Such connection would not pass firewalls without explicit security rules. It is useful for connection within the same node e.g. SC ⇔ Server.

SC cascading is used for performance and/or for security reasons. It is transparent for the application.

2.2 Network Security

The SC does not implement any security feature. The environment where SC is used must provide all required authentication, authorization, encryption, tunneling etc. features. Message transport over https will not be supported.

The IP address of the client and the IP of the incoming TCP/IP traffic will be available in the message header and can be evaluated by the server. This can be used to authenticate and authorize the client when VPN tunnel is used. From the security viewpoint, 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.

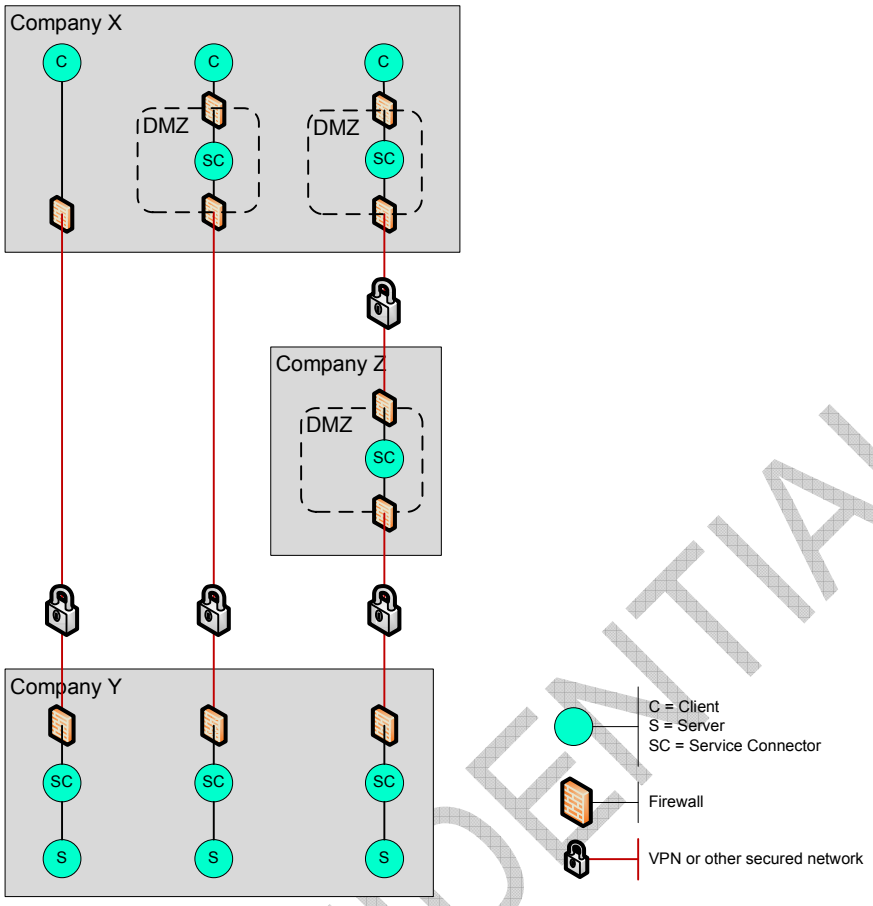


Figure 3 Network Security

Protocol

The message transport between each of the SC components (green bullet) may be configured as plain TCP/IP or HTTP. The connection is always 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 SC and the next downstream component is configured in the SC configuration. TCP/IP is strongly recommended between SCs through VPN tunnels as well as between SC and the Server for performance reasons. HTTP is recommended between the client and 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)

IP address list

Multiple SCs may be placed on the communication path between the client and the server. In order to allow comprehensive authorization all IP addresses are collected and made available to the server as a list. The list contains of IP addresses in the form 999.999.999.999. The order in the list has a dedicated meaning. The list has at least three entries.

1. IP of the client at Company X
2. Incoming IP received by SC = IP of the VPN Tunnel
3. IP of the SC at Company Y

Client connected via cascaded SC placed in company's X DMZ will have the list in the format:

1. IP of the client at Company X
2. Incoming IP received by SC in company's X DMZ.
3. IP of the SC in company's X DMZ
4. Incoming IP received by SC = IP of the VPN Tunnel.
5. IP of the SC at Company Y

If the pairs 1,2 or 3,4 have different values, then the corresponding network segment uses VPN or NAT. As long as there is only one SC behind the VPN, the second last address is always the tunnel IP used for authentication.

2.3 Network Connection Monitoring

The network connections between client ⇔ SC, SC ⇔ SC or SC ⇔ server is managed in a connection pool. New connections are created when necessary and deleted when they are idle for a long time.

Keepalive messages are sent in regular intervals on all idle connections initiated by a network peer. They are not sent on busy connections currently used for message exchange. E.g. while client is waiting for server response, the connection is busy and no keepalive message will be sent. On http connection statefull firewall inspection would reject two subsequent GET/POST request without previous response from the server in such case. Using of keepalive messages can be disabled by setting the *keepaliveInterval* = 0.

The only purpose of keepalive messages is to preserve the connection state in the firewall placed between the communicating peers and resets its internal timeout. Keepalive message is always initiated by the peer who has initiated the connection, because only outbound traffic may refresh the firewall timeout.

Client connections used for session or file services are mostly idle and will utilize the keepalive mechanism. Client connections used for publishing services are mostly busy. The mechanism to refresh the firewall timeout is built in the SCMP layer.

Server connections used for registering server are mostly idle and will utilize the keepalive mechanism. SC connections back to the server are also mostly idle and will utilize the keepalive mechanism.

When a connection breaks down because of an error while sending or receiving a keepalive message, or the response does not timely arrive a log entry is created, the connection is closed and deleted from the pool. **No error is signalled to the server application.**

Broken network connection between the server and SC on which the server has registered to the service is detected in SC and an immediate cleanup is done. This speeds up the detection of lost servers. The message has only a headline and no attributes and no body. The format is:

```
KRQ 0000000 00000 1.3
```

for request and:

```
KRS 0000000 00000 1.3
```

for response.

2.4 Load balancing

The SC does not provide any load balancing features. Established communication session will not be redirected to another server node during its life time.

2.5 Failover

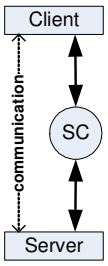
The SC does not provide any failover features. Aborted communication must be re-established by the communicating partners. The client application must find out a service which is alive.

2.6 Intrusion and Virus Protection

The entire network where SC is used is assumed to be safe and secure. No virus protection is embedded in SC. The customer may use screening firewall to protect the SC components. It is recommended to use SC within a DMZ.

SC is not designed to withstand network attacks like DOS or SYNC flood, or any other.

3 Service Model

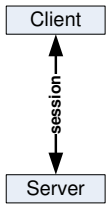


The SC supports message exchange between requesting application (client) and another application providing a service (server). The client and the server are the logical communication end-points. The SC never acts as a direct executor of a service. The client can communicate to multiple services at the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

Multiple clients may request the same service at the same time. Parallel execution of the service is implemented by starting multiple server instances (multiprocessing), by starting a multisession server (multithreading) or by a combination of both. The server process can provide one or multiple service and decides how many sessions it can serve. Serving multiple services within one server process is possible for multisession servers only.

All services are independent on each other. Server application may request another service and so play the client role.

3.1 Session Services



This is a statefull Request/Response service initiated by the client. For session services the client and the server exchange messages in context of a logical session through the SC.

The client controls the creation and deletion of the session and so allocation of the server. The client can have multiple concurrent sessions to the same or to different services. **Because the server execution is always synchronous, from the client perspective only one pending request per session is possible at any time.**

When a session is created SC will choose a free server and pass this and all subsequent requests within this session to the same server. Session information is passed in each message as a part of the message header. Session is deleted intentionally by the client or aborted by SC upon an error.

3.1.1 Synchronous message delivery

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

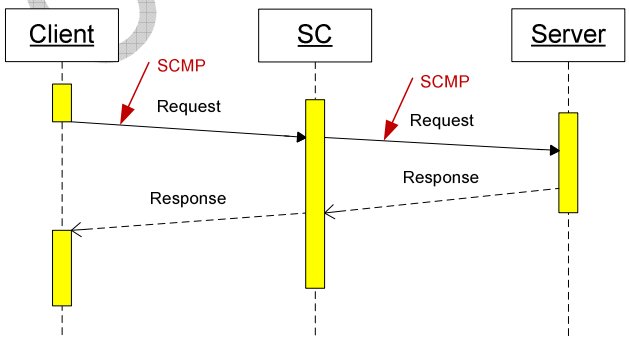


Figure 4 Synchronous Request/Response

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.

The message flow looks as follows:

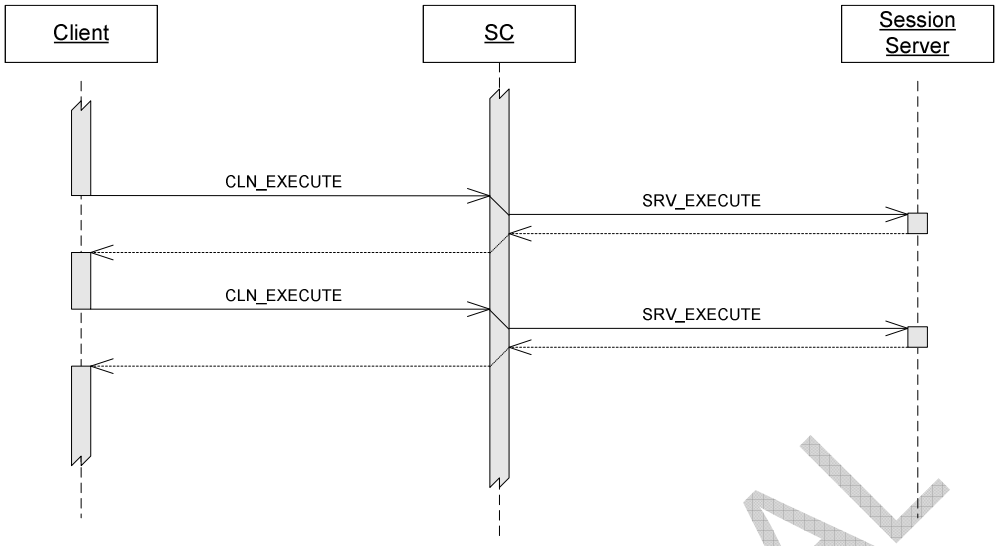


Figure 5 Synchronous Message Exchange.

3.1.2 Asynchronous message delivery

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

The client must declare a notification method that is invoked when the response message arrives.

Note! **Only one pending request per session is allowed at any time. Subsequent client request must be hold in by the client API until the response for the previous message arrives.**

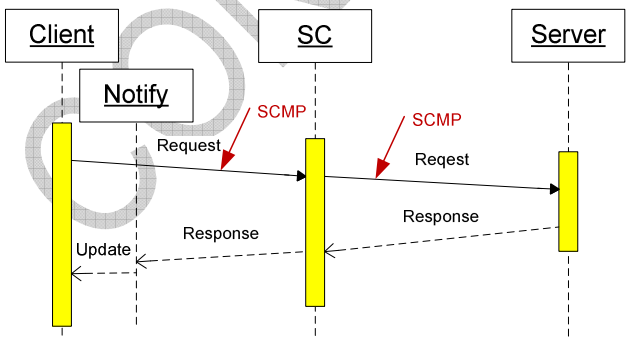


Figure 6 Asynchronous Request/Response

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. The message flow looks as follows:

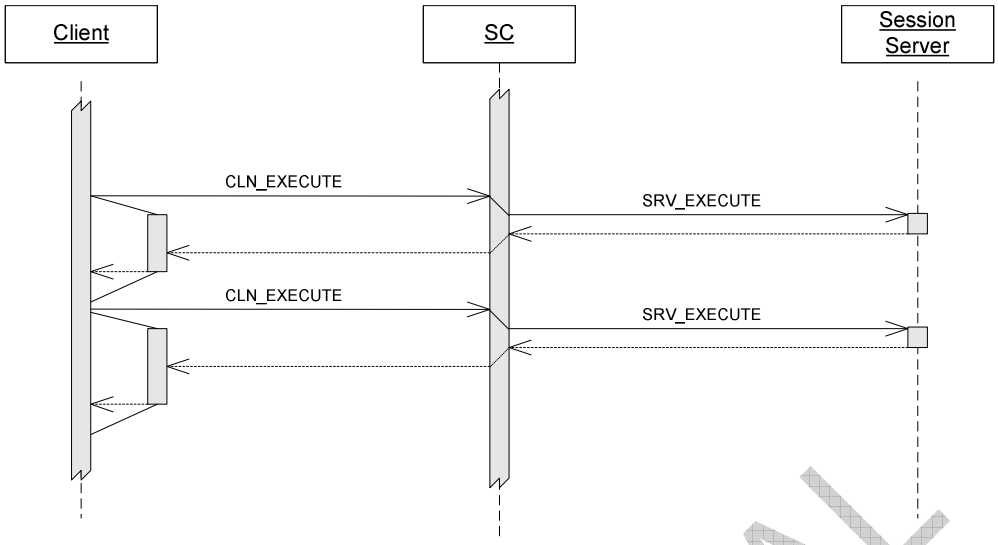


Figure 7 Asynchronous Message Exchange.

3.1.3 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 PAC (part acknowledge). For large request the message flow looks as follows:

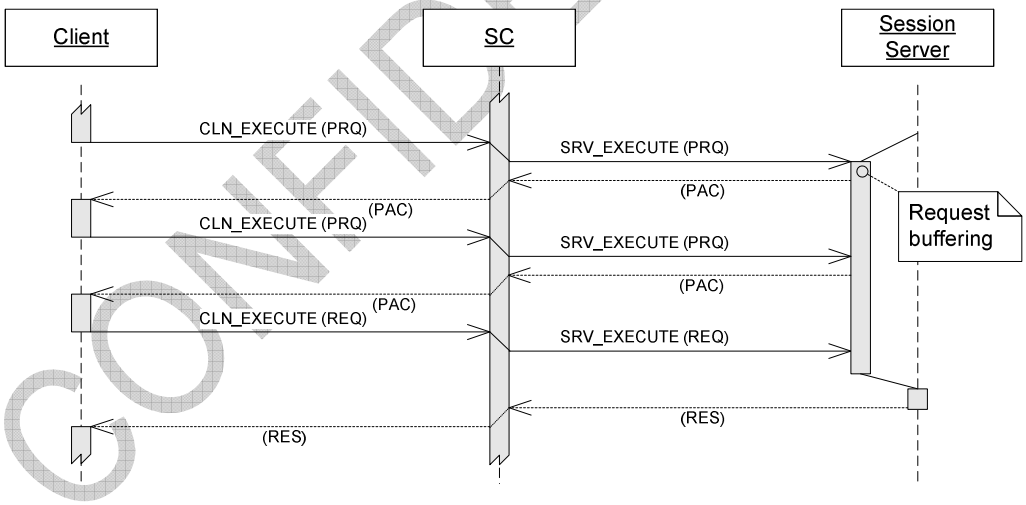


Figure 8 Large Request.

1. The client sends the first the message part (PRQ) to the server.
2. The message part is transported to the server and buffered here
3. The server answers with PAC in order to receive the next part.
4. When the final message part (REQ) arrives, the complete message is made available to the server.
5. The sever will process the message and send back the response message (RES).

For large response the message flow looks as follows:

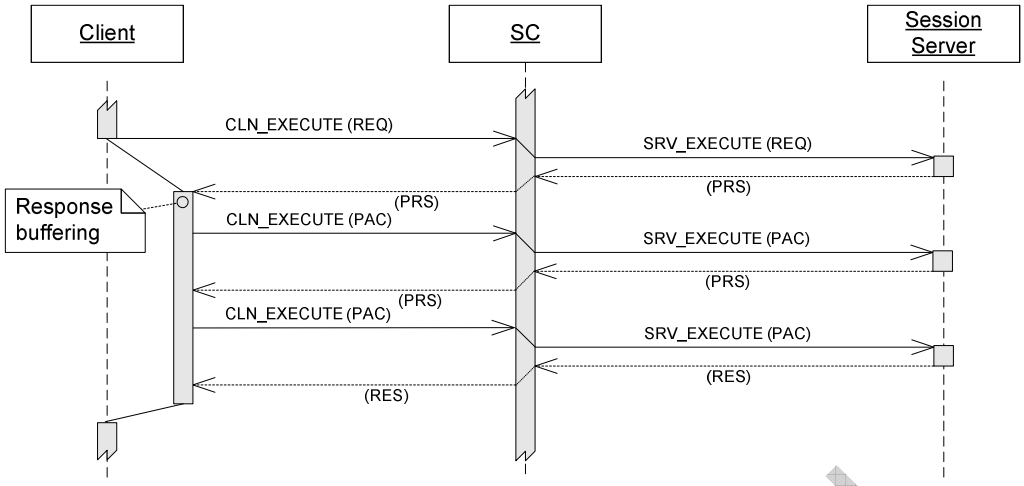


Figure 9 Large Response.

