STABILIT
Intelligente Informatik

SC

Service Connector

SC API proposal for C programming language

SC_1_SCMP-V1.0_E (Version V2.4)

This document presents a proposal for Application Programming Interface (API) for the C programming language.

This Document has been created with Microsoft Word 2003 (11) with template file C:\STABILIT\STANDARD\TEMPLATES\S_REP_E.DOT and printed at 18 November 2010 16:35.

Identification

Project:	SC
Title:	Service Connector
Subtitle:	SC API proposal for C programming language
Version:	V2.4
Reference:	SC_1_SCMP-V1.0_E
Classification:	Confidential
Keywords:	Concepts, Communication, Protocol, Programming Interface
Comment:	This document presents a proposal for Application Programming Interface (API) for the C programming language.
Author(s):	STABILIT Informatik AG Jan Trnka, Daniel Schmutz, Joel Traber
Approval (Reviewed by):	Signature ..... Jan Trnka
Audience:	Project team, Eurex IT management, Review team
Distribution:	Project team, SIX development team
Filename	c:\stabilit\projects\eurex\sc\documents\sc_0_api_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	Separated from SCMP description
16.6.2010	V2.4	Jan Trnka	Adapted to the latest Java Interface and SCMP

Table of Contents

1 PREFACE 4
1.1 Purpose & Scope of this Document 4
1.2 Definitions & Abbreviations 4
1.3 External References 4
1.4 Typographical Conventions 4
1.5 Outstanding Issues 5

2 INTRODUCTION 6

3 SERVICE MODEL 7
3.1 Session Services 7
3.2 Publishing Services 8
3.3 File Services 9
3.4 HTTP Proxy Services 9

4 CONTEXT HIERARCHY 10

5 PROGRAMMING INTERFACE 11
5.1 Connect 12
5.2 Disconnect 12
5.3 Get Connection Context Parameter 12
5.4 Set Connection Context Parameter 13
5.5 Timeout Handler (callback) 13
5.6 Error Handler (callback) 13
5.7 Create Session 14
5.8 Delete Session 14
5.9 Get Service Context Parameter 14
5.10 Send And Receive 15
5.11 Subscribe 15
5.12 Change Subscription 16
5.13 Unsubscribe 16
5.14 Message Receiver (callback) 16
5.15 Register Service 17
5.16 Deregister Service 17
5.17 Publish 18
5.18 Session Handler (callback) 18
5.19 Message Handler (callback) 19

6 GLOSSARY 20

APPENDIX A 21

INDEX 22

Tables

Table 1 Abbreviations & Definitions 4
Table 2 External references 4
Table 3 Typographical conventions 4

Figures

Figure 1 Communication Layers....6
Figure 2 Synchronous Request/Response....7
Figure 3 Asynchronous Request/Response....8
Figure 4 Asynchronous Subscribe / Publish....8
Figure 5 Context Hierarchy....10

1 Preface

1.1 Purpose & Scope of this Document

The SC project does not implement API for other language but java. In order to keep similar terminology and granularity of the API this document presents a proposal for the C programming language. It uses using as much analogy to the Java API as possible to ensure the C and java developer will understand each other.

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

1.2 Definitions & Abbreviations

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

Table 1 Abbreviations & Definitions

1.3 External References

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

Table 2 External references

1.4 Typographical Conventions

Convention	Meaning
text in italics	features not implemented in the actual release
`text in Courier font`	code example
[phrase]	In syntax diagrams, indicates that the enclosed values are optional
{ phrase1 | 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 in this document version:

- Handling of large messages. Parameters to control the message flow are missing
- Server for multiple services not supported / not verified
- Flag for message compression not implemented
- Flag for immediate server connection not implemented
- Session event not clearly structured. Should be enumerator.

2 Introduction

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.

