Document Title	Testability Protocol and Service Primitives
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	778
Document Classification	Auxiliary
Document Status	Final
Part of AUTOSAR Product	Acceptance Tests for Classic Platform
Part of Product Release	1.1.0

Document Change History		
Release	**Changed by**	**Change Description**
1.1.0	AUTOSAR Release Management	• Initial release

Disclaimer

Advice for users

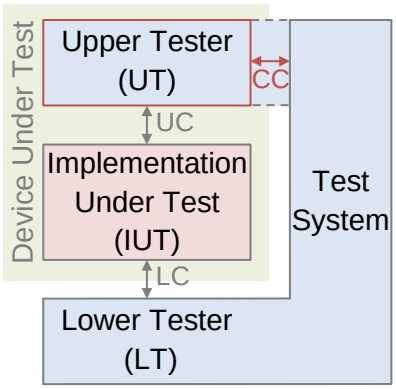
AUTOSAR specifications may contain exemplary items (exemplary reference models, "use cases", and/or references to exemplary technical solutions, devices, processes or software).

Any such exemplary items are contained in the specifications for illustration purposes only, and they themselves are not part of the AUTOSAR Standard. Neither their presence in such specifications, nor any later documentation of AUTOSAR conformance of products actually implementing such exemplary items, imply that intellectual property rights covering such exemplary items are licensed under the same rules as applicable to the AUTOSAR Standard.

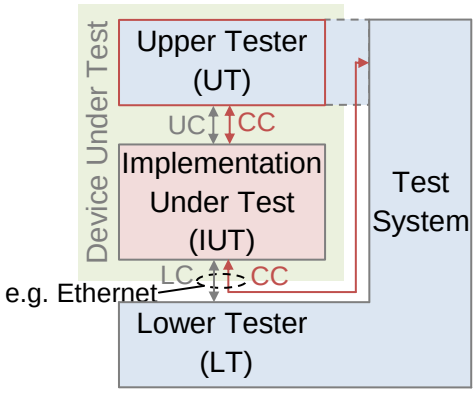
Table of Contents

1 Introduction and Functional Overview

This document details the specification of a communication control protocol with the objective of triggering service primitives (Service Primitives) that imply actions or observations on an implantation under test (IUT). To trigger the actions and observations a testability module/upper tester (UT) that implements the service primitives is located inside the device under test (DUT). The control communication using this protocol takes place on the control channel (CC) between Test System and UT. The actions and observations are exercised through the upper interface that the IUT exposes to its upper layers, the upper test channel (UC). The actions are intended to cause the IUT to communicate with the lower tester (LT) on the lower test channel (LC), wherein the test system verifies the IUT behavior. The test system can also stimulate the IUT to negative scenarios in order to validate the robustness of the IUT. There are several ways to setup the test environment.



Logic setup of the test environment



Scheme of the test environment using the control channel though the IUT itself

2 Acronyms and Abbreviations

Acronyms and abbreviations which have a local scope and therefore are not contained in the AUTOSAR glossary.

Abbreviation / Acronym:	***Description:***
IUT	Implementation Under Test
SP	Service Primitive (for triggering actions or observations on the IUT)
UT	Upper Tester (Part of TS that contains the SPs located within the DUT on top of the IUT)
TSB	Test Stub (same as UT)
TM	Testability Module (same a UT)
LT	Lower Tester (Part of TS located outside the DUT on bottom of the IUT)
UC	Upper Test Channel (channel between UT and IUT within the DUT)
LC	Lower Test Channel (channel between LT and IUT that can be accessed from outside the DUT)
CC	Control Channel (The channel between TS and UP used to call SPs that can be accessed from outside the DUT)
TS	Test System (The system that contains the test cases and control for UT and LT)
EVB	Event Bit (Protocol field that is set in case of an event)
GID	Group Identifier (Protocol field: determines a group of services)
PID	Service Primitive Identifier (Protocol field: determines a service)
TID	Type Identifier (Protocol field: to determine the message type)
RID	Result Identifier (Protocol field: similar to a Return Error Code)
DUT	Device Under Test (contains the UT and IUT that is tested)

3 Related Documentation

In this chapter lists all related documentation.

3.1 Input documents

[1] AUTOSAR SOME/IP Protocol Specification
AUTOSAR_PRS_SomeIPProtocol.pdf

[2] AUTOSAR Standard Datatypes
AUTOSAR_SWS_StandardTypes.pdf

3.2 Related Standards and Norms

No related standards

3.3 Related specification

Thus, the specification AUTOSAR SOME/IP Protocol Specification [1] shall be considered as additional and required specification for the testability protocol

4 Constraints and Assumptions

4.1 Limitations

Although the testability protocol format is compatible to the SOME/IP protocol the message exchange behavior is different. “Request/response” communication is supported but extended by optional notification events. There is no “fire and forget” or “publish/subscribe” communication.

A secure mechanism to prevent unauthenticated access to service primitives is not part of this document but should be realized.

4.2 Applicability to car domains

There are no known dependencies to certain car domains.

5 Intended context and applicability of protocol

5.1 Dependencies to other protocol layers

The testability protocol will most likely be used on top of UDP or TCP.

