OSEK/VDX

COM test procedure

Version 1.0

October, 2nd, 1998

What is OSEK/VDX?

OSEK/VDX is a joint project of the automotive industry. It aims at an industry standard for an open-ended architecture for distributed control units in vehicles.
A real-time operating system, software interfaces and functions for communication and network management tasks are thus jointly specified.
The term OSEK means ”Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug” (Open systems and the corresponding interfaces for automotive electronics).
The term VDX means „Vehicle Distributed eXecutive“. The functionality of OSEK operating system was harmonized with VDX. For simplicity OSEK will be used instead of OSEK/VDX in the document.

OSEK partners:

Adam Opel AG, BMW AG, Daimler-Benz AG, IIIT University of Karlsruhe, Mercedes-Benz AG, Robert Bosch GmbH, Siemens AG, Volkswagen AG.
GIE.RE. PSA-Renault (Groupement d’intérêt Economique de Recherches et d’Etudes PSA-Renault).

Motivation:

- High, recurring expenses in the development and variant management of non-application related aspects of control unit software.
- Incompatibility of control units made by different manufacturers due to different interfaces and protocols.

Goal:

Support of the portability and reusability of the application software by:

- Specification of interfaces which are abstract and as application-independent as possible, in the following areas: real-time operating system, communication and network management.
- Specification of a user interface independent of hardware and network.
- Efficient design of architecture: The functionality shall be configurable and scaleable, to enable optimal adjustment of the architecture to the application in question.
- Verification of functionality and implementation of prototypes in selected pilot projects.

Advantages:

- Clear savings in costs and development time.
- Enhanced quality of the control units software of various companies.
- Standardized interfacing features for control units with different architectural designs.
- Sequenced utilization of the intelligence (existing resources) distributed in the vehicle, to enhance the performance of the overall system without requiring additional hardware.
- Provides absolute independence with regards to individual implementation, as the specification does not prescribe implementation aspects.

OSEK conformance testing

OSEK conformance testing aims at checking conformance of products to OSEK specifications. Test suites are thus specified for implementations of OSEK operating system, communication and network management.

Work around OSEK conformance testing is supported by the MODISTARC project sponsored by the Commission of European Communities. The term MODISTARC means ”Methods and tools for the validation of OSEK/VDX based DISTributed ARChitectures”.

This document has been drafted by MODISTARC members:

Harald Heinecke	BMW AG
Wolfgang Kremer	BMW AG
Didier Stunault	Dassault Electronique
Benoit Caillaud	INRIA
Dirk John	IIIT, Karlsruhe University
Yevgeny Shakuro	Motorola GmbH
Barbara Ziker	Motorola GmbH
Jean-Paul Cloup	Peugeot Citroën S.A.
Jean-Emmanuel Hanne	Peugeot Citroën S.A.
Samuel Boutin	Renault S.A.
Patrick Palmieri	Siemens Automotive SA

TABLE OF CONTENTS

1. Introduction

1.1. Scope

This document specifies a test procedure for services and protocols of the OSEK COM as defined in specification document [4].

This document applies to conformance test suites for testing implementations which claim conformance to the OSEK COM specification. The test procedure consists of a list of test cases building the OSEK COM test suite. A test case consists of a sequence of statements corresponding to one or more test purposes specified in document [2].

1.2. References

[1] OSEK/VDX Conformance Testing Methodology - Version 1.0. - 19 December 1997.

[2] OSEK/VDX - COM test plan - Version 1.0. - July 24th, 1998.

[3] OSEK/VDX Operating System - Version 2.0 - revision 1 - 15 October 1997.

[4] OSEK/VDX Communication - Version 2.1 - revision 1 - 17th June 1998.

[5] OSEK Network Management - Concept and Application Programming Interface-Version 2.50 - 31st of May 1998.

[6] ISO/IEC 9646-1 - Information technology, Open Systems Interconnection, Conformance testing methodology and framework, *part 1 : General Concepts*, 1992.

[7] ISO/IEC 9646-3 - Information technology, Open Systems Interconnection, Conformance testing, methodology and framework, *part 3 : The Tree and Tabular Combined Notation (TTCN),* 1992.

1.3. Abbreviations

API	Application Programming Interface
CF	Consecutive Frame
ECU	Electronic Control Unit
EUT	Equipment Under Test
FC	Flow Control
FF	First Frame
ISO	International Standard Organization
IUT	Implementation Under Test
LSB	Low Significant Bit
LT	Lower Tester
MSB	Most Significant Bit
CAN	Car Area Network

COM	COMmunication
COM PDU	COMmunication - Protocol Data Unit
OS	Operating System
PDU	Protocol Data Unit
PICS	Protocol Implementation Conformance Statement
PIXIT	Protocol Implementation eXtra Information for Testing
SDL	Specification and Description Language
SF	Single Frame
TE	Test Equipment
TMP	Test Management Protocol
TM_PDU	Test Management - Protocol Data Unit
TTCN	Tree and Tabular Combined Notation
UT	Upper Tester
USDT	Unacknowledged and Segmented Data Transfer
UUDT	Unacknowledged and Unsegmented Data Transfer

2. Test environment

2.1. Test architecture

According to the methodology described in document [1], the test architecture for COM conformance is split into two parts:

- the Equipment Under Test (EUT) which encompasses the COM implementation to be tested, also called Implementation Under Test (IUT),
- the Test Equipment (TE) which implements the test suite and is connected to the Equipment Under Test by the network data bus.

The test suite makes up the Lower Tester (LT) which communicates through the Test Management Protocol (TMP) with its counterpart of the EUT called Upper Tester (UT). UT's role is on one hand to perform all actions requested by the LT and on the other hand to send back the information collected at the COM API.

To exchange information with the LT, the UT makes use of the services offered by the COM API. TMP information is encapsulated in the OSEK/COM protocol and occupies the data field of OSEK/COM data frames. It is expressed in terms of application messages called TM_PDUs (Test Management - Protocol Data Units).

During tests execution, the IUT will therefore send and receive two types of PDUs:

- COM PDUs allowing to achieve the test objectives and test IUT's behaviour. User data are not interpreted by the UT or the LT.
- COM PDUs supporting TM_PDUs. User data are meaningful for UT or LT.

During tests execution, TMP_PDUs are exchanged is either direction between LT and UT:

- TM_PDUs are sent by the LT in order to simulate the COM activity of the other network nodes,
- TM_PDUs are received by the LT and analysed in order to determine whether or not the IUT behaviour conforms to the COM specification.