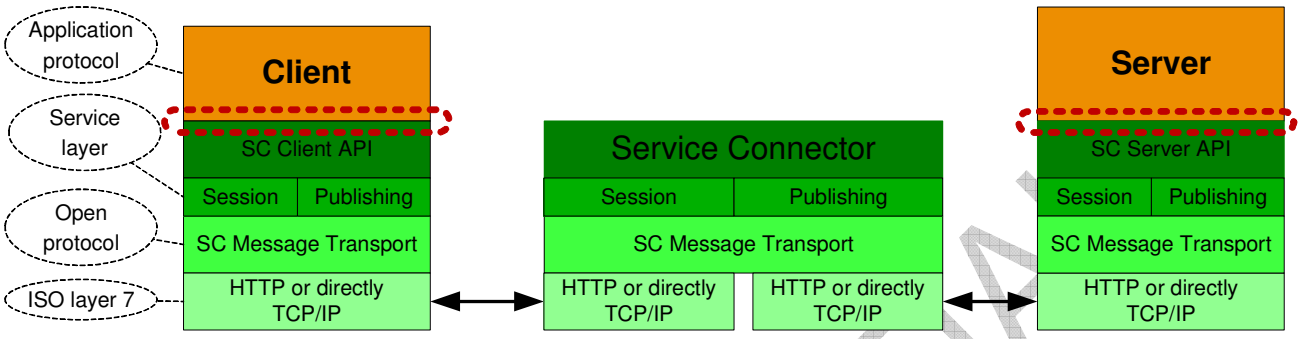
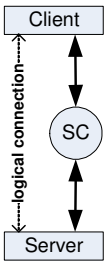


Figure 1 Communication Layers

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.

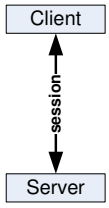
3 Service Model



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

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

3.1 Session Services



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

Synchronous

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

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

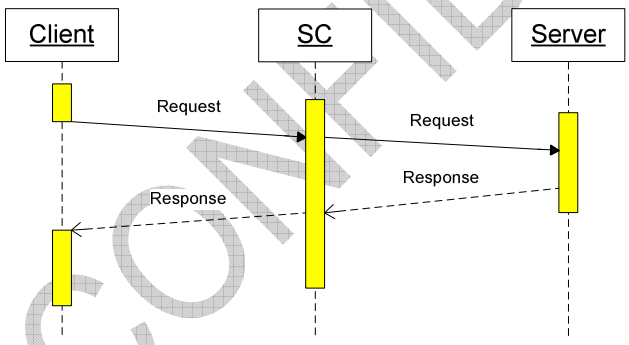


Figure 2 Synchronous Request/Response

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

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

Asynchronous

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

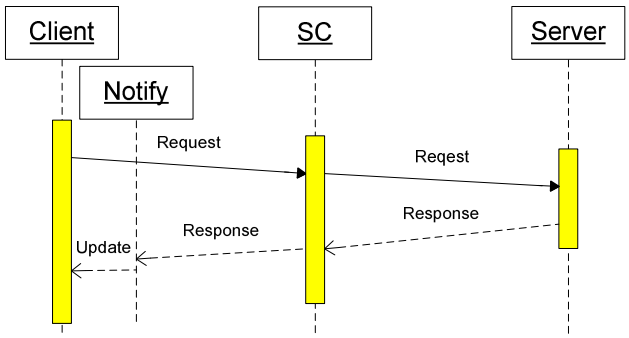
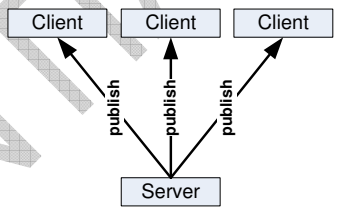