5.2 Dependencies to other standards and norms

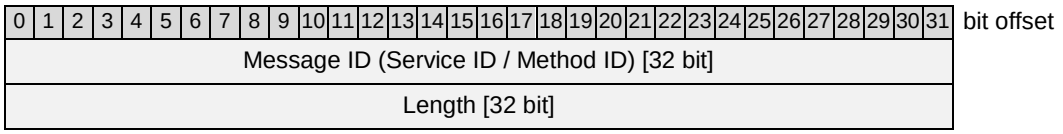
The testability protocol format is conform to the SOME/IP protocol [1] and can be configured and used with the same.

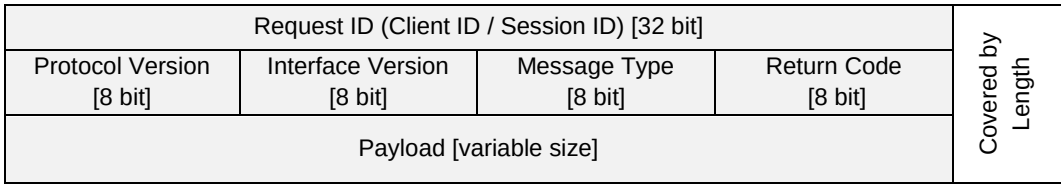
6 Protocol Specification

6.1 Message Format and Protocol Fields

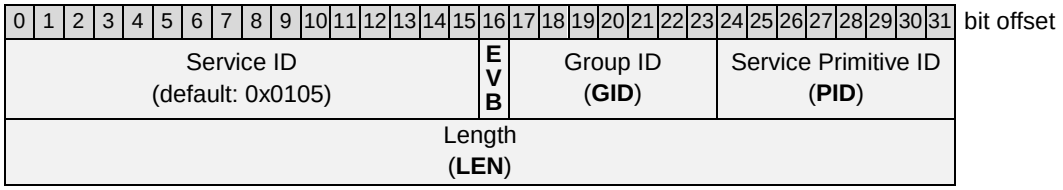
The message format and serialization format is derived from the SOME/IP standard. The protocol fields GID, PID, LEN, RID, DAT are used to select, control and get feedback from service primitives. Those fields may be used independently from the protocol.

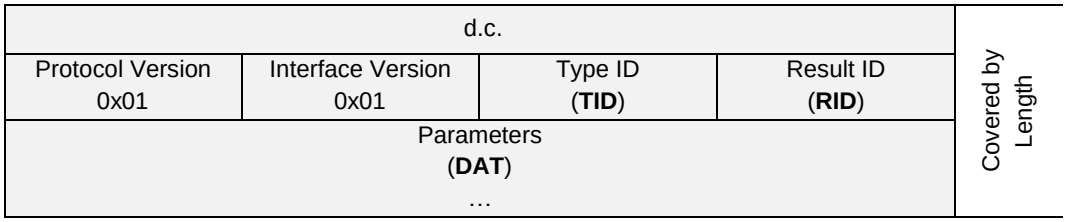
SOME/IP Message Format





Testability Message Format





("d.c" = don't care)

Field	*Name*	*Description*
SID	Service ID	defining the Testability Service: default 0x0105 (configurable)
TID	Message Type ID	Selects request, response or event (see 6.2 Message Exchange)
EVB	Event Bit	This bit is set for event messages (see 6.2 Message Exchange)
GID	Service Group ID	used to group service primitives (see 6.9 Service Groups)
PID	Service Primitive ID	select or identify a service primitive (see 6.10 Service Primitives)

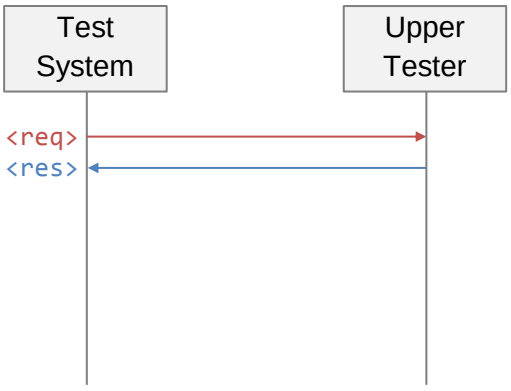
Field	*Name*	*Description*
RID	Result ID	signals the outcome of a request (see 6.8 Result IDs)
LEN	Data Length	amount of following bytes (8 bytes + amount parameter bytes)
DAT	Parameter Data	optional parameters (see 6.7 Data Types)

For all Message Types the Protocol Version field must have a constant value of 0x01 and the Interface Version field must have a constant value of 0x01. The Request ID containing the Client ID and Session ID must be ignored.