1. The client sends a request message (REQ) to the server.
2. The server receives the request messages and start producing the response. It sends part message (PRS) to the client and keeps the context.
3. The client receives the partial response (PRS) and starts buffering of the data. It sends part acknowledge (PAC) in order to receive more results from the server.
4. At the end the server sends the last message part as a regular response (RES)
5. When the final response arrives, the message is made available to the client.

Combination of large request and large response is also possible.

Message traffic with large request followed by a large response looks like (simplified):

```
REQ ..   msn=3
RES ..   msn=64
...
PRQ ..   msn=4
PAC ..   msn=65
PRQ ..   msn=5
PAC ..   msn=66
PRQ ..   msn=6
PAC ..   msn=67
REQ ..   msn=7
PRS ..   msn=68
PAC ..   msn=8
PRS ..   msn=69
PAC ..   msn=9
RES ..   msn=70
...
REQ ..   msn=10
RES ..   msn=71
```

3.1.4 Multi Session Server

Multi session server serves multiple sessions at the same time. SC uses a dynamic connection pool to exchange messages with the server. The server registers itself for one or more services and defines reasonably high *maxSessions* > 1 and *maxConnections* > 1. Depending on the *immediateConnect* the connection pool to the server is created immediately (*immediateConnect* = true) or when the first session is started (*immediateConnect* = false). The same connection is reused unless it is busy. Connections are closed when they are idle for a long time i.e. several subsequent keep alive messages have been sent. One connection is kept open for all times.

When necessary, new connections are created on the fly. When the server deregisters, the connection pool is deleted and all existing connections are closed.

The attribute *maxConnections* specified in server registration tells the SC how many connections should be created and so limits the pool size. This allows network resource optimization when one server serves many rarely used sessions. When all connections in the pool are exhausted, SC will wait for a free connection and finally return an error to the client.

The following schema shows message flow with multi session server with *immediateConnect* = true.:

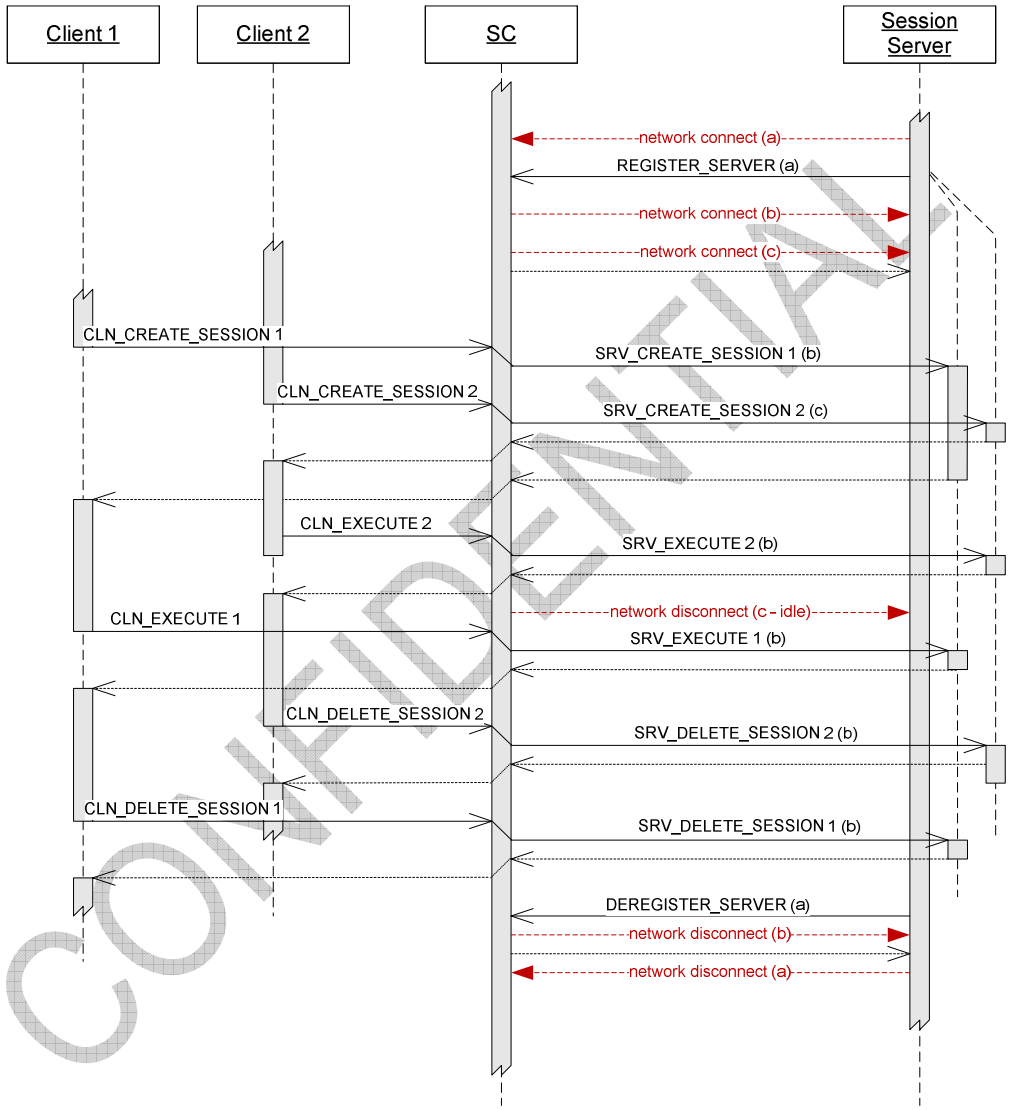


Figure 10 Multi Session server (immediateConnect = true).

Client

1. The client establishes a network connection to SC and starts communication with the ATTACH message.
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 SRV_CREATE_SESSION message. It also creates a unique sessionId for this session. If there is no free server instance available to this service, the SC will retry several times and finally returns the error “no free server available”.
4. Then the client can exchange messages with the server via CLN_EXECUTE messages.

5. When the session with this service is no longer needed, the client will send the CLN_DELETE_SESSION message. The server is notified with the SRV_DELETE_SESSION message.
6. Before the client terminates, it should send DETACH message and then terminate the network connection to SC.

When session is abnormally terminated, the client will receive an error message in the response to next request (regular message or echo). The reasons for this can be:

- Server sends DEREGISTER_SERVER
- Unexpected server exit
- Underlying communication error (e.g. unreachable node)

Note **The client may have multiple SCs connected at the same time. Per connected SC it may have multiple active sessions (even for the same service) at the same time. From the client perspective only one request may be pending at any time for a session.**

Server

1. The server establishes a network connection (a) to SC and starts communication and registers itself as an instance of a service with the REGISTER_SERVER message. At that time it should have a listener that will accept the connection (b) initiated by the SC to this server.
2. The SC registers the server instance and will create network connection (b) and (C) back to it. Keepalive messages will also be sent on these connections, depending on the *keepaliveInterval* set in REGISTER_SERVER.
3. 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.
4. The server receives messages from the client as SRV_EXECUTE and sends them back after execution of the service that it implements. The server receives requests on any of the network connections created by SC. It may use any available technique (e.g. multithreading), but must ensure that all requests are processed in parallel.
5. When the client deletes the session, the appropriate server is notified with the SRV_DELETE_SESSION message.
6. When the server is no longer needed it sends the message DEREGISTER_SERVER. From this point the SC will not allocate it for a session and terminates the connection (b) and (c) to it. Then it can terminate the network connection (a) to SC.

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

- Unexpected client exit and the subsequent expiration of the echo timeout
- Underlying communication error (e.g. unreachable node)

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

Note **The server must be active before the client will create a session. Otherwise the client will receive an error message.**

With flag *immediateConnect* = false connections are created by SC when the session is started. The same connection is reused unless it is busy. Connections are closed when they are idle for long time. The last active connection is closed when the session is deleted or aborted.

The following schema shows message flow with multi session server with *immediateConnect* = false:

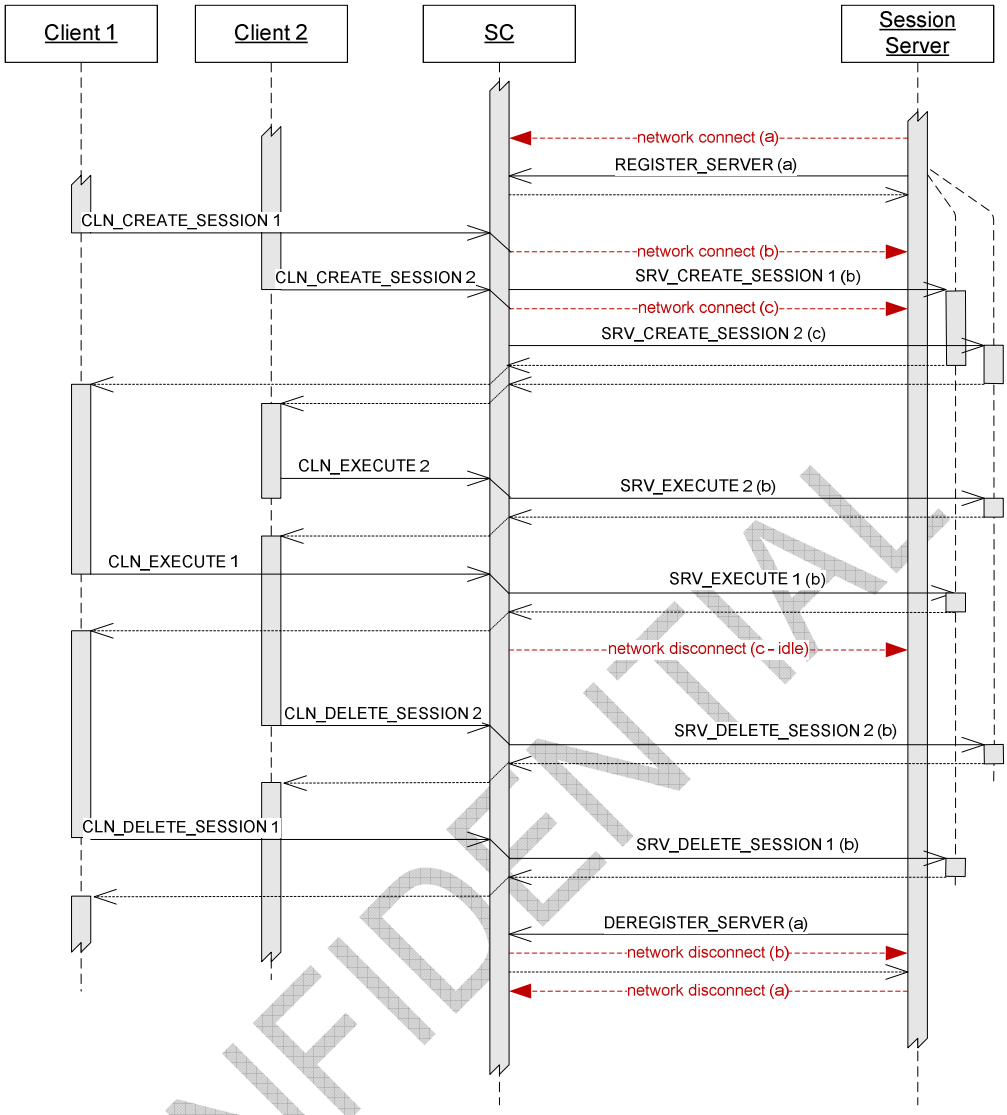


Figure 11 Multi-Connection server (immediateConnect = false).

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

Note!

The request execution on the server may take long time. In case the operation timeout (set by the client) expires, the SC will close the connection to the server on which the SRV_EXECUTE request was sent and send CLN_EXECUTE error message back to the client. SC will NOT abort the session! The client should react properly in this situation and close the session. In case it will send a subsequent CLN_EXECUTE to the server, the previous execution may still be pending!

The message flow looks as follows:

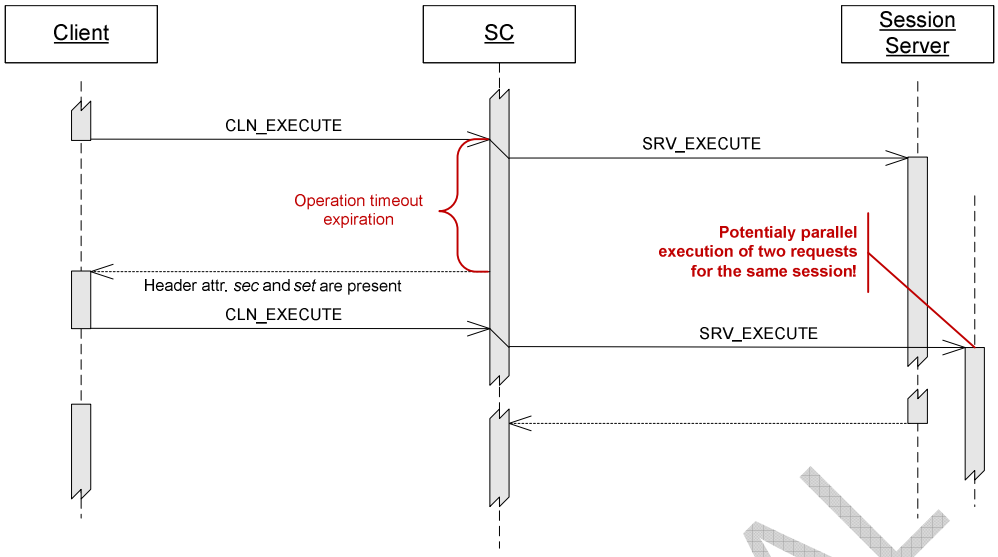


Figure 12 Execution exceeding operation timeout.

3.1.5 Single Session Server

Single-session behaves like a multi session server but has only one session and one connection. It registers itself for one service and may serve only one session at the same time. It must define *maxSessions = 1, maxConnections = 1* and may define *immediateConnect* = true or false. The required parallelism is achieved by starting multiple instances of the same server. The SC keeps track of the registered servers and allocates / de-allocates sessions to them. The following schema shows message flow with single session server.

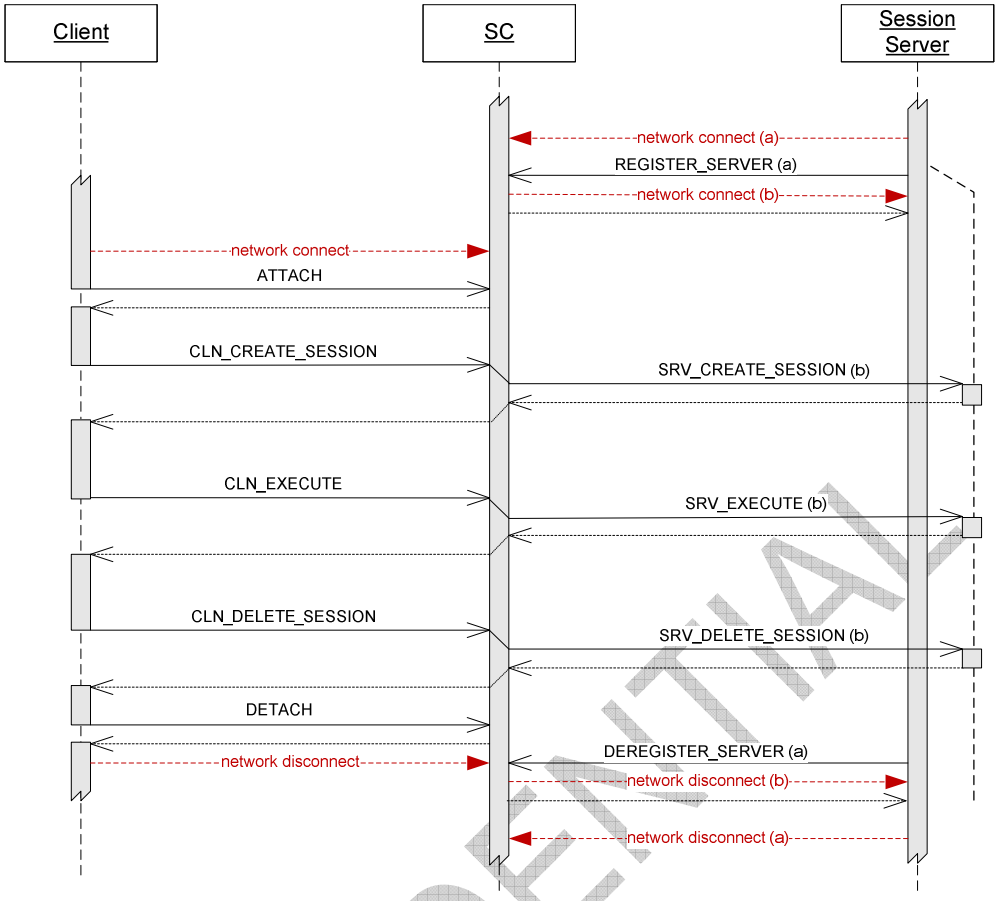


Figure 13 Single Session Server

Note! **The request execution on the server may take long time. In case the operation timeout (set by the client) expires, the SC will close the connection to the server on which the SRV_EXECUTE request was sent and send CLN_EXECUTE error message back to the client. SC will NOT abort the session! The client should react properly in this situation and close the session. In case it will send a subsequent CLN_EXECUTE to the server, the previous execution may still be pending!**

3.1.6 Application Server (Tomcat)

SCMP messaging with an application server (e.g. Tomcat) works exactly like a multi-connection server but utilizes the HTTP protocol between server and SC. The server must register itself for all services it will serve. It must define reasonable high *maxSessions* and *immediateConnect* = false. It is recommended to define also a reasonable high *maxConnections* attribute.

