OSEK/VDX

COM test plan

Version 1.0

July 24th, 1998

This document is an official release and replaces all previously distributed documents. The OSEK group retains the right to make changes to this document without notice and does not accept any liability for errors.
All rights reserved. No part of this document may be reproduced, in any form or by any means, without permission in writing from the OSEK/VDX steering committee.

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 the COM/NM project group of MODISTARC:

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 plan for services and protocols of the OSEK COM as defined in specification document [3]. It applies to conformance test suites for testing implementations which claim conformance to the OSEK COM specification.

According to the Conformance Methodology [1], definition of conformance tests is a two-stage process. This test plan document corresponds to the first step. It specifies a list of test purposes extracted from the COM specification. In the second step, test cases will be derived from the test purposes to build up the OSEK COM conformance test suite. Basically, a test case specifies the sequence of interactions between a tester and the COM implementation in order to verify a test purpose of this document. However, it is possible to have individual test cases that address multiple test purposes and likewise multiple test cases that address the same test purpose.

According to the Conformance Methodology this document follows the principle of black box testing. There is no test purpose for explicitely checking the COM sublayer interfaces, such as the Network Layer API and the Data Link Layer API. Also, interfaces between COM and NM have also been considered not mandatory and no test purpose has either been specified for the related APIs.

The test purposes are organised according to a tree structure described in Chapter 2.

1.2. References

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

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

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

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

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

[6] 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
COM	Communication

CTS	Clear To Send
ECU	Electronic Control Unit
FC	Flow Control
FF	First Frame
ISO	International Standard Organization
ISR	Interrupt Service Routine
OS	Operating System
MUDBPF	Maximum User Data Bytes Per Frame
PCI	Protocol Control Information
PDU	Protocol Data Unit
SF	Single Frame
SDL	Specification and Description Language
TTCN	Tree and Tabular Combined Notation
USDT	Unacknowledged and Segmented Data Transfer
UUDT	Unacknowledged and Unsegmented Data Transfer
WFT	WaitFrameTransmissions

2. Test purposes structure

2.1. Description

The test purposes for the OSEK COM services and protocols are arranged in groups and subgroups following a hierarchical structure. This organisation follows the COM specification structure. It intends to facilitate cross-checking with the specification and verification of completeness. It does not preclude a different approach for test cases organisation inside the test suite.

The tree structure of COM test purposes is illustrated in Figure 1 below:

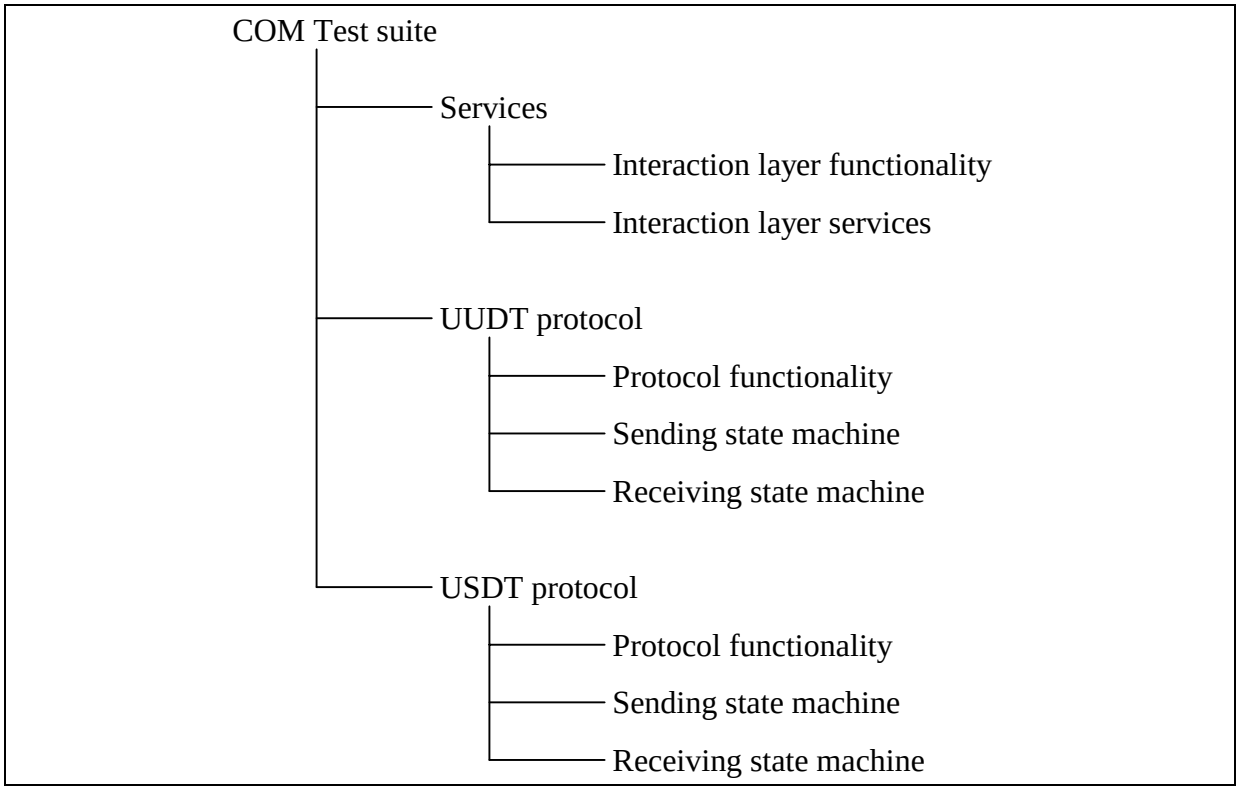

Figure 1 Hierarchy of COM test purposes

The service tests are subdivided on a per service basis. There is at least one test for each API in order to demonstrate that all implemented services can be successfully called by an application.

The protocol tests are subdivided according to protocol states and substates defined in the COM specification. They intend to verify that the COM implementation behaves as specified in all implemented (sub)states. They also check all transitions between the different (sub)states.

Both the service and the protocol test purposes include verification of:

- valid behaviour: the implementation is actually waiting for the stimuli received from the tester,
- error cases: the implementation has not received the expected stimulus after a given

time-out, or has received an unexpected stimulus.

Test purposes are brought together into tables corresponding to the leaves of the tree structure. Each table is made up of four columns providing:

- a reference number,
- the test assertion,
- the paragraph or picture of the COM specification from which the assertion was extracted,
- the specification variant needing to be implemented for the test purpose to be verified. The main variants are the conformance classes defined in the specification, CCC0, CCC1, CCC2, CCC3. As conformance classes are upward compatible, the variant column indicates the lowest class that must verify the test assertion. For example, a CCC2 indication means that the test is valid for CCC2 and CCC3 implementations.

  The "SDL" indication means that the test assertions is derived from the SDL models attached to the specification.

Each test assertion contains:

- the stimulus to be sent to verify the test purpose and if necessary the COM specification state needing to be reached before sending the stimulus,
- the action that shall be performed by the implementation to verify the test purpose and the subsequent output that should be observed by the tester. Note that the output can be "nothing" in which case the tester shall verify that the implementation did not send anything.

2.2. Detailed structure

- COM
  - Services
    - Interaction layer services
      - General
      - Application notification
      - Transmission modes
      - COM deadline monitoring on transmissions
      - COM deadline monitoring on receptions
    - Interaction layer API
      - StartCom
      - SendMessage/SendMessageTo - With copy
      - SendMessage/SendMessageTo - Without copy
      - ReceiveMessage/ReceiveMessageFrom - Unqueued messages with copy
      - ReceiveMessage/ReceiveMessageFrom - Unqueued messages without copy
      - ReceiveMessage - Queued messages without copy
      - GetMessageResource/ReleaseMessageResource
      - GetMessageStatus
      - Extended stati
      - Usage of COM services in OS routines
  - UUDT
    - UUDT protocol
      - Protocol format
      - Protocol errors
      - Protocol capabilities
    - UUDT sending state machine
      - Successful transfers
      - Unexpected frames
      - Error cases
    - UUDT receiving state machine
      - Successful transfers
      - Unexpected frames
      - Error cases
  - USDT
    - USDT protocol
      - Protocol format
      - Protocol errors
      - Protocol capabilities
    - USDT sending state machine
      - Successful transfers
      - Unexpected frames
      - Error cases
    - USDT receiving state machine
      - Successful transfers
      - Unexpected frames
      - Error cases

Table 1 Structure of COM Test Purposes

3. OSEK/COM test purposes

This clause contains a set of test purposes relevant to COM services and protocols. These test purposes provide ground material for developing the TTCN test suite which will be used to evaluate conformance to COM specification [3].

3.1.1. Service test group

This section specifies tests purposes relative to the COM Interaction Layer and the COM API as defined in chapter 3 of the COM specification.

Each test purpose defines both the test stimulus to be sent and the subsequent output(s) to be observed at the COM API.

The test stimuli include:

- API calls with different sets of input parameters,
- COM messages received by the implementation under test.
- COM deadline alarms (internal stimuli).

The observable outputs are either:

- the stati and output parameters returned by API, or
- the tasks activations or event settings performed by the implemention on message transmission/reception or on deadline occurrence.

Each test purpose also gives information on the specification variant(s) that need to be implemented for the test purpose to be verified.

The implementation variants are the conformance classes defined in the specification.

Remark:

Two properties of the specification will not be verified in the test suite because testing them requires implementation specific means:

- OSEK/COM adopts an asynchronous communication. There is no test purpose to verify that application is not blocked while data is transmitted and APIs for sending or receiving COM messages return immediately to the application.
- Message consistency during transfers or read/write operations by concurrent tasks. There is no test purpose regarding the return status E_COM_LOCKED of the OSEK/COM API.

