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 Standard	Acceptance Tests for Classic Platform
Part of Standard Release	1.2.0

Document Change History

Date	Release	Changed by	Change Description
2016-12-15	1.2.0	AUTOSAR Release Management	• New: ○ Service Primitives for ICMP, ICMPv6, IP, IPv6, ETH, DHCP, DHCPv6, PHY ○ Result IDs for TCP API Error Codes according to IETF RFC793 ○ Result ID E_IIF • Changed/Fixed: ○ Parameter in Service Primitive GET_VERSION ○ Moved Result ID E_INV ○ Sequence Diagram Client Receive and Forward step 3 • More Details: ○ Result IDs might be used in event messages too. ○ Sequence Diagram Client Receive and Forward and in Service Primitive RECEIVE_AND_FORWARD ○ Service Primitive CREATE_AND_BIND
2015-10-31	1.1.0	AUTOSAR Release Management	• Initial release

Disclaimer

Advice for users

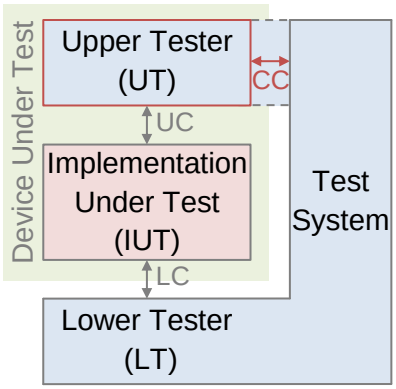
AUTOSAR specifications may contain exemplary items (exemplary reference models, "use cases", and/or references to exemplary technical solutions, devices, processes or software).

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.

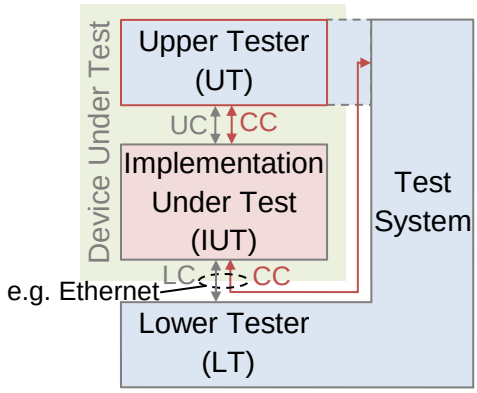
Table of Contents

1 Introduction and Functional Overview

This document details the specification of a communication control protocol with the objective of triggering service primitives (SP) that imply actions or observations on an implementation 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 that is located inside the DUT
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

[3] IETF RFC 793 “TRANSMISSION CONTROL PROTOCOL”

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

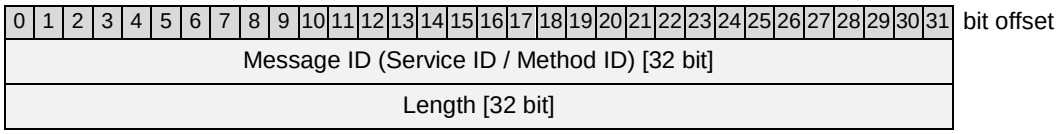
The testability protocol format is conform to the SOME/IP protocol [1] and can be configured and used with the same.

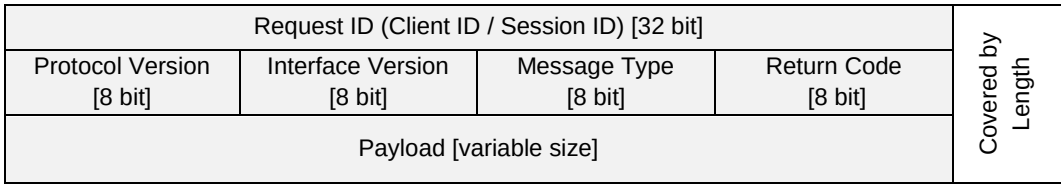
6 Protocol Specification

6.1 Message Format and Protocol Fields

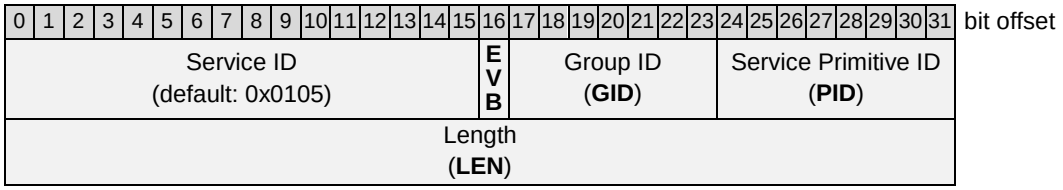
The message format and serialization format is derived from the SOME/IP standard. The protocol fields GID, PID, LEN, RID, DAT are used to select, control and get feedback from service primitives. Those fields may be used independently from the protocol.

