ISO/DIS 15765-2 Transport Protocol

CANoe DLL – Version 4.10

Table of contents

1 Introduction

Efforts over the past several years to standardize diagnostic protocols and procedures internationally have largely been completed with the introduction of the diagnostic protocol "Keyword Protocol 2000" (KWP2000) at all European automobile manufacturers and with the definition by ISO/TF2 of a Transport Layer for diagnostic purposes (YAPTL or OSEK/COM 2.0).

The transport protocol standardized by ISO/TF2 is the basis for data transfer during diagnosis, but is also used in other communication applications in which messages must be exchanged that are larger than the transport capability of a CAN message. The purpose of the CANoe expansion DLL described in this document is to model and simulate bus nodes which utilize this protocol.

Its implementation and specification are based on information of the following document:

- YAPTL - Yet Another Proposal for Transport Layer; Version 1.5; 06-04-97
- Releases from 3.0 implement the OSEK TP defined in the standard ISO/DIS 15765-2 (1999-11-30)
- Releases from 3.2 implement the OSEK TP defined in the standard ISO/DIS 15765-2 (2001-11-01)
- Releases from 4.1 implement the OSEK TP defined in the standard ISO/DIS 15765-2 (2003-11-11)

The version of the DLL described here is 4.13.27 (cf. 1.2 for details).

1.1 Abbreviations

Abbrev.	**Meaning**
BS	Block size, is defined by the receiver and transmitted to the sender in FC
CF	Consecutive Frame
FC	Flow Control
FF	First Frame
SA	Source Address (address of sender)
SF	Single Frame: Transport layer message that fits into a single CAN message (6 (extended A.) or 7 (normal A.) data bytes)
Stmin	Minimum interframe space, is defined by the receiver and transmitted to the sender in FC
TA	Target Address
TPCI	Transport Protocol Control Information, is transmitted in Byte 0 (normal address) or Byte 1 (extended address).
WFTmax	The maximum number of consecutive wait frames the receiver sends before aborting the transmission.

1.2 New Features of DLL version 4.14

- The DLL reports its functions with more information in the CAPL browser selection dialog. In addition, the functions have been divided into two groups, one for normal and one for fault injection functions.
- The extended (29 bit) CAN ID format is now fully available in normal and extended addressing modes. Note that a 29 bit base address for extended mode can only be set using the CAPL function `OSEKTL_SetTpBaseAdr` and that the usage of extended IDs has to be activated with the call `OSEKTL_UseExtId( 1);`.
- It is possible to use a special ASDT (*acknowledged* segmented data transfer) mode extension. Please cf. to section 3.3 for details.

1.3 New Features of version 4.9

- First- and Single-Frames are also indicated in the `OSEKTL_FI_PreSend` callback (cf.7.3.2).
- In *extended* addressing mode, it is possible to specify a range of addresses for reception, instead of using only the rx-mask (cf. 3.1.3.2, `OSEKTL_SetMsgCount()`).

1.4 New Features of version 4.7

1.4.1 Extended fault injection feature

The fault injection feature has been extended. It is now possible to delay any message for any given time, and to change the id of the message that will be sent. In addition, the position of the protocol configuration byte in the message to sent may be retrieved, allowing mode independent access to the data.

Important note: the "presend" callback (cf. 7.3.2) is now called **before** a message is delayed! This may change the timing behaviour of CAPL programs when a timer is started from within this callback.

1.4.2 New API Functions

Function Name	Description
`void OSEKTL_SetFCDelay( dword milliseconds)`	Set the delay of all FC frames to this value. (default: 0)
`void OSEKTL_SetWFTMax( dword WFTmax)`	Set the maximum number of FC.WT frames the receiver will send before aborting the reception. (default: 10)
`void OSEKTL_UseBSOfFC()`	Use the BS sent in the FC.CTS (default)
`void OSEKTL_SetFixedBS( long blocksize)`	Set the block size used by the sender to a fixed value, i.e. the BS returned in the FC.CTS is ignored.
`void OSEKTL_SetMinSTmin( dword milliseconds)`	The STmin value sent by the receiver in a FC.CTS is only used if it is larger than this value. (default: 0)

These are additional fault injection functions that will only be active if fault injection has been activated. Refer to 7.3.2 for details and 7.5.2 for an example.

`void OSEKTL_FI_AllCTS( long flag)`	Treat all FC frames as CTS. This function may be called any time and will take effect for all following transmissions.
`void OSEKTL_FI_SetDelay( dword MICROseconds)`	(*) Set the delay of the message that will be sent next. NOTE: the delay is given in *micro* seconds (µs).
`Dword OSEKTL_FI_GetDelay()`	(*) Returns the delay the message will experience if the value is not changed with `OSEKTL_FI_SetDelay`.
`Long OSEKTL_FI_IsFC()`	(*) Returns 1 if the message is a FC frame.
`Dword OSEKTL_FI_GetPCIOff()`	(*) Returns the index of the PCI byte in the message data.
`Void OSEKTL_FI_SetId( dword id)`	(*) Set the message id to this value.
`Dword OSEKTL_FI_GetId()`	(*) The message will be sent with this id.

(*) These functions may only be called from the `OSEKTL_FI_PreSend` callback! They access the current message only.

1.5 Corrections and clarifications

- The deactivation of the usage of IDs from addresses and addresses from IDs in extended mode (cf. 2.1.2) also works for data transfers that do not fit into a single frame.
- The documentation of the Functions for Data Transfer (cf. 3.1.1) has been corrected: the data type is not `char*`, but `byte*`. An example has been added.

1.6 New Features of version 4.4

1.6.1 Standard (2003-11-11) extensions

- The “11 bit mixed” addressing mode is supported (cf. 2.1.5).
- For compatibility, the different handling of empty single frames and the changed interpretation of the STmin value have to be activated explicitly (cf. 2.1.7).

1.6.2 New functionality

CAPL nodes are now able to send wait-frames to the sender (cf. 2.1.6)

1.6.3 New API Functions

Function Name	**Description**
`Long OSEKTL_GetWaiting()`	Returns the wait state of the node. If set, the node will send FC.WT frames instead of FC.CTS frames to a sender (cf. 2.1.6)
`OSEKTL_SetWaiting(long state)`	Set the wait state of the node. The node will send wait frames to a sender if set to 1.
`OSEKTL_SetUseAllAE(long state)`	If set to a value other than 0, the node will accept transmissions in (11 bit) Mixed mode using any address extension value. Otherwise the value has to match (default, cf. 2.1.4).

Function Name	Description
`Long OSEKTL_GetUseAllAE()`	Returns 0 if the address extension has to match exactly in the mixed addressing modes.
`Long OSEKTL_GetRecentAE()`	Returns the address extension of the most recent reception for (11 bit) Mixed mode (cf. 2.1.5).
`OSEKTL_Set11Mixed()`	Toggles to 11 bit mixed addressing

1.6.4 Additional database attribute

Attribute Name	Belongs to	Description
`TpUse2003Ext`	Node	If 1, the 2003 extensions will be used (cf. 2.1.7).

1.7 Corrections and clarifications

The functions `OSEKTL_SetFrameDlcVar` and `OSEKTL_SetFrameDlc8` are actually called `OSEKTL_SetDlcVar` and `OSEKTL_SetDlc8` in the DLL.

Examples for the usage of the `OSEKTL_FI_SendXByte` fault injection function are included.

A new section “Frequently asked questions” is included.

1.8 New Features of Version 4.0

In this version, the Fault Injection support was introduced. Please refer to section 7. Additionally, the delay of the first Consecutive Frame can be set to STmin or 0 with the INI file switch `STmin-BeforeFirstCF` (default: 1, i.e. the first CF is delayed) and it is possible to use functional addressing in normal addressing mode (i.e. set the functional address range).

1.9 New Features of Version 3.3

This version supports the new multibus mode introduced in CANoe 3.2 SP2. It allows extended support for gateways.

1.10 New Features of Version 3.2

1.10.1 Bug Fixes

- Corrected wait frame behaviour: the reception of a different type of flow control frame will reset the wait frame counter.