3.1.7 Session Monitoring

Session monitoring is based on ECHO messages, exchanged periodically for each active session between the client and the SC and on the network connection breakdown detection between SC and the server. Session monitoring has nothing in common with the keepalive messages exchanged on active network connections.

The client sends ECHO message in predefined intervals for each session service when no other request is in progress. The *echoInterval* is extended by *echoIntervalMuliplier* in order to allow time for network transmission. The client receives a response message back from the SC. The SC monitors the *echoInterval* in which ECHO messages arrive and aborts the session when the

interval is exceeded. SRV_ABORT_SESSION message is sent to the allocated server in such case.

The SC monitors the network connection to the server, which is assumed to be reliable. When the network connection on which it registers is broken or the SC cannot send a message to the server, server is assumed to be dead and clean up will be performed. This will delete all sessions of the server. When the client send a subsequent ECHO or CLN_EXEUTE message it will be notified about the deleted session.

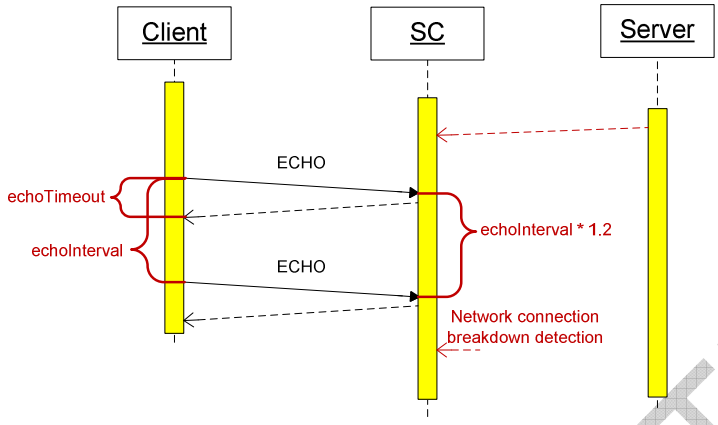


Figure 14 Session Monitoring

Based on the *oparationTimeout* SC monitors the response time of the server for request message sent from client to the server. When this timeout expires for SRV_EXECUTE, SC will notify the client but will NOT clean up the session. It may still be valid, just the server may be slow. When the timeout expires for SRV_CREATE_SESSION, the session is not created. When the operation timeout expires for SRV_DELETE_SESSION, or SRV_ABORT_SESSION the session is deleted.
The session maintained in SC will be deleted only due to these reasons:

1. The client deliberately closes the session by sending CLN_DELETE_SESSION
2. The SC will not receive ECHO from the client, for longer than the *echoInterval* * *echoIntervalMultiplier*.
3. The server crashes and loses connection to the SC

Client Abort

When client aborts its activity abruptly while a request is pending, then SC will not be able to deliver the response. The SC will be clean up the session and the server will be notified with SRV_ABORT_SESSION message.

When idling client aborts its activity abruptly, then SC will not detect the session breakdown before the *echoInterval* is exceeded. During this time new sessions can be created by the restarted client. Old sessions may still exist in SC some time until they will expire. Consequently the server may be still allocated and may receive SRV_ABORT_SESSION message for the old session after the SRV_CREATE_SESSION for the new session.

When idling client loses all connection to SC due to short temporary network unavailability, then the connection may be re-established without loss of the sessions. This is true only if the breakdown is shorter than the *echoInterval*, and no other message is sent to SC in the mean time.

Server Abort

When server aborts its activity abruptly (without a neat deregister), then SC will detect the connection breakdown immediately (on the connection on which the server registers) and will clean up all its sessions. Subsequent client messages for the deleted sessions may cause the error response “Invalid Session ID”. After server restart, new connections to and from SC will be established. The server can then be allocated to new sessions.

SC Abort

When SC crashes, client will detect the connection breakdown when a new message is sent. It should perform cleanup followed by a reconnect. The server will detect the connection SC

breakdown on the connection on which it listens and should perform a cleanup followed by reconnect and service registration.

After SC restart, new connections will be established. The servers must register before the client starts new session. Otherwise the client gets the errors “no free server available” or “server not available”.

3.1.8 Server Allocation

When a session is created SC will choose a free server and allocate it to the session. The SC uses a modified round-robin algorithm to find a free server.
Example:

When 3 server instances A1, A2, A3 have been started and each instance can serve 10 sessions named Ax-y, then the sessions will be allocated in this sequence:
A1-1, A2-1, A3-1, A1-2, A2-2, A3-2, A1-3, A2-3, A3-3, … A1-10, A2-10, A3-10

The algorithm looks for next free server in the sequence. Because the session can be deleted in any order, holes in the sequence many emerge and the server load will be not homogenous after a time.

3.1.9 Message Caching

Response messages for session services sent by server to clients can be cached within SC. Clients may fetch the message from the SC cache without a server interaction. This feature massively reduces the server load.

In order to insert the message into the cache the server must designate the response message with the attributes *cacheId* and *cacheExpirationDateTime* like the following example:

cid=CBCD_SECURITY_MARKET
ced=2010-08-16T20:00:00.000+0000

SC passes the message to the requested client and stores it also in a cache under the ID defined by *cacheId.* The *cacheId* is unique per service. If an older message exists in the cache, it will be overwritten. The *cacheExpirationDateTime* controls the cache validation. The server defines the expiration of the cache by using the *cacheExpirationDateTime* attribute. SC looks periodically for cached messages which has already expired and deletes them from the cache. In order to fetch a message from the cache the client must use the same *cacheId*. The request message must contain these attributes like the following example:

cid=CBCD_SECURITY_MARKET

The following picture shows the impact of the *cacheExpirationDateTime* in the messages.

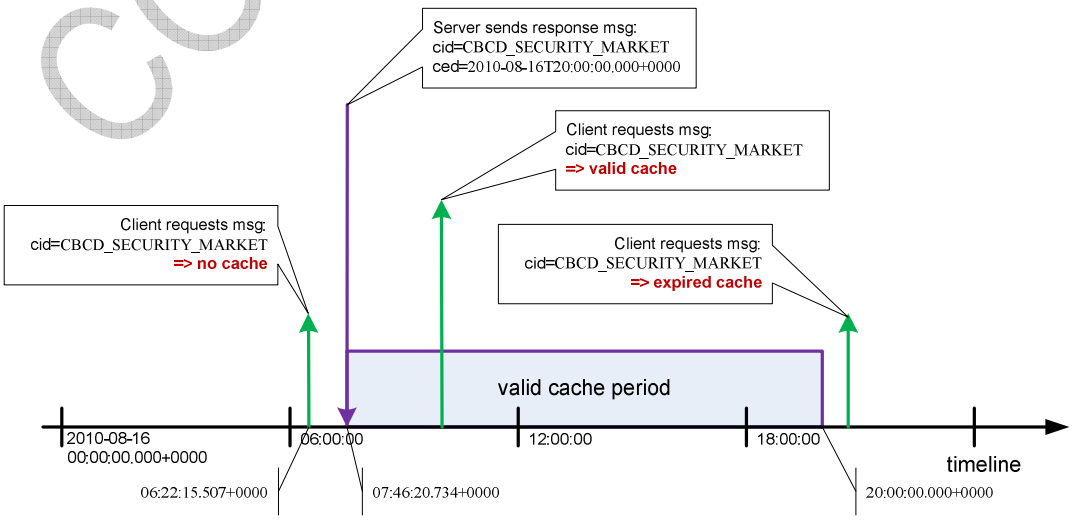


Figure 15 Caching rules

This mechanism allows using the cache in different time zones. The server can terminate the particular message validity by an absolute date and time.

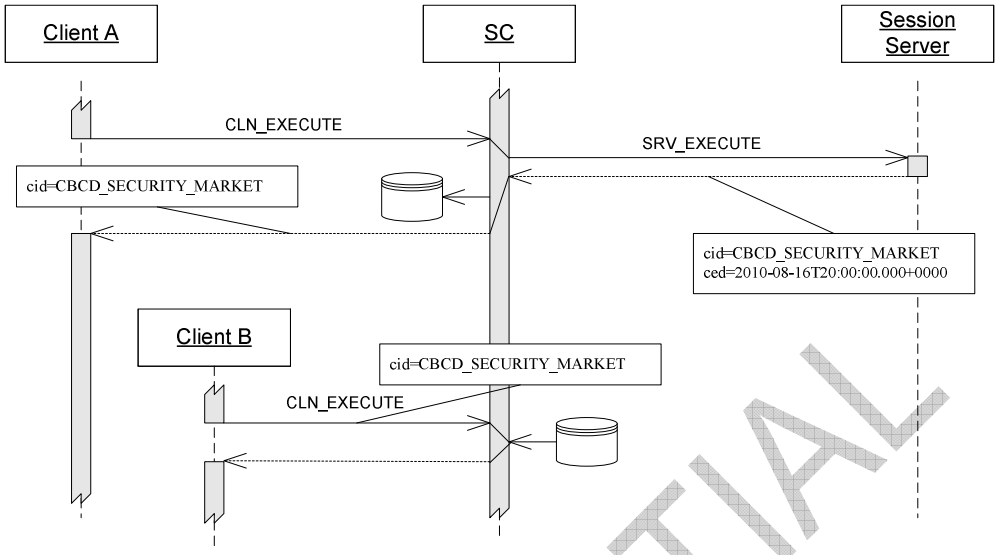


Figure 16 Message Caching

If the given *cacheId* is not found in the cache or does not exist in the request message, then SC will pass the request to the server. If client doesn't want a cached message he should omit the *cacheId* in the request.

Client and server must agree on *cacheId* which must be unique across all services. In cascaded SC configurations cache is created in all SC nodes along the path from client to the server.

Note: **While SC is building up the cache, client request may be rejected because the cache is temporarily invalid. The client must retry to take advantage of the cached message**

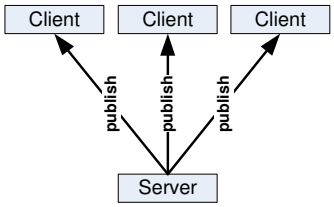
Caching of large messages is supported. For the transport of large cached message to the client SC uses the original parts of the message. The *cacheId* is modified by SC and must be returned unchanged in the client PAC request. This is how SC keeps track on parts which must be delivered next.

3.1.10 Message Sequencing

Message sequence number is used to identify the message during the transport. The message path from client to the server and the path from server to the client are independent and thus have independent message sequence numbers. The number is generated by the client or by the server and are passed unchanged through SC to the counterparty. SC does not validate it. ECHO messages have no message sequence numbers.

For regular operation the number is steadily increasing. This is true for the message path from client to the server. For the path from server to the client message can be fetched from cache instead of a real server reply. Such message will have its original message sequence number. The client must not validate the message sequence number in such case.

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 mask. When the message mask 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.

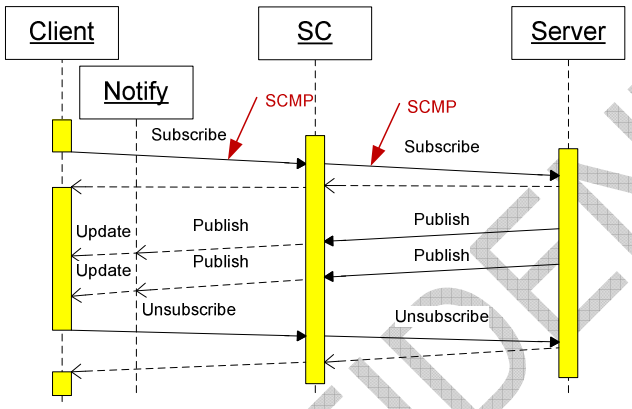


Figure 17 Subscribe / Publish

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

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.

The following schema shows message flow with a publishing server.

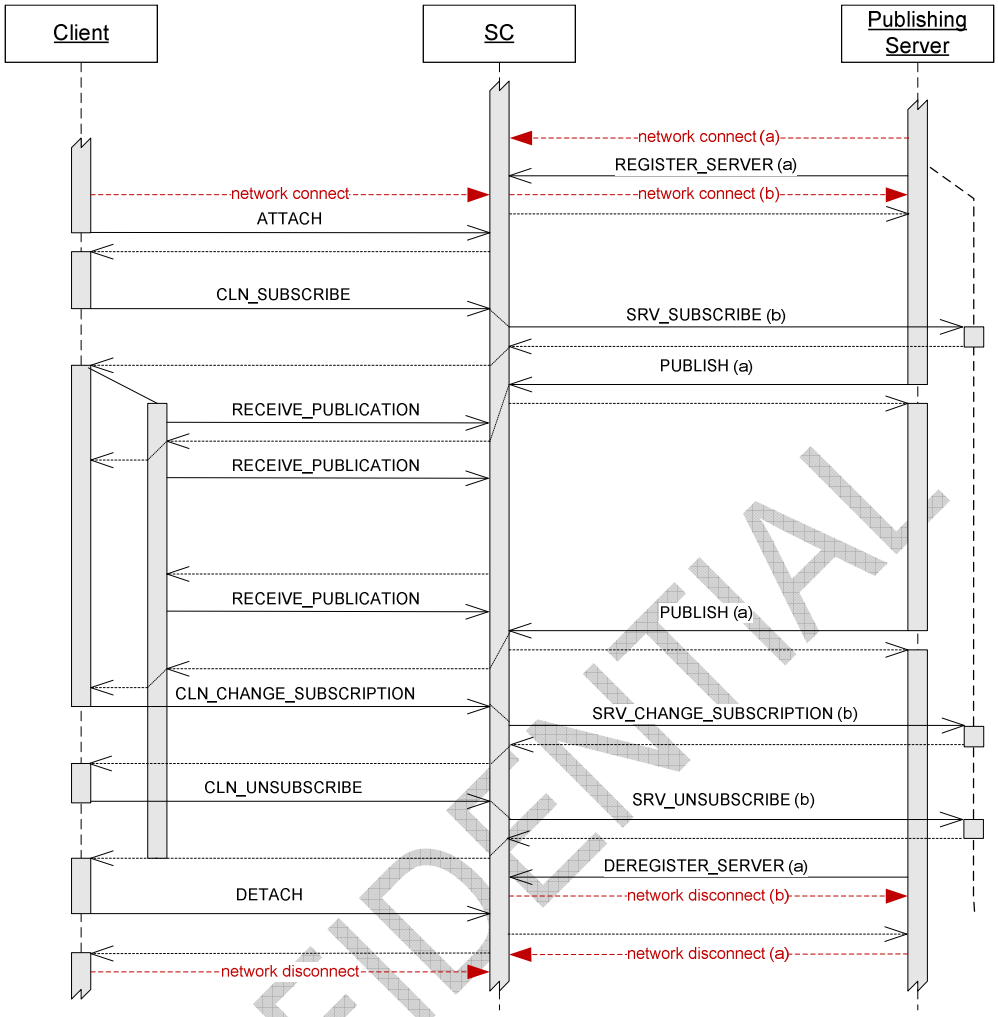


Figure 18 Publishing Service

Client

7. The client establishes a network connection to SC and starts communication with the ATTACH message. Keepalive messages can be sent on this connection.
8. Then it subscribes to a service with the CLN_SUBSCRIBE message and starts a listener that will receive the incoming messages.
9. The SC remembers the subscription and also creates a unique sessionId for it. It also notifies the server registered to this service with the SRV_SUBSCRIBE message. If there is no server, the client gets an error.
10. 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.
11. When no message is published within a period of time (defined by *noDataInterval*) then SC sends an empty message to the client and this initiates the next receipt with the RECEIVE_PUBLICATION message.
12. The client can change the publication mask with the CLN_CHANGE_SUBSCRIPTION message or terminate the subscription with CLN_UNSUBSCRIBE message. In both cases the server is notified with the SRV_CHANGE_SUBSCRIPTION or SRV_UNSUBSCRIBE message.
13. Before the client terminates, it should send DETACH message and then terminate the network connection to SC.