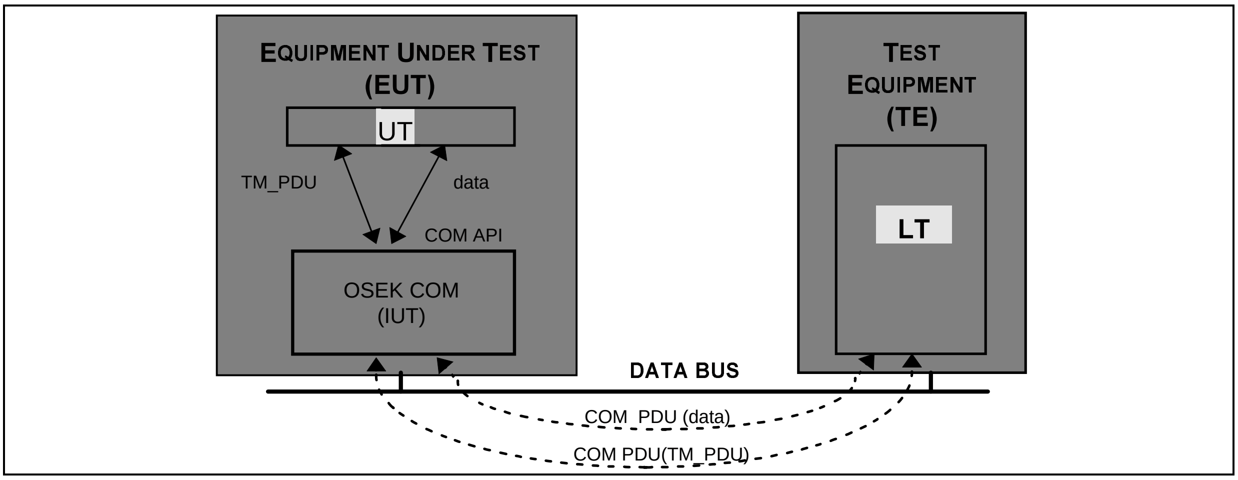

Figure 1 Test architecture for COM conformance

Special TM_PDUs are specified to simulate network errors. They are not transmitted to the UT but interpreted by the lower communication layers which shall perform the requested actions. A possible approach is described in the next section.

2.2. Requirements

2.2.1. Communication requirements

To enable execution of the test cases, the IUT shall be capable of conveying TM_PDUs between the UT and the LT in either direction. Therefore the IUT must at least contain two messages which will respectively support the transmission and the reception of TMP data. The UT implementor has to choose them in the available set of messages implemented in the IUT. The minimum requirements regarding the associated communication parameters are the following:

- direct transmission mode (mandatory),
- unqueued,
- static,
- UUDT protocol.

Queued or dynamic messages can also be used to support the TMP. It should be pointed out the test suite specification assumes that a TMP message can be transported in a single bus frame. Therefore, messages mapped on USDT protocol cannot be selected, except dynamic messages (because size of data is not fixed but explicitly specified in the Send/Receive calls).

2.2.2. OS requirements

The test architecture for COM conformance includes a test application called UT and implemented in the same equipment as the IUT. UT implementation does not require special OS functionality. The UT can be integrated in the same environment as the COM module. Like the COM, it only needs task and alarm management services and it can be based on a non-OSEK OS providing equivalent functionality.

The configuration of the UT can vary according to the COM module configuration itself. For instance, one or more tasks need to be implemented depending on the number of tasks that can be activated by the COM implementation. The configuration will also depend on the OS conformance class, the scheduling mechanisms and the inter-task communication (task activation or event setting).

Therefore, this document does not specify a configuration for the UT. It describes the operation of UT when it receives commands from the LT or information from the COM implementation, independently of the type and distribution of tasks and events.

2.2.3. Network perturbations

To verify conformance of a COM implementation, the test environment needs to simulate two types of events:

- no reception of a frame expected by the OSEK/COM module,
- no transmission of a frame sent by the OSEK/COM module.

Simulation of no reception is quite easy. The LT must simply no to send the expected frame. Simulation of no transmission is more difficult. A special TM_PDU has been defined and must be sent by the LT to trigger the simulation. It not transmitted to the UT. It can be interpreted either inside the EUT or in the TE.

- In the local option, the PDU is analysed by special test software implemented at the network driver interface inside the EUT. This software shall be able to notify the IUT of transmission errors.
- In the remote option, the PDU is processed by special test software inside the TE. This software is in charge of controlling a bus-specific equipment called "Bus Manipulator" and able to generate the requested perturbations on the Data Bus.

The picture below illustrates these two options. It shows the location of the added "test software" and the path of error simulation TM_PDUs in both configurations.

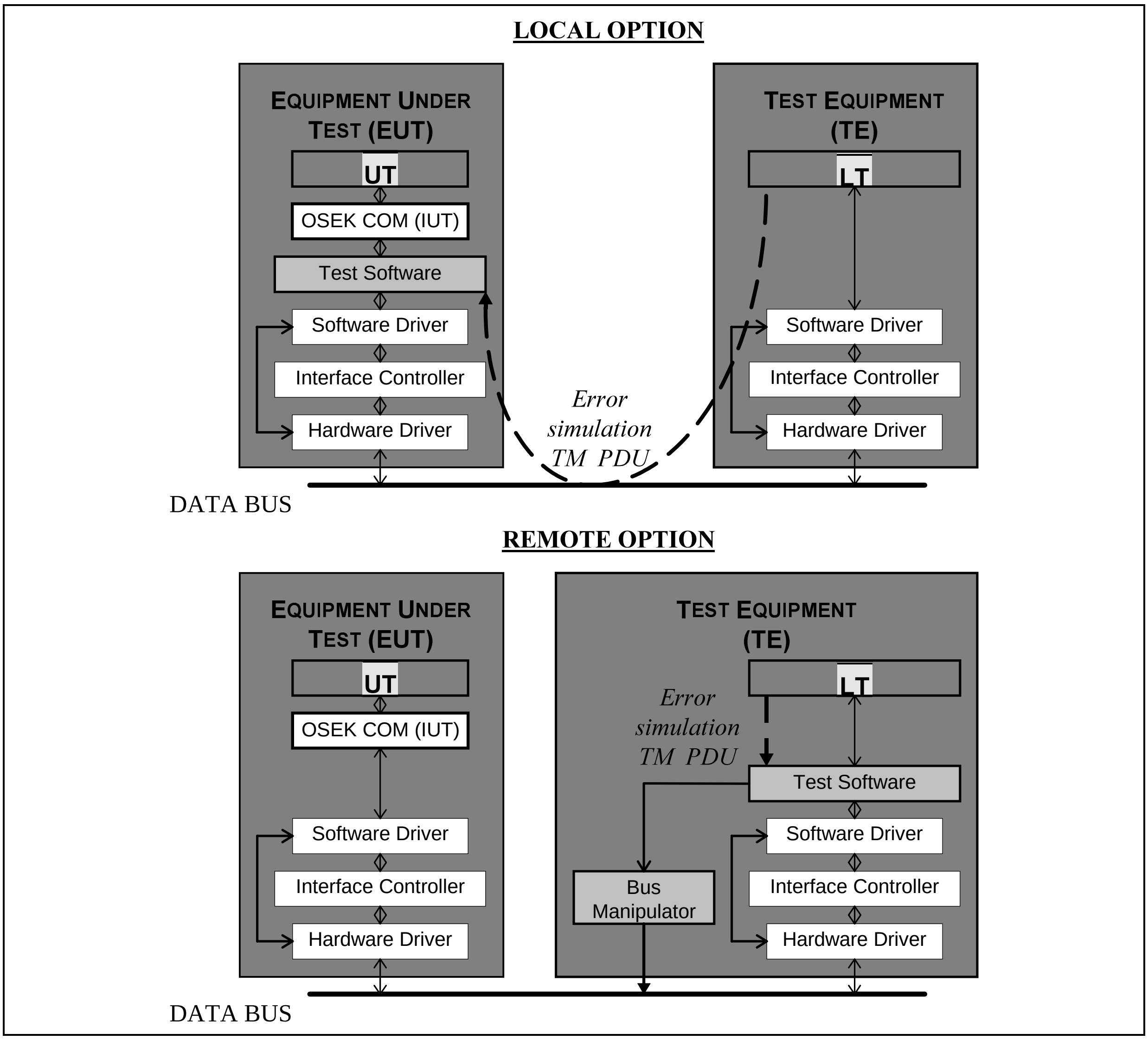

Figure 2 Architectures for error simulation

The local option is the more flexible but it requires modifications of the EUT software and it is therefore generally not applicable to conformance of ECU embedded implementations. The remote option requires additional hardware means and it may be more difficult to implement.