1.10.2 New Functionality

Version 3.2 supports the sending and reception of overflow flow control frames (FC.OVFLW). A receiver may decide to abort the reception of a TP message after it received a First Frame by sending this new flow control frame type, because the message size is larger than a specified value. The sender must not send any Consecutive Frames. Sender and receiver are notified of this event by a call to the error indication callback function `OSEKTL_ErrorInd(int)` with value 8 (receiver buffer overflow).

Also, for the Mixed addressing mode the Protocol Data Unit Format (PF) value was changed to the value specified in the standard (8.3.4), i.e. instead of the values 218 (0xDA) and 219 (0xDB) like for the NormalFixed addressing mode, the values 206 (0xCE) and 205 (0xCD) are used.

1.10.3 New API Functions

Function Name	Description
`Long OSEKTL_GetMaxMsgLen()`	Return the maximum TP message length the node will accept without sending an overflow frame. (default: 4095)
`OSEKTL_GetMaxMsgLen (long len)`	Set the maximum acceptable length (<= 4095 byte) of a TP message. The transmission of longer messages will be aborted by sending an overflow (FC.OVFLW) frame.
`Long OSEKTL_GetTimeoutBr()`	Returns the value of the timer Br [ms]
`OSEKTL_GetTimeoutBr(long val)`	Sets the value of the timer Br to val ms

1.10.4 New INI file switches

Switch Name	Description
`Tbr`	Timeout value Br [ms] ”Time until transmission of next FlowControl N_PDU”

1.11 New Features of Version 3.1

1.11.1 Bug Fixes

- Databaseattribute TpUseFC corrected
- UseFC=0: No FC at all is expected even after FF
- OSEKTL_SetCAN() works properly only in ‘PreStart’, and after this bugfix without exception. Affects physical and functional addressing.

1.11.2 New API Functions

Function Name	Description
`long OSEKTL_IsUseFFPrio()`	Priority for FC is used 0: as configured 1: as received in FF
`OSEKTL_SetUseFFPrio(long prio)`	Priority for FC will be used 0: as configured 1: as received in FF
`Long OSEKTL_GetFCPrio()`	Returns the priority used in FC frames
`OSEKTL_SetFCPrio(long prio)`	Sets the priority used in FC frames

1.11.3 New INI file switches

Function Name	Description
`TxPriority`	NormalFixed & mixed addressing mode: Priority used for SF, FF & CF
`FCPriority`	NormalFixed & mixed addressing mode: Priority used for FC
`UseFFPriority`	NormalFixed & mixed addressing mode: Priority used for FC derived from received FF

Function Name	Description
WriteLevel	Controlling debug outputs to the write window. The higher the write level, the more debug information is printed to the write window.

1.11.4 Usability

After setting the write level in the INI file to at least 1, parameter settings of the nodes will be checked for consistency and potential problems will be reported to the write window.

1.12 New Features of Version 3.0

1.12.1 New Addressing Modes

Version 3.0 supports the Addressing Modes 'NormalFixed' and 'Mixed'.

1.12.2 Additional Database Attributes

Function Name	Belongs to	Description
TpTxAdrMode	Node	node addresses target in physical (0) or functional (1) mode (address modes normal fixed and mixed)
NWM-Stationsadresse	Node	alternate description of ECU (compare with 'Diagnose-Stationsadresse')
TpAddressExtension	Node	Sets the address extension used for mixed addressing mode
TpSTMin	Node	Minimum Separation Time required for this node
TpBlockSize	Node	Block Size for this node

1.12.3 New API functions

Function Name	Description
OSEKTL_SetAdrExt(BYTE ae)	Sets the address extension to ae (has only effects in mixed addressing mode)
long OSEKTL_GetAdrExt()	Gets the address extension
OSEKTL_SetMixedMode()	Sets the addressing mode to 'mixed'
OSEKTL_SetNrFixMode()	Sets the addressing mode to 'normalfixed'
OSEKTL_UseExtId(long val)	val=1: Nodes use Extended CAN IDs val=0: Nodes use Standard CAN IDs
OSEKTL_SetTxPhys()	Sets the addressing mode of the following transmissions to physical addressing
OSEKTL_SetTxFunc()	Sets the addressing mode of the following transmissions to functional addressing
long OSEKTL_IsTxModePhys()	1 if Tx mode is physical or else 0

Function Name	Description
`long OSEKTL_IsUseExtId()`	1: Nodes use Extended CAN IDs 0: Nodes use Standard CAN IDs
`long OSEKTL_GetLoRxId()`	Lowest CAN ID which passes Rx Filter
`long OSEKTL_GetHiRxId()`	Highest CAN ID which passes Rx Filter
`long OSEKTL_GetStartSN()`	Sequence number used for first CF
`long OSEKTL_GetFixedDLC()`	0: Minimum DLC for CAN frames used x=1-8: At least DLC x used
`long OSEKTL_GetTimeoutXX()`	Reads the timeout value XX [ms] (XX may be: Ar, Cr, As or Bs) (described in ISO/TF2)
`OSEKTL_SetTimeoutXX(long val)`	Sets the timeout value XX (XX may be: Ar, Cr, As or Bs) (described in ISO/TF2) to val ms
`long OSEKTL_GetCAN()`	Reads the channel on which a node receives and sends frames
`OSEKTL_SetCAN(long can)`	Sets the channel on which a node receives and sends frames. (Only works properly in 'PreStart')
`long OSEKTL_IsUseFC()`	Flow Control is 1: used 0: not used
`OSEKTL_SetUseFC(long val)`	Set Flow Control is 1: used 0: not used
`OSEKTL_UseAdrFromId(long val)`	see below
`long OSEKTL_IsAdrFromId()`	see below
`OSEKTL_UseIdFromAdr(long val)`	see below
`long OSEKTL_IsIdFromAdr()`	see below
`OSEKTL_SetTxPrio(long val)`	sets priority of transmission in normal fixed and mixed addressing mode
`long OSEKTL_GetTxPrio()`	reads the priority of transmission in normal fixed and mixed addressing mode

1.12.4 Other changes

- Timeouts are calculated as described in ISO/TF2. Transmit Acknowledgements for sent frames are taken into account.
- All available CAN channels can be used (not only channels 1 and 2)
- More features can be configured via INI file
- The following CAPL APIs are no longer supported:
  - `OSEKTL_SetIdFromAdr()`
  - `long OSEKTL_GetIdFromAdr()`

- `OSEKTL_ClrIdFromAdr()`

Their functionality has been implemented by the following APIs:

- `long OSEKTL_IsAdrFromId()`
- `long OSEKTL_IsIdFromAdr()`
- `OSEKTL_UseAdrFromId()`
- `OSEKTL_UseIdFromAdr()`

1.13 New Features of Version 2.5

1.13.1 New API functions

Function Name	Description
`OSEKTL_SetEvalOneFC()`	Only the First FC is evaluated, as specified in ISO.
`OSEKTL_SetEvalAllFC()`	All FCs are evaluated. This is the default. Same behavior as DLL 2.4 and older
`long OSEKTL_GetEvalOneFC()`	True, if only one FC is evaluated
`OSEKTL_Set0Pattern (long pattern)`	Sets the pattern to be used for zero padding
`OSEKTL_Get0Pattern()`	Returns the pattern, default is 0xff
`OSEKTL_Set0Padding()`	Switches zero padding on
`OSEKTL_Clr0Padding()`	Switches zero padding off. This is the default
`OSEKTL_Set1FC_BS(byte bs)`	According to ISO, BS=0 means that there is only one FC (the one after the FF). Some manufacturer decided, to give BS=255 this meaning.
`long OSEKTL_Get1FC_BS()`	Returns the BS with the special meaning explained above.
`OSEKTL_ClrIdFromAdr()`	Extended Addressing only: Usually, the CAN-ID is derived from the sender address (and the base address). After the call of this function the CAN-ID is not automatically calculated and can be set using the function `OSEKTL_SetTxId(canId)`, (like in normal addressing mode). This function also disables the inverse calculation: On reception of a transport layer message, the target address for a FlowControl frame is not derived from the ID. The target address set with `OSEKTL_SetTgtAdr(ta)` is used.
`OSEKTL_SetIdFromAdr()`	Extended Addressing only: Switches back to default behavior, that is the CAN-ID is derived from the base address and the sender address.