3.1.1.1. Interaction Layer services

The functionnality described in sections “Application notification”, “Transmission modes” and “COM deadline monitoring” of the table below may be supported or not, depending on application requirements. The test purposes have to be verified on COM messages requiring that functionality.

Test purposes marked with * have to be verified for both ECU internal and ECU external communication. The others are specific to external communication.

Nr	Assertion	Paragraph in spec.	Affected variants
General			
1	The OSEK COM supports communication within ECUs.	1 + Table 2.2	CCC0 (1)
2	The OSEK COM supports communication between networked ECUs.	1 + Table 2.2	CCC0
3	An unqueued message is overwritten whenever a new message arrives.	2.5	CCC0*
4	Queued messages are delivered in the same order as they were sent.	2.5	CCC3*
Application notification			
5	The application is informed of message transmission by means of task activation or event setting depending on the selected mechanism at system generation time.	2.3.1	CCC1
6	The application is informed of message reception by means of task activation or event setting depending on the selected mechanism at system generation time.	2.3.1	CCC1*
Transmission modes			
7	In direct transmission mode, each call to the API transmission service updates the message object and issues a transmission request to the COM layer.	2.7.1	CCC0
8	In periodic transmission mode, each call to the API transmission service updates the message object. The transmission is performed on a cyclic time basis according to the time period defined at system generation time.	2.7.2	CCC1 (2)
9	In periodic transmission mode, the first transmission occurs once StartCOM is successfully executed.	2.7.2	CCC1 (2)
10	In mixed transmission mode, the COM module issues transmissions on relevant changes in the message value. Possible relevant changes are: – value less than constant – value greater than constant – value equal to constant – (value - oldvalue) less than constant – (value - oldvalue) greater than constant – (value - oldvalue) equal to constant	2.7.3	CCC1 (2)

11	In mixed transmission mode, the COM module issues periodic transmission requests as in periodic mode. Intermediate transmissions on relevant changes in the message value do not modify the base cycle.	2.7.3	CCC1 (2)
COM deadline monitoring on transmissions			
12	In direct transmission mode a monitoring alarm can be started on each transmission request by the application. If it expires due to failed transmission, the application is informed by means of task activation or event setting as specified at system generation time.	2.8	CCC1
13	In direct transmission mode the monitoring alarm related to a given message is not started if it is already running at the time of transmission request by the application.	2.8	CCC1
14	If direct transmission succeeds, the monitoring alarm is cancelled. No task is activated and no event is set.	2.8	CCC1
15	In periodic transmission mode a monitoring alarm can be started on each periodic transmission request by the COM module. If it expires due to failed transmission, the application is informed by means of task activation or event setting as specified at system generation time.	2.8	CCC1 (2)
16	In periodic transmission mode the monitoring alarm related to a given message is not started if it is already running at the time of transmission request by the COM module.	2.8	CCC1 (2)
17	If the periodic transmission succeeds the monitoring alarm is cancelled. No task is activated and no event is set.	2.8	CCC1 (2)
18	In mixed transmission mode a monitoring alarm is started on each periodic transmission request by the COM module. If it expires due to failed transmission, the application is informed by means of task activation or event setting as specified at system generation time.	2.8	CCC1 (2)
19	In mixed transmission mode a monitoring alarm is started on each transmission request due to relevant change in the message value. If it expires due to failed transmission, the application is informed by means of task activation or event setting as specified at system generation time.	2.8	CCC1 (2)
20	In mixed transmission mode the monitoring alarm related to a given message is not started if it is already running at the time of periodic transmission request.	2.8	CCC1 (2)

21	In mixed transmission mode the monitoring alarm related to a given message is not started if it is already running at the time of transmission request due to relevant change in the message value.	2.8	CCC1 (2)
22	If the periodic transmission succeeds in mixed transmission mode, the monitoring alarm is cancelled. No task is activated and no event is set.	2.8	CCC1 (2)
23	If the transmission due to relevant change in the message value succeeds in mixed transmission mode, the monitoring alarm is cancelled. No task is activated and no event is set.	2.8	CCC1 (2)
COM deadline monitoring on receptions			
24	In periodic reception mode a monitoring alarm can be started on each reception. If it expires due to no further reception, the application is informed by means of task activation or event setting as specified at system generation time.	2.8	CCC1 (2)
25	In periodic reception mode the monitoring alarm is automatically started once StartCOM is successfully executed. A special value can be chosen for first alarm.	2.8	CCC1 (2)
26	In periodic reception mode the monitoring alarm is restarted after expiration due to no reception of the expected message.	2.8	CCC1 (2)

(1) Characteristics of ECU-internal communication according to Table 2.2 of specification:
- direct transmission mode,
- static configuration.

(2) Characteristics of periodic/mixed transmissions according to Table 2.2 of specification:
- ECU-external communication using UUDT protocol,
- unqueued messages,
- static configuration.

3.1.1.2. Interaction layer API

As previously, the different variants of SendMessage/SendMessageTo and ReceiveMessage/ ReceiveMessageTo may be supported or not depending on application requirements.