Figure 3 Asynchronous Request/Response

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

3.2 Publishing Services



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

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

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

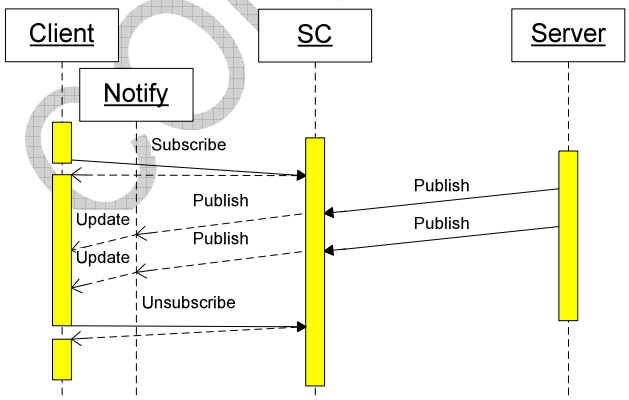


Figure 4 Asynchronous Subscribe / Publish

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

3.3 File Services

This SC service provides API for these file operations:

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

3.4 HTTP Proxy Services

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

4 Context hierarchy

Each object interfacing the application has an individual context. The contexts are hierarchically organised.

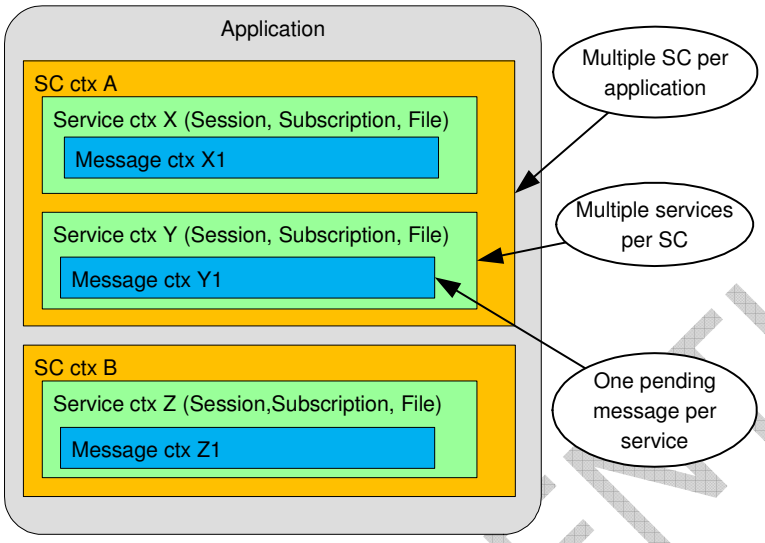


Figure 5 Context Hierarchy

The application may connect to multiple SCs and so must hold a context for each connection. Within a SC the application can use multiple services at the same time. Each service has its own context. Within one service only one pending message at any time is allowed.

5 Programming Interface

In order to make the C method names unique, a general prefix **SC_** has been introduced.

The function names correspond with the method names in java. The contexts correspond roughly with the java objects.

Client

The semantic of the session client interface has the following sequence:

1. SC_Connect (establish SC context)
2. SC_CreateSession A (establish service A context)
3. SC_SendAndReceive (to service A)
   ...
4. SC_CreateSession B (establish service B context)
5. SC_SendAndReceive (to service B)
   ...
6. SC_DeleteSession A (delete service A context)
7. SC_DeleteSession B (delete service B context)
8. SC_Disconnect (delete SC context)

The semantic of the publishing client interface has the following sequence:

1. SC_Connect
2. SC_Subscribe A
   ... receiving messages via Receiver A from service A
3. SC_Subscribe B
   ... receiving messages via Receiver A from service A and Receiver B from service B
4. SC_ChangeSubscription A
   ... receiving messages via Receiver A from service A and Receiver B from service B
5. SC_Unsubscribe A
6. SC_Unsubscribe B
7. SC_Disconnect

The session services and publishing services may share the same connection, when they communicate to the same SC.