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

- Unexpected client exit
- Underlying communication error (e.g. unreachable node)

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. The server establishes a network connection to SC and starts communication and registers itself as an instance of a service with the REGISTER_SERVER message. Keepalive messages can be sent on this connection.
2. The SC registers the server instance and will create network connection (b) back to it. Keepalive messages will also be sent on this connection, depending on the *keepaliveInterval* set in REGISTER_SERVER.
3. When a client subscribes or changes the subscription the server is notified with the SRV_SUBSCRIBE or SRV_CHANGE_SUBSCRIPTION message. The message contains the subscription mask and additional information about the client. The server can accept or reject the subscription or its change. Like for a session service the server instance is allocated for the subscription until the subscription is deleted. The same sever instance will get all subscription actions of one client. The sessionId identifies uniquely each subscription.
4. Now the server can publish messages to the service. SC immediately responds when the PUBLISH message has been queued. The server does not wait for message delivery to the clients. The published message must have a mask designating its contents.
5. The SC compares the mask 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 or if any at all.
6. Messages are delivered in the order of their publishing. E.g. in order SC receives them. For this reason they are queued within SC.
7. When the client unsubscribes, the allocated server is notified with the SRV_UNSUBSCRIBE message.
8. When the server is no longer needed it sends the message DEREGISTER_SERVER. Then it can terminate the network connection (a) to SC.

When the client terminates abnormally, the server is notified with the SRV_UNSUBSCRIBE message.
The reasons for this can be:

- Unexpected client exit
- Underlying communication error (e.g. unreachable node)

Note **The server must be active before the client will subscribe. Otherwise the client will receive an error message.**

When the server terminates timely before the client, the client will receive an error in the RECEIVE_PUBLICATION response.

Multiple publishing servers may register to the same service. Also like a session server multiple subscriptions per server are allowed. SC chooses one server instance which is not busy when a notification must be processed. Unlike a session server the chosen server instance is allocated only for the time of the notification processing (one request).

3.2.1 Large Messages

Publishing of large messages works on server like sending a large response and on client like receiving a large response.

The server publishes the messages parts to the service. SC immediately responds when the PUBLISH message part has been queued. All message parts must have the same mask. The server does not wait for message delivery to the clients. The SC delivers the message parts to the subscribed clients in the right sequence like any other message. The message parts are buffered on the client until the final part arrives. Then it is passed to the application.

Note **Publishing of large messages is <u>not</u> possible from multiple servers to the same service. Only one server may publish a large message to a service at the same time.**

The queuing of messages in SC and buffering of message parts in the client is independent. The message flow looks as follows:

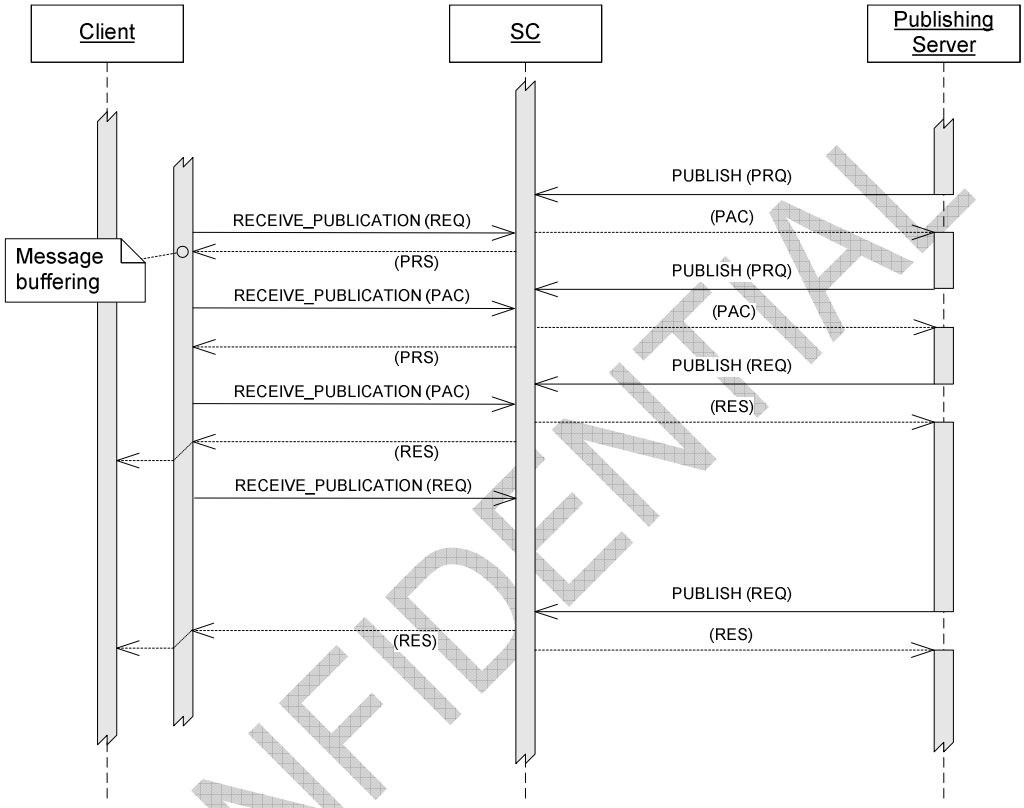


Figure 19 Large Published Message

3.2.2 Subscription Monitoring

Subscription monitoring is based on the RECEIVE_PUBLICATION messages, exchanged periodically between the client and the SC for each active subscription. Subscription monitoring is independent on the keepalive messages exchanged on active network connections.

The client sends RECEIVE_PUBLICATION message in order to get the published data. The SC sends either a response message with data or an empty response with the *noData* flag when no data is available and the *noDataInterval* is exceeded. The client starts then immediately a new RECEIVE_PUBLICATION request.

Whenever SC cleans up a subscription it sends a SRV_UNSUBSCRIBE or SRV_CHANGE_SUBSCRIPTION message to the publishing server.

The SC monitors the network connection to the server, which is assumed to be reliable. When the network connection on which it registers is broken or the SC cannot send a message to the server, server is assumed to be dead and clean up will be performed.

Note: Server exit has no an immediate impact on the client subscription! The client may still receive messages from other servers. If the server allocated to the subscription is no longer alive, the client will get error on subsequent CHANGE_SUBSCRIPTION or UNSUBSCRIBE operations.

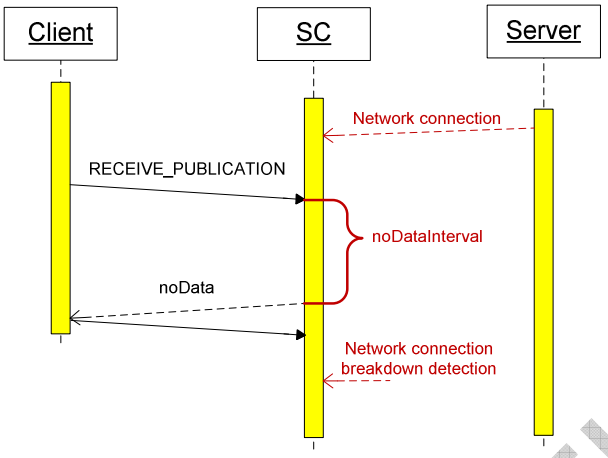


Figure 20 Subscription Monitoring

Client Abort Client subscribed to a service has always a pending RECEIVE_PUBLICATION request. When it aborts its activity abruptly the delivery of the response message will fail and the SC will clean up its subscription.

Server Abort When a publishing server aborts its activity abruptly (without a neat deregister), then SC will detect the connection breakdown immediately (on the connection on which the server registers) and will clean up all its registration. The subscribed clients are informed in the next response. New server should be started immediately in order to handle incoming client subscriptions. Otherwise the subscription will fail.

SC Abort When SC crashes, client will detect the connection breakdown immediately, because it has a pending request. It should perform cleanup followed by a reconnect. The server will detect the connection breakdown on the connection on which it listens and should perform a cleanup followed by reconnect and service registration.

After SC restart, new subscriptions will be established. The servers must register before clients will subscribe. Otherwise the client gets the errors “no free server available” or “server not available”.

3.2.3 Server Allocation

When a subscription is created SC will choose a free server and pass the request to it. The SC uses a modified round-robin algorithm.
Example:

> When 3 server instances A1, A2, A3 have been started and each instance can serve 10 sessions named Ax-y, then the subscriptions will be allocated in this sequence:
> A1-1, A2-1, A3-1, A1-2, A2-2, A3-2, A1-3, A2-3, A3-3, … A1-10, A2-10, A3-10

The algorithm looks for next free server in the sequence. Because the subscription can be deleted in any order, holes in the sequence many emerge and the server load will be not homogenous after a time.

3.2.4 Message Fan-Out

Subscription for a publishing service is kept in the SC to which the client is attached. In cascaded SC configurations this is the nearest SC node. The client subscription is combined

with subscriptions of all other clients and passed to the next SC node. I.e. the SC behaves itself like a subscribing client.

When a SC receives a published message it will pass it to all clients with matching subscription. So it does also for a cascaded SC that subscribes on behalf of its clients. In this way only one message is sent to the cascaded SC where it is distributed to all clients according to their mask. This feature massively reduced outbound network traffic to the clients.

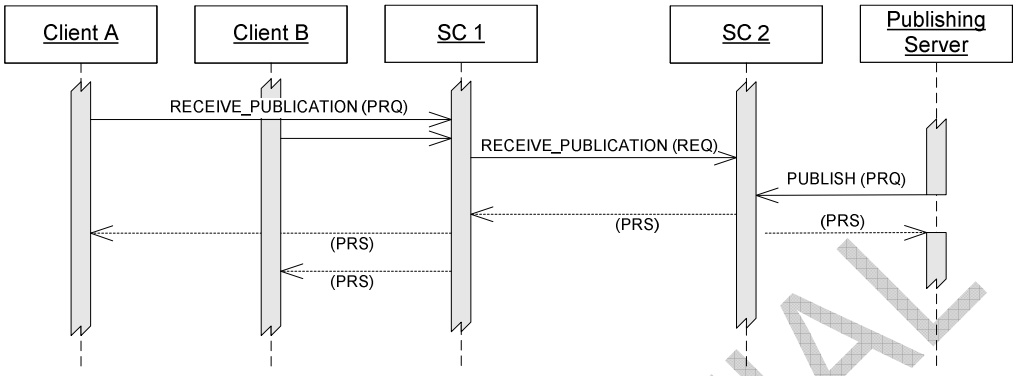


Figure 21 Message Fan-Out

Because the SC1 subscribes on behalf of its clients, it propagates new subscriptions or deletion of a subscription as subscription change on the SC2.

3.2.5 Message Sequencing

Message sequence number is used to identify the message during the transport. The message path from client to the server and the path from server to the client are independent and thus have independent message sequence numbers. The number is generated by the client or by the server and are passed unchanged through SC to the counterparty. SC does not validate it. Reply message to RECEIVE_PUBLICATION with *nod* (no data) flag set and reply to PUBLISH message have no message sequence numbers. Regular reply message to RECEIVE_PUBLICATION has the message sequence number created by the server and passed in PUBLISH request message.

For regular operation the number is steadily increasing. The message sequence numbers generated by the client will never reach the server. The message sequence numbered generated by the server will be passed to the client. When multiple servers are publishing messages to the same service, the message sequence number received by the client may have duplicates.

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.

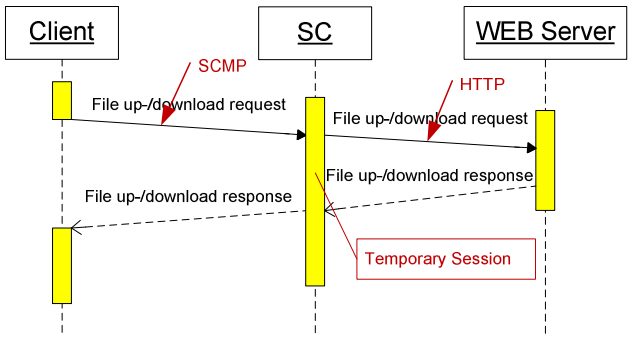


Figure 22 File service

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.

The client may initiate only one file operation to a service at the same time. All operations are synchronous. The SC configuration maps a service name to a directory on the web server (virtual host). The client initiates the upload or download of the file for a service. The client may upload or download file in different server locations. Directory structures (tree) are not supported. No security checks are done. The upload must be enabled and configured in the web server by installing a PHP script. This script is provided in the SC kit.

List of files is provided as an array of strings (file names). Within the message body the file list is represented as a delimited string in format:

```
{name<LF>}…
```

The Web Server is not registered to the service. It is configured to allow file upload, download and directory browsing for the specific directories (virtual hosts). The message flow looks as follows:

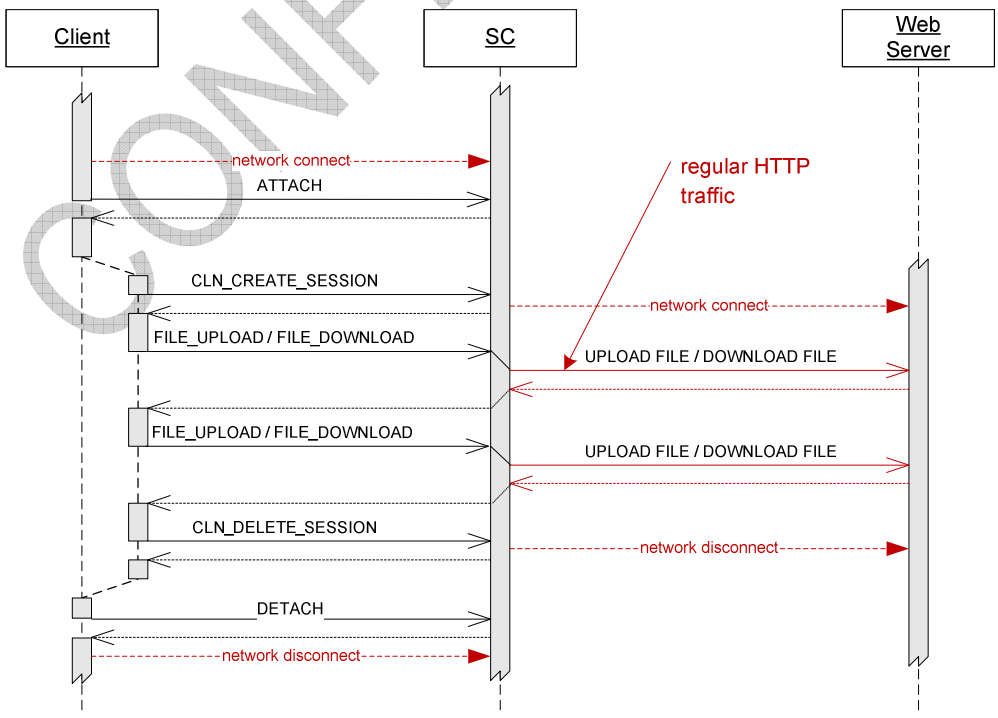


Figure 23 File upload and download.

The file is transferred in parts, analogically to large message. In order to keep track of the message parts and a temporary session is created. This session is automatically deleted when the file is completely uploaded or downloaded. The client API should hide the session creation and deletion. The file operation should be presented as synchronous method. On the web server the PHP script handles the message flow. It also does format the FILE_LIST message to the appropriate format.

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. Cascaded SCs on the path are transparent for the client.

3.4 HTTP Proxy Services

The SC supports redirecting of regular HTTP traffic to another server. It is acting like normal HTTP proxy without caching.