Function Name	Description
`long OSEKTL_GetIdFromAdr()`	Extended Addressing only: Returns true in the default mode (ID calculated from address)

1.13.2 New Callback functions

There is one new callback function. It indicates the reception of a FirstFrame. As proposed by the standard, the arguments contain both address information and the data length.

Function Name	Description
`void OSEKTL_FirstFrameIndication (long source, long target, long length)`	Indicates, that a first frame has been received. The meaning of the parameters depends on the addressing mode: Extended addressing: • source: Source (sender) address • target: Target (receiver) address Normal addressing: • source: CAN-ID • target: CAN-Id `length` is the length of the message for both addressing modes.

1.13.3 Additional Database Attributes

Function Name	Belongs to	Values	Description
`TpAddressingMode`	Node	0,1	0 means: normal addressing 1 means: extended addressing This value overwrites the value given in the ini file. If the addressing mode is specified neither in the ini file nor in the database, normal addressing is used.
`TpTargetAddress`	Node	0..ff	This attribute is relevant for extended addressing only: It specifies the target address. Can also be set in CAPL using `OSEKTL_SetTgtAdr(ta)`.

1.13.4 Other changes

- The addressing mode can now be changed at runtime without problems.
- The default of the receive mask has been changed from 0xff to 0xffff.

  - Normal addressing: Just any receive identifier can be used and selected during measurement
  - Extended addressing: The receive range can be changed during measurement
- Changing the identifiers is now also possible for 29-Bit CAN-Identifiers
- In extended addressing mode the “base address” and the “ECU number” are used to determine the Tx Identifier. SetTxIdentifier() has no effect in this mode.
- The flow state of FlowControl frames is checked: Only 0 (clear to send) and 1 (wait) are accepted. Otherwise stated: FCs with TPCI bytes from 0x32 to 0x3f are not considered as FCs and rejected.
- The length of rx messages is checked: The message as specified in the PDU (TPCI byte) may not be longer than the frame. For example: The dlc of a FC has to be 3 or greater. (4 for extended addressing). The dlc of a FirstFrame with 2 data bytes has to be at least 3 (4 for extended addressing).
- Extended Addressing only: After reception of a ConsecutiveFrame, the target **and source** address are compared to the ones currently used. In version 2.4 or older only the target address was taken into account.

2 Properties of OSEK_TP.DLL

2.1 Versions and Functional Features

The DLL implements the transport protocol of ISO/TF2.

Handling of FlowControlStatus and ErrorFrame as well as error-corrected transmission are not defined in this specification version, and consequently they are not implemented here.

The ISO protocol recognizes four different modes, normal addressing, extended addressing, normalfixed addressing and mixed addressing. The DLL supports all addressing modes.

The current version of the DLL is 3.1.

2.1.1 Normal Addressing

Source information and target information (SA and TA) are coded in the CAN identifier.

Two identifiers are needed per point-to-point connection.

2.1.2 Extended Addressing

The sender information is coded in the CAN identifier, and the target address is coded in the first data byte. The CAN ID is the sum of a so-called base address and the controller address:

```
CAN-ID= Base address + controller address
```

In default mode the tx identifier is automatically calculated using the formula above. This behavior can be changed by calling `OSEKTL_UseIdFromAdr(0)`. In this mode the CAN-Id has to be set using `OSEKTL_SetTxId(id)`. The default behavior is activated with `OSEKTL_UseIdFromAdr(1)`.

In default mode the target address to which the flow control frames are sent, are automatically calculated using from the received identifier. This behavior can be changed by calling `OSEKTL_UseAdrFromId(0)`. In this mode the target address has to be set using

`OSEKTL_SetTgtAdr(ta)` (which also influences the Tx section of the node). The default behavior is activated with `OSEKTL_UseAdrFromId(1)`.

2.1.3 Normal Fixed Addressing

The source and sender information (SA and TA) are coded in an extended CAN identifier.

Two identifiers are needed per point-to-point connection. Transmissions can be made using functional addressing or physical addressing. See TxAdrMode for details.

2.1.4 Mixed Addressing

The source and sender information (SA and TA) are coded in an extended CAN identifier as in normal fixed addressing mode.

Additionally there is an address extension coded in the first data byte of the CAN frame.

Two identifiers are needed per point-to-point connection. Transmissions can be made using functional addressing or physical addressing. See TxAdrMode for details.

As default, nodes will accept transmissions only if the address extension matches the value set for the node. This can be changed by calling `OSEKTL_SetUseAllAE(1)`: all transmissions will be accepted and the receiver can query the address extension with `OSEKTL_GetRecentAE()`.

2.1.5 11 bit Mixed Addressing

This mode is like the normal mode, but the first data byte in the CAN frame is used as address extension.

Two identifiers are needed per point-to-point connection, and the sender may set the address extension value to use. The address extension value is handled as in the Mixed mode (2.1.4).

2.1.6 Special Aspects of the Implementation

- The DLL reacts to WaitFrames (Flow control messages with the contents "Wait" instead of "ClearToSend") and can send WaitFrames itself if the node is brought into wait mode with the CAPL function `OSEKTL_SetWaiting(1)`. Note that if the number of WaitFrames sent by the receiver exceeds the specified maximum, the receiver will call the error indication callback with error number 7, “Too many FC.WFT sent”. On the sender side, the missing FlowControl frame will lead to an error indication 2, “Timeout waiting for FC”.
- There are only one transmit channel and one receive channel per simulated network node. If - during reception of a long message - a Single Frame or First Frame of a second long message arrives, the first message is canceled, an error indication sent to the application and the second long message will be received.

2.1.7 Standard 2003 extensions

If the DLL is configured to use the extensions introduced in the 2003-11-11 version of the standard, then

- Attempts to send data of size 0 (i.e. a single frame of length 0) will fail, calling the `OSEKTL_ErrorInd()` callback with error code 9, “Wrong parameter”.
- The new STmin value interpretation is used: The values 0x80-0xF0 and 0xFA-0xFF are reserved and will be interpreted as 127 ms if received. Values 0xF1-0xF9 will be interpreted as 100 to 900 µs (please note that the accuracy of the timing depends on the bus drivers used!).

Since these extensions are incompatible to the old standard, the DLL has to be configured to use them explicitly: via the new node attribute `TpUse2003Ext` or the `OSEKTL_Set2003Ext()` CAPL function.

2.2 Node-Specific Simulation

The implementation permits automatic instancing of the functionality for each bus node within the CANoe simulation. The options and data necessary for this are described in section *2.3* Parameterization.

2.3 Parameterization

All parameters given in the INI file will be read at measurement start. After this, the database is checked for the attributes described below. If any attribute is defined in the database, either the default value of this attribute or the value specified for the concerned node will overwrite the settings from the INI file.

On measurement start the parameterization is checked for consistency and if the write level is set to at least 1, potential problems will be reported to the write window. Certainly not all settings which may impair the communication can be checked, but the user will be informed about inappropriate combinations of baseaddress and rxmask, short termed timeouts and activated receive filters.

Finally, during the measurement the parameters can be overruled by using the appropriate CAPL APIs.