In addition to conformance classes, the variant *EStatus* means that the associated tests must be executed if the extended stati of the COM API have been implemented.

Test purposes marked with * have to be verified for both ECU internal and ECU external communication. The others are specific to external communication.

Nr	Assertion	Paragraph in spec.	Affected variants
StartCOM service			

1	StartCOM service initialises the communication hardware and calls the MessageInit function	4.4.2.3	CCC0
2	Assuming initialisation has succeeded, StartCOM returns E_OK if MessageInit returns E_OK	4.4.2.3	CCC0
3	Assuming initialisation has succeeded, StartCOM returns the MessageInit error code if MessageInit returns an error	4.4.2.3	CCC0
SendMessage/SendMessageTo (1) - With copy			
4	SendMessage updates the message object with the given data and returns E_OK. In case of network communication, it requests the transmission of the message object to the receiving entities.	3.3.1	CCC0*
5	SendMessageTo updates the message object with the given data and returns E_OK. Then it requests the transmission of the message object to the receiving entity. The receiving entity is selected by the <Recipient> parameter. The length of message is provided by the <Datalength> parameter.	3.3.6	CCC2
SendMessage/SendMessageTo (1) - Without copy			
6	SendMessage returns E_OK if the message object is not locked. In case of network communication, it requests the transmission of the message object to the receiving entities.	3.3.1	CCC0*
7	SendMessageTo returns E_OK if the message object is not locked. Then it requests the transmission of the message object to the receiving entity. The receiving entity is selected by the <Recipient> parameter. The length of message is provided by the <Datalength> parameter.	3.3.6	CCC2
ReceiveMessage/ReceiveMessageFrom (1) - Unqueued messages with copy			
8	If a message is available, ReceiveMessage delivers the data of the message object and returns E_OK.	3.3.2	CCC0*
9	If no message has been received so far, ReceiveMessage returns E_COM_NOMSG.	3.3.2	CCC0*
10	If no message has been received since the last call, ReceiveMessage delivers the data of the current message object and returns E_OK.	3.3.2	CCC0*

11	If a message is available, ReceiveMessageFrom delivers the data of the message object and returns E_OK. The length of the message is provided by the <DataLength> parameter and the reference of the message sender is provided by the <Sender> parameter.	3.3.7	CCC2
12	If no message has been received so far, ReceiveMessageFrom returns E_COM_NOMSG.	3.3.7	CCC2
13	If no message has been received since the last call, ReceiveMessageFrom delivers the data of the current message object and returns E_OK.	3.3.7	CCC2
ReceiveMessage/ReceiveMessageFrom (1) - Unqueued messages without copy			
14	If a message is available, ReceiveMessage returns E_OK.	3.3.2	CCC0*
15	If no message has been received so far, ReceiveMessage returns E_COM_NOMSG.	3.3.2	CCC0*
16	If a message is available, ReceiveMessageFrom returns E_OK. The length of the message is provided by the <DataLength> parameter and the reference of the message sender is provided by the <Sender> parameter.	3.3.7	CCC2
17	If no message has been received so far, ReceiveMessageFrom returns E_COM_NOMSG.	3.3.7	CCC2
ReceiveMessage - Queued messages with copy (2)			
18	If the reception FIFO contains at least one message, ReceiveMessage delivers the oldest message. The returned status is E_OK.	3.3.2	CCC3*
19	If the FIFO queue is empty, ReceiveMessage returns E_COM_NOMSG.	3.3.2	CCC3*
20	If case of a FIFO overflow, ReceiveMessage delivers the oldest message. The returned status is E_COM_LIMIT.	3.3.2	CCC3*
GetMessageResource/ReleaseMessageResource - (Without copy)			
21	GetMessageResource sets the given message object as busy if it is not already so and it returns E_OK.	3.3.3	CCC0
22	If the message object is already busy, GetMessageResource returns E_COM_BUSY.	3.3.3	CCC0
23	ReleaseMessageResource sets off the given message object from busy and it returns E_OK.	3.3.4	CCC0
GetMessageStatus			

24	GetMessageStatus returns the current status of the message object (see tests 4 to 20 and 22).	3.3.5	CCC0
Extended stati			
25	SendMessage returns E_COM_ID in case of invalid <Message> parameter.	3.3.1	CCC0 + EStatus
26	ReceiveMessage returns E_COM_ID in case of invalid <Message> parameter.	3.3.2	CCC0 + EStatus
27	GetMessageResource returns E_COM_ID in case of invalid <Message> parameter.	3.3.3	CCC0 + EStatus
28	ReleaseMessageResource returns E_COM_ID in case of invalid <Message> parameter.	3.3.4	CCC0 + EStatus
29	GetMessageStatus returns E_COM_ID in case of invalid <Message> parameter.	3.3.5	CCC0 + EStatus
30	SendMessageTo returns E_COM_ID in case of invalid <Message> parameter.	3.3.6	CCC0 + EStatus
31	ReceiveMessageFrom returns E_COM_ID in case of invalid <Message> parameter.	3.3.7	CCC0 + EStatus
Usage of COM services in OS routines (3)			
32	SendMessage can be called at ISR level in case of unqueued message with copy.	3.5	CCC0
33	ReceiveMessage can be called at ISR level in case of unqueued message with copy.	3.5	CCC0
34	GetMessageStatus can be called at ISR level.	3.5	CCC0
35	ReceiveMessage can be called in ErrorHook routine in case of unqueued message with copy.	3.5	CCC0 (4)
36	GetMessageStatus can be called in ErrorHook routine.	3.5	CCC0 (4)