Server

The semantic of the session server interface has the following sequence:

1. SC_RegisterService
2. ... executing either *SessionHandler* or *MessageHandler*
3. SC_DeregisterService

The semantic of the publishing server interface has the following sequence:

1. SC_RegisterService
2. SC_Publish
   ... publish messages
3. SC_DeregisterService

5.1 Connect

SC_Connect (
OUT SCctx: scContext,
IN String: SCIPaddress,
IN Integer: SCportNumber,
IN String: protocol,
IN Callback: timeoutHandler,
IN Callback: errorHandler)

Connects the client to the SC. From now on keep alive messages may be sent between the client and the SC in order to detect broken connection. This method returns control when the connection to SC is established.
Parameters:

- Handle to the SC context. The SC context must be used in subsequent calls. The client may have multiple SCs connected at the same time. The properties of the SC context can be set with methods described in 5.4.
- IP address or DNS name of the node on which the SC is running
- Port number the SC expects the connection
- Communication protocol to be used (HTTP or TCP/IP)
- Address of the timeout handler that is invoked when an operation timeout expires.
- Address of the error handler that is invoked when an error occurs.

Possible errors:

- SC is not available
- Communication error

5.2 Disconnect

SC_Disconnect (
IN SCctx: scContext)

Disconnects the client from SC. All existing sessions should be deleted and subscriptions cleared before this call. No further communication with SC is possible after this call. The SC context cannot be reused. This method returns the control when the connection is effectively discarded.
Parameters:

- Context of the connection to be destroyed.

Possible errors:

- none

5.3 Get Connection Context Parameter

SC_GetSCcontextParameter (
IN SCctx: scContext,
IN String: parameterName,
OUT String: parameterValue)

Returns various connection parameters. This method gets control back immediately.
Parameters:

- Context of the connection.
- Name of the parameter. Possible parameter names are:
  - IPaddress - IP address or DNS name of the node on which the CS is running.
  - portNumber - Port number the SC took the connection
  - protocol - Communication protocol in use (HTTP or TCP/IP)

- OperationTimeout – Timeout in seconds for the issued operation (default is set in Connect()). Value = 0 means no timeout.
- ReceiverTimeout - Interval in seconds in which keep alive messages are sent to SC. (default = 300). Value = 0 means no keep alive.
- SenderTimeout - Timeout in seconds for receipt of the keep alive response (default = 60)

- Parameter value

Possible errors:

- Unknown parameter

5.4 Set Connection Context Parameter

SC_SetSCcontextParameter (
IN SCctx: scContext,
IN String: parameterName,
IN String: parameterValue)

Set connection parameters. This method returns control when the parameter was set.
Parameters:

- Context of the connection.
- Name of the parameter. Possible parameter names are:
  - OperationTimeout – Timeout in seconds for the issued operation (default = 60). Value = 0 means no timeout.
  - ReceiverTimeout - Interval in seconds in which keep alive messages are sent to SC. (default = 300). Value = 0 means no keep alive.
  - SenderTimeout - Timeout in seconds for receipt of the keep alive response (default = 60)
- New parameter value

Possible errors:

- Unknown parameter

5.5 Timeout Handler (callback)

SC_TimeoutHandler (
OUT SCctx: scContext,
OUT Integer: timeoutType)

Callback that is invoked when a timeout expires
Parameters:

- Context of the connection
- Type of the timeout (operation or keep alive)

5.6 Error Handler (callback)

SC_ErrorHandler (
OUT SCctx: scContext,
OUT Integer: errorCode,
OUT String: errorText)

Callback that is invoked when an error occurs.
Parameters:

- Context of the connection.
- Error code
- Error text

5.7 Create Session

SC_CreateSession (
IN SCctx: scContext,
OUT ServiceCtx: serviceContext,
IN String: serviceName,
IN String: sessionInformation,
OUT Integer: applErrorCode
OUT String: applErrorText)

Creates logical session between client and server. One free server is allocated. On the server the *SessionHandler* callback is invoked with the event = *SessionStart*. This method returns control when the allocated server accepts the new session.
Parameters:

- Context of the SC connection to be used.
- Context of the service. This context handle must be used in subsequent calls. The client may have multiple sessions with the same or with different service.
- Name of the service the client wants to connect to.
- Session information passed to the server
- Optional application error code filled by the server when the session was rejected
- Optional application error text filled by the server when the session was rejected

Possible errors:

- Service does not exist
- Service is not Request/Response type
- No free server is available
- Communication error

5.8 Delete Session

SC_DeleteSession (
IN ServiceCtx: serviceContext)

Terminates the logical session between the client and the server. On server the *SessionHandler* is invoked with the event = *SessionEnd*. This method returns control when the server is de-allocated and the session is deleted.
Parameters:

- Context of the service to be deleted.