2.3.1 Ini File

Only node-independent values are taken from the initialization file osek_tp.ini . An example of this is the initialization file for the Demo shown below. Please refer to the description of the corresponding CAPL function for details on a value's effect.

```
[Intern]
WriteLevel = 3

#-----------------------------------------------------#
# Default values - may be changed in CAPL             #
# Database attributes may overwrite these values      #
#-----------------------------------------------------#

[Common]
# Addressing Mode
# 0: normal(*)/ 1: extended/ 2: normalfixed/ 3: mixed/
# 4: mixed11
TP_AddressingMode =  Normal

# Used CAN Channel (default=1)
CANBus = 1

#----- extended addressing --------------------------#
Target_Offset= 0x600
Start_Functional_Addresses= 0xE0
End_Functional_Addresses  = 0xEF

#----- mixed addressing -----------------------------#
# Address Extension (default=0)
TP_AddressExtension = 0

# Determines the receive range and activation of Rx Filter
```

```
# (default=0: Rx Filter deactivated)
TpRxIdentifier_Lo  = 0x601
TpRxIdentifier_Hi  = 0x6FE
TpUseRxFilter      = 0
#-----------------------------------------------------#

#----- normal addressing --------------------------#
# Default value for Rx identifier
# May be overwritten by value in CANdb
#
TpRxIdentifier = 0x603
TpTxIdentifer = 0x6F3
#-----------------------------------------------------#

#-----------------------------------------------------#
# Configuration Parameters which change behavior     #
# of the nodes                                       #
#-----------------------------------------------------#

# Blocksize to send to source node (default=2)
Blocksize = 2

# STMin to send to source node (default=0)
STMin = 20


#-----------------------------------------------------#
# Configuration Parameters which change the behavior #
#          of the transport layer                    #
# (all default values correspond to ISO/TF2)         #
#-----------------------------------------------------#

# Maximum Number of waitframes (default=5)
WFTMax = 5

# Timeouts (defaults=1000)
Tas = 1000
Tbs = 1000
Tar = 1000
Tcr = 1000

# Frame Length
# Use Frames with fixed length (3..8)
# (default: 0: variable minimum length of frames)
FixedFrameLength = 0

# ZeroPadding
# 1: Use, 0: Don't use (*)
ZeroPadding = 0
PaddingValue = 0x00

# Extended CAN Identifier
# 1: use extended CAN Ids, 0: use Std CAN Ids (*)
# (only for normal and extended addressing)
ExtendedId=0
```

```
# Delay the first CF by STmin?
# 0: send immediately after receiving FC, if FCs are used.
STminBeforeFirstCF=1
# The standard is not completely clear on the behaviour,
# since so far STmin was kept, it is the default here.

# ---   Block Size    ---#
# Block size to switch flow control off (default=0)
BSOff = 0

# --- Separation time ---#
# ~ used, if STMin from FC is ignored (default=0)
FixedSTMin = 100
# 1: use STMin from FC (*), 0: ignore STMin from FC
STMinFromFC = 0

# --- BS & STMin ---#
# 0: Using only first FC to determine BS and STMin (default)
# 1: Using all FC to determine BS and STMin
UseAllFC = 0

# Use Flow Control (default=1)
UseFC=1

# Sequence Number for first CF (default=1)
FirstSN=1

# Calculation of Id or address from received frame
# (defaults=1)
IdFromAdr = 1
AdrFromId = 1

# Calculation of Id or address from received frame
# (defaults=1)
//IdFromAdr = 1
//AdrFromId = 1

# Mixed and NormalFixed addressingmode
# 1: Using Priority of received FF for FC;
# 0: Using configured priority for FC (default)
//UseFFPrio = 0

# Mixed and NormalFixed addressingmode
# Priority of FC (0..8) (default=3)
//FlowControlPrio = 3
# Priority of SF, FF & CF (0..8) (default=3)
//TransmitPrio = 3
```

- Changes to the initialization file are assumed by CANoe at the start of measurement.
- The first entry, `WriteLevel`, controls the number of messages that are output to the Write window. The higher the number, the more messages.
- `ExtendedId` indicates whether standard (11 bit) or extended (29 bit) identifiers should be used.

- `TP_AddressingMode` must be set to `normal, extended, normalfixed or mixed`. Alternatively the values 0..3 can be used for the selection of the address mode. The mode may be toggled during measurement time.
- `Target_Offset`: Base address, is only used with extended addressing.
- `Start_Functional_Addresses` and `End_Functional_Addresses` indicate the address range that is interpreted as functional (only for normal and extended addressing)
- `TpRxIdentifier_Lo` and `TpRxIdentifier_Hi` indicate the receive range. Can be overwritten using `OSEKTL_SetRxRange(idLo, idHi)`. If the so described Receive-Filter shall be used `TpUseRxFilter` must be set to 1.
- `TpRxIdentifier` indicates the receive identifier (only for normal addressing) and `TpTxIdentifier` the sending identifier.

2.3.2 Old Format

The format described below is obsolete and might not be supported in future releases of the DLL!

In older versions of the transport layer DLL (Name: osek20tl.dll or tp_ext.dll) all information was taken from the initialization file.

This has been made possible again (effective from Version 2.2 on) to permit working with old models without requiring significant changes. Old initialization files have the following format:

```
#-----------------------------------------------------#
#----- old style ini file ----------------------------#
#-----------------------------------------------------#

[Intern]
Ausgabe = 3

[NodeA]
TP_Request = 0x6f1
TP_Response = 0x6ff
CANBUS = 1
STMIN = 45
BS_FC = 9
TimeoutFC = 1000
TimeoutCF = 1000
TransmitCF = 40


[NodeB]
TP_Request  = 0x734
TP_Response = 0x4f4
STMIN       = 40
BS_FC       = 8
TimeoutFC   = 40
TimeoutCF   = 70
TransmitCF  = 41
CANBUS      = 1
```

The primary difference compared to the current format is that information must be input for each node (node-specific information can now be stored in the database).

If the DLL finds node-specific information in the ini file for the transmit identifier, then it is assumed that the old format applies. A message is output in the Write window telling the user that in order to utilize the latest performance features the ini file must be converted to the new format. Significant limitations when using the old format include:

- Only normal addressing
- Only standard identifiers (no 29-bit identifiers)
- The receive identifier of a node cannot be changed at runtime.

Other limitations of the old DLL versions (maximum data length 255 bytes, block size < 16, no evaluation of STmin in FC) do not apply any longer, independent of which initialization file format is used.

2.3.3 Database

The following table provides an overview of the attributes relevant to the DLL:

Attribute	**Belongs to**	**Comment**
`TpNodeBaseAddress`	Node	0...2031[1] - set to a default value for all nodes, or use an individual node value.
`NodeLayerModules`	Node	Must be set to osek_tp.dll. If other entries are made they must be separated by commas.
`TpTxIdentifier`	Node	Transmit ID (normal addressing)
`TpRxIdentifier`	Node	Receive ID (normal addressing)
`Diagnose-Stationsadresse`	Node	0...255 (extended addressing)
`TpRxMask`	Node	0...7ff (0...2031): Identifies the receive range
`TpCanBus`	Node	Identifies the CAN channel used.
`TpUseFC`	Node	Indicates whether flow control messages should be used (= 1) or not (= 0)
`TpAddressingMode`	Node	0: normal addressing 1: extended addressing 2: normal fixed addressing 3: mixed addressing 4: 11 bit mixed addressing This value overwrites the value given in the ini file. If the addressing mode is specified neither in the ini file nor in the database, normal addressing is used.

[1] A CAN identifier consists of 12 bits, that is 2048 different ids. But the protocol allows only the ids from 0...2031 to be used, as the first CAN chip implementation used the bit pattern from 2031...2047 for internal purposes. So 0x7ff = 2048 actually means 2032.

Attribute	Belongs to	Comment
`TpTargetAddress`	Node	This attribute is relevant for extended addressing only: It specifies the target address. Can also be set in CAPL using `OSEKTL_SetTgtAdr(ta)`.
`TpTxAdrMode`	Node	node addresses target in physical (0) or functional (1) mode (address modes normal fixed and mixed)
`NWM-Stationsadresse`	Node	alternate description of ECU number (compare with 'Diagnose-Stationsadresse')
`TpAddressExtension`	Node	Sets the address extension used for (11 bit) mixed addressing mode
`TpSTMin`	Node	Minimum Separation Time required for this node
`TpBlockSize`	Node	Block Size for this node
`TpUse2003Ext`	Node	Value of 1 will activate 2003 extensions (cf. 2.1.7)

In the database that is used, the *NodeLayerModules* attribute must be defined as a node-specific attribute.

The user-specific attributes of each node using the transport layer must have the entry *NodeLayerModules* set to osek_tp.dll.

2.3.4 CANoe Options

The node to be simulated is entered in CANoe's *Simulation Setup* window. In doing so, it is important that the name used in the database agree exactly with the name assigned in the simulation setup. If configured properly, a "NL" appears in the symbol for the controller to be simulated.

2.3.5 Model Parameterization

No parameters need to be configured in the model itself. Only the data buffers for communication with the DLL need to be created.

3 Functional Description

3.1 CAPL Expansions

The DLL expands all configured simulation nodes by the CAPL commands listed below for utilization of the transport layer. Unless otherwise specified, the return value of functions is `void` (no return value).