If actual implementation of the test architecture does not enable error simulation, a reduced test suite can be executed. But the COM functionality will not be completely checked.

3. Features and parameters

The COM specification defines optional features and allows different configurations of the specification parameters. Prior to any test suite execution, it is necessary to get a precise knowledge of what features and functions are supported and what parameter values or range of values are permissible. Such information has to be supplied by implementors in standard questionnaires defined hereafter. It will be then used to configure the test environment and to determine which tests can be executed.

Two questionnaires are to be provided. The first one is called PICS. It contains a statement of the capabilities and options which have been implemented. Each question pertains to one of the specification requirements, mandatory or optional. The PICS helps to determine whether all the mandatory features have been implemented and hence it allows a static evaluation of IUT conformance before test suite execution. The PICS is a fixed-format questionnaire in which the questions are simply answered Yes or No.

The second questionnaire is called PIXIT. It provides with additional information required to run the conformance tests. PIXIT questions ask for parameter values pertaining to the IUT and to the testing environment such as time-out values or addressing information. Answers are used to parameterize the test suite and configure the LT and the UT.

3.1. Format of the questionnaires

The questionnaire tables consists of four columns for the PICS and five for the PIXIT:

- <u>Item</u>: specifies an identifier which can be used as a reference in other questions
- <u>Service / protocol features or parameters</u>: specifies the nature of the requested information
- <u>Status</u>: gives a status of the feature/parameter in the specification (Mandatory, Optional)
- <u>Support</u>: indicates whether the feature/parameter has been implemented or not. This column is to be filled in by IUT implementers.
- <u>Value</u>: specifies the related parameter value (PIXIT only). This column is to be filled in by IUT implementers.

The questionnaires make use of the following symbols or abbreviations:

- <u>Status column</u>:
  - M Mandatory
  - O Optional
  - Oi Exclusive option. Support of one and only one *Oi* item (i = option reference number) is mandatory.
  - *pred:* Conditional expression where *pred* refers to the item that needs to be supported for the condition to apply. Conditions may contain logical expressions using the following symbols:
    - | logical OR,
    - . (dot) logical AND.

- Support column:
  Yes feature/parameter supported
  No feature/parameter not supported
  N/A Not Applicable due to not matched condition

The support column does only propose answers meeting compliance requirements. For instance, if the feature or parameter is mandatory only a Yes answer is presented. Answering No means non-compliance. Doing that, static conformance analysis becomes straightforward.

Whenever a condition is specified in the status column, a "N/A" answer is proposed and should be ticked if the IUT does not match the condition. The condition defines what should be answered to some previous questions in order to keep the present statement meaningful. No condition is expressed when the statement is depending on previous answers relating to mandatory features (since such answers should normally be Yes).

3.2. Questionnaires

3.2.1. PICS

The following questionnaires intend to provide a comprehensive list of COM features and options in order to determine the IUT capabilities with great accuracy. Protocol capabilities are listed before services features since the latter are directly connected to protocol implementation.

3.2.1.1. Overall capabilities

Item	Protocol Feature	Status	Support	
	Maximum conformance class supported (select only one option):			
Cc0	– CCC0	O1	_Yes	
Cc1	– CCC1	O1	_Yes	
Cc2	– CCC2	O1	_Yes	
Cc3	– CCC3	O1	_Yes	
	Network protocols supported:			
	– UUDT			
Uus	• as data sender	M	_Yes	
Uur	• as data receiver	M	_Yes	
	– USDT			
Uss	• as data sender	Cc2|Cc3:M	_Yes	N/A
Usr	• as data receiver	Cc2|Cc3:M	_Yes	N/A
	Is local (inter-task) communication supported ?	M	_Yes	
	Transmission concepts supported:			
Dtr	– Direct	M	_Yes	
Ptr	– Periodical	¬Cc0:M	_Yes	N/A
Mtr	– Mixed	¬Cc0:M	_Yes	N/A

	Support of deadline monitoring:		
Dmd	– as data sender, direct transmission	¬Cc0:M	_Yes N/A
Dmp	– as data sender, periodic transmission	¬Cc0:M	_Yes N/A
Dmm	– as data sender, mixed transmission	¬Cc0:M	_Yes N/A
Dmr	– as data receiver	¬Cc0:M	_Yes N/A
	Message types supported:		
Uqm	– Unqueued	M	_Yes
Qum	– Queued	Cc3:M	_Yes N/A
	Message configurations supported:		
Sts	– Static, as sender	M	_Yes
Str	– Static, as receiver	M	_Yes
Dys	– Dynamic, as sender	Cc2|Cc3:M	_Yes N/A
Dyr	– Dynamic, as receiver	Cc2|Cc3:M	_Yes N/A

3.2.1.2. Protocol events

Item	Protocol Feature	Status	Support
	Support of UUDT PDUs:		
Ufs	– as sender	M	_Yes
Ufr	– as receiver	M	_Yes
	USDT PDUs supported:		
	– SF		
Sfs	• as sender	Uss:M	_Yes _N/A
Sfr	• as receiver	Usr:M	_Yes _N/A
	– FF		
Ffs	• as sender	Uss:M	_Yes _N/A
Ffr	• as receiver	Usr:M	_Yes _N/A
	– CF		
Cfs	• as sender	Uss:M	_Yes _N/A
Cfr	• as receiver	Usr:M	_Yes _N/A
	– FC frame		
Fcs	• as sender	Usr:M	_Yes _N/A
Fcr	• as receiver	Uss:M	_Yes _N/A

3.2.1.3. COM PDU fields

Item	Protocol Feature	Status	Support
	Addressing modes supported (at least one option must be supported)		
Am1	– normal	O	_Yes _No
Am2	– extended	O	_Yes _No
	UUDT PDU fields supported		
Uf1	– User data	Uss|Usr:M	_Yes _N/A