(1) Usage of SendMessageTo/ReceiveMessageFrom (dynamic configuration):
- direct transmission mode,
- unqueued messages,
- ECU-external communication

(2) Characteristics queued messages according to Table 2.2 of specification:
- direct transmission mode,
- ECU-internal or ECU-external communication (UUDT protocol only)
- static configuration

(3) It is assumed that all previous tests will be executed at task level. Therefore, there is no assertion regarding possible execution at task level.

(4) Assumes usage of an OSEK/OS.

3.1.2. UUDT protocol test group

This section specifies tests purposes relative to the UUDT protocol, as defined in chapter 4 of the COM specification. The test purposes define actions expected from the implementation under test on a given input in order to verify that its behaviour conforms to the specification.

Each test purpose defines both the test stimulus or stimuli to be sent and the subsequent output(s) to be observed at the COM API. Some actions can also be triggered by internal events. The test stimuli include:

- COM API procedure calls.
- UUDT frames received by the implementation under test.
- Timer expiration (internal stimuli): TA.

The observable outputs are as follows:

- Status of application messages.
- Reception of application messages or absence of reception.
- UUDT frames sent by the implementation under test.

Each test purpose also gives information on the specification variant(s) that need to be implemented for the test purpose to be verified.

3.1.2.1. UUDT protocol

Beside conformance classes, two variants have been identified. They concern the type of protocol encoding:

- *normal* for normal frame format,
- *extended* for extended frame format,

Nr	Assertion	Paragraph in spec.	Affected variants
Protocol formats			
1	For a message using normal addressing, address information is encoded in the frame header. User data occupy the user data field.	4.1	CCC0 + Normal
2	For a message using extended addressing, there is eight bits of additional address information in the first byte of user data field. User data start with the second byte.	4.1	CCC0 + Extended
Protocol errors			
3	Frames with bad addressing information (e.g. CAN identifier) are ignored.	4.1 + SDL	CCC0
4	Frames with bad extended address byte are ignored.	4.1 + SDL	CCC0 + Extended
Transfer capabilities			
5	When using normal addressing the UUDT protocol supports 1 to N user data bytes (N = size of frame data field).	4.1	CCC0 + Normal

6	When using extended addressing the UUDT protocol supports 1 to N-1 user data bytes (N = size of frame data field).	4.1	CCC0 + Extended

3.1.2.2. UUDT sending state machine

The sending protocol consists of only one SDL state.

In the table below, the test stimuli for “successful transfers” are transmission requests from the interaction layer. The tests are expected to end with a transmission notification by the interaction layer. Successful transmission is notified by setting the message status and possibly activating a task or setting an event depending on system configuration.

“Error cases” lead to abort message transmission. The interaction layer shall not send any notification.

Nr	Assertion	Paragraph in spec.	Affected variants
Successful transfer			
1	A unique UUDT data frame is transmitted.	SDL	CCC0
Error case			
2	TA expiration due to bad UUDT frame transmission.	SDL	CCC0

3.1.2.3. UUDT receiving state machine

The receiving protocol consists of only one SDL state.

In the table below, tests of “successful receptions” are expected to end with a message reception notification by the interaction layer. Successful reception is notified by setting the message status and possibly activating a task or setting an event depending on system configuration.

Nr	Assertion	Paragraph in spec.	Affected variants
Successful reception			
1	End of message reception resulting from SF reception (data length < MUDBPF).	4.1	CCC0

3.1.3. USDT protocol test group

This section specifies tests purposes relative to the USDT protocol, as defined in chapter 4 of the COM specification. Test purposes have been established from the SDL diagrams presented in the specification, according to the Conformance Methodology described in document [1]. They intend to verify that the COM implementation behaviour conforms to the specification. They include:

- tests of state activity: tests are specified to verify actions performed by the

implementation on a given input,

- tests of state transitions: one test is specified for each event leading to move from a given state to another state of the COM specification.

Each test purpose defines both the test stimulus or stimuli to be sent and the subsequent output(s) to be observed at the COM API. Some actions can also be triggered by internal events. The test stimuli include:

- COM API procedure calls.
- USDT frames received by the implementation under test.
- Timer expirations (internal stimuli): TA, TB1, TB2, TD2.

The observable outputs are as follows:

- Status of application messages.
- Reception of application messages or absence of reception.
- USDT frames sent by the implementation under test.