All OSEKTL_ functions can be called in all addressing modes. However changes take only effect when the addressing mode is set to the mode mentioned in the headline of the appropriate chapter.

3.1.1 Functions for Data Transfer

Name	Comment
`OSEKTL_GetRxData (byte* rxData, int bufferSize)`	Retrieves the received data
`OSEKTL_DataReq (byte* txData, int length)`	Places a transmit request
`Long OSEKTL_GetSrcAdr()`	Returns the Source Address of the last received message

Example:

```
On key 's' {                    // Send some data after pressing key s
   Byte data[100];
   OSEKTL_DataReq( data, 27);    // Requested the sending of data.
   // Transmission completion is indicated by callback OSEKTL_DataCon( long txLength).

}

OSEKTL_DataInd( long count) {    // Indicate the reception of data.
   Byte data[ 100];
   OSEKTL_GetRxData( data, elcount( data));
   Write("Received %d byte from 0x%x: [%02x ...]", count, OSEKTL_GetSrcAdr(), data[0]);
}
```

3.1.2 Addressing Mode

Name	Comment
`OSEKTL_SetExtMode()`	Toggles to extended addressing
`OSEKTL_SetNrmlMode()`	Toggles to normal addressing
`OSEKTL_SetMixedMode()`	Toggles to mixed addressing
`OSEKTL_SetNrFixMode()`	Toggles to normalfixed addressing
`OSEKTL_Set11Mixed()`	Toggles to 11 bit mixed addressing
`Long OSEKTL_GetAdrMode()`	Returns the active addressing mode (0: normal, 1: extended, 2: normalfixed, 3: mixed, 4: 11 bit mixed)

3.1.3 Protocol Parameters for all addressing modes

Name	Comment
`Long OSEKTL_GetBS()`	Returns the active block size (requested as receiver)
`OSEKTL_SetBS(long bs)`	Sets the block size
`void OSEKTL_UseBSOfFC()`	As sender, use the BS sent in the FC.CTS (default)
`void OSEKTL_SetFixedBS( long blocksize)`	Set the block size used as sender to a fixed value, i.e. the BS returned in the FC.CTS is ignored.

Name	Comment
`Long OSEKTL_GetSTMIN()`	Returns the active Stmin
`OSEKTL_SetSTMIN(long stmin)`	Sets STmin
`OSEKTL_SetFixedST(long st)`	st is used as separation time instead of STmin. If st < STmin, this results in a protocol violation
`OSEKTL_UseSTminOfFC()`	STmin of Flow Control messages is used. This is the default behavior
`void OSEKTL_SetMinSTmin( dword milliseconds)`	The STmin value sent by the receiver in a FC.CTS is only used if it is larger than this value. (default: 0)
`OSEKTL_SetStartSN(long sn)`	sn is the sequence number used for the first consecutive frame in a segmented message. Default value is 1. Some manufacturers specifiy 0. All other values are incorrect, but not rejected by the DLL.
`OSEKTL_SetDlcVar()`	The length of the CAN frames is adapted to the data length. For instance a flow control frame has dlc=3 (4 if extended Addressing is used). The length of first frames and the last consecutive frame of a segmented message is also adapted.
`OSEKTL_SetDLC8()`	Length (dlc) of the CAN frames is always 8, independent of the amount of data to be transmitted. This is the default behavior.
`OSEKTL_SetEvalOneFC()`	Only the First FC is evaluated, as specified in ISO.
`OSEKTL_SetEvalAllFC()`	All FCs are evaluated. This is the default. Same behavior as DLL 2.4 and older
`Long OSEKTL_GetEvalOneFC()`	True, if only one FC is evaluated
`OSEKTL_Set0Pattern(pattern)`	Sets the pattern to be used for zero padding
`Long OSEKTL_Get0Pattern()`	Returns the pattern, default is 0x00
`OSEKTL_Set0Padding()`	Switches zero padding on
`OSEKTL_Clr0Padding()`	Switches zero padding off. This is the default
`OSEKTL_Set1FC_BS(byte bs)`	According to ISO, BS=0 means that there is only one FC (the one after the FF). Some manufacturers decided, to give BS=255 this meaning.
`Byte OSEKTL_Get1FC_BS()`	Returns the BS with the special meaning explained above.
`OSEKTL_UseExtId(long val)`	val=1: Nodes use Extended CAN IDs val=0: Nodes use Standard CAN IDs
`Long OSEKTL_IsUseExtId()`	1: Nodes use Extended CAN IDs 0: Nodes use Standard CAN IDs

Name	Comment
`Long OSEKTL_GetLoRxId()`	Lowest CAN ID which passes Rx Filter
`Long OSEKTL_GetHiRxId()`	Highest CAN ID which passes Rx Filter
`Long OSEKTL_GetStartSN()`	Sequence number used for first CF
`Long OSEKTL_GetFixedDLC()`	0: Minimum DLC for CAN frames used x=1-8: At least DLC x used
`Long OSEKTL_GetCAN()`	Reads the channel on which a node receives and sends frames
`OSEKTL_SetCAN(long can)`	Sets the channel on which a node receives and sends frames. Only works properly in 'PreStart'
`Long OSEKTL_IsUseFC()`	Flow Control is 1: used 0: not used
`OSEKTL_SetUseFC(long val)`	Set Flow Control is 1: used 0: not used
`void OSEKTL_SetFCDelay( dword milliseconds)`	Set the delay of all FC frames sent to this value. (default: 0)
`Long OSEKTL_GetMaxMsgLen()`	Return the maximum TP message length the node will accept without sending an overflow frame. (default: 4095)
`OSEKTL_GetMaxMsgLen (long len)`	Set the maximum acceptable length (<= 4095 byte) of a TP message. The reception of longer messages will be aborted by sending an overflow (FC.OVFLW) frame.
`Long OSEKTL_GetWaiting()`	Returns the wait state of the node. If set, the node will send FC.WT frames instead of FC.CTS frames to a sender (cf. 2.1.6)
`OSEKTL_SetWaiting(long state)`	Set the wait state of the node. The node will send wait frames to a sender if set to 1.
`void OSEKTL_SetWFTMax( dword WFTmax)`	Set the maximum number of FC.WT frames a receiver will send before aborting the reception. (default: 10)

3.1.3.1 Times

The times are taken from the initialization file at the start of measurement. They may be changed at any time from CAPL.

Name	Comment
`long OSEKTL_GetTimeoutXX()`	Reads the timeout value XX [ms] (XX may be: Ar, Br, Cr, As or Bs) (described in ISO/TF2)
`OSEKTL_SetTimeoutXX(long val)`	Sets the timeout value XX (XX may be: Ar, Br, Cr, As or Bs) (described in ISO/TF2) to val ms

Name	Comment (further description in ISO/DIS 15765-2)
Ar	maximum time from send request of a frame to reception of transmit acknowledgement on receiver side
Br	time until transmission of next FlowControl N_PDU
Cr	maximum time from of transmit acknowledgement for a sent frame until reception of CF on receiver side
As	maximum time from send request of a frame to reception of transmit acknowledgement on sender side
Bs	maximum time from of transmit acknowledgement for a sent frame until reception of FC on sender side

The APIs mentioned below are obsolete and are only provided for compatibility reasons. As described above version 3.0 of the TP DLL implements a more precise handling of timeouts. `SetTimeoutFC` sets As and Bs and `SetTimeoutCF` sets Ar and Cr to the specified value.