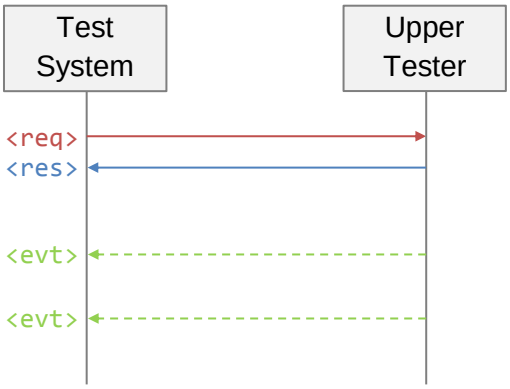
6.2 Message Exchange

The message exchange is based on a simple request response mechanism. Every request is followed by a response message immediately to indicate the success of the request. Therefore a non-blocking behavior is required by service primitives, meaning they do not implement any event or wait criteria in between their request and response. To support such behavior some service primitives may trigger one or more event messages when active. Service primitives can be terminated or switched to an inactive state calling END_TEST. The Message types can be interpreted as follows:

Message	*TID*	*Description*
Request	0x00	corresponds to a non-blocking function call
Response	0x80	corresponds to a non-blocking function return that is always followed after a request
Event	0x02	corresponds to a callback function call (**EVB** set to 1)



Simple Service Primitive not providing event parameters



Service Primitive providing event parameters (0…n events)

6.3 States of Service Primitives

Service primitives switch to an active state when called and might stay in this state until a certain condition applies or their task has simply finished. While in active state a service primitive might trigger events. A service primitive will always return to an inactive state[1] in case END_TEST has been called.

6.4 Default Behavior

Some service primitives require a default behavior even if not called and active.

Service Primitive	*Default behavior*
RECEIVE_AND_FORWARD	Received bytes of data will be counted on socket basis starting with 0 in case the SP is in an inactive state. In the moment the SP goes back to the inactive phase, the counter will be reset to zero.

6.5 Constraints

When using or implementing the protocol the following contains have to be considered:

[PRS_TPSP_00001] [Service primitive calls of groups other than GENERAL are only valid in between the calls of START_TEST and END_TEST.]()

[PRS_TPSP_00002] [A response or event will always send to the requester of a service primitive triggering the same.]()

[PRS_TPSP_00003] [A request is only allowed to send after the response of the previous request that was received, expect for the END_TEST request.]()

6.6 Extensibility

It is allowed to add non-standard service primitives to existing groups or to add non-standard groups. IDs for non-standard service primitives (PIDs) and non-standard service groups (GIDs) will be assigned invers and counted backwards starting with 0xFF, 0xFE… and so forth for PIDs or 0x7F, 0x7E… and so forth for GIDs. It is not allowed to assign an already existing standard PID or GID to a non-standard extension.

[1] Inactive state: the phase prior the first call of a SP or the phase between the point in time the SP has fished their task or END_TEST has been called and the next call of the service primitive.

6.7 Data Types and Format

The basic AUTOSAR data types [2] and additionally variable length types of unsigned 8-bit arrays are supported as defined by the SOME/IP standard [1]. All parameters are transferred within the payload field of a request, response or event message and must be conform to the data types defined in this chapter. Parameters do not need further formalization in order to allow the data exchange between Tester and Service Primitives. Both, the tester and the service primitive share the knowledge about used parameters, their order, and data types.
As for SOME/IP the on-wire format is big endian, MSB first and the Upper Tester adapts the data types to the endianness and bit-order of the used platform.

6.7.1 Boolean

Name	*Bits*	*Range*
bool	8	0, 1 … 255 [0x00, 0x01 - 0xFF] (false, true)

6.7.2 Unsigned

Name	*Bits*	*Range*	
uint8	8	0 … 255	[0x00 … 0xFF]
uint16	16	0 … 65535	[0x00 … 0xFFFF]
uint32	32	0 … 4294967295	[0x000000 … 0xFFFFFFFF]
uint64	64	0 … 18446744073709551615	[0x00000000000000 … 0xFFFFFFFFFFFFFFFF]

6.7.3 Signed

Name	*Bits*	*Range*	
sint8	8	-128 … 127	[0x80 … 0x7F]
sint16	16	-32768 … 32767	[0x8000 … 0x7FFF]
sint32	32	-2147483648 … 2147483647	[0x80000000 …0x7FFFFFFF]
sint64	64	-9223372036854775808 … 922337203685477580	[0x8000000000000000 … 0x7FFFFFFFFFFFFFFF]

6.7.4 Floating Point

Name	*Bits*	*Range*	
float32	32	$1.17549 \cdot 10^{-38}$ – $3.40282 \cdot 10^{38}$	IEEE-754
float64	64	$2.22507 \cdot 10^{-308}$ – $1.79769 \cdot 10^{308}$	IEEE-754

6.7.5 Variable Length

A variable length type always contains a ***uint16*** variable named *n* to indicate the length of following elements of type ***uint8***.

Name	*Definition*	*Number of Bytes*
vint8	n · uint8	2 + 0 … 65535

6.7.5.1 IP Addresses

IP addresses can be placed without any convention into a variable length type. The type of IP address is recognizable by its length (4 for IPv4 or 16 for IPv6).
Example: An IPv4 Address (192.168.0.1) will result in:

Length n (uint16)	*Data (n · uint8)*
0x00 0x04	0xC0 0xA8 0x00 0x01

Example: An IPv6 Address (2001:DB9::1) will result in:

Length n (uint16)	*Data (n · uint8)*
0x00 0x10	0x20 0x01 0x0D 0xB9 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01

6.7.5.2 Text

A default text will be encoded using UTF-8 with BOM and null termination and is placed into a variable length data type.
Example: A Text “AbCd€” will result in:

Length n (uint16)	*Data (n · uint8)*		
	BOM	Text	Termination
0x00 0x0B	0xEF 0xBB 0xBF	0x41 0x62 0x43 0x64 0xE2 0x82 0xAC	0x00

6.8 Result IDs

The Result Identifier is represented by the RID field in the protocol header of a response message. The list of result IDs may be extended in further Versions of this document.