	SF PDU fields supported			
Sf1	– PCI-opcode	Uss|Usr:M	_Yes	_NA
Sf2	– DL	Uss|Usr:M	_Yes	_N/A
Sf3	– User data	Uss|Usr:M	_Yes	_N/A
	FF PDU fields supported			
Ff1	– PCI-opcode	Uss|Usr:M	_Yes	_NA
Ff2	– XDL	Uss|Usr:M	_Yes	_NA
Ff3	– DL	Uss|Usr:M	_Yes	_N/A
Ff4	– User data	Uss|Usr:M	_Yes	_N/A
	CF PDU fields supported			
Cf1	– PCI-opcode	Uss|Usr:M	_Yes	_NA
Cf2	– SN	Uss|Usr:M	_Yes	_N/A
Cf3	– User data	Uss|Usr:M	_Yes	_N/A
	FC PDU fields supported			
Fc1	– PCI-opcode	Uss|Usr:M	_Yes	_NA
Fc2	– FS	Uss|Usr:M	_Yes	_NA
Fc3	– BSmax	Uss|Usr:M	_Yes	_NA
Fc4	– STmin	Uss|Usr:M	_Yes	_N/A

3.2.1.4. COM API capabilities

Item	Service Feature	Status	Support		
	COM API calls supported:				
Sv0	– StartCOM	M	_Yes		
Sv1	– SendMessage	M	_Yes		
Sv2	– ReceiveMessage	M	_Yes		
Sv3	– GetMessageResource	M	_Yes		
Sv4	– ReleaseMessageResource	M	_Yes		
Sv5	– GetMessageStatus	M	_Yes		
Sv6	– SendMessageTo	Dys:M	_Yes		_N/A
Sv7	– ReceiveMessageFrom	Dyr:M	_Yes		_N/A
	COM indication capabilities				
	– Indication of end of transmission				
Iett	• by task activation	O	_Yes	_No	
Iete	• by event setting	¬Iett:O	_Yes	_No	_N/A
	– Indication of end of reception				
Iert	• by task activation	O	_Yes	_No	
Iere	• by event setting	¬Iert:O	_Yes	_No	_N/A
	– Deadline indication on periodic/mixed transmission				
Idtt	• by task activation	Dms:O2	_Yes	_No	_N/A
Idte	• by event setting	Dms:O2	_Yes	_No	_N/A
	– Deadline indication on periodic reception				
Idrt	• by task activation	Dmr:O3	_Yes	_No	_N/A
Idre	• by event setting	Dmr:O3	_Yes	_No	_N/A

3.2.1.5. COM API parameters

Item	Service Feature	Status	Support	
	SendMessage parameters:			
Sm1	– Message (SymbolicName)	M	_Yes	
Sm2	– Data	M	_Yes	
	ReceiveMessage parameters:			
Rm1	– Message (SymbolicName)	M	_Yes	
Rm2	– Data	M	_Yes	
	GetMessageResource parameters:			
Gr1	– Message (SymbolicName)	M	_Yes	
	ReleaseMessageResource parameters:			
Rr1	– Message (SymbolicName)	M	_Yes	
	GetMessageStatus parameters:			
Gs1	– Message (SymbolicName)	M	_Yes	
	SendMessageTo parameters:			
Smt1	– Message (SymbolicName)	Sv6:M	_Yes	_N/A
Smt2	– Data	Sv6:M	_Yes	_N/A
Smt3	– DataLength	Sv6:M	_Yes	_N/A
Smt4	– Recipient	Sv6:M	_Yes	_N/A
	ReceiveMessageFrom parameters:			
Rmf1	– Message (SymbolicName)	Sv7:M	_Yes	_N/A
Rmf2	– Data	Sv7:M	_Yes	_N/A
Rmf3	– DataLength	Sv7:M	_Yes	_N/A
Rmf4	– Sender	Sv7:M	_Yes	_N/A

3.2.1.6. COM API return codes

Note that E_COM_LOCKED return code is not verified in the test suite (see document [2]). So, no information is requested about this value in the PICS.

Item	Service Feature	Status	Support	
	Is E_OK return code supported by:			
Eok0	– StartCOM	M	_Yes	
Eok1	– SendMessage	M	_Yes	
Eok2	– ReceiveMessage	M	_Yes	
Eok3	– GetMessageResource	M	_Yes	
Eok4	– ReleaseMessageResource	M	_Yes	
Eok5	– GetMessageStatus	M	_Yes	
Eok6	– SendMessageTo	Sv6:M	_Yes	_N/A
Eok7	– ReceiveMessageFrom	Sv7:M	_Yes	_N/A
	Is E_COM_BUSY return code supported by:			
Ebu3	– GetMessageResource	M	_Yes	
Ebu5	– GetMessageStatus	M	_Yes	

 Eid1 Ed2 Eid3 Eid4 Eid5 Eid6 Eid7	Is E_COM_ID return code supported by: – SendMessage – ReceiveMessage – GetMessageResource – ReleaseMessageResource – GetMessageStatus – SendMessageTo – ReceiveMessageFrom	 O O O O O Sv6:O Sv7:O	 _Yes _No _Yes _No _Yes _No _Yes _No _Yes _No _Yes _No _N/A _Yes _No _N/A
 Ecl2 Ecl5	Is E_COM_LIMIT return code supported by: – ReceiveMessage – GetMessageStatus	 M M	 _Yes _Yes
 Ecn2 Ecn5 Ecn7	Is E_ COM_NOMSG return code supported by: – ReceiveMessage – GetMessageStatus – ReceiveMessageFrom	 M M Sv7:M	 _Yes _Yes _Yes _N/A

3.2.2. PIXIT

The following questionnaires intend to provide actual values for implementation-dependent parameters stated in the COM specification. They also ask for some test parameters required to run the test cases. The values supplied by the IUT designer will be picked up to parameterize the test suite.

3.2.2.1. Protocol parameters

- MUDBPF (Maximum User Data Bytes Per Frame)

  This value represents the size of user data field in a USDT/SF using normal addressing format. For CAN, MUDBPF = 7 (8 - PCI byte).

  The resulting size of user data in the various OSEK/COM frames is given in the following table:

Type of frame	normal addressing	extended addressing
UUDT frame	MUDBPF + 1	MUDBPF
SF	MUDBPF	MUDBPF - 1
FF	MUDBPF - 1	MUDBPF - 2
CF	MUDBPF	MUDBPF - 1

- WFTmax (WaitFrameTransmissions max.)

  This parameter represents the maximum number of FC(Wait) accepted by the tester before declaring the IUT blocked off.

- BSmax (Block Size max)

  This parameter represents the expected block size parameter transmitted by the IUT in a FC frame after reception of the First Frame (FF) of a long message.