Possible errors:

- Communication error

5.9 Get Service Context Parameter

SC_GetServiceCtxParameter (
IN ServiceCtx: serviceContext,
IN String: parameterName,
OUT String: parameterValue)

Returns various service parameters. This method returns the control immediately.
Parameters:

- Existing context of the service.
- Name of the parameter. Possible parameter names are:
  - String: serviceName – name of the service
  - String: serviceType – type of the service
  - ?
- Parameter value

Possible errors:

- Unknown parameter

5.10 Send And Receive

SC_SendAndReceive(
IN ServiceCtx: serviceContext,
IN String: requestMsg,
IN/OUT String: messageInformation,
IN String: cacheID,
IN Timestamp: cacheExpirationDateTime,
OUT String: responseMsg,
OUT Integer: applErrorCode
OUT String: applErrorText)

Sends a synchronous request to the allocated server identified by the service context and waits for the arrival of the response message. On the server the *MessageHandler* callback is invoked and performs the service. This method returns control when the response message from the server arrives.
Parameters:

- Context of the service.
- Message information passed to and from the server (e.g. message type)
- Acceptable cache time to live
- Request message passed to the server
- Optional cache identification (empty = do not look into cache, otherwise look for cached message with this ID)
- Optional cache expiration date time of the message required by the client. When a message is found in cache, it must have expiration date time after this value.
- Response message received from the server
- Optional application error code filled by the server
- Optional application error text filled by the server

Possible errors:

- Communication error

5.11 Subscribe

SC_Subscribe (
IN SCctx: scContext,
OUT ServiceCtx: serviceContext,
IN String: serviceName,
IN String: subscriptionMask,
IN String: subscriptionInformation,
IN Callback: msgReceiver)

Subscribes the client on a subscribe/publish service. When a message is published and matches the client subscription mask, the publish message receiver will be invoked. This method returns control when the subscription becomes effective.
Parameters:

- Context of the existing connection to be used.
- Context created for this service. This context handle must be used in subsequent calls.
- Name of the service the client wants to subscribe to.
- Subscription mask used to designate interest on certain published messages.
- Subscription information passed to the server
- Address of the message receiver that is invoked when a published message arrives.

Possible errors:

- Service does not exist

- Service is not Subscribe/Publish type
- Communication error

5.12 Change Subscription

SC_ChangeSubscription (
IN ServiceCtx: serviceContext,
IN String: subscriptionMask)

Changes the client subscription. Old subscription is discarded. This method returns control when the new subscription becomes effective.

Parameters:
- Context of the service.
- New subscription mask used to designate interest on certain published messages

Possible errors:
- Communication error

5.13 Unsubscribe

SC_Unsubscribe (
IN ServiceCtx: serviceContext)

Deletes the client subscription. From now on the client will not receive published messages from the service any longer. This method returns control when the subscription is effectively deleted in SC.

Parameters:
- Context of the service.

Possible errors:
- Communication error

5.14 Message Receiver (callback)

SC_MessageReceiver (
OUT ServiceCtx: serviceContext,
OUT String: messageInformation,
OUT String: publishedMsg,
OUT String: msgMask)

Routine that receives the published messages. This method gets control when the message arrives.

Parameters:
- Context of the service to which the subscription has been done
- Message information filled by the server
- Message published by the server
- Effective mask of the message

Possible errors:
- Communication error

5.15 Register Service

SC_RegisterService (
OUT SCctx: scContext,
IN String: SCIPaddress,
IN Integer: SCportNumber,
OUT ServiceCtx: serviceContext,
IN String: serviceName,
IN Integer: acceptPortNumber,
IN Integer: maxSessions,
IN Callback: sessionHandler,
IN Callback: messageHandler)

Connects the server to the SC and registers it for a service. It must be ready to accept connection on its own IP address and the acceptPortNumber. The SC will establish as many connections to it as specified by maxSessions. Depending on the service type the server can now be assigned to a session and receive messages from the client or will receive subscription notifications and can publish data to the clients.
The server may register only once for one service but do this several times for different services.