6.8.1 Standard Results

This range is used for the standard AUTOSAR return types (refer to [2]).

Name	*ID*	*Description*
E_OK	0x00	The service primitive has performed successfully
E_NOK	0x01	General error (same as E_NOT_OK)
0x02 - 0x7F		Range of AUTOSAR specific error codes (returns of API function calls other than E_OK or E_NOT_OK)

6.8.2 Testability Specific

Name	*ID*	*Description*
E_NTF	0xFF	The requested service primitive was not found
E_PEN	0xFE	The Upper Tester or a service primitive is pending
E_ISB	0xFD	Insufficient buffer size

6.8.3 Service Primitive Specific

Name	*ID*	*Description*
E_ISD	0xEF	Invalid socket ID
E_UCS	0xEE	Unable to create socket or no free socket
E_UBS	0xED	Unable to bind socket, port taken
E_INV	0xEC	Invalid Input or Parameter

6.9 Service Groups

Service primitives are grouped in service groups. While service primitives define the functionality, a service group defines the functional context. The Group Identifier is represented by the 7-Bit GID field in the protocol header.

Group Name	*GID*
GENERAL	0x00
UDP	0x01
TCP	0x02
ICMP	0x03
ICMPv6	0x04
IP	0x05
IPv6	0x06
DHCP	0x07
DHCPv6	0x08
ARP	0x09
NDP	0x0A

6.9.1 General Group

Group	GENERAL
GID	0x00
Description	Contains general service primitives that must be provided by the Upper Tester

Service Primitives Name	*PID*	*Type*
GET_VERSION	0x01	mandatory
START_TEST	0x02	mandatory
END_TEST	0x03	mandatory

6.9.2 UDP Group

Group	UDP
GID	0x01
Description	Group of UDP related service primitives

Service Primitives		
Name	*PID*	*Type*
CLOSE_SOCKET	0x00	mandatory
CREATE_AND_BIND	0x01	mandatory
SEND_DATA	0x02	mandatory
RECEIVE_AND_FORWARD	0x03	mandatory
CONFIGURE_SOCKET	0x06	mandatory
SHUTDOWN	0x07	extension

6.9.3 TCP Group

Group	TCP
GID	0x02
Description	Group of TCP related service primitives

Service Primitives		
Name	*PID*	*Type*
CLOSE_SOCKET	0x00	mandatory
CREATE_AND_BIND	0x01	mandatory
SEND_DATA	0x02	mandatory
RECEIVE_AND_FORWARD	0x03	mandatory
LISTEN_AND_ACCEPT	0x04	mandatory
CONNECT	0x05	mandatory
CONFIGURE_SOCKET	0x06	mandatory
SHUTDOWN	0x07	extension

6.10 Service Primitives

The Service Primitive Identifier is represented by the 8-Bit PID field in the protocol header. Depending on a service group a service primitive (SP) may have a different set of parameters. The separation between the different parameter sets for each group is done by creation of separate and atomic service primitives using the same service identifier (PID) but a different GID and set of parameters.
The following table gives an overview on the service primitives supported by this specification and corresponding service groups:

SP Name	*PID*	GENERAL	UDP	TCP
GET_VERSION	0x01	m		
START_TEST	0x02	m		
END_TEST	0x03	m		
CLOSE_SOCKET	0x00		m	m
CREATE_AND_BIND	0x01		m	m
SEND_DATA	0x02		m	m
RECEIVE_AND_FORWARD	0x03		m	m
LISTEN_AND_ACCEPT	0x04			m
CONNECT	0x05			m
CONFIGURE_SOCKET	0x06		m	m
SHUTDOWN	0x07		e	e

(m= mandatory, o = optional, e = extension)

6.10.1 Get Version

SP	GET_VERSION
Group	GENERAL
PID	0x01
Description	This SP will return the testability protocol version of the used protocol and service primitive implementation. The minor version is changed in case of additions to the service primitives or the protocol that do not break backward compatibility. The major version is changed in case of changes on existing SPs and parameters or the introduction of new service groups. The current version is V1.0.

Response Parameters			
Name	*Type {Range}*	*Group*	*Description*
majorVer	uint16	GENERAL	Major version (X of “X.Y”)
minorVer	uint16	GENERAL	Minor version (Y of “X.Y”)

6.10.2 Start Test

SP	START_TEST
Group	GENERAL
PID	0x02
Description	The purpose of this SP is to have a defined entry tag in trace at the point in time the test case was started. This SP does not have any request parameters.

6.10.3 End Test

SP	END_TEST
Group	GENERAL
PID	0x03
Description	The purpose of this SP is to reset the Upper Tester. All sockets of the test channel will be closed, counters are set to the default value, buffers are cleared and active service primitives will be terminated. Another purpose of this SP is to have a defined entry tag in trace at the point in time the test case was stopped. The parameters may be ignored by the testability module.