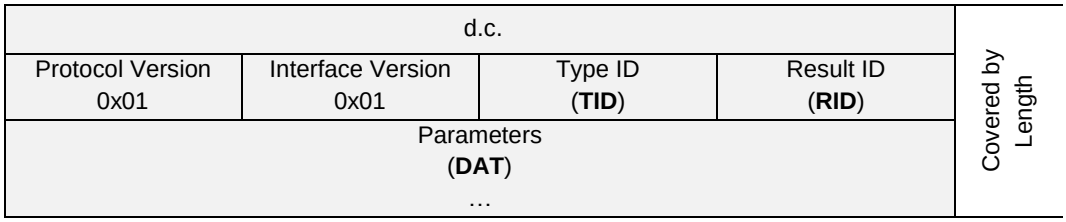
SOME/IP Message Format





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)

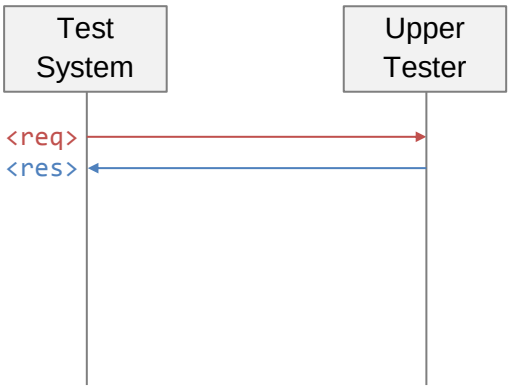
Field	*Name*	*Description*
RID	Result ID	signals the outcome of a request (see 6.8 Result IDs)
LEN	Data Length	amount of following bytes (8 bytes + amount parameter bytes)
DAT	Parameter Data	optional parameters (see 6.7 Data Types)

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.

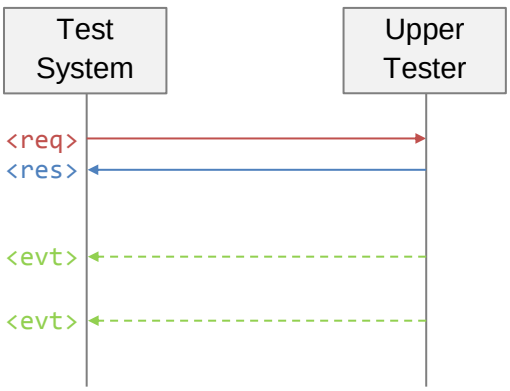
6.2 Message Exchange

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 Type

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

6.7.2 Unsigned Type

Type	*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 Type

Type	*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 Type

Type	*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 Type

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

Type	*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 (vint8). The type of IP address is recognizable by its length (4 for IPv4 or 16 for IPv6).

Type	Definition	Number of Bytes
ip4addr	vint8 n = 4	2 + 4
ip6addr	vint8 n =16	2 + 16
ipxaddr	ip4addr | ip6addr	2 + 4 or 2 + 16

Example: An IPv4 Address (192.168.0.1) will result in:

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

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

Example	
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.

Type	Definition	Number of Bytes
text	vint8 n = 0 | 4 … 65535	2 + 0 or 2 + 4 … 65535

Example: A Text “AbCd€” will result in:

Example			
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 or event 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]).

Result	*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

Result	*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
E_INV	0xFC	Invalid Input or Parameter

6.8.3 Service Primitive Specific

Result	*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_IIF	0xEC	Invalid network or virtual interface
E_TCP_PNA	0xEB	TCP error: “precedence not allowed” [3]
E_TCP_FSU	0xEA	TCP error: “foreign socket unspecified” [3]
E_TCP_ILP	0xE9	TCP error: “connection illegal for this process" [3]
E_TCP_INR	0xE8	TCP error: “insufficient resources" [3]
E_TCP_CAE	0xE7	TCP error: “connection already exists" [3]
E_TCP_COC	0xE6	TCP error: “connection closing" [3]
E_TCP_CNE	0xE5	TCP error: “connection does not exist" [3]
E_TCP_CRE	0xE4	TCP error: “connection reset" [3]
E_TCP_CAT	0xE3	TCP error: “connection aborted due to user timeout" [3]
E_TCP_COR	0xE2	TCP Error: “connection refused” [3]

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	*GID*
GENERAL	0x00
UDP	0x01
TCP	0x02
ICMP	0x03
ICMPv6	0x04
IP	0x05
IPv6	0x06
DHCP	0x07
DHCPv6	0x08
ARP	0x09
NDP	0x0A
ETH	0x0B
PHY	0x0C

6.9.1 General Group

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

6.9.4 ICMP Group

Group	ICMP	
GID	0x03	
Description	Group of ICMPv4 related service primitives	
Service Primitives	*PID*	*Type*
ECHO_REQUEST	0x00	extension

6.9.5 ICMPv6 Group

Group	ICMPv6	
GID	0x04	
Description	Group of ICMPv6 related service primitives	
Service Primitives	*PID*	*Type*
ECHO_REQUEST	0x00	extension