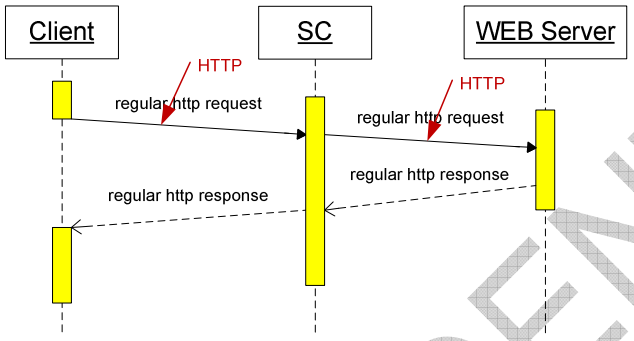


Figure 24 HTTP proxy service

HTTP traffic through the SC is possible without a service and session context. SC receives the http requests on a dedicated port and passes all HTTP traffic to the configured server. It acts like a HTTP proxy. The requested url must be resolved (dns) by the underlying network infrastructure. SC does not provide any name resolution. The message flow looks as follows:

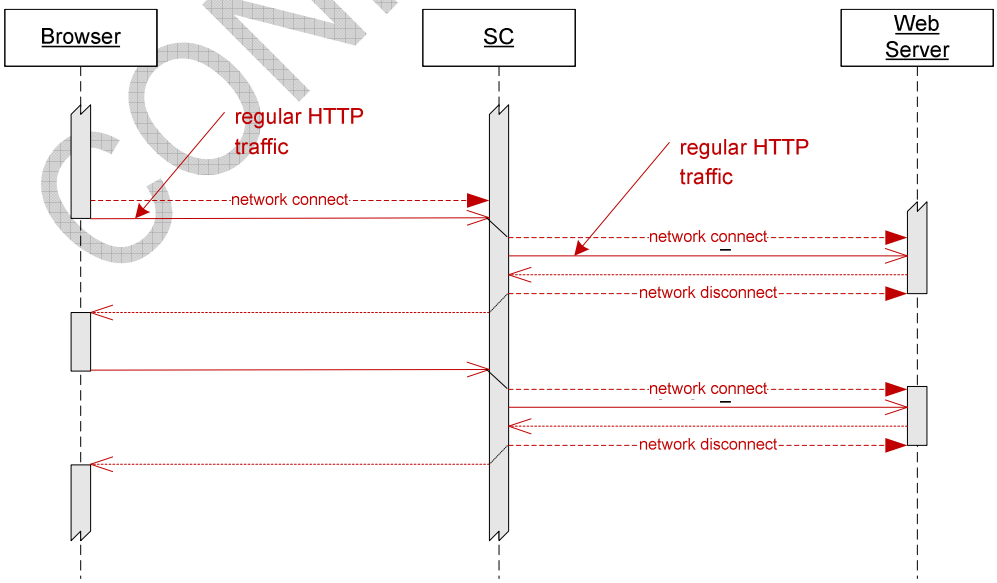


Figure 25 HTTP proxy service

The Web Server is not registered to any service. The SC configuration has one destination (host + port) for redirection of the HTTP traffic. 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.

4 Cascaded Services

Cascaded Services on cascaded SCs and add security, performance and scalability to the solution built with SC. They allow distributing services on different places within the network or redirection of the communication to another location. All types of services can be configured as cascaded service. Cascaded services are fully transparent to the client and server applications.
The following rules apply:

- Cascaded service redirects message flow to service with the same name but on another host.
- Service names are unique in the entire network

4.1 Session Service

A cascaded session service does not hold any context information. It simply passes all traffic to the next service configured in SC. The session exits only on SC to which the server has registered. Server cannot register to a cascaded service.

Caching can be enabled on both SCs. This allows caching of data closer to the consumer.

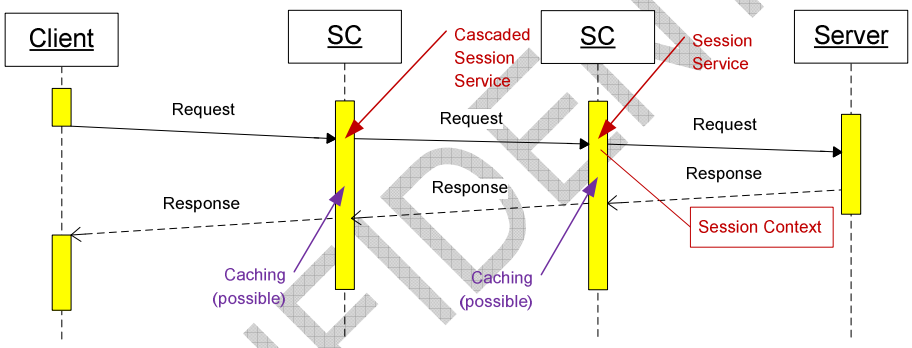


Figure 26 Cascaded Session Service (simplified)

4.2 Publishing Service

A cascaded publishing service behaves like a regular service, but combines all subscriptions and acts like a client to the next service configured in SC. In this way it receives all messages only once and distributes them to the subscribed clients. This behaviour is called fan-out.

The client subscription, change subscription and unsubscribe are passed unchanged to the server in their original form. Combined subscriptions are not passed to the server at all.

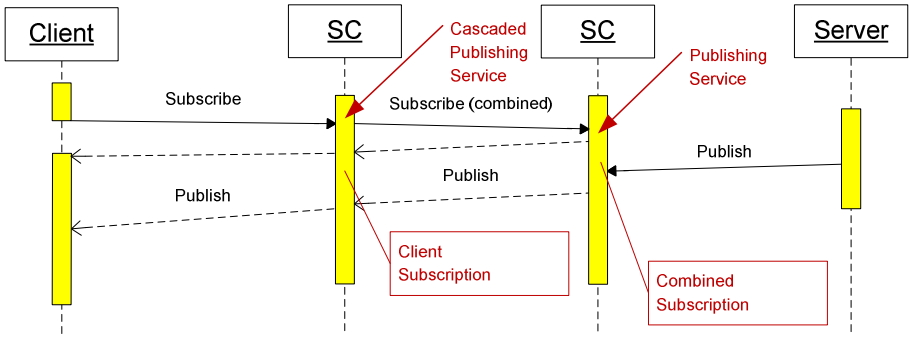


Figure 27 Cascaded Publishing Service (simplified)

4.3 File Service

A cascaded file service does not hold any context information. It simply passes all traffic to the next service configured in SC.

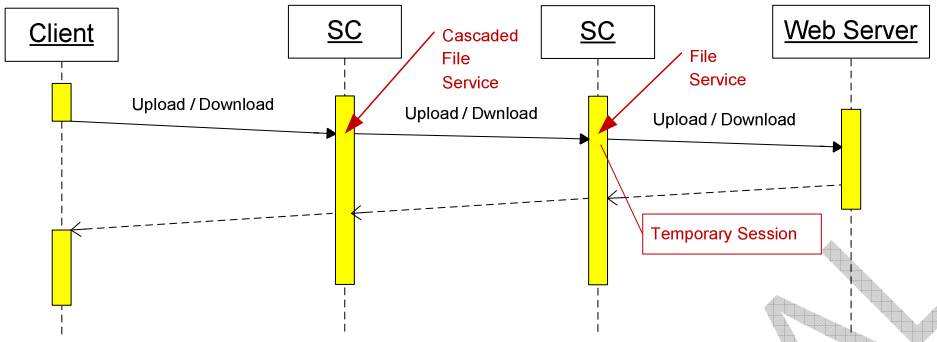


Figure 28 Cascaded File Service

4.4 Cascaded HTTP Proxy Service

HTTP cascaded service is simple proxy and thus passes all http traffic to the next configured host. The SC configuration has one destination (host + port) for redirection of the HTTP traffic. There is no difference between redirection to another SC or to a Web Server.

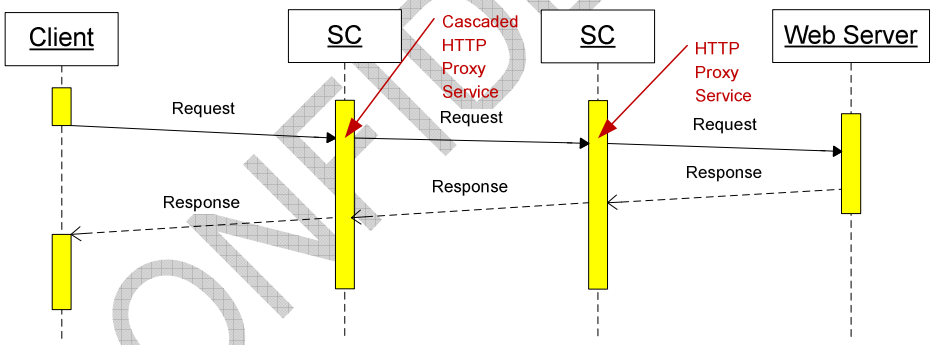


Figure 29 Cascaded HTTP Proxy Service

4.5 Service Routing

Cascaded SC configurations with cascaded services allow service routing. Service routing is fully transparent to the client and server applications.

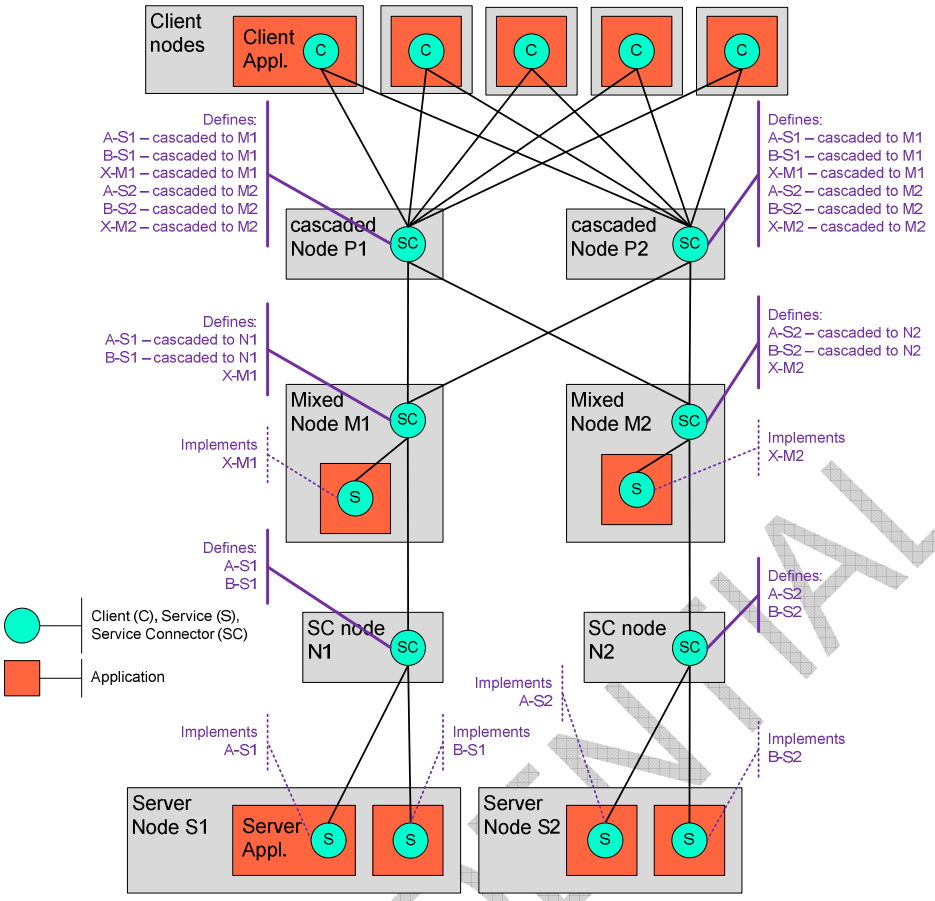


Figure 30 Service Routing

5 State Diagrams

The following chapter shows state diagrams for the client and the server.

5.1 Client

Session Service

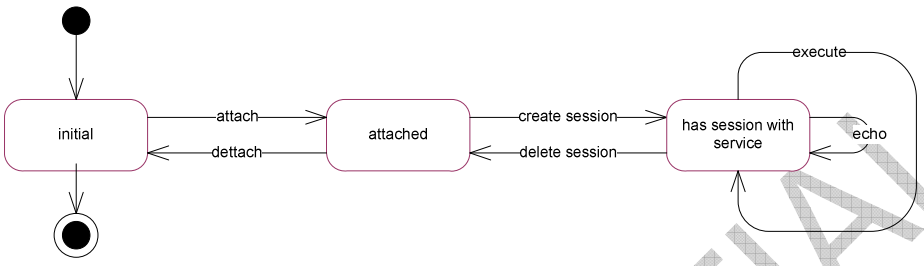


Figure 31 Client states for session service

Message exchange is only possible within a session. File services are available in attached state. On error the following state is established:

- Error in ATTACH results in *initial* state.
- Error in DETACH results in *initial* state.
- Error in CLN_CREATE_SESSION results in *attached* state.
- Error in CLN_DELETE_SESSION results in *attached* state.
- Error in CLN_EXECUTE results in *has session with service* state. Except in case the message contains sessionId which is no longer valid.
- Error in ECHO results in *attached* state. The session is deleted.

Client getting error on CLN_EXECUTE should issue CLN_DELETE_SESSION and ignore all errors.

Publishing Service

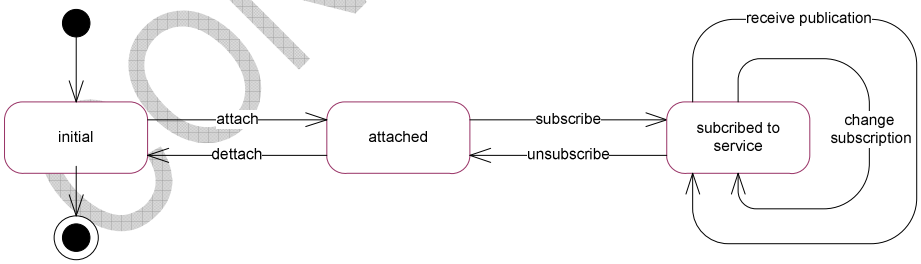


Figure 32 Client states for publishing service

On error the following state is established:

- Error in ATTACH results in *initial* state.
- Error in DETACH results in *initial* state.
- Error in CLN_SUBSCRIBE results in *attached* state.
- Error in CLN_UNSUBSCRIBE results in *attached* state.
- Error in CLN_CHANGE_PUBLICATION results in *subscribed to service* state. Except in case the message contains sessionId which is no longer valid.
- Error in RECEIVE_PUBLICATION results in *subscribed to service* state.

Client getting error on RECEIVE_PUBLICATION should issue CLN_UNSUBSCRIBE and ignore all errors.

5.2 Server

Session Service

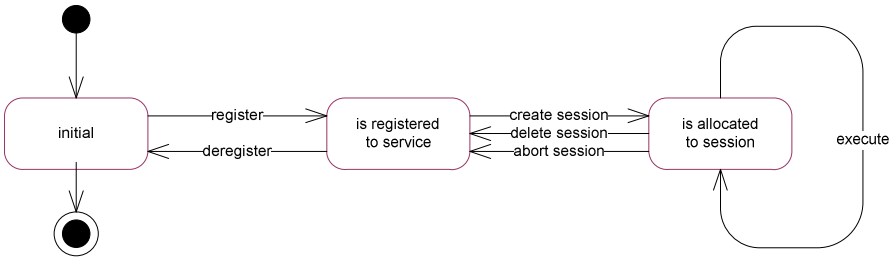


Figure 33 Server states for session service

The de-allocation of the server may happen either due to a regular client action (delete session) or die to an error handled by SC (abort session). On error the following state is established:

- Error in REGISTER_SERVER results in *initial* state.
- Error in DEREGISTER_SERVER results in *initial* state.
- Error in SRV_CREATE_SESSION results in *is registered to service* state.
- Error in SRV_DELETE_SESSION results in *is registered to service* state.
- Error in SRV_ABORT_SESSION results in *is registered to service* state.
- Error in SRV_EXECUTE results in *is allocated to session* state.

Publishing Service

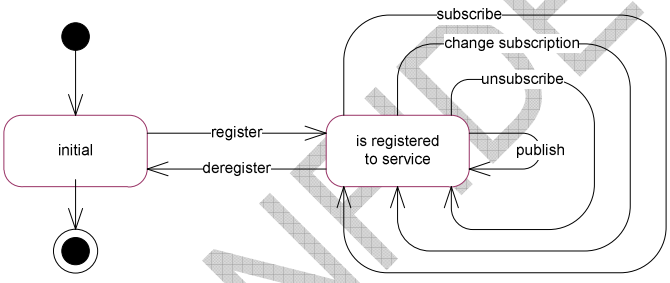


Figure 34 Server states with publishing service

Publishing server gets short requests to verify the subscribing client when it subscribes, unsubscribes or changes its subscription. No durable session is created. On error the following state is established:

- Error in REGISTER_SERVER results in *initial* state.
- Error in DEREGISTER_SERVER results in *initial* state.
- Error in SRV_SUBSCRIBE results in *is registered to service* state.
- Error in SRV_CHANGE_PUBLICATION results in *is registered to service* state.
- Error in SRV_UNSUBSCRIBE results in *registered to service* state.
- Error in PUBLISH results in *is registered to service* state.