Depending on the implementation this method can either return control immediately and handle all incoming messages in a designated handler or can declare a session and message handler and return control at server exit.
Parameters:

- Handle to the SC context. The SC context must be used in subsequent calls. The client may have multiple SCs connected at the same time. The properties of the SC context can be set with methods described in 5.4.
- IP address or DNS name of the node on which the SC is running
- Port number the SC expects the connection
- Context of the service. This context handle must be used in subsequent calls.
- Name of the service this server will serve
- Port number on which this server accepts the connection from SC
- Maximal number of concurrent sessions this server will serve.
- Address of the session handler that is invoked when a session event occurs.
- Address of the message handler that receives the client messages, implements the service and sends responses back to the client.

Possible errors:

- Service does not exist
- Communication error
- Timeout expired

5.16 Deregister Service

SC_DeregisterService (
IN SCctx: scContext,
IN ServiceCtx: serviceContext)

Deregisters the server from the service. The server will no longer receive messages for this service. If the server has an active session, it will get the event *SessionAbort* from SC first.

Parameters:

- Handle to the SC context.
- Context of the service

Possible errors:

- Service does not exist
- Service is not request/response type

- Communication error
- Timeout expired

5.17 Publish

SC_Publish(
IN ServiceCtx: serviceContext,
IN String: messageMask,
IN String: messageInformation,
IN String: publishMsg)

Sends a publish message to a service. This message is then distributed to the subscribed clients. On client the *PublishMessageReceiver* is invoked and gets this message. This method returns control when the message was accepted by SC, but before it was distributed or received by the clients.
Parameters:

- Context of this service.
- Mask designating the message contents.
- Message information passed to the client
- Message to be published to subscribed clients

Possible errors:

- Communication error

5.18 Session Handler (callback)

SC_SessionHandler (
OUT SCctx: scContext,
OUT ServiceCtx: serviceContext
OUT String: event,
OUT String: sessionInformation,
OUT String: IPaddressList
IN Boolean: rejectFlag,
IN Integer: applErrorCode,
IN String: applErrorText)

Server method that receives session control messages from SC.
Parameters:

- Context of the connection.
- Context of the service.
- Event that occurred. The possible events are:
  - *SessionStart* – the client has connected and a logical session was established. The session information contains the information passed by the client.
  - *SessionEnd* – the client has disconnected from this service
  - *SessionAbort* – the session was aborted, due to reasons explained in session information.
  - *Subscribe* – the client subscribes for this service.
  - *Unsubscribe* – the client unsubscribes for this service.
  - *ChangeSubscribe* – the client changes the subscription.
- Session information passed by the client (e.g. used for authentication)
- IP addresses on the path between the client and the server (always pairs)
- Reject flag passed back to SC and the client. True means the server rejects this event.
- Optional application error code passed to the client when the reject flag is true.
- Optional application error text passed to the client when the reject flag is true.

Possible errors:

- Communication error

5.19 Message Handler (callback)

SC_MessageHandler (
OUT SCctx: scContext,
OUT ServiceCtx: serviceContext
OUT String: requestMsg,
IN/OUT String: messageInformation,
IN String: responseMsg,
IN String: cacheID,
IN Timestamp: cacheExpirationDateTime
IN Integer: applErrorCode,
IN String: applErrorText)

Server routine that implements the service.
Parameters:

- Context of this connection.
- Optional routing information passed by the client (e.g. message type)
- Request message information passed from and to the client
- Response message returned to the client
- Optional cache ID (empty = message will be not cached, otherwise identification of the cached message)
- Optional expiration date time of the message in the cache.
- Optional application error code passed to the client
- Optional application error text passed to the client

Possible errors:

- Communication error

6 Glossary

Glossary item
glossary text

Appendix A

Index

No index entries found.