Item	Protocol parameter	Status	Support	Value
Pp1	MUDBPF	M	_Yes	
Pp2	WFTmax	Uss|Usr:M	_Yes _N/A	
Pp3	BSmax	Usr:M	_Yes _N/A	
	Protocol timers on the sender side:			
Ts1	TAs	Uss:M	_Yes _N/A	
Ts2	TB1	Uss:M	_Yes _N/A	
Ts3	TB2	Uss:M	_Yes _N/A	
Ts4	TD2	Uss:M	_Yes _N/A	
Ts5	ST	Uss:M	_Yes _N/A	
	Protocol timers on the receiver side:			
Tr1	TAr	Usr:M	_Yes _N/A	
Tr2	TC	Usr:M	_Yes _N/A	
Tr3	TE	Usr:M	_Yes _N/A	

3.2.2.2. Message information

The test user shall provide general information on the user messages that will be used in the test suite to check IUT conformance. Such data make up a message information table. Each element of the table describes the characteristics of a given message and contains the following items:

Name	Message information
mesg_id	Message identifier (Symbolic name identifier)
mesg_len	Message length (max length if dynamic)
conf	Configuration (dynamic/static)
nwprot	Network protocol (uudt/usdt)
addr_md	Addressing mode (normal/extended)
daddr	Data link address
eaddr	Extended address (if extended addressing)
endpt	Logical address of remote end point (if dynamic)

In case of local transmission, only mesg_id and mesg_len must be specified.

In the test suites a message is always identified by an index to the message information table, called from now on message handle. The first two handles are assigned to the messages supporting the TMP, handle 0 for TMP reception by the UT and handle 1 for TMP transmission.

The test user shall specify the messages handles to be used for checking each particular functionnality of the IUT. If required, he has also to provide additional parameters needed to test the functionnality. A given handle can be referenced as many times as necessary. TMP message handles can also be referenced. They can be used to verify IUT conformance while supporting the TMP at the same time.

Item	Message information	Status	Support	Value
	Handles for testing the COM protocols:			
Ph1	– UUDT receiver	M	_Yes	
Ph2	– UUDT sender	M	_Yes	
Ph3	– USDT/SF receiver	Uss:M	_Yes _N/A	
Ph4	– USDT/SF sender	Usr:M	_Yes _N/A	
Ph5	– USDT receiver / FF + one CF	Uss:M	_Yes _N/A	
Ph6	– USDT sender / FF + one CF	Usr:M	_Yes _N/A	
Ph7	– USDT receiver / FF + at least 2 blocks	Uss:M	_Yes _N/A	
Ph8	– USDT sender / FF+at least 3 CFs	Usr:M	_Yes _N/A	
Ph9	– USDT receiver / maximum length	Uss:M	_Yes _N/A	
Ph10	– USDT sender / maximum length	Usr:M	_Yes _N/A	
	Handles for testing send/receive static:			
Sh1	– SendMessage without copy	M	_Yes	
Sh2	– SendMessage with copy	M	_Yes	
Sh3	– ReceiveMessage without copy	M	_Yes	
Sh4	– ReceiveMessage with copy	M	_Yes	
Sh5	– Send/Receive inter-task without copy	M	_Yes	
Sh6	– Send/Receive inter-task with copy	M	_Yes	
	Data for testing periodic transmission::			
Sp1a	– Message handle	Ptr:M	_Yes _N/A	
Sp1b	– Transmission period	Ptr:M	_Yes _N/A	
	Data for testing mixed transmission (*):			
Sp2a	– Message handle	Mtr:M	_Yes _N/A	
Sp2b	– Transmission period	Mtr:M	_Yes _N/A	
Sp2c	– Relevant value (to be transmitted)	Mtr:M	_Yes _N/A	
Sp2d	– No relevant value (not transmitted)	Mtr:M	_Yes _N/A	
	Data for direct transmission deadline:			
Sp3a	– Message handle	Dmd:M	_Yes _N/A	
Sp3b	– Transmission deadline	Dmd:M	_Yes _N/A	
	Data for periodic transmission deadline:			
Sp4a	– Message handle	Dmp:M	_Yes _N/A	
Sp4b	– Transmission period	Dmp:M	_Yes _N/A	
Sp4c	– Transmission deadline	Dmp:M	_Yes _N/A	
	Data for mixed transmission deadline (*):			
Sp5a	– Message handle	Dmm:M	_Yes _N/A	
Sp5b	– Transmission period	Dmm:M	_Yes _N/A	
Sp5c	– Transmission deadline	Dmm:M	_Yes _N/A	
Sp5d	– Relevant value (to be transmitted)	Dmm:M	_Yes _N/A	
Sp5e	– No relevant value (not transmitted)	Dmm:M	_Yes _N/A	
	Data for reception deadline:			
Sp6a	– Message handle	Dmr:M	_Yes _N/A	
Sp6b	– First deadline	Dmr:M	_Yes _N/A	
Sp6c	– Other deadlines	Dmr:M	_Yes _N/A	

(*) relevance/no relevance of message change is estimated from the initial value set in MessageInit ().

Item	Message information	Status	Support	Value
	Handles for testing send/receive dynamic:			
Sd1a	– SendMessageTo without copy	Dys:M	_Yes _N/A	
Sd2a	– SendMessageTo with copy	Dys:M	_Yes _N/A	
Sd3a	– ReceiveMessageFrom without copy	Dyr:M	_Yes _N/A	
Sd4a	– ReceiveMessageFrom with copy	Dyr:M	_Yes _N/A	
	Additional information on Sd1 to Sd4:			
Sd.b	– logical address of 2nd remote end point			
Sd.c	– data link address of 2nd end point			
Sd.d	– extended address of 2nd end point (if extended addressing mode)			
	Data for testing queued transfers			
Sq1a	– Handle for network reception	Qum:M	_Yes _N/A	
Sq1b	– Size of network message queue	Qum:M	_Yes _N/A	
Sq2a	– Handle for local transfer	Qum:M	_Yes _N/A	
Sq2b	– Size of local message queue	Qum:M	_Yes _N/A	

3.2.2.3. API parameters

Item	Service parameter	Status	Support	Value
	API return status:			
Rs1	– E_OK	M	_Yes	
Rs2	– E_COM_BUSY	M	_Yes	
Rs3	– E_COM_ID	Eid1/7:M	_Yes _N/A	
Rs4	– E_COM_LIMIT	M	_Yes	
Rs5	– E_COM_NOMSG	M	_Yes	
Rs6	– Error status returned by MessageInit()	M	_Yes	
	Miscelleanous:			
Ap1	– Bad identifier (to test E_COM_ID)	M	_Yes	

3.2.2.4. Network parameters

Network data associated with user messages are already supplied in the message information table. The following table provides with additional information required to test the OSEK/COM protocols.

Item	Data bus parameter	Status	Support	Value
Np1	bad address information	M	_Yes	
Np2	bad extended address	Am2:M	_Yes _N/A	

3.2.2.5. Test suite parameters

- Test execution timers

  The following timers are defined to manage the test execution:

  Tresp: this timer is started when the LT is waiting for a PDU from the EUT. If it expires, the test will conclude that no response is forthcoming.

  Twait: this timer is started when the LT must wait for a certain amount of time before sending the next PDU. This can happen when the LT has to send two PDUs consecutively and the IUT needs to terminate the first action before being able or entitled to accept the second PDU. The latter is sent after Twait expiry.

  Tlat: to check protocol timer implementation, a time latency has to be defined for IUT outputs triggered by timer expiry. For instance, to check an assertion such as "a CF is transmitted after ST time-out", the LT will firstly verify that nothing has been received within the ST period, then verify that a CF has been received within the subsequent Tlat period.

  Tstart: this timer represents the time needed by the IUT to execute the StartCOM function.