Request Parameters			
Name	*Type {Range}*	*Group*	*Description*
tcId	uint16	GENERAL	The test case ID going to be terminated Example: 42 (0x2A) of test case ATS_DIAG_42
tsName	vint8 {0-100}	GENERAL	The test suite name (UTF-8 encoded with BOM and null termination → see 6.7.5.2 Text) Example: “ATS_DIAG” of test case ATS_DIAG_42

6.10.4 Close Socket

SP	CLOSE_SOCKET
Group	UDP/TCP
PID	0x00
Description	Closes a socket.
SP Results	E_ISD

Request Parameters			
Name	*Type {Range}*	*Group*	*Description*
socketId	uint16	UDP/TCP	Socket that should be closed
abort	bool	TCP	When true: closes the socket immediately, the stack is not waiting for outstanding transmissions and acknowledgements

6.10.5 Create and Bind

SP	CREATE_AND_BIND
Group	UDP/TCP
PID	0x01
Description	Creates a socket and optionally binds this socket to a port and a local IP address.
SP Results	E_UCS, E_UBS

Request Parameters			
Name	*Type {Range}*	*Group*	*Description*
doBind	bool	UDP/TCP	true: bind will be performed false: no bind will be performed
localPort	uint16	UDP/TCP	Local port to bind (0xFFFF: PORT_ANY)
localAddr	vint8 {4,16}	UDP/TCP	Local address (4:IPv4, 16:IPv6) Any IP: If all address bytes are zero The domain is selected by the type of address

Response Parameters			
Name	*Type {Range}*	*Group*	*Description*
socketId	uint16	UDP/TCP	The resulting socket ID. (Only valid in case the return code was E_OK)

6.10.6 Send Data

SP	SEND_DATA
Group	UDP/TCP
PID	0x02
Description	Sends data to a target. Please note: because of the non-blocking behavior of Service Primitives a positive response does NOT signal the success of the transmission, but the success of issuing the transmission.
SP Results	E_ISD

Request Parameters			
Name	*Type {Range}*	*Group*	*Description*
socketId	uint16	UDP/TCP	Local socket used to perform the transmission
totalLen	uint16	UDP/TCP	Total length: repeat data up to that length (in bytes). In case the value of totalLen is smaller than the length of data, the full length of data will be transmitted.
destPort	uint16	UDP	Destination port
destAddr	vint8 {4, 16}	UDP	Destination address (n = 4:IPv4, 16:IPv6)
data	vint8 {0-65535}	UDP/TCP	Data to transmit

6.10.7 Receive and Forward

SP	RECEIVE_AND_FORWARD
Group	UDP/TCP
PID	0x03
Description	Data will be forwarded to the test system that will be received after the call of this SP. The amount of forwarded data per received datagram (UDP) or bulk of stream data (TCP) can be limited using maxFwd. The original length of this data unit can be obtained by fullLen. The process will repeat itself (active phase) until the maximum amount of data defined by maxLen was received or END_TEST was called. **UDP**: No further requirements. (see 6.12.2 UDP Receive and Count) **TCP**: When called all data that was received prior the call of this SP will be consumed[2] and discarded in order to open the TCP receive window. All data that is received during the active phase of this SP will be consumed up to the maximum amount of data defined by maxLen. (see 6.12.4 TCP Client Receive and Forward)
SP Results	E_ISD

Request Parameters			
Name	*Type {Range}*	*Group*	*Description*
socketId	uint16	UDP/TCP	The Socket selected for forwarding
maxFwd	uint16	UDP/TCP	Maximum length of payload to be forwarded per event
maxLen	uint16	UDP/TCP	Maximum count of bytes to receive over all (0xFFFF: limitless)

Response Parameters			
Name	*Type {Range}*	*Group*	*Description*
dropCnt	uint16	UDP/TCP	Count of received and dropped bytes within the inactive phase of this SP. Will reset to zero when called.

Event Parameters			
Name	*Type {Range}*	*Group*	*Description*
fullLen	uint16	UDP/TCP	The full length of available data in bytes
srcPort	uint16	UDP	Source port of the received datagram
srcAddr	vint8 {4,16}	UDP	Source address of the received datagram
payload	vint8 {0-maxFwd}	UDP/TCP	The payload that was received

[2] *consumed*: obtaining the received data from the TCP/IP stack or notify the Stack that the data has been processed

6.10.8 Listen and Accept

SP	LISTEN_AND_ACCEPT
Group	TCP
PID	0x04
Description	Marks a socket as listen socket that will be used to accept incoming connections. Whenever a new connection was established this SP provides the socket ID of the new connection together with the listen socket, client port, and address in an event.
SP Results	E_ISD

Request Parameters			
Name	*Type {Range}*	*Group*	*Description*
listenSocketId	uint16	TCP	Local socket that should listen
maxCon	uint16	TCP	Maximum number of connections allowed to establish

Event Parameters			
Name	*Type {Range}*	*Group*	*Description*
listenSocketId	uint16	TCP	Listen socket where the connection was established
newSocketId	uint16	TCP	Socket of the newly created connection
port	uint16	TCP	Client Port
address	vint8 {4, 16}	TCP	Client IP address (n=4:IPv4, n=16:IPv6)