Name	Comment
`Long OSEKTL_GetTimeoutFC()`	Returns the time up to which a Flow Control message (FC) must have arrived
`OSEKTL_SetTimeoutFC(long t)`	Sets the timeout for flow control messages
`Long OSEKTL_GetTimeoutCF()`	Returns the time up to which a Consecutive Frame (CF) must have arrived
`OSEKTL_SetTimeoutCF(long t)`	Sets the timeout for Consecutive Frames

3.1.3.2 Protocol Parameters for Extended Addressing

Name	Comment
`Long OSEKTL_GetTgtAdr()`	Returns the active Target Address
`OSEKTL_SetTgtAdr(long ta)`	Sets the Target Address
`Long OSEKTL_GetTpBaseAdr()`	Returns the active base address
`OSEKTL_SetTpBaseAdr (long b)`	Sets the base address
`Long OSEKTL_GetEcuNumber()`	Returns the active address (Number) of the controller
`OSEKTL_SetEcuNumber (long nr)`	Sets the controller address
`Long OSEKTL_GetLoFctAdr()`	Returns the lowest functional address (Diagnosis)

Name	Comment
`OSEKTL_SetLoFctAdr (long adr)`	Sets the lowest functional address (Diagnosis)
`Long OSEKTL_GetHiFctAdr()`	Returns the highest functional address (Diagnosis)
`OSEKTL_SetHiFctAdr(long h)`	Sets the highest functional address (Diagnosis)
`OSEKTL_UseIdFromAdr(long val)`	Usually, the CAN-ID is derived from the sender address (and the base address). After the call of this function with `val=0` the CAN-ID is not automatically calculated and can be set using the function `OSEKTL_SetTxId(canId)`, (like in normal addressing mode). 1: Switches back to default behavior, that is the CAN-ID is derived from the base address and the sender address.
`long OSEKTL_IsIdFromAdr()`	1: Id calculated from ECU + BaseAdr 0: Id from `OSEKTL_SetTxId`
`OSEKTL_UseAdrFromId(long val)`	Usually the source address (address where the FC frames for this transmission shall be sent) is calculated from the received CAN identifier. 1: target address calculated from CAN identifier 0: target address from `OSEKTL_SetTgtAdr(ta)`
`long OSEKTL_IsAdrFromId()`	1: target address calculated from CAN identifier 0: target address from `OSEKTL_SetTgtAdr(ta)`
`OSEKTL_SetMsgCount( DWORD n)`	Only messages with ids in the interval [base-address; base-address + n –1] will be considered.
`DWORD OSEKTL_SetMsgCount()`	Get the number of messages considered.

With extended addressing the transmit identifier is defined by the base address and the controller address (ECU number). The user assumes responsibility for ensuring that the consistency condition is satisfied:

```
Tx identifier= Base address + Station address
```

3.1.3.3 Protocol Parameters for Normal Addressing

Name	Comment
OSEKTL_GetTxId()	Returns the active transmit ID
OSEKTL_SetTxId()	Sets the transmit ID
OSEKTL_GetRxId()	Returns the active receive ID
OSEKTL_SetRxId()	Sets the receive ID
`Long OSEKTL_GetLoFctAdr()`	Returns the lowest functional address (Diagnosis)

Name	Comment
`OSEKTL_SetLoFctAdr(long adr)`	Sets the lowest functional address (Diagnosis)
`Long OSEKTL_GetHiFctAdr()`	Returns the highest functional address (Diagnosis)
`OSEKTL_SetHiFctAdr(long h)`	Sets the highest functional address (Diagnosis)

3.1.3.4 Protocol Parameters for Normal Fixed Addressing

Name	Comment
`OSEKTL_SetTxPhys()`	Sets the addressing mode of the following transmissions to physical addressing
`OSEKTL_SetTxFunc()`	Sets the addressing mode of the following transmissions to functional addressing
`long OSEKTL_IsTxModePhys()`	1 if Tx mode is physical or else 0
`OSEKTL_SetTxPrio(long val)`	sets priority of transmission in normal fixed and mixed addressing mode
`long OSEKTL_GetTxPrio()`	reads the priority of transmission in normal fixed and mixed addressing mode
`long OSEKTL_IsUseFFPrio()`	Priority for FC is used 0: as configured 1: as received in FF
`OSEKTL_SetUseFFPrio(long prio)`	Priority for FC will be used 0: as configured 1: as received in FF
`Long OSEKTL_GetFCPrio()`	Returns the priority used in FC frames
`OSEKTL_SetFCPrio(long prio)`	Sets the priority used in FC frames

`Long OSEKTL_GetEcuNumber()`
`OSEKTL_SetEcuNumber(long)`
`Long OSEKTL_GetTgtAdr()`
`OSEKTL_SetTgtAdr(long ta)`
	These functions are used to set the source and target address in the CAN id. Please refer to the descriptions under extended addressing mode above.

Depending on the Tx mode, the mask with which the extended identifier is generated is set to:

- physical: 0x0CDA0000
- functional: 0x0CDB0000

Note that the priority (bits 28-26) is set to 3 as default, but this can be changed with the function `OSEKTL_SetTxPrio`. Activate the usage of the FirstFrame-priority if the receiver should use the same priority as the sender, i.e. call `OSEKTL_SetUseFFPrio(1)`.

3.1.3.5 Protocol Parameters for Mixed Addressing

The functions defined for Normal Fixed addressing (3.1.3.4) may be used here, plus:

Name	Comment
`void OSEKTL_SetAdrExt(BYTE ae)`	Sets the address extension to ae (has only effects in mixed addressing mode)
`BYTE OSEKTL_GetAdrExt(void)`	Gets the address extension
`OSEKTL_SetUseAllAE(long state)`	If set to a value other than 0, the node will accept transmissions using any address extension value. Otherwise the value has to match (default, cf. 2.1.4).
`Long OSEKTL_GetUseAllAE()`	Returns 0 if the address extension has to match exactly, 1 if the node accepts any AE value.
`Long OSEKTL_GetRecentAE(void)`	Returns the address extension of the most recent reception (available after reception of the FirstFrame). -1 indicates that no such value is available.

Depending on the Tx mode, the mask with which the extended identifier is generated is set to (further explanations see above and the note for Normal Fixed mode):

- physical: 0x0CCE0000
- functional: 0x0CCD0000

3.1.3.6 Protocol Parameters for 11 bit Mixed Addressing

(Cf. 2.1.5) The functions for Normal addressing (3.1.3.3) are available here, plus the additional functions described for the Mixed mode (3.1.3.5), but NOT the functions listed for the Normal Fixed mode!

3.2 Callback Functions

The first three of following functions must be provided in CAPL by the application programmer. The last callback function is optional.

Name	Comment	
`OSEKTL_ErrorInd(int error)`	Error message from transport layer: error =	
	1:	Timeout waiting for CF
	2:	Timeout waiting for FC
	3:	Received Wrong SN
	4:	TxBusy, only one transmission possible at the same time
	5:	Received unexpected PDU
	6:	Timeout while trying to send a CAN frame
	7:	Too many FC.WFT sent
	8:	Receiver buffer overflow
	9:	Wrong parameter
	<0:	Unknown Error

Name	Comment
`OSEKTL_DataInd(long rxLength)`	Returns the number of data received
`OSEKTL_DataCon(long txLength)`	Returns the number of data successfully sent
`Void OSEKTL_FirstFrameIndication(long source, long target, long length)`	Indicates, that a FF has been received. The meaning of the parameters depends on the addressing mode: source: normal: Tx Identifier of source node extended: ECU(of source node) + Base normal fixed: SA mixed: SA target: normal: Tx Identifier of destination node extended: ECU(of destination node) + Base normal fixed: TA mixed: TA `length` is the length of the message for both addressing modes.

3.3 Acknowledged data transfer (extension)

Normally, ISO TP is an unacknowledged transport protocol (USDT), i.e. the sender does not get a confirmation that the data it sent to the receiver has been received correctly. The DLL now supports an optional acknowledged mode (ASDT, acknowledged segmented data transfer).

WARNING: The user has to make sure that ASDT is available in the network and has to activate this feature explicitly.

3.3.1 Description

This extension to the protocol will do the following, if activated:

- A receiver node will send a FC.Ack frame (new type) to the sender after it has correctly received the last data frame (SF or CF) expected. If an error occurred, the receiver will NOT send this frame, i.e. the sender will run into a FC timeout.
- The sender node waits for the FC.Ack after it has sent the last data frame. If no such frame is received within the N_Bs timeout, the transmission error is reported with a call to the CAPL function `OSEKTL_ErrorInd(2)`.
- The format of the FC.Ack frame is as follows: The “FlowStatus” value of this frame is set to 0xF, and the second byte of the PCI is defined to be “AckRes”. At the moment, only the value AckRes = 0 is defined, but this can be overwritten (see below).