6.9.6 IP Group

Group	IP	
GID	0x05	
Description	Group of IPv4 related service primitives	
Service Primitives	*PID*	*Type*
STATIC_ADDRESS	0x00	extension
STATIC_ROUTE	0x01	extension

6.9.7 IPv6 Group

Group	IPv6	
GID	0x06	
Description	Group of IPv6 related service primitives	
Service Primitives	*PID*	*Type*
STATIC_ADDRESS	0x00	extension
STATIC_ROUTE	0x01	extension

6.9.8 DHCP Group

Group	DHCP	
GID	0x07	
Description	Group of DHCPv4 related service primitives	
Service Primitives	*PID*	*Type*
INIT_DHCP_CLIENT	0x00	extension
STOP_DHCP_CLIENT	0x01	extension
SET_DHCP_OPTION	0x02	extension

6.9.9 DHCPv6 Group

Group	DHCPv6	
GID	0x08	
Description	Group of DHCPv6 related service primitives	
Service Primitives	*PID*	*Type*
INIT_DHCP_CLIENT	0x00	extension
STOP_DHCP_CLIENT	0x01	extension
SET_DHCP_OPTION	0x02	extension

6.9.10 ETH Group

Group	ETH	
GID	0x0B	
Description	Group of Ethernet Interface related service primitives	
Service Primitives	*PID*	*Type*
INTERFACE_UP	0x00	extension
INTERFACE_DOWN	0x01	extension

6.9.11 PHY Group

Group	PHY	
GID	0x0C	
Description	Group of Broadr-Reach physical interface related service primitives	
Service Primitives	*PID*	*Type*
READ_SIGNAL_QUALITY	0x00	mandatory
READ_DIAG_RESULT	0x01	mandatory
ACTIVATE_TEST_MODE	0x02	mandatory
SET_PHY_TX_MODE	0x03	mandatory

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	ICMP	ICMPv6	IP	IPv6	DHCP	DHCPv6	ETH	PHY
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								
ECHO_REQUEST	0x00				e	e						
STATIC_ADDRESS	0x00						e	e				
STATIC_ROUTE	0x01						e	e				
INTERFACE_UP	0x00										e	
INTERFACE_DOWN	0x01										e	
INIT_DHCP_CLIENT	0x00								e	e		
STOP_DHCP_CLIENT	0x01								e	e		
SET_DHCP_OPTION	0x02								e	e		
READ_SIGNAL_QUALITY	0x00											m
READ_DIAG_RESULT	0x01											m
ACTIVATE_TEST_MODE	0x02											m
SET_PHY_TX_MODE	0x03											m

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

6.10.1 Get Version

Service	GET_VERSION		
Group	GENERAL		
PID	0x01		
Definition	This SP will return the testability protocol version of the used protocol and service primitive implementation. The testability protocol version is bound to the TC release version the protocol is based on. The current version is TC1.2.0.		
Response			
Parameter	*Type*	*Group*	*Description*
majorVer	uint16	GENERAL	Major version (X of “X.Y.Z”)
minorVer	uint16	GENERAL	Minor version (Y of “X.Y.Z”)
patchVer	uint16	GENERAL	Minor version (Z of “X.Y.Z”)

6.10.2 Start Test

Service	START_TEST
Group	GENERAL
PID	0x02
Definition	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

Service	END_TEST		
Group	GENERAL		
PID	0x03		
Definition	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			
Parameter	*Type*	*Group*	*Description*
tcId	uint16	GENERAL	The test case ID going to be terminated Example: 42 (0x2A) of test case ATS_DIAG_42
tsName	text	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

Service	CLOSE_SOCKET		
Group	UDP/TCP		
PID	0x00		
Definition	Closes a socket.		
Results	E_ISD, E_TCP_ILP, E_TCP_CNE, E_TCP_CRE(in Event)		
Request			
Parameters	*Type*	*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

Service	CREATE_AND_BIND		
Group	UDP/TCP		
PID	0x01		
Definition	Creates a socket and optionally binds this socket to a port and a local IP address. Note: Some TCP/IP-Stacks may need to know at socket creation time whether it is a client or a server socket. For those kind of implementations the SP may create and return a higher-level ID that maps to the corresponding data needed to create the socket later and the real socket ID once created.		
Result	E_UCS, E_UBS, E_TCP_ILP, E_TCP_INR, E_TCP_PNA		
Request			
Parameter	*Type*	*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	ipxaddr	UDP/TCP	Local address (n=4:IPv4 or n=16:IPv6) Any IP: If all address bytes are zero The domain is selected by the type of address
Response			
Parameter	*Type*	*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

Service	SEND_DATA		
Group	UDP/TCP		
PID	0x02		
Definition	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.		
Result	E_ISD, E_TCP_FSU, E_TCP_ILP, E_TCP_INR, E_TCP_COC, E_TCP_CNE		
Request			
Parameter	*Type*	*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	ipxaddr	UDP	Destination address (n = 4:IPv4, 16:IPv6)
flags	uint8	TCP	Bit 7: reserved Bit 6: reserved Bit 5: URG Bit 4: reserved Bit 3: PSH Bit 2: reserved Bit 1: reserved Bit 0: reserved
data	vint8 {0 … 65535}	UDP/TCP	Data to transmit