Item	Test suite parameter	Status	Support	Value
	Test execution timers:			
Tt1	– Tresp	M	_Yes	
Tt2	– Twait	M	_Yes	
Tt3	– Tlat	M	_Yes	
Tt4	– Tstart	M	_Yes	

4. Test Management Protocol

4.1. Test scenarios

Figure 2 below describes the different communication scenarios between the UT and the LT. To simplify, protocol messages that do not carry out TM_PDUs are called COM PDUs.

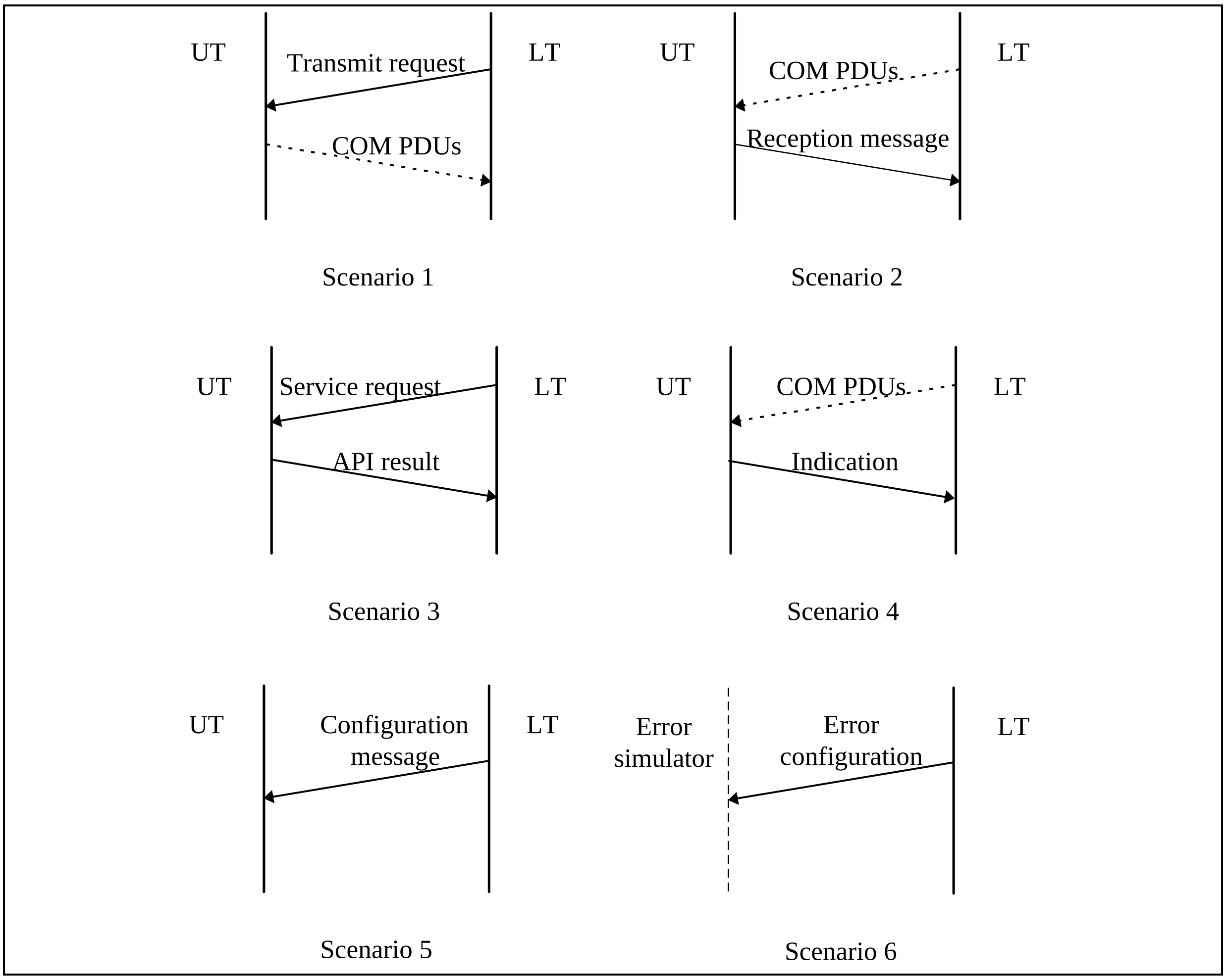

Figure 3 Test scenarios

Scenarios 1 and 2 are used to test the OSEK/COM protocol:

- Scenario 1 allows to test the data sending protocol. On *Transmit request* reception, the UT issues a SendMessage(To) and the LT analyses the COM PDUs generated by the IUT.
- Scenario 2 allows to test the data receiving protocol. The LT generates the necessary COM PDUs leading to a message reception at the COM API. On reception, the UT sends back a *Reception message* to the LT.

Scenario 3 is used to request the UT to call a service of the COM API. The *Service request* message conveys a service identifier and the associated parameters. The UT then returns the results of the service execution in the *API status* message, i.e. the API status and if any, the API's output parameters.

In Scenario 4, the LT sends out (or not) COM PDUs causing an indication from the IUT to the UT (task activation or event signalling). The indication is returned to the LT in the *Indication message.* It may inform the LT of internal events such as end of message transmission, end of reception or deadline expiration.

Scenario 5 aims at configuring the UT behaviour. The *Configuration message* specifies which of the possible COM indications shall be returned to the LT and for which message.

Scenario 6 aims at configuring the network interface behaviour. The *Configuration message* specifies whether or not the network perturbations shall be simulated.

4.2. Data Types

The test management protocol makes use of the following data types of the COM specification:

Data Types	**Remark**
StatusType	Type of returned status information after a service call
SymbolicName	Unique name identifying a message object
DataLength	Data length of the application message to send/receive
AddressType	Logical reference of a remote communication peers

Table 1 Reused data types of COM specification

Data types specific to the test management protocol are defined hereafter.

The first octet of TMP messages describes the nature of the COM service to execute. It is coded as follows:

Format 1:

MsgType	MsgDir	TMPDUName

Format 2:

MsgType	MsgDir	ConfBit	Data1Type	0	0

Figure 4 First octet of TMP messages

Name: **`MsgType`** `(formats 1 and 2)`

Description: This data type helps to determine the nature of received bus frames in the LT. The first octet of user data in bus frames can be either the first data of the user message (UUDT protocol) or the PCI byte of USDT frames. The two bits coded in MsgType allow to determine whether the frame is a UUDT one or a USDT one. As PCI values only occupy the two LSBs of the higher nibble, non-zero values of MsgType can be used to specify the type of message.

Values:

"usdtPCI"	('00'B):	always '00' in USDT PCI bytes
"form1"	('01'B):	TMP message , format 1

"form2"	(’10’B):	TMP message , format 2
"dataFrm"	(’11’B):	application data, do not interpret

Name: **MsgDir** (formats 1 and 2)

Description: This data type defines the direction of the message.

"toIUT"	(’0’B):	message from LT to IUT/UT
"fromIUT"	(’1’B):	message from UT/IUT to LT.

Name: **ConfBit** (format 2)

Description: This data type defines the user’s message configuration.