N_PCI	Byte #1	Byte #2	Byte #3
Value	0x3F	AckRes (default: 0)	0 (reserved)

3.3.2 Activation

The ASDT mode can be activated by

- From the INI file: In section `[COMMON]`, set parameter `SetAckMode` to a value of 1.
- From CAPL: The function `OSEKTL_ActivateAck( 1)` has to be called.

3.3.3 Additional functions

Function name	Parameters	Description
`OSEKTL_ActivateAck`	`Long flag`	0: Use USDT mode (default) 1: Activate ASDT mode The call will only affect future transmissions and receptions!
`OSEKTL_SetAckRes`	`DWORD result`	Value of AckRes to send to sender (8 bit)
`Long OSEKTL_GetAckRes`	`(none)`	Value of last AckRes received, only valid during FinishTransmission callback

3.3.4 Examples

Transfer of a single frame:

```
OSEKTL_ActivateAck(1);
OSEKTL_DataReq( buffer, 5);
```

Result (normal addressing, protocol data in brackets):

```
441   [05] 53 68 6f 72 74 00 00  // SingleFrame, 5 byte
442   [3f 00 00] 00 00 00 00 00  // FlowControl.Ack, AckRes = 0
```

Transfer of a larger block:

```
OSEKTL_ActivateAck(1);
OSEKTL_DataReq( buffer, 26);
```

Result (normal addressing, protocol data in brackets):

```
442   [10 1a] 54 68 69 73 20 69  // FirstFrame, 26 byte
441   [30 02 14] 00 00 00 00 00  // FlowControl.ClearToSend
442   [21] 73 20 61 20 6c 6f 6e  // ConsecutiveFrame
442   [22] 67 65 72 20 54 65 53  // ConsecutiveFrame
441   [30 02 14] 00 00 00 00 00  // FlowControl.ClearToSend
442   [23] 74 54 65 58 74 2e 53  // ConsecutiveFrame (last)
441   [3f 00 00] 00 00 00 00 00  // FlowControl.Ack, AckRes = 0
```

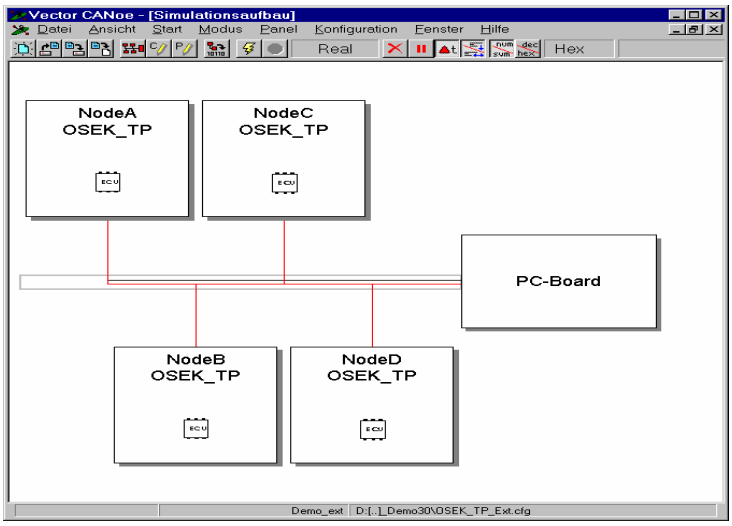
4 Demo

The demo consists of the following components:

- CANoe configuration: OSEK_TP_Ext.cfg and OSEK_TP_Nor.cfg
- Database: candb\demo_ext.dbc and candb\demo_nor.dbc
- CAPL models: `nodes\*.can`
- Panels: panels*.cnp

4.1 Model Description

The demo configurations consist of four network nodes each, which can transmit and receive long messages.



The demo folder contains two demo configurations which both use the same CAPL programs.

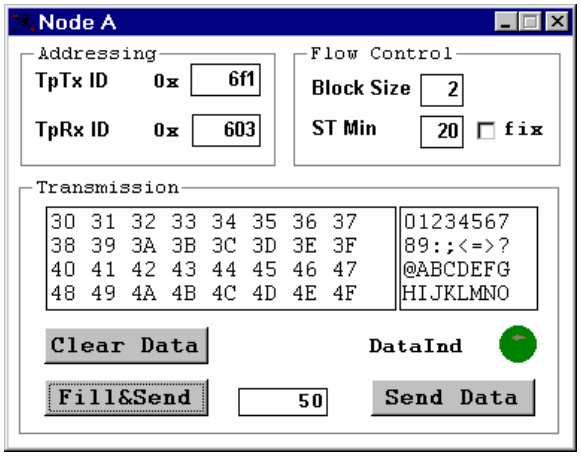
With `OSEK_TP_Nor.cfg` the database `demo_nor.dbc` is loaded. This database defines the default value of the node attribute 'TpAddressingMode' to 0. Therefore all nodes communicate in normal addressing mode. The panels `NodeAn.cpn` are shown, which allow to configure the Rx Ids and the Tx Ids of the nodes.

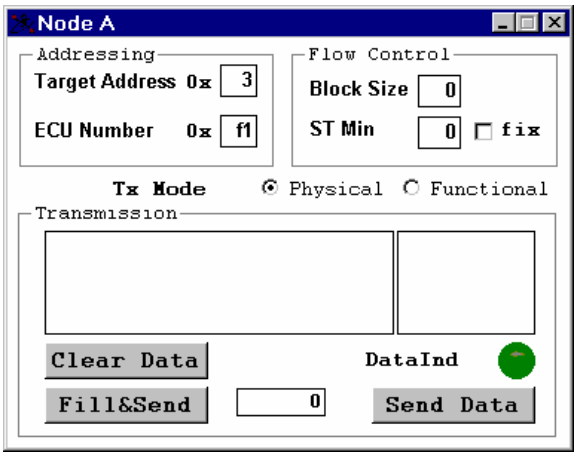
With `OSEK_TP_Ext.cfg` the database `demo_ext.dbc` is loaded. This database defines the default value of the node attribute 'TpAddressingMode' to 1. Therefore all nodes which have no special value defined as their attribute 'TpAddressingMode', communicate in extended addressing mode. This applies to Node A and Node B. Node C and Node D have a value set for their 'TpAddressingMode'. Thus they communicate either in normal fixed or mixed addressing mode. The panels `NodeAx.cpn` are shown, which allow to configure the source and target addresses of the nodes.

Except from the 'TpAddressingMode' attribute the databases are identical. The configurations show two trace windows. One in which only observer messages are shown, if the OSEK TP observer is activated. And another window where all CAN and observer messages are shown.

4.1.1 Description of the Panels

There is a main panel which is shown for each node, which allows to enter data to be transmitted and to configure the essential parameters of the transport layer.





GUI Element	Description
Hex Editor	Allows to enter data to be sent and shows data recently received
Fill & Send Button	Fills the hex editor with the amount of bytes specified in the edit field placed to the right of the button
Send Button	Sends the data entered into the hex editor to the specified address
Clear Data	Deletes all data in the hex editor
Fix Checkbox	If activated, the concerned node doesn't evaluate the STMin from the received flow control frame, but instead uses the STMin entered in the panel.
Tx Mode (physical/functional)	effects the CAN identifier used for transmission in normal fixed and mixed addressing mode only. The id mask is then set according to ISO/TF2.

5 Installation

5.1 Product Components Supplied

The following components are supplied with the product:

- This document as PDF file.
- `osek_tp.dll`, CANoe expansion DLL for 32-bit (exec32)
- osek_tp.ini, parameter file for osek_tp.dll
- Demo directory

5.2 Installation

DLL and initialization file are copied to the directory which also contains CANoe; this is usually `canwin\exec32`.

The initialization files are configured for the Demo. They must be adapted to your application as necessary.

6 Frequently asked questions

This section collects some of the most commonly encountered problems and suggests solutions.

Problem:	On measurement start, “Could not find mandatory callback function X” is printed in the Write window.
Cause:	The DLL searches for the callback functions described in 3.2 and none were declared in the CAPL program of the node.
Recommendation:	Declare the appropriate function in the CAPL program of the node.