6.10.9 Connect

SP	CONNECT
Group	TCP
PID	0x05
Description	Triggers a TCP connection to a remote destination.
SP Results	E_ISD

Request Parameters			
Name	*Type {Range}*	*Group*	*Description*
socketId	uint16	TCP	TCP socket that should connect to a remote destination
destPort	uint16	TCP	Port of the remote destination
destAddr	vint8 {4, 16}	TCP	IP address of the remote destination (n=4:IPv4, n=16:IPv6)

6.10.10 Configure Socket

SP	CONFIGURE_SOCKET
Group	UDP/TCP
PID	0x06
Description	This SP is used to select and set parameters that can be configured on a socket basis. More parameters may be supported in following versions of this document or by non-standard extensions (Parameter IDs starting with 0xFFFF, 0xFFFE… and so forth).
SP Results	E_ISD, E_INV

Request Parameters			
Name	*Type {Range}*	*Group*	Description
socketId	uint16	UDP/TCP	socket that should be configured
paramId	uint16	UDP/TCP	Selects the parameter to be configured: 0x0000 (1 Byte): TTL/Hop Limit 0x0001 (1 Byte): Priority (traffic class/DSCP & ECN)
paramVal	vint8 {0-65535}	UDP/TCP	The value of the selected parameter that must match the corresponding length.

6.11 Standard Extensions

The set of service primitives defined in Chapter 6.10 is aligned with the supported functionalities of the AUTOSAR TCP/IP Module (Revision 4.2.1). However there are other well-known socket API's, like the Berkeley Socket API, that define some further functions. In order to make the Testability Protocol futureproof, especially with respect to the AUTOSAR Adaptive Platform, additional service primitives for the most important functions, shall be specified to ensure the compatibility and interoperability with such TCP/IP implementations.

6.11.1 Shutdown

SP	SHUTDOWN
Group	UDP/TCP
PID	0x07
Description	Shuts down a socket.

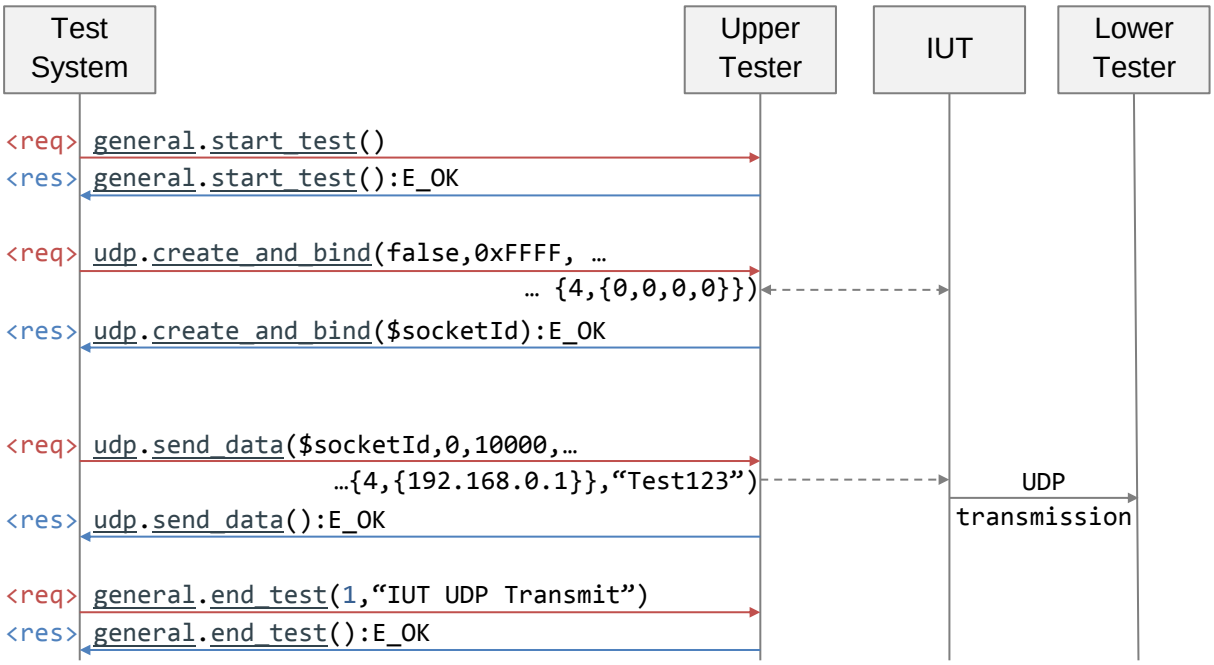
Request Parameters			
Name	*Type {Range}*	*Group*	*Description*
socketId	uint16	UDP/TCP	Socket that should shutdown
typeId	uint8	UDP/TCP	Selects the way the socket is shutdown: 0x00: further reception will be disallowed 0x01: further transmission will be disallowed. 0x02: further transmission and reception will be disallowed.

6.12 Use Cases

For the use cases described in this chapter the Lower Tester shall have the IPv4 address 192.168.0.1, a test server port for UDP 10000 and TCP 20000. The IUT shall have the IPv4 address 192.168.0.2 and will be configured by the test system during the test execution. Requests are symbolized by <req>, responds by <res> and events by <evt>.