6.10.7 Receive and Forward

Service	RECEIVE_AND_FORWARD		
Group	UDP/TCP		
PID	0x03		
Definition	Data that will be received after the call of this SP will be forwarded to the test system. 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 (inactive phase). **UDP**: No further requirements. (see 6.12.2 UDP Receive and Count) **TCP**: In the inactive phase (e.g. prior the first call) all data received will be discarded or ignored. When called all data that was received on the specified socked prior the call of this SP will be consumed[2] 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)		
Results	E_ISD, E_TCP_ILP, E_TCP_INR , E_TCP_COC		
Request			
Parameter	*Type*	*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			
Parameter	*Type*	*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			
Parameter	*Type*	*Group*	*Description*
fullLen	uint16	UDP/TCP	The full length of available data in bytes
srcPort	uint16	UDP	Source port of the received datagram
srcAddr	ip	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

Service	LISTEN_AND_ACCEPT		
Group	TCP		
PID	0x04		
Definition	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.		
Result	E_ISD, E_TCP_ILP, E_TCP_FSU, E_TCP_INR, E_TCP_CAE, E_TCP_CRE(in Event) , E_TCP_PNA		
Request			
Parameter	*Type*	*Group*	*Description*
listenSocketId	uint16	TCP	Local socket that should listen
maxCon	uint16	TCP	Maximum number of connections allowed to establish
Event			
Parameter	*Type*	*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	ipxaddr	TCP	Client IP address

6.10.9 Connect

Service	CONNECT		
Group	TCP		
PID	0x05		
Definition	Triggers a TCP connection to a remote destination.		
Results	E_ISD, E_TCP_PNA, E_TCP_FSU, E_TCP_ILP, E_TCP_INR, E_TCP_CAE, E_TCP_CRE (in Event), E_TCP_CAT (in Event) , E_TCP_PNA, E_TCP_COR (in Event),		
Request			
Parameter	*Type*	*Group*	*Description*
socketId	uint16	TCP	TCP socket that should connect to a remote destination
destPort	uint16	TCP	Port of the remote destination
destAddr	ipxaddr	TCP	IP address of the remote destination

6.10.10 Configure Socket

Service	CONFIGURE_SOCKET		
Group	UDP/TCP		
PID	0x06		
Definition	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).		
Results	E_ISD, E_TCP_ILP, E_TCP_CNE		
Request			
Parameter	*Type*	*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) 0x0002 (1 Byte): IP DF DontFragment 0x0003 (N Bytes): IP Timestamp Option data as stored in the IP header option 4 as described by RFC 791 page 22 0x0004 (1 Byte): IP Type of Service TOS encoded as defined by RFC 791 page 29 (Delay, Throughout, Reliability, Cost, MBZ) 0x0005 (2 Byte): Set MSS MaxSegmentSize (valid values 5001460) 0x0006 (1 Byte): Enable/disable Nagle Algorithm parameter (enabled=1) 0x0007 (1 Byte): Enable/disable the transmission of the UDP checksum (enabled=1)
paramVal	vint8 {0-65535}	UDP/TCP	The value of the selected parameter that must match the corresponding length.

6.10.11 Read Signal Quality

Service	READ_SIGNAL_QUALITY		
Group	PHY		
PID	0x00		
Definition	Returns the current signal quality in percent by reading the value from the related Ethernet transceiver		
Results	E_IIF		
Request			
Parameter	*Type*	*Group*	*Description*
ifName	text	PHY	The name of the network interface (e.g. “eth1.5” or “\Device\NPF_{6F111E2E-41B6-4147-BE6E-101110033111}” or “0”)
Response			
Parameter	*Type*	*Group*	Description
sigQuality	uint8	PHY	Signal quality in percent

6.10.12 Read Cable Diagnostics Result

Service	READ_DIAG_RESULT		
Group	PHY		
PID	0x01		
Definition	Returns the result of the cable diagnostics.		
Results	E_IIF		
Request			
Parameter	*Type*	*Group*	*Description*
ifName	text	PHY	The name of the network interface (e.g. “eth1.5” or “\Device\NPF_{6F111E2E-41B6-4147-BE6E-101110033111}” or “0”)
Response			
Parameter	*Type*	*Group*	Description
diagResult	uint8	PHY	Result of the cable diagnostics: 0x00: Cable diagnostic ok 0x01: Cable diagnostic failed 0x02: Short circuit detected 0x03: Open circuit detected