Each test purpose also gives information on the specification variant(s) that need to be implemented for the test purpose to be verified. According to table 2.2 of COM specification, the USDT protocol is allowed in the following conditions:

- direct transmission mode,
- unqueued messages.

3.1.3.1. USDT protocol

Beside conformance classes, two variants have been identified. They concern the type of protocol encoding:

- *normal* for normal frame format,
- *extended* for extended frame format,

Nr	Assertion	Paragraph in spec.	Affected variants
Protocol formats			
1	For a message using normal addressing, address information is encoded in the frame header. PCI byte occupies the first byte of user data field.	4.1	CCC2 + Normal
2	For a message using extended addressing, there is eight bits of additional address information in the first byte of user data field. PCI byte occupies the second byte.	4.1	CCC2 + Extended
3	SF format is as follows: – high nibble of PCI byte equals 0 – low nibble of PCI byte contains data length (excluding PCI byte) – the following bytes contain the user data	4.3.2.2.1	CCC2

4	FF format is as follows: – high nibble of PCI byte equals 1 – low nibble of PCI byte contains the higher 4 bits of data length (excluding PCI byte) – the next byte contains the lower 8 bits of data length – the following bytes contain the user data	4.3.2.2.2	CCC2
5	CF format is as follows: – high nibble of PCI byte equals 2 – low nibble of PCI byte contains the Sequence Number – the following bytes contain the user data	4.3.2.2.3	CCC2
6	The Sequence Number is initialised to 0 when starting transmission of a message	4.3.2.2.3	CCC2
7	The Sequence Number is incremented up to 15 and re-initialised to 0 after wraparound.	4.3.2.2.3	CCC2
8	FC frame format is as follows: – high nibble of PCI byte equals 3 – low nibble of PCI byte specifies the flow status value: 0 for Clear To Send or 1 for WaiT – the next byte contains the maximum Block Size – the following byte contains the minimum Separation Time	4.3.2.2.3	CCC2
9	Padding of unused data bytes shall be performed in a single frame message, a flow control frame and the last consecutive frame of a multiple frame message. Padding pattern is not specified.	4.2.2	CCC2
Protocol errors			
10	Frames with bad addressing information (e.g. CAN identifier) are ignored.	4.1 + SDL	CCC2
11	Frames with bad extended address byte are ignored.	4.1 + SDL	CCC2 + Extended
12	Frames with unauthorised PCI values (i.e. > 3) are ignored.	SDL	CCC2
Transfer capabilities and segmentation			
13	An unsegmented message is carried out in a single frame. The USDT protocol supports 1 to max (MUDBPF, 15) user data bytes per single frame	4.2.3	CCC2
14	The USDT protocol supports up to 4095 user data bytes per message	4.2.4	CCC2

15	A multiple frame message consists of: – a First Frame containing the first (MUDBPF-1) user data bytes – 0 or more Consecutive Frames containing successive segments of MUDBPF user data bytes – the last Consecutive Frame containing the remaining (1 to MUDBPF) user data bytes	4.2.4	CCC2
16	On receipt of a First Frame, the receiving entity shall start assembling the segmented message. Then it shall buffer the received data bytes on receipt of successive Consecutive Frames until the whole message is received (number of bytes defined in the First Frame).	4.3.2.2.2 4.3.2.2.3	CCC2
17	For segmented messages, the maximum block size is 255 frames.	4.2.4	CCC2
18	For segmented messages, if block size is set to 0 by the receiver, no further flow control shall be performed during the transmission of Consecutive Frame(s).	4.2.4	CCC2
19	The USDT protocol shall be capable of carrying out parallel transmission of different messages, provided that they are not mapped onto the same (normal or extended) address.	4.4.4	CCC2

3.1.3.2. USDT sending state machine

Test purposes have been established from the SDL diagrams of the specification, according to the Conformance Methodology described in document [1]. The sending protocol consists of four SDL states, as follows:

- *idle*: no transfer of message is pending,
- *await_frst_fc_frm*: awaiting next FC frame after sending FF until timeout B1,
- *await_nxt_fc_frm*: awaiting next FC frame after sending last CF of block until timeout B2,
- *await_st_tim*: awaiting time-out of separation time (= STmin).

In the table below, the test stimuli for “successful transfers” are transmission requests from the interaction layer. The tests are expected to end with a transmission notification by the interaction layer. Successful transmission is notified by setting the message status and possibly activating a task or setting an event depending on the selected mechanism at system generation time.

“Error cases” lead to abort message transmission. The interaction layer shall not send any notification.