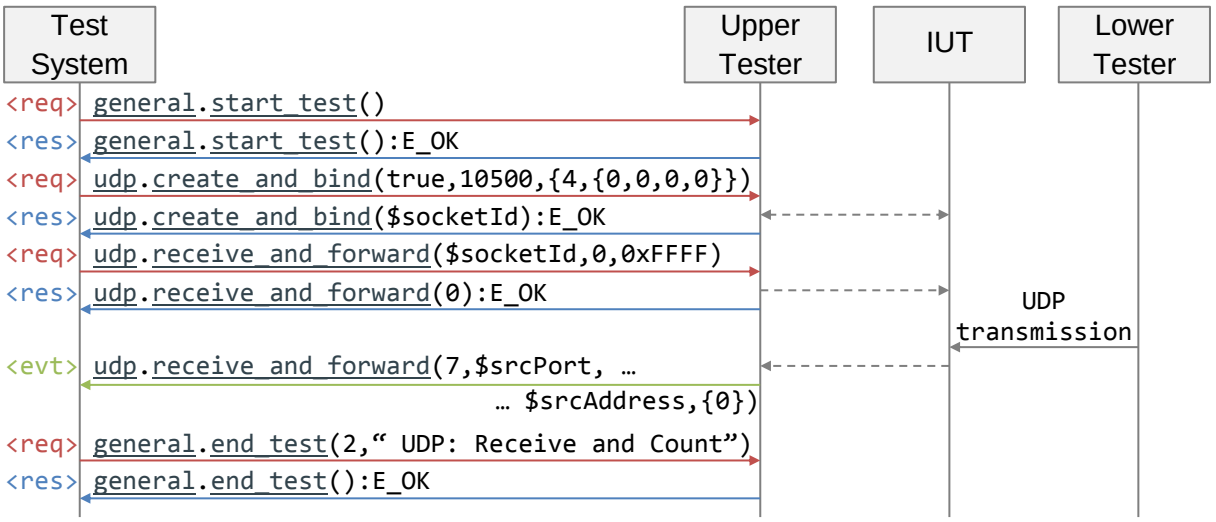
6.12.1 UDP Transmit

The test system creates a UDP socket without any specific bind and issues a transmission from the IUT to the Lower Tester.



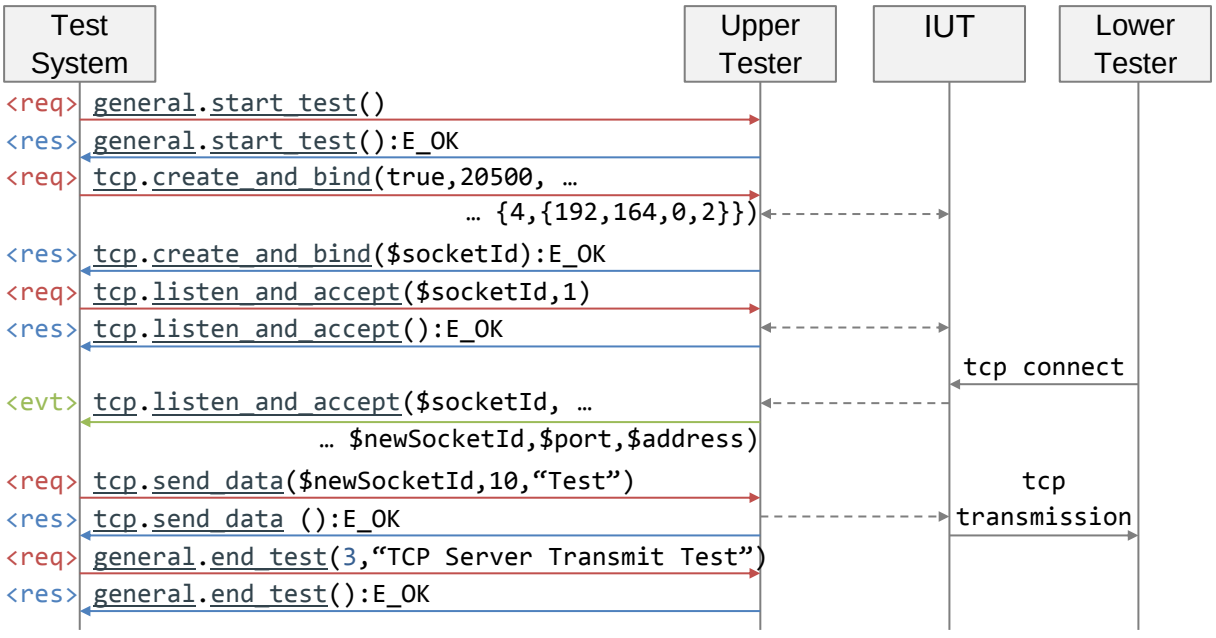
6.12.2 UDP Receive and Count

The test system creates a UDP socket and binds it to the local port 10500 valid for every IP interface. The test system calls RECEIVE_AND_FORWARD and requests the Upper Tester to receive limitless but not to forward the data to the test system. The Upper Tester returns a drop count of zero, meaning there was no previous data received on this socket. The lower tester transmits seven bytes to the IUT. The data will be consumed and dropped but the byte count will be forwarded to the test system.