6.10.13 Activate PHY Test Mode

Service	ACTIVATE_TEST_MODE		
Group	PHY		
PID	0x02		
Definition	Activates a given PHY test mode.		
Results	E_IIF		
Request			
Parameter	*Type*	*Group*	*Description*
ifName	text	PHY	The name of the network interface (e.g. “eth1.5” or “\Device\NPF_{6F111E2E-41B6-4147-BE6E-101110033111}” or “0”)
testMode	uint8	PHY	Test mode to be activated: 0x00: normal operation 0x01: test transmitter droop 0x02: test master timing jitter 0x03: test slave timing jitter 0x04: test transmitter distortion 0x05: test power spectral density (PSD) mask

6.10.14 Set PHY Tx Mode

Service	SET_PHY_TX_MODE		
Group	PHY		
PID	0x03		
Definition	Activates a given transmission mode.		
Results	E_IIF		
Request			
Parameter	*Type*	*Group*	*Description*
ifName	text	PHY	The name of the network interface (e.g. “eth1.5” or “\Device\NPF_{6F111E2E-41B6-4147-BE6E-101110033111}” or “0”)
txMode	uint8	PHY	Transmission Mode to be activated: 0x00: normal operation 0x01: transmitter disabled 0x02: scrambler disabled

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

Service	SHUTDOWN		
Group	UDP/TCP		
PID	0x07		
Definition	Shuts down a socket.		
Results	E_TCP_ILP, E_TCP_CNE, E_TCP_COC		
Request			
Parameter	*Type*	*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.

6.11.2 Interface Up

Service	INTERFACE_UP		
Group	ETH		
PID	0x00		
Definition	Enables an Ethernet interface or virtual interface. This SP is not affecting the persistent configuration.		
Results	E_IIF		
Request			
Parameter	*Type*	*Group*	*Description*
ifName	text	ETH	The name of the network interface (e.g. "eth1.5" or "\Device\NPF_{6F111E2E-41B6-4147-BE6E-101110033111}" or "0")

6.11.3 Interface Down

Service	INTERFACE_DOWN		
Group	ETH		
PID	0x01		
Definition	Disables an Ethernet interface or virtual interface. This SP is not affecting the persistent configuration.		
Results	E_IIF		
Request			
Parameter	*Type*	*Group*	*Description*
ifName	text	ETH	The name of the network interface (e.g. “eth1.5” or “\Device\NPF_{6F111E2E-41B6-4147-BE6E-101110033111}” or “0”)

6.11.4 Static Address

Service	STATIC_ADDRESS		
Group	IP/IPv6		
PID	0x00		
Definition	Assigns a static IP address and Netmask to the given network interface.		
Results	E_IIF		
Request			
Parameter	*Type*	*Group*	*Description*
ifName	text	IP/IPv6	The name of the network interface (e.g. “eth1.5” or “\Device\NPF_{6F111E2E-41B6-4147-BE6E-101110033111}” or “0” depending of the OS)
addr	ipxaddr	IP/IPv6	The subnet for the route
netMask	uint8	IP/IPv6	The subnet mask for the route in CIDR-notation

6.11.5 Static Route

Service	STATIC_ROUTE		
Group	IP/IPv6		
PID	0x01		
Definition	Adds a static route for the network. This SP is not affecting the persistent configuration.		
Results	E_IIF		
Request			
Parameter	*Type*	*Group*	*Description*

ifName	text	IP/IPv6	The name of the network interface (e.g. “eth1.5” or “\Device\NPF_{6F111E2E-41B6-4147-BE6E-101110033111}” or “0” depending of the OS)
subNet	ipxaddr	IP/IPv6	The subnet for the route
netMask	uint8	IP/IPv6	The subnet mask for the route in CIDR-notation
gateway	ipxaddr	IP/IPv6	The gateway IP address for the route

6.11.6 Initialize DHCP Client

Service	INIT_DHCP_CLIENT		
Group	DHCP/DHCPv6		
PID	0x00		
Definition	Initialize the DHCP Client by use of network interface and port.		
Results	E_IIF		
Request			
Parameter	*Type*	*Group*	*Description*
ifName	text	DHCP/ DHCPv6	The name of the network interface (e.g. “eth1.5” or “\Device\NPF_{6F111E2E-41B6-4147-BE6E-101110033111}” or “0” depending of the OS)

6.11.7 Stop DHCP Client

Service	STOP_DHCP_CLIENT		
Group	DHCP/DHCPv6		
PID	0x01		
Definition	Shutdown the DHCP Client by use of network interface and port.		
Results	E_IIF		
Request			
Parameter	*Type*	*Group*	*Description*
ifName	text	DHCP/ DHCPv6	The name of the network interface (e.g. “eth1.5” or “\Device\NPF_{6F111E2E-41B6-4147-BE6E-101110033111}” or “0” depending of the OS)