"stBit"	('0'B):	static message
"dynBit"	('1'B):	dynamic message.

Name: **Data1Type** (format 2)

Description: This data type defines the format for user data encoding (see 4.6).

"encode0"	('00'B):	encoding format 1
"encode1"	('01'B):	encoding format 2
"encode2"	('10'B):	encoding format 3
"badData"	('11'B):	bad data, do not match any encoding format

Name: **TMPDUName** (format 1)

Description: This data type defines the type of TMP message. Messages regarding the COM API can be either a "request to call" when going from LT to UT or the "result of the API call" when going from UT to LT.

Values:

"startCOM"	('00000'B):	call to/result of StartCOM
"sendMsg"	('00001'B):	call to/result of SendMessage
"sendTo"	('00010'B):	call to/result of SendMessageTo
"rcvMsg"	('00011'B):	call to/result of ReceiveMessage
"rcvFrom"	('00100'B):	call to/result of ReceiveMessageFrom
"getRes"	('00101'B):	call to/result of GetMessageResource
"relRes"	('00110'B):	call to/result of ReleaseMessageResource
"getStat"	('00111'B):	call to/result of GetMessageStatus
"UTEvent"	('01000'B):	report from UT task activation or event setting
"configUT"	('01001'B):	configuration of UT's behaviour
"setError"	('01111'B):	configuration of network interface's behaviour

The other data types implemented in TM_PDUs are as follows:

Name: **MesgIdType**

Description: This data type defines an identifier for the message to be transmitted or received. It may or not be equal to the message handle. In UT application, it has to be associated with the "symbolic name" defined in the COM/API specification.

Name: **MixedValType**

Description: This data type defines the type of user message used to test the mixed transmission mode.

Name: **StatusModeType**

Description: This data type defines how the API return code must be handled by the UT.

Values:
"never": the return code is never returned to the LT,
"always": the return code is always returned to the LT,
"ifError": the return code is returned if different from E_OK,

Name: **ActionType**

Description: This data type specifies a mask defining what information collected at the COM API must be reported to the LT. It also defines special actions to be performed by the UT.

Values: This data type includes one bit for each possible action:
One bit: (do not) report from end of message transmission or reception,
One bit: (do not) report from deadline expiration,
One bit: inhibit/activate reception of a queued message,
One bit: (do not) call the next COM function at ISR level,
One bit: (do not) call the next COM function from ErrorHook routine.

Name: **EventIdType**

Description: This data type defines a mask defining what information collected at the COM API is being reported to the LT.

Values: This data type includes one bit for each possible information:
One bit reporting from end of message transmission or reception,
One bit reporting from deadline expiration.

Name: **NetErrorType**

Description: This data type specifies the network errors to be simulated.

Values:
"noNetError": no error simulation,
"noTransmission": simulation of no transmission (e.g. no frame acknowledgement at the data bus)

4.3. TMP messages from LT to UT

TMP messages are transmitted from LT to UT to request the UT to either:

- execute a service of the COM API,
- or configure UT's behaviour.

Message Name:	**`CallSM`**
Scenario:	1 - Transmit request

Parameters:	MsgType <msg_typ>; // "form2"
	MsgDir <dir>; // "toIUT"
	ConfBit <conf>; // "stBit" or "dynBit"
	Data1Type <encode>;
	MesgIdType <message>;
	DataLength <dlength>; // OPTIONAL
	AddressType <recipient>; // OPTIONAL
	MixedValType <mixedval>; // OPTIONAL

Purpose:	This message requests the UT to execute either "status = SendMessage(message, access)" if "conf" = stBit (static), or "status = SendMessageTo(message, access, recipient, dlength)" if "conf" = dynBit (dynamic). "message" identifies the message to be transmitted. "access" is the reference of the user data buffer. The parameter is not transmitted. It must be known locally by the UT. "dlength" (dynamic message) is the length of message data in octets. "recipient" (dynamic message) is the logical address of the message recipient. The parameter is not transmitted. It must be defined before UT and LT implementation. Message data shall be initialised by the UT before transmission according to the format defined by "encode" (see § 4.6), except in case of mixed transmission mode. In that case the message value is supplied by the mixedval parameter. Note that presence of dlength/recipient and mixvalue in the TM_PDU structure are exclusive options. The status returned by SendMessage(To) must be saved. It can be requested later by the LT with a CallAPI message. Only the last status must be kept.

Message Name:	**`CallStart`**
Scenario:	3 - Service request

Parameters:	MsgType <msg_typ>; // "form1"
	MsgDir <dir>; // "toIUT"
	TMPDUName <name>; // "startCOM"
	StatusModeType <statusMode>;
	StatusType <status>;

Purpose: This message requests the UT to execute "status = StartCOM()". Parameter status of the message represents the status code which must be returned by the MessageInit function.

Depending on both the returned status and the statusMode option, the UT will send back or not the status to the LT. Status transmission is done with the APIStatus message.

Message Name: **`CallAPI`**

Scenario: 3 - Service request

Parameters:

MsgType	<msg_typ>;	// "form1"
MsgDir	<dir>;	// "toIUT"
TMPDUName	<name>;	
MesgIdType	<message>;	
StatusModeType	<statusMode>;	

Purpose: This message can be used

1. to request the UT to execute a service of the COM API, except StartCOM, SendMessage and SendmessageTo.
2. to get the status returned by the last call to SendMessage or SendmessageTo

In the latter case, the parameter "name" is set to "sendMsg" or "sendTo". The UT will send back by the status using the APIStatus message. It does not need to test the "message" or "status" parameter. The last status must be sent anyway.

In the first case, the service is defined by the parameter "name" as follows:

"name"	API call
rcvMsg	status = ReceiveMessage(message, access)
rcvFrom	status = ReceiveMessageFrom(message, access, sender, dlength)
getRes	status = GetMessageResource(message)
relRes	status = ReleaseMessageResource(message)
getStat	status = GetMessageStatus(message)

Depending on both the returned status and the statusMode option, the UT will send back or not the status to the LT. Status transmission is done using either:

- the RMStatus message after a call to ReceiveMessage or ReceiveMessageFrom, or
- the APIStatus message after a call to GetMessageResource, ReleaseMessageResource or GetMessageStatus.

Message Name: **`CallConfigUT`**

Scenario: 5 - Configuration message

Parameters:

MsgType	<msg_typ>;	// "form1"
MsgDir	<dir>;	// "toIUT"
TMPDUName	<name>;	
MesgIdType	<message>;	
ActionType	<action>;	

Purpose: This message allows to configure UT's behaviour according to the value of parameter "action".

1. If information "report from message transmission/reception" or " report from deadline expiration" is set, the UT shall transmit a UTEvent message whenever the task or event associated to the given "message" is activated/set.

   Default: end of transmission/reception and deadline expiration are not reported.

2. If information "inhibit message reception" is set, the UT shall not call ReceiveMessage or ReceiveMessageFrom when the given "message" is received.

   Default: ReceiveMessage or ReceiveMessageFrom must be called whenever a message reception is detected. The result is sent using the RMStatus message.