6.12.3 TCP Server Transmit

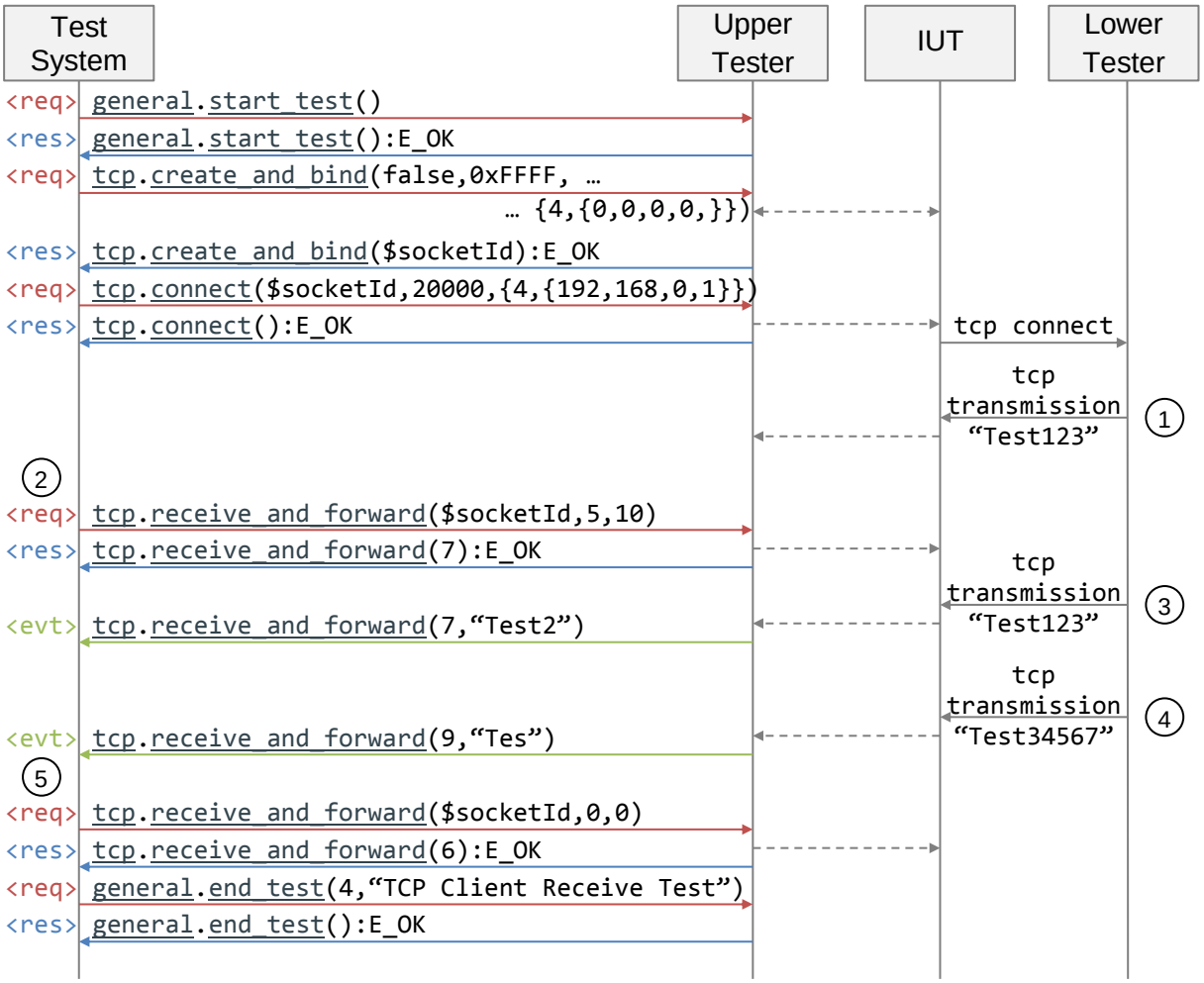
The test system sets up a TCP server (IP: any, Port: 20500), issues a TCP connection from the lower test to the IUT Server and issues a data transmission from the server to the client (lower tester).



6.12.4 TCP Client Receive and Forward

The test system sets up a TCP client and issues a connection from the IUT to the server that is the lower tester. For reasons of comprehensibility the following steps are listed below.

1. The lower tester transmits seven bytes to the IUT. The received data will be ignored by the Upper Tester and not consumed but counted.
2. The test system calls RECEIVE_AND_FORWARD and requests the Upper Tester to receive 10 bytes but only to forward at maximum five bytes per received bulk of steam data. Previously received bytes will be consumed and dropped but, the count will be returned to the test system and will then be reset to zero.
3. The lower tester transmits another seven bytes to the IUT. The data will be consumed, five bytes will be forwarded to the test system and two bytes will be dropped.
4. The lower tester transmits nine bytes to the IUT. Three bytes will be consumed and forwarded to the test system and six bytes will not be consumed but dropped and counted.
5. The test system calls RECEIVE_AND_FORWARD again and requests to receive and forward nothing. Previously received bytes will be consumed and the count will be returned to the test system.



7 Changes to Previous Versions

Since this is the initial release, there are no changes to previous versions.