6.11.8 Set DHCP Option

Service	SET_DHCP_OPTION		
Group	DHCP		
PID	0x02		
Definition	Sets DHCP Client options		
Results	E_IIF		
Request			
Parameter	*Type*	*Group*	*Description*
ifName	text	DHCP	The name of the network interface (e.g. “eth1.5” or “\Device\NPF_{6F111E2E-41B6-4147-BE6E-101110033111}” or “0” depending of the OS)
code	uint8	DHCP	DHCP option code: 51_d [4 Byte]: IP address lease time 57_d [2 Byte]: Maximum message size 61_d [1…* Byte]: Client identifier by name 161_d [6 Byte]: Client identifier by hardware address
value	vint8 {0… 65535}	DHCP	DHCP option value selected by the code parameter using the corresponding byte size

6.11.9 Echo Request

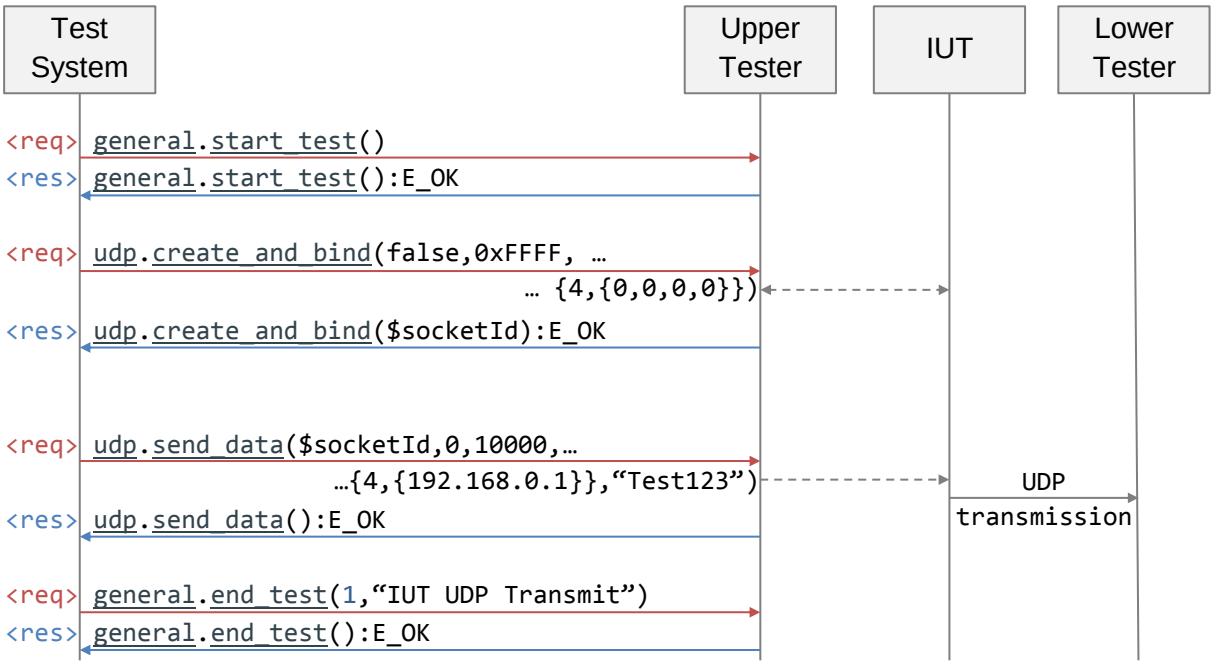
Service	ECHO_REQUEST		
Group	ICMP/ICMPv6		
PID	0x00		
Definition	Issues the transmission of an ICMP Echo Request.		
Results	E_IIF		
Request			
Parameter	*Type*	*Group*	*Description*
ifName	text	ICMP/ ICMPv6	Optional: The name of the network interface (e.g. “eth1.5” or “\Device\NPF_{6F111E2E-41B6-4147-BE6E-101110033111}” or “0” depending of the OS)
destAddr	ipxaddr	ICMP/ ICMPv6	The destination address
data	vint8 {0 … 65535}	ICMP/ ICMPv6	Payload to transmit

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>.