6 Relationship Diagram

The following chapter explains the relationship between the SC entities.

6.1 Client

Client may have one or more concurrent connections to SC instances. For one SC connection client may have one or more concurrent sessions or subscriptions to services. Multiple sessions or subscriptions to the same service are possible.

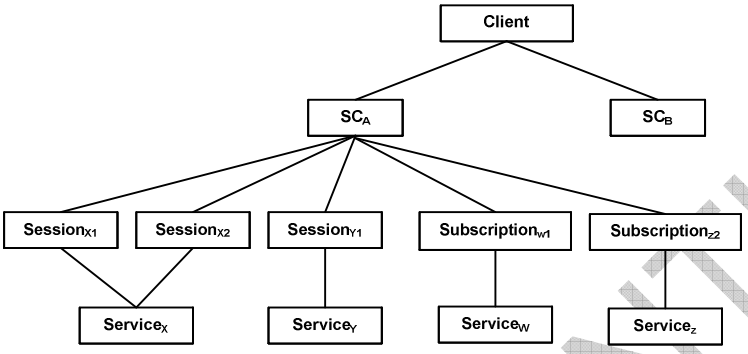


Figure 35 Client relationships

6.2 Server

Single Session Server

Single session server may have only one concurrent connection to a SC. It may register for only one service and only one session.

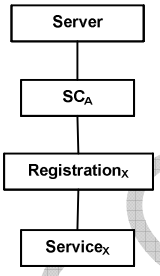


Figure 36 Single Session Server relationships

Multi Session Server

Multi session server may have multiple concurrent connections to SC instances. Within one connection it may register to serve multiple sessions of the same service. When it will serve multiple services, it can use the same connection to SC.

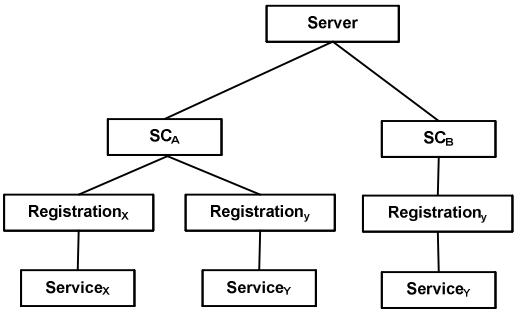


Figure 37 Multi Session Server relationships

7 Message Structure

The Service Connector Message Protocol (SCMP) uses a simple header – body structure.

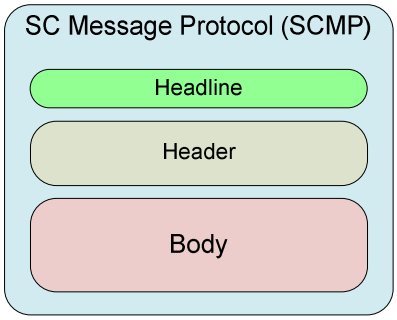


Figure 38 Message Structure

Message example as visible with Wireshark:

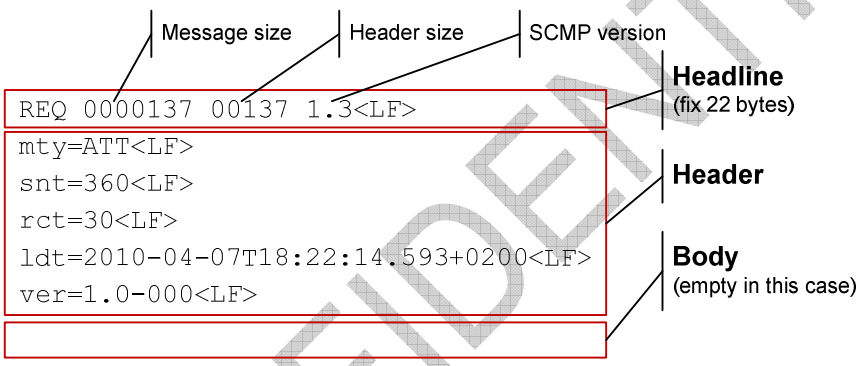


Figure 39 Message Structure Example

7.1 Headline

The fix size (22 bytes) headline defines the header key, the total size of the message, size of the header and the SCMP version. It is encoded in ISO-8859-1 (Latin 1) and terminated by <LF>.

HeaderKey The header key defines the purpose of the message and can be:

- REQ – Request from client or server to SC or request from SC to server
- RES – Response from server to SC or SC to client or to server
- PRQ – Large request part from client or server
- PRS – Large response part from server to client
- PAC – Part acknowledge, request to get more parts
- KRQ – Keepalive request
- KRS – Keepalive response
- EXC – Exception returned after REQ, PRQ or KRQ in case of an error

Message size The complete size of the message in bytes counted from the beginning of the header until the end of the message body. The number is 7 bytes long and has leading zeros. (The maximal allowed message size on OpenVMS platform is 64kB. On other platforms the limit is 9’999’999 bytes.)

Header Size

The size of the message header in bytes counted from the beginning of the header until the end of the message header. The number is 5 bytes long and has leading zeros.

Version

The SCMP version is the version of the **protocol specification** to which this message adheres. It has fixed format 9.9 and it ensures that the receiver knows how this message is structured. The version number (e.g. 2.5) means: 2 = Release number, 5 = Version number.
The receiver may implement multiple protocol versions, thus "understand" older versions. The following matching rule applies:

- Message: 2.5 + receiver implements: 2.5 => compatible
- Message: 2.5 + receiver implements: 2.6 => compatible
- Message: 2.7 + receiver implements: 2.5 => not compatible
  (message may have new headers unknown to the receiver)
- Message: 1.4 + receiver implements: 2.5 => not compatible
  (old message structure and possibly not understood here)
- Message: 2.5+ receiver implements: 1.8 => not compatible
  (new message structure and surely not understood here)

7.2 Header

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

Note:
"=" or <LF> characters are not allowed within attribute names and/or values.

Example:

```
mty=ATT
kpt=10
ldt=2010-04-07T18:22:14.593+0200
```

Unknown header attributes will be ignored and **not** forwarded. The sequence order of the attributes is meaningless.

7.3 Body

The message body has variable length and contains 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.

7.4 SCMP over TCP/IP

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

7.5 SCMP over HTTP

For transport over HTTP the headline and messages header have content type **text/plain** and the message body is multipart, the content type **application/octet-stream**.
The request uses method POST to send the request. 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 0000081 00081 1.3
mty=ATT
kpt=10
ldt=2010-04-07T18:22:14.593+0200