Nr	Assertion	Paragraph in spec.	Affected variants
Successful transfers			

1	End of message transmission in *idle* state (data length < MUDBPF). A SF is transmitted.	4.2.3 + SDL	CCC2
2	End of message transmission in *await_frst_fc_frm* state (MUDBPF <= data length < 2*MUDBPF). – a FF is transmitted – a CF is transmitted after FC(CTS,BS>=1) reception.	4.2.4 + SDL	CCC2
3	End of message transmission after wait in *await_frst_fc_frm* state (MUDBPF <= data length < 2*MUDBPF). – a FF is transmitted – nothing is transmitted after a FC(wait) reception – a CF is transmitted after FC(CTS,BS>=1) reception.	4.2.4 + SDL	CCC2
4	End of message transmission in *await_nxt_fc_frm* state (2*MUDBPF <= data length < 3*MUDBPF). – a FF is transmitted – a CF is transmitted after FC(CTS,BS=1) reception – a CF is transmitted after FC frame reception (BS>=1)	4.2.4 + SDL	CCC2
5	End of message transmission in *await_nxt_fc_frm* state after wait (2*MUDBPF <= data length < 3*MUDBPF). – a FF is transmitted – a CF is transmitted after FC(CTS,BS=1) reception – nothing is transmitted after a FC(wait) reception – a CF is transmitted after FC(CTS,BS>=1) reception	4.2.4 + SDL	CCC2
6	End of message transmission in *await_st_tim* state (2*MUDBPF <= data length < 3*MUDBPF). – a FF is transmitted – a CF is transmitted after FC(CTS,BS>=2) reception – a CF is transmitted after ST time-out (triggered in *await_frst_fc_frm* state*)*	4.2.4 + SDL	CCC2
7	End of message transmission in *await_st_tim* state (3*MUDBPF <= data length < 4*MUDBPF). – a FF is transmitted – a CF is transmitted after FC(CTS,BS>=3) reception – a CF is transmitted after ST time-out (triggered in *await_frst_fc_frm* state*)* – a CF is transmitted after ST time-out (triggered in *await_st_tim* state*)*	4.2.4 + SDL	CCC2
8	End of message transmission in *await_nxt_fc_frm* state. TB2 triggered in *await_st_tim* state (3*MUDBPF <= data length < 4*MUDBPF). – a FF is transmitted – a CF is transmitted after FC(CTS,BS=2) reception – a CF is transmitted after ST time-out – a CF is transmitted after FC(CTS,BS>=1) reception	4.2.4 + SDL	CCC2

9	End of message transmission in *await_nxt_fc_frm* state. TB2 triggered in *await_nxt_fc_frm* state (3*MUDBPF <= data length < 4*MUDBPF). – a FF is transmitted – a CF is transmitted after FC(CTS,BS=1) reception – a CF is transmitted after FC(CTS,BS=1) reception – a CF is transmitted after FC(CTS,BS>=1) reception	4.2.4 + SDL	CCC2
10	End of message transmission in *await_st_tim* state. ST triggered in *await_nxt_fc_frm* state (3*MUDBPF <= data length < 4*MUDBPF). – a FF is transmitted – a CF is transmitted after FC(CTS,BS=1) reception – a CF is transmitted after FC(CTS,BS>=1) reception – a CF is transmitted after ST time-out	4.2.4 + SDL	CCC2
Unexpected frames			
11	FC frames received in *idle* state are ignored	SDL	CCC2
Error cases			
12	TA expiration in *idle* state due to bad SF transmission	4.4 + SDL	CCC2
13	TA expiration in *idle* state due to bad FF transmission	4.4 + SDL	CCC2
14	TA expiration in *await_frst_fc_frm* state due to bad CF transmission	4.4 + SDL	CCC2
15	TA expiration in *await_nxt_fc_frm* state due to bad CF transmission	4.4 + SDL	CCC2
16	TA expiration in *await_st_tim* state due to bad CF transmission	4.4 + SDL	CCC2
17	TB1 expiration in *await_frst_fc_frm* state due to no FC frame reception after FF transmission	4.4 + SDL	CCC2
18	TB2 expiration in *await_nxt_fc_frm* state due to no FC frame reception after CF transmission in *await_frst_fc_frm* state	4.4 + SDL	CCC2
19	TB2 expiration in *await_nxt_fc_frm* state due to no FC frame reception after CF transmission in *await_nxt_fc_frm* state	4.4 + SDL	CCC2
20	TB2 expiration in *await_nxt_fc_frm* state due to no FC frame reception after CF transmission in *await_st_tim* state	4.4 + SDL	CCC2
21	TD2 expiration in *await_frst_nxt_frm* state after FC frame reception with Flow Status set to WaiT in *await_frst_fc_frm* state	4.4 + SDL	CCC2

22	TD2 expiration in *await_frst_nxt_frm* state after FC frame reception with Flow Status set to WaiT in *await_nxt_fc_frm* state	4.4 + SDL	CCC2
23	FC frame reception in *await_st_tim* state	SDL	CCC2
24	SF with data length > MUDBPF	SDL	CCC2

3.1.3.3. USDT receiving state machine

Test purposes have been established from the SDL diagrams of the specification, according to the Conformance Methodology described in document [1]. The receiving protocol consists of three SDL states, as follows:

- *idle*: no transfer of message is pending,
- *await_frst_cf_frm*: awaiting first CF of next block after sending of FC frame until timeout C,
- *await_nxt_cf_frm*: awaiting next CF of current block after receipt of previous until timeout D.