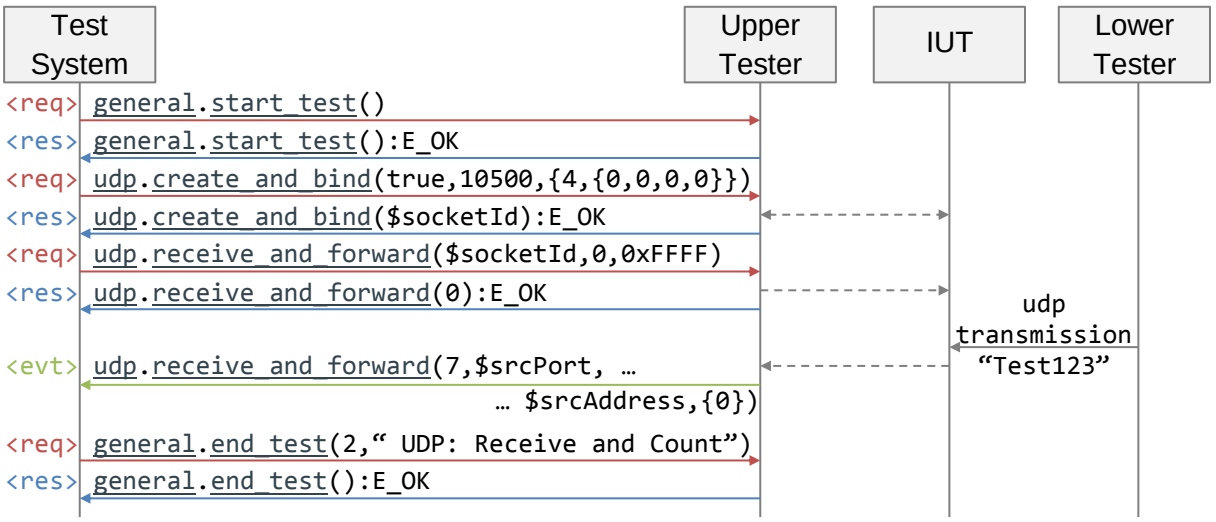
6.12.1 UDP Transmit

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



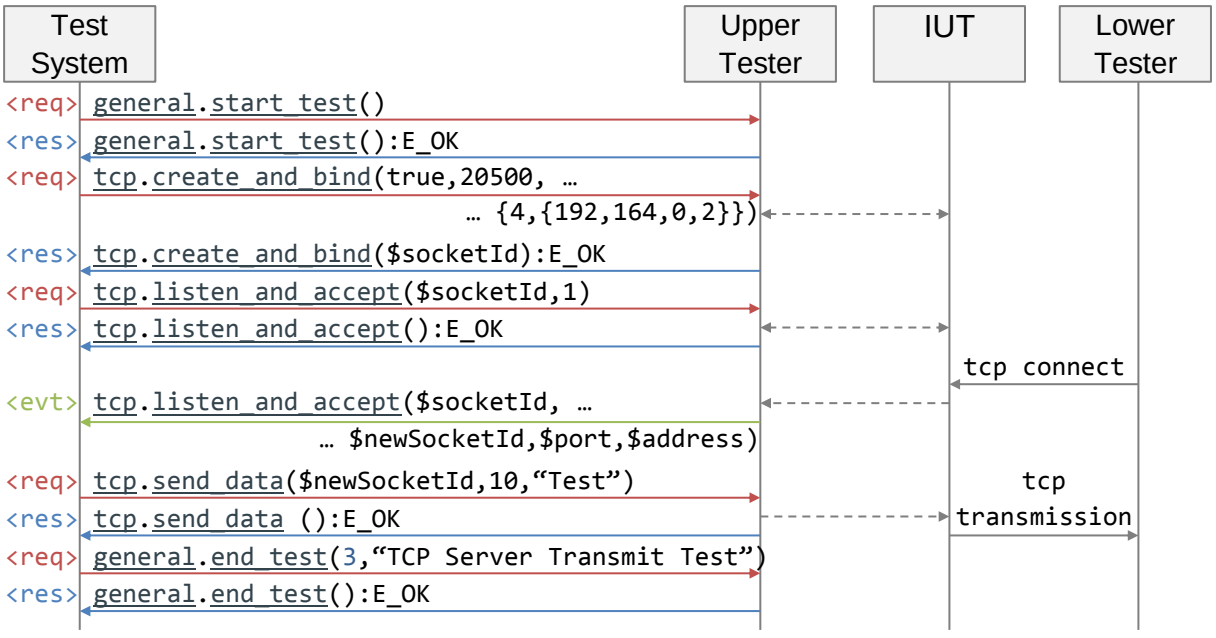
6.12.2 UDP Receive and Count

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

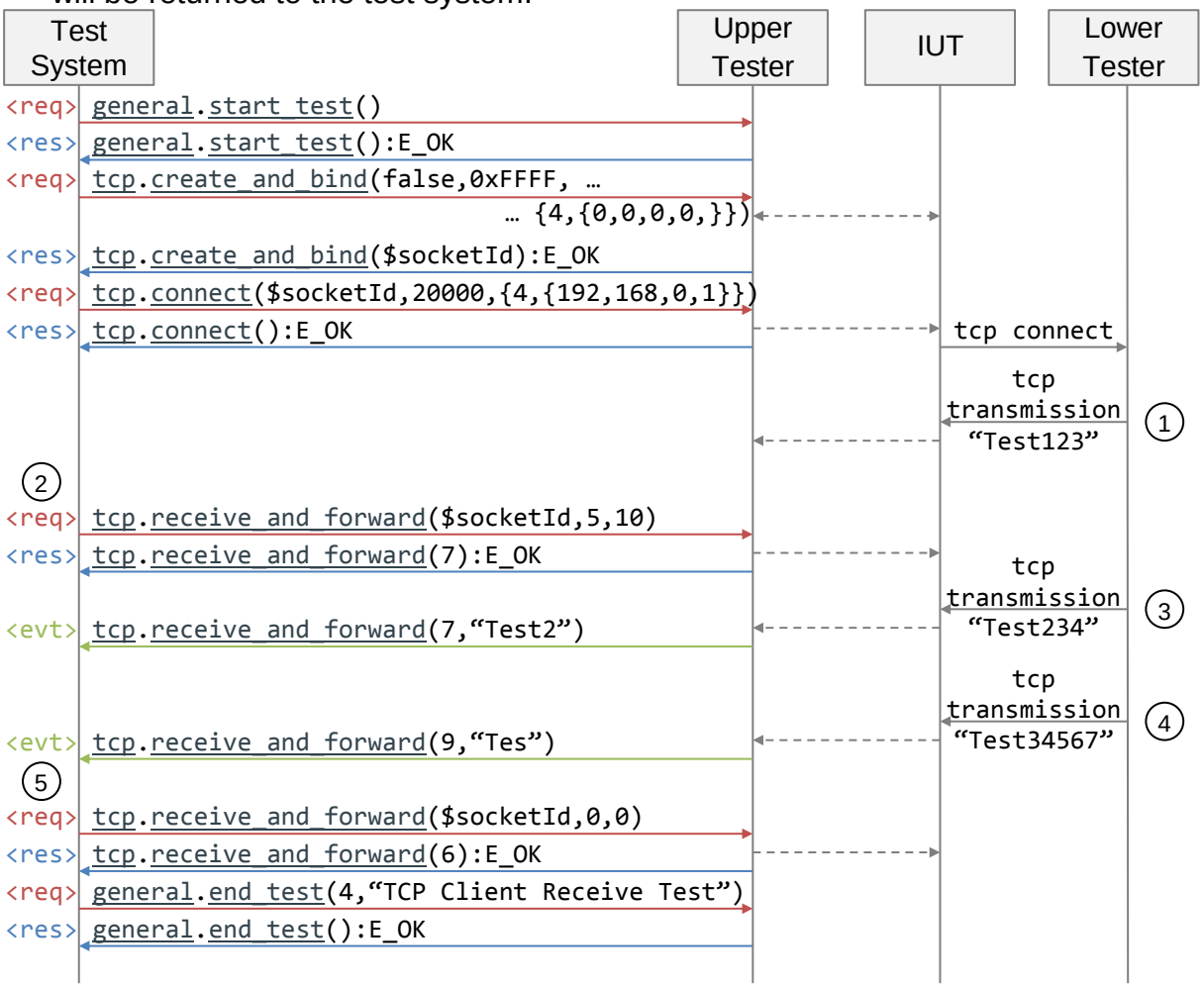
The test system sets up a TCP server (IP: any, Port: 20500), issues a TCP connection from the lower test to the IUT Server and issues a data transmission from the server to the client (lower tester).



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 (the receive window gets smaller).
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 stream data. Previously received bytes will be consumed and dropped (meaning the receive window will get reopened), the byte 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

Changed from TC1.1.0 to TC1.2.0:

- **New** Parameter in Service Primitive GET_VERSION: The version returned is now bound to the current TC release number. A patch version parameter (Z of X.Y.Z) is now contained in the response message.
- **Moved** Result ID E_INV: The Result ID E_INV (Invalid Input or Parameter) has been moved from category “Service Primitive Specific” to category “Testability Specific” in order to indicate that this Result ID is valid for every Service Primitive. The binary expression has been changed from 0xEC to 0xFC. 0xEC might be reused for a different Result ID in future.