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

RES 0000041 00041 1.3
mty=ATT
ldt=2010-04-07T18:22:14.593+0200
```

8 SCMP Messages

Server may play the role of a client and consume other services. In such configuration message which belong to the client and to the server must have different types. For this reason messages initiated by the client have CLN_ prefix and messages sent to the server the SRV_ prefix.

8.1 Keepalive

Keepalive messages are sent in regular intervals on all idle connections initiated by a network peer. The only purpose of keepalive messages is to preserve the connection state in the firewall placed between the communicating peers and resets its internal timeout. See also Chapter 2.3

Using of keepalive messages can be disabled with the *keepaliveInterval (kpi)* attribute. The message has only a headline and no attributes and no body. The format is:

KRQ 0000000 00000 1.3

for request and

KRS 0000000 00000 1.3

for response.

8.2 ATTACH (ATT)

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

mty=ATT
ver=1.0-023
ldt=1997-07-16T19:20:30.064+0100
oti=2000

SC receives the message and sends back the response:

mty=ATT
ldt=1997-07-16T19:20:34.044+0200

When an SC error occurs the response message contains the attributes:

mty=ATT
ldt=1997-07-16T19:20:30.453+0100
sec=3000
set=SCMP version mismatch (Received=1.0-23, Required 1.1.-34)

Actually this message is only used to check the availability of the SC and to check the compatibility of the communicating partners.

8.3 DETACH (DET)

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

mty=DET
oti=2000

SC receives the message and sends back the response:

mty=DET

8.4 INSPECT (INS)

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

mty=INS
bty=txt
ipl=10.0.4.32
oti=2000

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

mty=INS
bty=txt
ipl=10.0.4.32

The request and response message body has the format according to the following table:

Request format	Response format	Comment
state=name	*state*=state	Shows the state of a service identified by name. Returns *enabled* or *disabled* according to the current service state.
sessions=name	*sessions*=9999/9999	Shows the sessions available and allocated for the service. Returns *available / allocated* count of sessions.

When an SC error occurs the response message contains no body and the attributes:

mty=INS
sec=370
set=Unkown service: P01_RTXS_RPRWS3

8.5 MANAGE (MGT)

This message is sent from the client to SC in order to change the SC behaviour. The message has body of type *text* and has these attributes:

mty=MGT
bty=txt
ipl=10.0.4.32
oti=2000

The request message body has the format according to the following table:

Request format	Comment
disable=name	Disables the service identified by name. When service is disabled, clients cannot create new sessions or subscriptions. Existing sessions or subscriptions are not aborted. The initial (default) state of the service is defined in the SC configuration.
enable=name	Enables the service identified by name
kill	Terminates the SC. The will immediately exit without any cleanup action. **The SC will execute this request only if the IP address in ipl is equal to the IP address of the local node.** This prevents shutdown issued from remote nodes.

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

mty=MGT
ipl=10.0.4.32

When an SC error occurs the response message contains no body and the attributes:

mty=MGT

sec=370
set=Unkown service: P01_RTXS_RPRWS3

8.6 CLN_CREATE_SESSION (CCS)

This message is sent from the client to SC in order to create a new session for a service. The message has optional a body and contains these attributes (as seen by the SC):

mty=CCS
msn=1
nam=P01_RTXS_RPRWS1
ipl=10.0.4.32/10.0.4.32
sin=SNBZHP - TradingClientGUI 10.2.7
eci=300
oti=10000

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 does not contain the attribute rejectFlag, the SC keeps the session and sends back to client the message with optional body and the following attributes:

   mty=CCS
   msn=1
   nam=P01_RTXS_RPRWS1
   sid=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d

6. If the response message contains the attribute rejectFlag, the SC deletes the session, de-allocates the server and sends back to client the message with optional body and the following attributes:

   mty=CCS
   msn=1
   nam=P01_RTXS_RPRWS1
   rej
   aec=4334591
   aet=%RTXS-E-NOPARTICIPANT, Authorization error – Unknown participant

When an SC error occurs the response message contains the attributes:

mty=CCS
sec=330
set=Unkown service: P01_RTXS_RPRWS3

If SC cannot allocate a free server instance for the session it will respond with the error set=No free server available for service: P01_RTXS_RPRWS3.

Large messages are not supported in this context.

8.7 SRV_CREATE_SESSION (SCS)

This message is sent from the SC to the server when a server instance has been be allocated to a session. Optionally the message has a body and contains these attributes:

mty=SCS
msn=1

nam=P01_RTXS_RPRWS1
sid=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d
sin=SNBZHP - TradingClientGUI 10.2.7
ipl=10.0.4.32/10.0.4.32/10.2.54.12
oti=10000

The server receives the message and must decide to accept or reject this request. If it accepts, then the reply has an optional body and the following attributes:

mty=SCS
msn=1
nam=P01_RTXS_RPRWS1
sid=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d

If it rejects the session, then the reply has an optional body and the following attributes:

mty=SCS
msn=1
nam=P01_RTXS_RPRWS1
sid=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d
rej
aec=4334591
aet=%RTXS-E-NOPARTICIPANT, Authorization error – Unknown participant

The *aec* / *aet* attributes should contain readable information describing the rejection reason.

8.8 CLN_DELETE_SESSION (CDS)

This message CLN_DELETE_SESSION is sent from the client to SC in order to terminate an existing session. The message has no body and contains these attributes:

mty=CDS
msn=974834
nam=P01_RTXS_RPRWS1
sid=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d
sin=rollback
oti=10000

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:
   mty=CSD
   msn=974834
   nam=P01_RTXS_RPRWS1
   sid=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 handles such situations and responds with a appropriate error message.

When an SC error occurs the response message contains the attributes:

mty=CDS
sec=330
set=Session does not exist

8.9 SRV_DELETE_SESSION (SDS)

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:

mty=SDS
msn=974834
nam=P01_RTXS_RPRWS1
sid=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d
sin=rollback
oti=10000

The server must return a message with the following attributes:

mty=SDS
msn=974834
nam=P01_RTXS_RPRWS1
sid=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d

In case the server does not respond within the given operation timeout (oti) the SC will abort the session, deregister the server and close all connections to the server.

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

8.10 SRV_ABORT_SESSION (SAS)

This message is sent from the SC to the server when the session is aborted due to errors or other unexpected event. The message has no body and contains these attributes:

mty=SAS
nam=P01_RTXS_RPRWS1
sid=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d
sec=302
set=Session timeout exceeded
oti=2000

The server must return a message with the following attributes:

mty=SAS
nam=P01_RTXS_RPRWS1
sid=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d

In case the server does not respond within the configured operation timeout (oti) the SC will abort the session, deregister the server and close all connections to the server.

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

8.11 REGISTER_SERVER (REG)

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:

mty=REG
nam=P01_RTXS_RPRWS1
mxs=10
mxc=5
imc
pnr=9100
ver=1.0-023
ldt=1997-07-16T19:20:30.064+0100
kpi=360
oti=2000

SC receives the message and performs 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. Starts monitoring the connection based on keep alive interval value
4. Sends back a message with the following attributes:

   mty=REG
   nam=P01_RTXS_RPRWS1
   ldt=1997-07-16T19:20:30.064+0100

When an SC error occurs the response message contains the attributes:

mty=REG
sec=330
set=Service name=P01_RTXS_RPRWS5 not found

8.12 CHECK_REGISTRATION (CRG)

This message can be sent from the registered server to SC in order to check its registration . The message has no body and contains these attributes:

mty=CRG
nam=P01_RTXS_RPRWS1

SC receives the message checks the registration and sends back message with the following attributes:

mty=CRG
nam=P01_RTXS_RPRWS1

When an SC error occurs the response message contains the attributes:

mty=CRG
sec=330
set=Server is not registered

8.13 DEREGISTER_SERVER (DRG)

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. The message has no body and contains these attributes:

mty=DRG
nam=P01_RTXS_RPRWS1
oti=2000

SC receives the message and does these actions:

1. Finds the server and performs a cleanup by 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.
2. Terminates all connections that have been established from SC to this server.
3. Sends back message with the following attributes:

   mty=DRG
   nam=P01_RTXS_RPRWS1

When an SC error occurs the response message contains the attributes:

mty=DRG
sec=330
set=Server is not registered

After this message the server may disconnect from the SC.

8.14 CLN_EXECUTE (CXE)

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:

mty=CXE
nam=P01_RTXS_RPRWS1
sid=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d
msn=974833
min=SECURITY_MARKET_QUERY
oti=60000

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

cid=CBCD_SECURITY_MARKET

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

mty=CXE
nam=P01_RTXS_RPRWS1
sid=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d
msn=974834
min=SECURITY_MARKET_RESULT

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

cid=CBCD_SECURITY_MARKET
ced=1997-08-16T17:00:00.000+0100

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

aec=4334591
aet=%RDB-F-NOTXT, no transaction open

In case the server does not respond within the given operation timeout (oti) the SC will abort the session, send CLN_EXECUTE response with the error "timeout expired" back to the client and send the SRV_ABORT_SESSION to the server.

When an SC error occurs the response message contains the attributes:

mty=CXE
sec=330
set=Session does not exist

Large messages are supported in this context.

8.15 SRV_EXECUTE (SXE)

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:

mty=SXE
nam=P01_RTXS_RPRWS1
sid=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d
msn=974833
min=SECURITY_MARKET_QUERY
oti=18000

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:

mty=SXE

nam=P01_RTXS_RPRWS1
sid=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d
msn=34834
min=SECURITY_MARKET_RESULT

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

cid=CBCD_SECURITY_MARKET
ced=1997-08-16T17:00:00.000+0100

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

aes=4334591
aet=%RDB-F-NOTXT, no transaction open

8.16 ECHO (ECH)

This message is sent from the client to SC in order to verify the session consistency. The client must send this message in periodic intervals in scope of every session. The message has no body and contains these attributes:

mty=ECH
nam=P01_RTXS_RPRWS1
sid=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d
msn=974833
oti=2000

SC receives the message and verifies the allocated session. Then it sends back a message without body and the following attributes:

mty=ECH
nam=P01_RTXS_RPRWS1
sid=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d
msn=974834

When an SC error occurs the response message contains the attributes:

mty= ECH
sec=330
set=Session does not exist

8.17 CLN_SUBSCRIBE (CSU)

This message is sent from the client to SC in order to subscribe for a publishing service. The message has optional a body and contains these attributes (as seen by the SC):

mty=CSU
msn=1
nam=P01_BCST_CH_RPRWS2
msk=000012100012832102FADF----------
ipl=10.0.4.32/10.0.4.32
sin=SNBZHP - TradingClientGUI 10.2.7
noi=300
oti=20000

SC receives the message and does these actions:

1. Generates a unique session id
2. Sends the message SRV_SUBSCRIBE to the server registered for this service and awaits the server response.
3. If the server response message does not contain the attribute rejectFlag, the SC remembers the subscription mask of the client for this service and sends back to client the message with the following attributes:

   mty=CSU

msn=1
nam=P01_BCST_CH_RPRWS2
sid=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d

4. If the server response message contains the attribute rejectFlag, the SC deletes the session, and sends back to client the message with the following attributes:
   mty=CSU
   msn=1
   nam=P01_BCST_CH_RPRWS2
   rej
   aec=4334591
   aet=%RTXS-E-NOPARTICIPANT, Authorization error – Unknown participant

When an SC error occurs the response message contains the attributes:
mty=CSU
sec=400
set=Unkown service: P01_BCST_CH_RPRWS2

The subscription is synchronous operation. The client gets control when all SC components on the path to the publishing server are aware of the subscription.
If SC cannot find server registered for this service it will respond with the error set=No server available for service: P01_BCST_CH_RPRWS2.

Large messages are not supported in this context.

8.18 SRV_SUBSCRIBE (SSU)

This message is sent from the SC to the server in order to process the client subscription (e.g. perform authentication). The message has optional a body and contains these attributes:
mty=SSU
msn=1
nam=P01_BCST_CH_RPRWS2
sid=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d
msk=000012100012832102FADF----------
ipl=10.0.4.32/10.0.4.32/10.2.54.12
sin=SNBZHP - TradingClientGUI 10.2.7
oti=10000

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:
mty=SSU
msn=1
nam=P01_BCST_CH_RPRWS2
sid=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d

If it rejects the session, then it must return a message with the following attributes:
mty=SSU
msn=1
nam=P01_BCST_CH_RPRWS2
sid=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d
rej
aec=4334591
aet=%RTXS-E-NOPARTICIPANT, Authorization error – Unknown participant

8.19 CLN_CHANGE_SUBSCRIPTION (CHS)

This message is sent from the client to SC in order to change the subscription for a publishing service. The message has optional a body and contains these attributes (as seen by the SC):

mty=CHS
msn=53834
nam=P01_BCST_CH_RPRWS2
sid=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d
msk=000012100012832102FADF----------
ipl=10.0.4.32/10.0.4.32
sin=SNBZHP - TradingClientGUI 10.2.7
oti=10000

SC receives the message and does these actions:

1. Sends the message SRV_CHANGE_SUBSCRIPTION to the server registered for this service and awaits the server response.
2. If the server response message does not contain the attribute rejectFlag, the SC changes the subscription mask of the client for this service and sends back to client the message with the following attributes:

   mty=CHS
   msn=974834
   nam=P01_BCST_CH_RPRWS2
   sid=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d

3. If the server response message contains the attribute rejectFlag, the SC keeps the previous subscription and sends back to client the message with the following attributes:

   mty=CHS
   msn=974834
   nam=P01_BCST_CH_RPRWS2
   sid=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d
   rej
   aec=4334591
   aet=%RTXS-E-NOPARTICIPANT, Authorization error – Unknown participant

When an SC error occurs the response message contains the attributes:

mty=CHS
sec=320
set=Client is not subscribed

The change of the subscription is synchronous operation. The client gets control when all SC components on the path to the publishing server are aware of the new subscription. If SC cannot find server registered for this service it will respond with the error set=No server available for service: P01_BCST_CH_RPRWS2.

Large messages are not supported in this context.

8.20 SRV_CHANGE_SUBSCRIPTION (SHS)

This message is sent from SC to the server in order to delete the client subscription. The message has optional a body and contains these attributes:

mty=SHS
msn=53834
nam=P01_BCST_CH_RPRWS2
sid=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d
oti=10000

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:

mty=SHS
msn=1
nam=P01_BCST_CH_RPRWS2
sid=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d

If it rejects the session, then it must return a message with the following attributes:

mty=SHS
msn=1
nam=P01_BCST_CH_RPRWS2
sid=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d
rej
aec=4334591
aet=%RTXS-E-NOPARTICIPANT, Authorization error – Unknown participant

8.21 CLN_UNSUBSCRIBE (CUN)

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:

mty=CUN
msn=53834
nam=P01_BCST_CH_RPRWS2
sid=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d
sin=rollback
oti=10000

SC receives the message and sends the message SRV_UNSUBSCRIBE to the server registered for this service and awaits the server response. Then it sends back to client the message with the following attributes:

mty=CUN
msn=1
nam=P01_BCST_CH_RPRWS2
sid=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d

When an SC error occurs the response message contains the attributes:

mty=CUN
sec=230
set=Client is not subscribed

This operation is synchronous. The client gets control when all SC components on the path to the publishing server have deleted the subscription. If SC cannot find server registered for this service it will respond with the error set=No server registered for service: P01_BCST_CH_RPRWS2.

8.22 SRV_UNSUBSCRIBE (SUN)

This message is sent from the SC to the server in order to remove the client subscription (e.g. delete the session). The message has no body and contains these attributes:

mty=SUN
msn=53834
nam=P01_BCST_CH_RPRWS2
sid=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d
sin=rollback
oti=10000

The server must return a message with the following attributes:

mty=SUN
msn=974834
nam=P01_BCST_CH_RPRWS2

sid=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d

8.23 RECEIVE_PUBLICATION (CRP)

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:

mty=CRP
sid=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d
nam=P01_BCST_CH_RPRWS2
msn=974833
oti=2000

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:

      mty=CRP
      sid=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d
      nam=P01_BCST_CH_RPRWS2
      msn=974834
      min=CH_AUCTION
      msk=%%%%%%%%%%%%%%%%%%%%%%%%%X%%%%%%%%%%%%%%%%%%%%

   b. The *noDataInterval (noi)* timeout expires. Then it sends back a message with the no body and the following attributes:

      mty=CRP
      sid=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d
      nam=P01_BCST_CH_RPRWS2
      msn=974834
      nod

When an SC error occurs the response message contains the attributes:

mty=CRP
sec=130
set=Client is not subscribed to service: P01_BCST_CH_RPRWS2

8.24 PUBLISH (SPU)

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:

mty=SPU
nam=P01_BCST_CH_RPRWS2
msn=65411
min=CH_AUCTION_IOI
msk=%%%%%%%%%%%%%%%%%%%%%%%%%X%%%%%%%%%%%%%%%%%%%%
oti=2000

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:

   mty=SPU
   nam=P01_BCST_CH_RPRWS2
   msn=65412

3. Starts distribution of the message to the subscribed clients based on their subscription mask and the mask of the message.

When an SC error occurs the response message contains the attributes:

mty=SPU
sec=100
set=Service P01_BCST_CH_RPRWS2 does not exist

8.25 FILE_DOWNLOAD (FDO)

This message is used to download file from a web server. The message has no body and contains these attributes:

mty=FDO
nam=P01_LOGGING
sid=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d
rfn=LSC TradingClientGUI_20100115_07h28m59s.log
oti=2000

The SC takes the message and initiates a download from the web server configured for this service. The requesting URL is constructed according to the configuration and the given remote file name. The response message has a body containing the file and has these attributes:

mty=FDO
bty=bin
nam=P01_LOGGING
sid=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d
rfn=LSC TradingClientGUI_20100115_07h28m59s.log

The maximal message length may exceed the 64kB limit.

On Error the SC may return message a response message contains the attributes:

mty=FDO
nam=P01_LOGGING
rfn=LSC TradingClientGUI_20100115_07h28m59s.log
sid=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d
sec=40
set=404 Not found

See http://www.w3.org/Protocols/rfc2616/rfc2616-sec6.html#sec6.1.1 for possible status codes and reasons.

8.26 FILE_UPLOAD (FUP)

This message is used to upload file to a web server. The message has a body containing the file and has these attributes:

mty=FUP
bty=bin
nam=P01_CONFIGURATION
sid=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d
rfn=SCconfig-P01-V3.3-433.properties
oti=2000

The SC takes the message and initiates an upload to the web server configured for this service. The requesting URL is constructed according to the configuration and the given remote file name. The response message has no body and has these attributes:

mty=FUP
nam=P01_LOGGING
sid=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d
rfn=SCconfig-P01-V3.3-433.properties

The maximal message length may exceed the 64kB limit. On Error the SC may return message a response message contains the attributes:

mty=FUP
nam=P01_LOGGING
sid=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d
rfn=SCconfig-P01-V3.3-433.properties
sec=40
set=406 Not Acceptable

See http://www.w3.org/Protocols/rfc2616/rfc2616-sec6.html#sec6.1.1 for possible status codes and reasons.

8.27 FILE_LIST (FLI)

This message is used to get list of files on web server. The Web Server must allow the directory browsing. The message has no body and these attributes:

mty=FLI
nam=P01_CONFIGURATION
sid=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d
oti=2000

The SC takes the message and creates an URL to explore the location on the web server corresponding to the service. The response message has body and has these attributes:

mty=FLI
bty=txt
nam=P01_ CONFIGURATION
sid=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d

The response message body has the following format:

```
File name1<LF>
File name2<LF>
File name3<LF>
```

No header or footer is used. Each file name is on a separate line terminated by <CR><LF> Filenames appear as they are stored on disk. Upper / lowercase is depending on the operating system used. No file attributes like date or size is provided. The list order is not predictable.The physical location corresponding to the service is not disclosed.

The maximal message length may exceed the 64kB limit. On Error the SC may return message a response message contains the attributes:

mty=FLI
nam=P01_ CONFIGURATION
sid=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d
sec=40
set=403 Forbidden

See http://www.w3.org/Protocols/rfc2616/rfc2616-sec6.html#sec6.1.1 for possible status codes and reasons.

9 SCMP Header Attributes

The following is a list of all possible attributes in a SC message header in alphabetical order. All attributes and their values are ASCII, encoded as ISO 8859-1 (Latin-1). "=" or <LF> are not allowed within attribute names and/or values. Unknown attributes will be ignored.

You can find the matrix describing which attribute is used in which message at the end of this document.

9.1 actualMask (ams)

Name	actualMask
Code	ams
Description	The actual client subscription mask filled by SC in SRV_CHANGE_SUBSCRIPTION.
Validation	Any printable character, length ≤ 256Byte.
Comment	This mask is filled by SC and may be used by the server to check the subscription changes.
Example	Subscription mask: ams=00001210001283210 2FADF----------X----------

9.2 appErrorCode (aec)

Name	appErrorCode
Code	aec
Description	Numeric value passed between server and the client used to implement error protocol on the application level. Can be set by server whenever it responds with a message body.
Validation	Numeric value ≥ 0
Comment	This can be used by the client to check a specific server error.
Example	aec=4334591

9.3 appErrorText (aet)

Name	appErrorText
Code	aet
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*. Can be set by server whenever it responds with a message body.
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	aet=%RDB-F-NOTXT, no transaction open

9.4 bodyType (bty)

Name	bodyType
Code	bty
Description	Type of the message body
Validation	Enumeration, 3 characters, fixed: • txt – message body is ISO-8859-1 (Latin 1) encoded text • bin – binary data • xml – XML data (not implemented yet)

Default	bin
Comment	When http transport is used, the content-type header is set according to this attribute.
Example	bty=txt

9.5 cacheExpirationDateTime (ced)

Name	cacheExpirationDateTime
Code	ced
Description	When sent by the server then it represents the absolute expiration date and time of the message in cache. It must be set together with *cacheId* attribute.
Validation	YYYY-MM-DDThh:mm:ss.fff+hhmm It is local date time plus zone information. The fff are seconds fractions and time zone offset is at the end.
Comment	The server uses *cacheExpirationDateTime* to define how long the message is valid. The client will get a cached message when the *cacheId* and *serviceName* matches a message in the cache and the cached message is timely valid.
Example	ced=1997-08-16T19:20:34.237+0200

9.6 cacheId (cid)

Name	cacheId
Code	cid
Description	Identification agreed by the communicating applications to uniquely identify the cached content. The *cacheId* is unique per service.
Validation	Any printable character except "/", 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 the *cacheId* and *serviceName* matches a message in the cache and the cached message is timely valid. During receipt of large cached message cacheId is modified by SC and contains parts identification like *cacheId/messageSequenceNr.*
Example	cid=CBCD_SECURITY_MARKET

9.7 cascadedMask (cma)

Name	cascadedMask
Code	cma
Description	Subscription mask exchanged between cascaded SCs.
Validation	length ≥ 0 and ≤ 256Byte It may contain "%" character.
Comment	This attribute is sent by SC to another SC along the communication path. It contains the cumulative subscription mask of the SC.
Example	cascaded mask: cma=0000121%%%%%%%%%%%%%%----------X----------

9.8 cascadedSubscriptionId (csi)

Name	cascadedSubscriptionId
Code	csi
Description	Unique identification of the subscription of the SC (on-behalf client) in cascaded configurations.
Validation	Known subscription, length ≥ 0 and ≤ 256Byte
Comment	This attribute is sent by SC to another SC along the communication path. It

	contains the subscription id of the SC performing the message fan-out.
Example	csi=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d

9.9 compression (cmp)

Name	compression
Code	cmp
Description	Flag true or false describing if the message body is compressed or not.
Validation	Present is true, missing is false
Default	true
Comment	The compression can be enabled or disabled on message level.
Example	cmp

9.10 echoInterval (eci)

Name	echoInterval
Code	eci
Description	Interval in seconds between two subsequent ECHO messages sent by the client to SC. The message is sent only when no EXECUTE message is pending.
Validation	Number ≥ 1 and ≤ 3600
Default	300
Comment	This is used by the SC to detect a broken session. The value should be set with respect to the throughput of the network connection. If this interval expires then the session is treated as dead and a cleanup is done.
Example	eci=300

9.11 immediateConnect (imc)

Name	immediateConnect
Code	imc
Description	Flag true or false to tell SC when connection to the server should be created.
Validation	Present is true, missing is false.
Default	true
Comment	After server registers to a service SC will create as many connections to it as defined by *maxSessions.* When *immediateConnect* = true SC will create the connections immediately and keep the session until DEREGISTER_SERVER is done. When *immediateConnect* = false SC will create the connections before a session is allocated and close the connection when the session is deleted.
Example	Imc

9.12 ipAddressList (ipl)

Name	ipAddressList
Code	ipl
Description	List of IP addresses on the network path between the client and the session server. The list contains IP addresses in the form 999.999.999.999:99999.
Validation	List in format {999.999.999.999/999.999.999.999}...
Comment	The list has at least three entries. 1. IP of the client, 2. Incoming IP received by SC (IP of the VPN Tunnel) 3. IP of the SC Client connected via cascaded SC placed in a customer DMZ will have the list in the format: 1. IP of the client, 2. Incoming IP received by SC in customer DMZ. 3. IP of the SC in customer DMZ 4. Incoming IP received by SC (IP of the VPN Tunnel)

	5. IP of the SC If any of the pairs at position1-2 or 3-4 have different values, then NAT occurs in this network segment. As long as there is only one SC behind the VPN, the second last address is always the tunnel IP used for the authentication.
Example	ipl=10.0.4.32/10.0.4.32/10.2.54.12

9.13 keepaliveInterval (kpi)

Name	keepaliveInterval
Code	kpi
Description	Interval in seconds between two subsequent keepalive requests (KRQ). The keepalive message is solely used to refresh the firewall timeout on the network path. Keepalive message is only sent on an idle connection. The value = 0 means no keep alive messages will be sent.
Validation	Number ≥ 0 and ≤ 3600
Default	60
Comment	The keepaliveInterval is exchanged at the beginning of the communication. In REGISTER_SERVER the *keepaliveInterval* defines how often the SC will sent keepalive messages to the server.
Example	kpi=360

9.14 localDateTime (ldt)

Name	localDateTime
Code	ldt
Description	String value describing the actual local date and time.
Validation	YYYY-MM-DDThh:mm:ss.fff+hhmm It is local date time plus zone information. The fff are seconds fractions and time zone offset is at the end after the + sign.
Comment	The local date time is exchanged at the beginning of the communication (ATTACH, REGISTER_SERVER). It is used to calculate the time difference between the communicating parties and to harmonize the log for troubleshooting purposes.
Example	ldt=1997-07-16T19:20:30.064+0100

9.15 messageInfo (min)

Name	messageInfo
Code	min
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 set by the sender and evaluated by the receiver of the message to simplify decision how the message should be processed. It can also be used for troubleshooting to identify the message during the message transmission.
Example	min=SECURITY_MARKET_QUERY

9.16 messageSequenceNumber (msn)

Name	messageSequenceNumber
Code	msn
Description	Identification generated by the sender of a message in order to identify and track it during a session. The sessionId + the messageSequenceNumber uniquely identify the message. Numbering of request and response messages is independent. Error messages generated by SC never contain a messageSequenceNumber. ECHO message has no msn.

Validation	Running number in format > 0, not validated by SC Gaps are possible. See also chapter 3.1.10 and 3.2.5
Comment	The message sequence number is reset at begin of the session and is steadily increasing, incremented by the sender.
Example	**REQ .. msn=3** **RES .. msn=64** ... **PRQ .. msn=4** PAC .. msn=65 PRQ .. msn=5 PAC .. msn=66 PRQ .. msn=6 PAC .. msn=67 **REQ .. msn=7** PRS .. msn=68 PAC .. msn=8 PRS .. msn=69 PAC .. msn=9 **RES .. msn=70** ... **REQ .. msn=10** **RES .. msn=71**

9.17 messageType (mty)

Name	messageType
Code	mty
Description	Unique message type
Validation	3 characters, fixed, uppercase, list of known message types
Comment	Message type that represents a certain command. The direction of the message is visible in the headline.
Example	mty=ATT

9.18 mask (msk)

Name	mask
Code	msk
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 Client may not subscribe with mask containing "%" character.
Comment	If the message mask matches the subscription mask, the client will get this message. The matching rules: • masks of unequal length do not match • % - matches any single character at this position • All other characters must exactly match (case sensitive)
Example	Subscription mask: msk=00001210001283210 2FADF----------X---------- Matching examples of message masks: msk=00001210001283210 2FADF----------X---------- msk=0000121%%%%%%%%%%%%%%%----------X---------- Not matching examples of message masks: msk=00001210001283210 2FADF---------- msk=0000121%%%%%%%%%%%%%%%----------X----------

9.19 maxConnections (mxc)

Name	maxConnections
Code	mxc
Description	Number of connections (pool size) SC will create initially to communicate with the server.
Validation	Number > 0 and ≤ 1024 If mxs = 1 then mxc = 1 If mxs > 1 then mxc > 1 and mxc ≤ mxs (multi-session server with only 1 connection is not supported)
Default	mxs
Comment	When a server registers to a service it must tell the SC how many connections it can handle. In case all connections are busy, SC will not start a new connection but keep trying to get a free connection until the *operationTiemout* defined in the message expires. Then it will return an error “no free connection available”.
Example	mxc=10

9.20 maxSessions (mxs)

Name	maxSessions
Code	mxs
Description	Number of sessions this server instance can serve.
Validation	Number > 0
Comment	When a 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 free/busy servers in SC. The value 1 means single session server. Value > 1 means multi-session server. See also *immediateConnect* flag. In case all servers are busy, SC will keep trying to get a free server until the *operationTiemout* defined in the message expires. Then it will return an error “no free server available”.
Example	mxs=10

9.21 noData (nod)

Name	noData
Code	nod
Description	NoData flag is used in RECEIVE_PUBLICATION to tell the subscribed client, that no data for publishing exists. The client must immediately send another RECEIVE_PUBLICATION to renew the interest.
Validation	Present is true, missing is false.
Comment	No msn is sent when this flag is set
Example	nod

9.22 noDataInterval (noi)

Name	noDataInterval
Code	noi
Description	Interval in seconds the SC will wait to deliver RECEIVE_PUBLICATION response with *noData* flag set.
Validation	Number > 0 and ≤ 3600
Default	300
Comment	The receiving client monitors the interval and treats the subscription as broken when this time is exceeded.
Example	noi=60

9.23 operationTimeout (oti)

Name	operationTimeout
Code	oti
Description	Maximal response time in milliseconds allowed for an operation (measured in the client). The client should set the value according to the expected duration of the operation.
Validation	Number ≥ 1000 and ≤ 3600000
Default	60000
Comment	This is used by to make any operation non-blocking. When this timeout expires, while a session is pending, the session is treated as dead and cleaned up.
Example	oti=10000

9.24 portNr (pnr)

Name	portNr
Code	pnr
Description	Number of the TCP/IP port the session server accepts the connection(s).
Validation	Number > 1 and ≤ 65535
Comment	When a session server registers to a service, 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.
Example	pnr=9100

9.25 rejectSession (rej)

Name	rejectSession
Code	rej
Description	Flag in SRV_CREATE_SESSION, SRV_SUBSCRIBE, SRV_CHANGE_SUBSCRIPTION response message set by the server when it rejects the session.
Validation	Present is true, missing is false.
Default	false
Comment	The server should also set the *appErrorCode* and *appErrorText* to explain the rejection reason.
Example	rej

9.26 remoteFileName (rfn)

Name	remoteFileName
Code	rfn
Description	Name used in FILE_UPLOAD to store the file on the web server or name used in FILE_DOWNLOAD to identify the file to be downloaded.
Validation	Any printable character allowed as filename, length ≤ 256Byte
Comment	
Example	rfn=LSC TradingClientGUI_20100115_07h28m59s.log

9.27 scErrorCode (sec)

Name	scErrorCode
Code	sec
Description	Numeric error code set by SC in other to inform the communication partner about an error. List of possible error codes will be published.
Validation	Number > 0 and < 1000
Comment	This is used to handle the SC error. The message must have EXC key in the headline.
Example	sec=453

9.28 scErrorText (set)

Name	scErrorText
Code	set
Description	English text set by the SC in other to describe the error signalled as *scErrorCode*. Precise error description must be here.
Validation	Any printable character, length > 0 and ≤ 256Byte
Comment	This is used to log or display the SC error. The message must have EXC key in the headline.
Example	set=Unknow service name: P01_RTXR_RPRWS4

9.29 scVersion (ver)

Name	scVersion
Code	ver
Description	Software version number of the producer of this message.
Validation	String format 9.9-999
Comment	This version number is sent in ATTACH or REGISTER_SERVER 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 or SC. The version number looks like 3.2-023: • 3 = Release number • 2 = Version number • 023 = Revision number The matching rules are: • Request: 3.2-023 + own: 3.2-023 => compatible • Request: 3.2-021 + own: 3.2-023 => compatible • Request: 3.1-006 + own: 3.2-023 => compatible • Request: 3.2-025 + own: 3.2-023 => not compatible (requestor may utilize new features unknown here) • Request: 3.3-005 + own: 3.2-023 => not compatible (requestor uses new functions unknown here) • Request: 2.2-023 + own: 3.2-023 => not compatible (possibly other incompatible interface) • Request: 4.0-007 + own: 3.2-023 => not compatible (possibly other incompatible interface)
Example	ver=3.2-023

9.30 serviceName (nam)

Name	serviceName
Code	nam
Description	Name of the service
Validation	Any printable character, length > 0 and ≤ 32Byte.
Comment	The service name is an abstract name and represents the logical address of the service. In order to allow message routing the name must be unique in scope of the entire SC network. Service names must be agreed at the application level and are stored in the SC configuration.
Example	nam=P01_RTXS_RPRWS1

9.31 sessionId (sid)

Name	sessionId
Code	sid
Description	Unique identification of the session

Validation	Known session, length ≥ 0 and ≤ 256Byte
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	sid=cdc50b36-1fc4-4f9e-8430-d2e3d7284d9d

9.32 sessionInfo (sin)

Name	sessionInfo
Code	sin
Description	Optional 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	sin=SNBZHP - TradingClientGUI 10.2.7

10 Glossary

Client

Piece of an application consuming services and initiating actions.

Server

Piece of an application providing services to clients.

Service

Abstract unit of work provided by the server and delivered to the client in order to implement a specific functionality. SC supports session, publishing and file services.

Session

Temporary allocation of a dedicated server to a client. Session ensures information flow between a client and the allocated server. SC supports request/response sessions and subscription sessions.

Call

A call represents a pair of a request and response.

Command

A command dispatches specific actions on a server according to the incoming request.

Request

Data structure created by the Client or SC in order to initiate an information exchange. It may contain one or more messages.

Response

Data structure created by the Server or SC in order to deliver the requested information. It may contain one or more messages.

Message

Basic transport instrument to exchange information between client, SC and the server. It belongs to a request or to a response.

Message Part

Message part is a piece of a large message. Large message is splittet into parts.

Composite

Data structure prepared to hold all message parts of a large message

Registry

Common list of known objects that ensures their uniqueness, organized in a way to find them easily.

Requester

Piece of code in SC or client initiating the information exchange

Responder

Piece of code in SC or server delivering the requested information

Connection

Network communication between client and SC or SC and the server

Endpoint

Network communication part on SC or server

Appendix A - Message Header Matrix

		actualMask (ams)	appErrorCode (aec)	appErrorText(aet)	bodyType (bty)	cacheId (cid)	cacheExpoirationDateTime (ced)	cascadedMask (cma)	cascadedSubscriptionId(csi)	compression (cmp)	echoInterval (eci)	immediateConnect (imc)	ipAddressList (ipl)	keepaliveInterval (kpi)	localDateTime (ldt)	messageInfo (min)	messageSequenceNumber (msn)	messageType (mty)	mask (msk)	maxConnections (mxc)	maxSessions (mxs)	noData (nod)	noDataInterval (noi)	operationTimeout (oti)	portNr (pnr)	rejectSession (rej)	remoteFileName (rfn)	scErrorCode (sec)	scErrorText (set)	scVersion (ver)	serviceName (nam)	sessionId (sid)	sessionInfo (sin)
ATTACH	REQ														x			x												x			
	RES														x			x										E	E				
DETACH	REQ																	x															
	RES																	x										E	E				
INSPECT	REQ				x					o			x					x															
	RES				x					o			x					x										E	E				
MANAGE	REQ				x					o			x					x															
	RES				x					o			x					x										E	E				
CLN_CREATE_SESSION	REQ				o					o	x		x				x	x						x						x		o
	RES		o	o	o					o							x	x								o		E	E		I	x	
SRV_CREATE_SESSION	REQ				o					o			x				x	x						x						x	x	o
	RES		o	o	o					o							x	x								o					I	x	
CLN_DELETE_SESSION	REQ																x	x						x						x	x	o
	RES																x	x										E	E		I	x	
SRV_DELETE_SESSION	REQ																x	x						x						I	x	o
	RES																x	x													I	x	
SRV_ABORT_SESSION	REQ																	x						x				x	x		I	x	
	RES																	x													I	x	
REGISTER_SERVER	REQ											o		x	x			x		x	x				x					x	x		
	RES														x			x										E	E		I		
CHECK_REGISTRATION	REQ																	x													x		
	RES																	x										E	E		x		
DEREGISTER_SERVER	REQ																	x													x		
	RES																	x										E	E		I		
CLN_EXECUTE	REQ/PRQ				o	o				o						o	x	x						x							x	x	
	RES/PRS		o	o	o	o	o			o						o	x	x										E	E		x	x	
	PAC					o											x	x						x							x	x	
SRV_EXECUTE	REQ/PRQ				o					o						o	x	x						x							x	x	
	RES/PRS		o	o	o	o	o			o						o	x	x													x	x	
	PAC																x	x						x							x	x	
ECHO	REQ																	x						x							x	x	
	RES																	x										E	E		x	x	
FILE_DOWNLOAD	REQ																	x						x			x				x	x	
	RES																	x									x	E	E		x	x	
FILE_UPLOAD	REQ																	x						x			x				x	x	
	RES																	x									x	E	E		x	x	
FILE_LIST	REQ																	x						x							x	x	
	RES									o								x										E	E		x	x	
CLN_SUBSCRIBE	REQ				o			o	o	o			x				x	x	x				x	x						x		o
	RES		o	o	o					o							x	x								o		E	E		I	x	
SRV_SUBSCRIBE	REQ				o					o			x				x	x	x					x							x	x	o
	RES		o	o	o					o							x	x								o					I	x	
CLN_CHANGE_SUBSCRIPTION	REQ				o			o	o	o			x				x	x	x					x							I	x	o
	RES		o	o	o					o							x	x								o		E	E		I	x	
SRV_CHANGE_SUBSCRIPTION	REQ	x			o					o			x				x	x	x					x							I	x	o
	RES		o	o	o												x	x								o					I	x	
CLN_UNSUBSCRIBE	REQ																x	x						x							I	x	o
	RES																x	x										E	E		I	x	
SRV_UNSUBSCRIBE	REQ																x	x						x							I	x	o
	RES																x	x													I	x	
RECEIVE_PUBLICATION	REQ/PAC																x	x													x	x	
	RES/PRS				o					o						o	x	x	I			o						E	E		I	x	
PUBLISH	REQ/PRQ				o					o						o	x	x	x												x		
	RES																	x										E	E		I		
	PAC																	x													I		

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

Headline Keys:
REQ => request
RES => response
PRQ => part request (large message)
PRS => part response (large message)
PAC => part acknowledge (large message)
KRQ => keepalive request
KRS => keepalive response
EXC => exception

Appendix B – Error Codes and Messages

Code ranges:

- 400 – 499 => Client
- 500 – 599 = > Server
- 600 – 699 => ServiceConnector

Errors are derived from HTTP error codes definitions. (http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html)

Code	*Internal Mnemonic*	*Text*
400	BAD_REQUEST	Bad request. The request could not be understood by the server due to malformed syntax.
404	NOT_FOUND	Not found.
408	REQUEST_TIMEOUT	Request Timeout. The client did not produce a request within the time that the server was prepared to wait.
420	HV_ERROR	Validation error occured.
	HV_WRONG_SC_VERSION_FORMAT	Invalid sc version format.
	HV_WRONG_SC_RELEASE_NR	Invalid sc release nr.
	HV_WRONG_SC_REVISION_NR	Invalid sc revision nr.
	HV_WRONG_SCMP_VERSION_FORMAT	Invalid scmp version format.
	HV_WRONG_SCMP_VERSION_NR	Invalid scmp version nr.
	HV_WRONG_SCMP_RELEASE_NR	Invalid scmp release nr.
	HV_WRONG_LDT	Parsing localDateTime failed.
	HV_WRONG_SCMP_RELEASE_NR	Invalid scmp release nr.
	HV_WRONG_LDT	Parsing localDateTime failed.
	HV_WRONG_IPLIST_FORMAT	Invalid iplist format.
	HV_WRONG_MAX_SESSIONS	Invalid maxSessions field.
	HV_WRONG_ECHO_TIMEOUT	Invalid echoTimeout field.
	HV_WRONG_ECHO_INTERVAL	Invalid echoInterval field.
	HV_WRONG_PORTNR	Invalid portNr field.
	HV_WRONG_KEEPALIVE_INTERVAL	Invalid keepalive interval field.
	HV_WRONG_NODATA_INTERVAL	Invalid not data interval field.
	HV_WRONG_MASK	Invalid mask.
	HV_WRONG_SESSION_INFO	Invalid session info field.
	HV_WRONG_SERVICE_NAME	Invalid service name field.
	HV_WRONG_MESSAGE_INFO	Invalid message info field.
	HV_WRONG_MESSAGE_ID	Invalid message id field.
	HV_WRONG_SESSION_ID	Invalid session id field.
	HV_WRONG_SC_ERROR_CODE	Invalid sc error code field.
	HV_WRONG_SC_ERROR_TEXT	Invalid sc error text field.
	V_WRONG_CONFIGURATION_FILE	Invalid configuration file.
500	SERVER_ERROR	Server error.
504	GATEWAY_TIMEOUT	Gateway Timeout. The server, while acting as a gateway or proxy, did not receive a timely response from the upstream server specified by the URI.
600	SC_ERROR	Service Connector error.
601	NO_FREE_SERVER	No free server.
602	SERVER_ALREADY_REGISTERED	Server already registered for the service.
606	FRAME_DECODER	Not possible to decode frame, scmp header line wrong.
610	CONNECTION_EXCEPTION	Connection error.

Index

No index entries found.