3. If information "call at ISR level" or "call from ErrorHook routine" is set, the next API calls must be issued at ISR level or from an ErrorHook routine respectively. The "message" parameter is meaningless.

   Default: all API calls are issued at the task level from a user application routine.

Note for implementation of items 1 and 2: only one message can be assigned a non default value. The UT need not hold an "action" parameter for each message. It has only to know which message is configured with a non default value.

4.4. TMP messages from UT to LT

TMP messages are transmitted from UT to LT to inform the UT of the result of a service call or of a COM indication.

Message Name: **`RMStatus`**

Scenario: 2 - Reception message
3 - Service request

Parameters:

MsgType	<msg_typ>;	// "form2"
MsgDir	<dir>;	// "fromIUT"
ConfBit	<conf>;	// "stBit" or "dynBit"
Data1Type	<encode>;	
MesgIdType	<message>;	

StatusType	<status>;	
DataLength	<dlength>;	// OPTIONAL
AddressType	<sender>;	// OPTIONAL

Purpose: This message provides the LT with the status returned by the ReceiveMessage or ReceiveMessageFrom function. A call to either function can be triggered either explicitly or implicitly:

1. The UT shall execute the receive function when requested explicitly by the CallAPI message with parameter name set to "rcvMsg" or "rcvFrom".
2. The UT shall execute the receive function whenever a new message is received by the Interaction Layer, provided reception is not inhibited by a previous CallConfigUT regarding this particular message.

After calling ReceiveMessage or ReceiveMessageTo, the UT shall determine the encoding format of message data and verify data values according to the rules specified in § 4.6

RMStatus parameters are as follows:

"encode" represents the encoding format of the received data. It shall be set to "badData" if wrong values have been detected in the sequence of data.

"message" is the message identifier (linked to first parameter of ReceiveMessage/ ReceiveMessageFrom).

"status" is the status returned by ReceiveMessage/ ReceiveMessageFrom.

"sender" (dynamic message) is the logical address of the message sender (same as sender parameter of ReceiveMessageFrom).

"dlength" (dynamic message) is the length of message data in octets (same as last parameter of ReceiveMessageFrom).

Message Name: **`APIStatus`**

Scenario: 3 - API result

Parameters:

MsgType	<msg_typ>;	// "form1"
MsgDir	<dir>;	// "fromIUT"
TMPDUName	<name>;	
StatusType	<status>;	

Purpose: This message provides the LT with the status returned by the COM/API service executed on reception of CallStartCOM or CallAPI. Parameter "name" defines the name of the service and can take one of the values "sendMsg", "sendTo", "startCOM", "getRes", "relRes" or "getStat".

Message Name: **`UTEvent`**

Scenario: 4 - Indication

Parameters:	MsgType	<msg_typ>;	// "form1"
	MsgDir	<dir>;	// "fromIUT"
	TMPDUName	<name>;	// "UTEvent"
	MesgIdType	<message>;	
	EventIdType	<eventId>;	

Purpose: This message informs the LT that an event reception or task activation from the COM module has just occurred. The type of indication is defined by "eventId" and the concerned message by "message".

This indication must be sent only when expressly authorised by the previous CallConfigUT message.

4.5. TMP messages from LT to Network Interface

TMP messages are transmitted from LT to Network Interface to configure the network error simulation.

Message Name: **`CallSetError`**

Scenario: 6 - Error configuration

Parameters:	MsgType	<msg_typ>;	// "form1"
	MsgDir	<dir>;	// "toIUT"
	TMPDUName	<name>;	// "UTEvent"
	NetErrorType	<netError>;	

Purpose: This message defines whether transmission errors shall be simulated or not. This information is supplied by the netError parameter.

Default: no error simulation.

If simulation of no message transmission is requested, the network interface shall behave so that a transmission error is returned to the COM after each transmission attempt until error simulation is stopped by another CallConfigError message with a different netError value.

4.6. Encoding rules for user data

The conformance tester shall verify the validity of user data transferred from message buffers to the network or from the network to message buffers. Simple rules have been defined to encode user data bytes with different values. These rules shall be applied by both the LT and the UT to generate and verify message data.

The format of data bytes consists of a fixed part (3 MSBs) and a variable part (5 LSBs)

(dataFrm)			variable part				
1	1	0					

Figure 5 Encoding of user data bytes

The variable part of consecutive data bytes is incremented modulo 32.

First value = (Message identifier + Encode flag) modulo 32, where "Encode flag" is set to 0, 1 or 2 according to the following rules:

- Transmission by LT in UUDT frames and USDT/SF or FF:

  Encode flag is incremented modulo 3 whenever a new message is transmitted.

  In case of segmented data, the rule of modulo 32 incrementation of "variable part" applies to consecutive data segments transmitted in successive data frames.

- Transmission by UT after CallSM reception:

  Encode flag is supplied by the LT in the *encode* parameter of CallSM. This parameter is incremented modulo 3 whenever a new CallSM is issued.

Remarks:

- Encode flag incrementation is performed globally for messages transmitted via CallSM or via UUDT/USDT frames. The first value generated in test suite execution is 0.
- These rules do not apply to messages configured for mixed transmission. Data of such messages are relevant to the Interaction Layer and special values have to be transmitted.

Attachment 1: OSEK/COM test suite

The COM test suite is specified in TTCN language [7].

The test cases are derived from the test purposes of document [2]. But the sequence of test cases and of test purposes are organised differently. The test purposes are listed according to the order of chapters and sections in the COM specification. On the contrary, the test cases are grouped in directories representing the main options of an implementation. Inside each directory, they are sequenced in a logical order to allow a progressive test of the associated functionnality.

The test case directories are defined in the table below:

Directory	**Test Objectives**
UUDTs	UUDT sending protocol
UUDTr	UUDT reception protocol
USDTs	USDT sending protocol
USDTr	USDT reception protocol
CCC0	CCC0 services of the OSEK/COM API
CCC1	CCC1 services of the OSEK/COM API
CCC2	CCC2 services of the OSEK/COM API
CCC3	CCC3 services of the OSEK/COM API

To facilitate cross-reference with the test plan, naming conventions have been defined. Test case names are derived from the location of the corresponding assertion in the test plan. Names consist of:

- a radix identifying the table of test assertion,
- the reference number of the assertion in the table. If the test case is linked to several assertions, the respective numbers are separated by "_". If several tests stem from the same assertion, the number is followed by a letter A, B, C...

Example: UUP1_2A is the first test case (final letter A) covering assertions Nr 1 and 2 of the table "UUDT protocol".

The correspondence between the test case names and the test plan is given in the following table:

Test plan section	Test case name
Interaction Layer services / network communication	SRV...
Interaction Layer services / local (inter-task) communication	LSRV...
Interaction Layer API / network communication	API...
Interaction Layer API / local (inter-task) communication	LAPI...
UUDT protocol	UUP...
UUDT sending state machine	UUS...
UUDT receiving state machine	UUR...
USDT sending state machine	USS...
USDT receiving state machine	USR...

Table 2 Test case names

Test purposes which are covered by many other test cases are not referenced in the test suite. For example, assertion "The OSEK COM supports communication within ECUs" is covered by all the tests dealing with local communication.