- **More Details** in Service Primitive CREATE_AND_BIND: Added a note for TCP/IP-Stacks that need to know at socket creation time whether it is a client or a server socket.
- **New** Result ID E_IIF: Invalid network or virtual interface
- **Bugfix** in Sequence Diagram Client Receive and Forward: correct data transmitted in step 3
- **More Details** in Sequence Diagram Client Receive and Forward and in Service Primitive RECEIVE_AND_FORWARD: If received data is not consumed by the stub the TCP receive window will get smaller. If the data is consumed the TCP receive window will get reopened.
- **New** Service Primitives ECHO_REQUEST, STATIC_ROUTE, INTERFACE_UP, INTERFACE_DOWN , INIT_DHCP_CLIENT, STOP_DHCP_CLIENT, SET_DHCP_OPTION, READ_SIGNAL_QUALITY, READ_DIAG_RESULT, ACTIVATE_TEST_MODE, SET_PHY_TX_MODE
- **New** Service Groups ICMP, ICMPv6, IP, IPv6, ETH, DHCP, DHCPv6, PHY
- **New** Result IDs for TCP API Tests according to IETF RFC793: E_TCP_PNA, E_TCP_FSU, E_TCP_ILP, E_TCP_INR, E_TCP_CAE, E_TCP_COC, E_TCP_CNE, E_TCP_CRE E_TCP_CAT
- **More Details** for the use of Result IDs. Result IDs might be used in event messages too.