In the table below, tests of "successful receptions" are expected to end with a message reception notification by the interaction layer. Successful reception is notified by setting the message status and possibly activating a task or setting an event depending on the selected mechanism at system generation time. The value of BSmax sent by the implementation in the first FC may influence the test description. In that case, the tests associated to different BSmax values are numbered by the same number followed by a letter, e.g. 3a, 3b. The corresponding BSmax values are specified in the "Affected variants" column.

"Error cases" lead to abort message reception. The interaction layer shall not send any notification.

Nr	Assertion	Paragraph in spec.	Affected variants
Successful receptions			
1	End of message reception in *idle* state resulting from SF reception (data length < MUDBPF).	4.2.3 + SDL	CCC2
2	End of message reception in *await_frst_cf_frm* state (MUDBPF <= data length < 2*MUDBPF) – a FF is received – a FC(CTS) is transmitted – a CF is received.	4.2.4 + SDL	CCC2
3a	End of message reception in *await_nxt_cf_frm* state (2*MUDBPF <= data length < 3* MUDBPF) – a FF is received – a FC(CTS,BSmax,STmin) is transmitted – two CFs are received (time separation STmin).	4.2.4 + SDL	CCC2 + BSmax > 1 or BSmax = 0

3b	End of message reception in *await_frst_cf_frm* state (BSmax*(MUDBPF+1) <= data length < BSmax*(MUDBPF+2)) – a FF is received – a FC(CTS,BSmax,STmin) is transmitted – a CF is received – a FC(CTS,BSmax,STmin) is transmitted – a CF is received.	4.2.4 + SDL	CCC2 + BSmax = 1
4a	End of message reception in *await_nxt_cf_frm* state (BSmax*(MUDBPF+1) <= data length < BSmax*(MUDBPF+2)) – a FF is received – a FC(CTS,BSmax,STmin) is transmitted – BSmax CFs are received (time separation STmin).	4.2.4 + SDL	CCC2 + BSmax > 1
4b	End of message reception in *await_nxt_cf_frm* state (3*MUDBPF <= data length < 4*MUDBPF) – a FF is received – a FC(CTS,BSmax,STmin) is transmitted – three CFs are received (time separation STmin).	4.2.4 + SDL	CCC2 + BSmax = 0
5	End of message reception in *await_frst_cf_frm* state (BSmax*(MUDBPF+2) <= data length < BSmax*(MUDBPF+3)) – a FF is received – a FC(CTS,BSmax,STmin) is transmitted – BSmax CFs are received (time separation STmin) – a FC(CTS,BSmax,STmin) is transmitted – a CF is received.	4.2.4 + SDL	CCC2 + BSmax > 1
6	End of message reception in *await_nxt_cf_frm* state after wait for buffer resources – a FF is received – a FC(CTS,BSmax,STmin) is transmitted – BSmax CFs are received (time separation STmin) – a FC(WaiT) is transmitted – a FC(CTS,BSmax,STmin) is transmitted once resources are again available – a CF is received.	4.2.4 + SDL	CCC2 + BSmax > 0
Unexpected frames			
7	CFs received in *idle* state are ignored	SDL	CCC2
8	SFs received in *await_frst_cf_frm* state are ignored	SDL	CCC2
9	FFs received in *await_frst_cf_frm* state are ignored	SDL	CCC2
10	SFs received in *await_nxt_cf_frm* state are ignored	SDL	CCC2
11	FFs received in *await_nxt_cf_frm* state are ignored	SDL	CCC2
Error cases			

12	Unexpected Sequence Number received in CF in *await_frst_cf_frm* state	4.3.2.2.3 + SDL	CCC2
13	Unexpected Sequence Number received in CF in *await_nxt_cf_frm* state	4.3.2.2.3 + SDL	CCC2
14	TA expiration due to bad FC(CTS) transmission after FF reception in *idle* state	4.4 + SDL	CCC2
15	TA expiration due to bad FC(CTS) transmission after CF reception in *await_frst_cf_frm* state.	4.4 + SDL	CCC2
16	TA expiration due to bad FC(CTS) transmission after CF reception in *await_nxt_cf_frm* state.	4.4 + SDL	CCC2
17	TA expiration due to bad FC(WaiT) transmission after CF reception.	4.4 + SDL	CCC2
18	TE expiration due to lack of resources after WFTmax transmissions of FC(WaiT).	4.4 + SDL	CCC2
19	TC expiration due to no CF reception in *await_frst_cf_frm* state (TC triggered in *idle* state)	4.4 + SDL	CCC2
20	TC expiration due to no CF reception in *await_frst_cf_frm* state (TC triggered in *await_nxt_cf_frm* state)	4.4 + SDL	CCC2
21	TD1 expiration due to no CF reception in *await_nxt_cf_frm* state	4.4 + SDL	CCC2