Problem:	It is not possible to send on a specific bus.
Cause:	If CANoe is in “standard mode”, the TP DLL will only work on the bus the node is attached to. Any setting or changing of the CAN channel is ignored. In “compatible” mode, the CAN channel settings *are* used, e.g. the “TpCan-Bus” database attribute.
Recommendation:	Assign the correct CAN channel to the network in standard mode or set the database attribute to the correct value in compatible mode.

Problem:	A gateway transferring data between two busses does not work correctly.
Cause:	CANoe not in “standard” mode. Gateway functionality is **not** available in compatible mode.
Recommendation:	Switch CANoe to standard mode and adapt databases and CAPL programs.

7 Appendix: Fault injection support

7.1 Introduction

During the development of a communication system, the correct behaviour of the components is the primary goal. But in many situations reliability is also a key issue and therefore the components (especially ECUs) should react in a predefined and controlled way on errors introduced by other components. For example, a protocol error generated by a communication partner should not prevent the ECU from communicating with other partners in later transmissions.

The goal of this extension to the transport layer DLL is to introduce faults in the transport protocol, thereby helping the system designer to evaluate the reactions of the components, identify possibly dangerous situations and find solutions for them.

7.2 WARNING

Using the Fault Injection features will introduce errors into the communication! Therefore only users with in depth knowledge of the transport protocol should activate it for development and test purposes and never in production situations.

7.3 Usage

Before the Fault Injection (FI) features can be activated, an "Enable" function has to be called in every node that should use it. All function names start with the prefix "`OSEKTL_FI`" and do not return a value.

The activation of a fault will only be active for the next transmission or reception. For transmissions, the possibility of a fault injection is determined during the `OSEKTL_DataReq()` function call. Any later changes will not affect the fault behaviour. For receptions, the possibilities of a fault injection is determined after the call to the CAPL callback `OSEKTL_FirstFrameInd()` has returned. Therefore it is possible to setup fault injection in this function.

In order to deactivate a fault feature before it is assigned to a transmission, the function may be called again with a negative value as the first parameter.

7.3.1 Functions

Function name	Parameters	Description
`OSEKTL_FI_Enable`	`(none)`	Allow fault injection
`OSEKTL_FI_Disable`	`(none)`	Prohibit Fault Injection
`OSEKTL_FI_SendXByte`	`long NumberOfBytesToSendTillStop, unsigned long MinLenLastFrame, long FillValue`	Abort transmission after the specified number of bytes has been sent. See below for more detailed information.
`OSEKTL_FI_DropCF`	`long NumberOfCFToBeLost`	Do not send one ConsecutiveFrame
`OSEKTL_FI_DoubleTx`	`long NumberOfFrameToBeDoubled`	Double a frame, where 0 is the FirstFrame, 1 the first ConsecutiveFrame etc.
`OSEKTL_FI_DelayCF`	`long NumberOfDelayedCF, unsigned long DelayInMsForThisCF`	Delay a Continuous Frame, where 1 is the first one, etc.
`OSEKTL_FI_DropFC`	`long NumberOfDroppedFC`	Do not send one FlowControl frame, where 0 is the first one, etc.

OSEKTL_FI_DelayFC	long NumberOfDelayedFlowControl, unsigned long DelayTime	Delay a FlowControl frame for Delay-Time ms, where 0 is the first FC etc.
OSEKTL_FI_DoubleFC	long NumberOfFCToBeDoubled	Send a FC two times, where 0 is the first one etc.
OSEKTL_FI_AllCTS	long flag	Treat all FC frames as CTS if parameter is not 0. This function may be called any time and will take effect for all following transmissions. (default: 0)

Detailed description of the `OSEKTL_FI_SendXByte` function:

If FillValue is >=0, the last frame sent will be filled with it until MinLenLastFrame is reached. If it is <0, the last frame will be filled with data bytes, i.e. effectively transmitting more data than specified. Note that the final frame of a transmission cannot be altered!

It is possible to specify a value larger than the value used in the “DataReq” function call; in this case the sender will try to send the given amount of data. The additional data will be set to the lower 8 bit of the FillValue.

Example:

```
OSEKTL_FI_SendXByte( 12, 8, -1);
OSEKTL_DataReq( buffer, 20);
```

Result (extended addressing, protocol data in brackets):

```
[00 10 14] 00 01 02 03 04        // First frame, 20 byte, target ECU 0, 5 data byte
[00 21] 05 06 07 08 09 0a        // Cont.Frame 1, 6 data byte
[00 22] 0b 0c 0d 0e 0f 10        // Cont.Frame 2, 6 data byte
```

In this case, 17 data byte are actually transferred to the receiver.

Example:

```
OSEKTL_FI_SendXByte( 12, 0, -1); // fillValue is not used if MinLenLastFrame == 0.
OSEKTL_DataReq( buffer, 20);
```

Result:

```
[00 10 14] 00 01 02 03 04        // First frame, 20 byte, target ECU 0, 5 data byte
[00 21] 05 06 07 08 09 0a        // Cont.Frame 1, 6 data byte
[00 22] 0b                       // Cont.Frame 2, one data byte
```

Here, only the specified 12 byte are transferred and the last frame is shortened.

Example:

```
OSEKTL_FI_SendXByte( 12, 6, 0x99);
OSEKTL_DataReq( buffer, 20);
```

Result:

```
[00 10 14] 00 01 02 03 04        // First frame, 20 byte, target ECU 0, 5 data byte
[00 21] 05 06 07 08 09 0a        // Cont.Frame 1, 6 data byte
[00 22] 0b 99 99 99              // Cont.Frame 2, one data byte, filled for DLC 6
```

The last frame is made 6 byte long, filling it with the FillValue.

Example:

```
OSEKTL_FI_SendXByte( 1, 8, -1);
OSEKTL_DataReq( buffer, 20);
```

Result:

```
[00 10 14] 00 01 02 03 04        // First frame, 20 byte, target ECU 0, 5 data byte
```

Here, only the FirstFrame is sent, then the sender aborts the transmission.

Example:

```
OSEKTL_FI_SendXByte( 30, 8, -1);
```

```
OSEKTL_DataReq( buffer, 20);
```

Result:

```
[00 10 14] 00 01 02 03 04        // First frame, 20 byte, target ECU 0, 5 data byte
[00 21] 05 06 07 08 09 0a        // Cont.Frame 1, 6 data byte
[00 22] 0b 0c 0d 0e 0f 10        // Cont.Frame 2, 6 data byte
[00 23] 11 12 13 ff ff ff        // Cont.Frame 3, 3 data byte, 3 fill byte
[00 24] ff ff ff ff ff ff        // Cont.Frame 4, 6 fill byte
```

This example for sending more data than specified in the FirstFrame demonstrates how an additional frame is sent containing fill bytes. Here, the sender did not finish the sending, since the receiver did not reply with an FlowControl message, i.e. only 29 byte were sent.

7.3.2 Callbacks

When defined for a node, this CAPL callback function is called directly after the frame was created. In the function, the message content and length may be changed, and also the actual delay can be changed.

CAPL callback name	Parameters	Description
`void OSEKTL_FI_PreSend`	`word dlc[], // length 1` `byte message[] // length 8 byte`	Length of message and message content

The following functions may only be called from the `OSEKTL_FI_PreSend` callback! They will access the current message only, i.e. the message that will be sent next by the node.

CAPL callback name	Parameters	Description
`Void OSEKTL_FI_SetDelay`	`dword MICROseconds`	Set the delay of the message that will be sent next. NOTE: the delay is given in *micro* seconds (µs).
`Dword OSEKTL_FI_GetDelay`	(none)	Returns the delay (in µs) the message will experience if the value is not changed with `OSEKTL_FI_SetDelay`.
`Long OSEKTL_FI_IsFC`	(none)	Returns 1 if the message is a FC frame.
`Dword OSEKTL_FI_GetPCIOff`	(none)	Returns the index of the PCI byte in the message data.
`Void OSEKTL_FI_SetId`	`dword id`	Set the message id to this value.
`Dword OSEKTL_FI_GetId`	(none)	The message will be sent with this id.

7.4 Feature interaction

In case more than one Fault Injection feature is activated for one transmission or reception, the following rules apply:

- The transmission faults and the reception faults do not interact since they cannot happen in the same transport context.
- The dropping of a message dominates a message delay or doubling, i.e. when a frame should be dropped and delayed or dropped and doubled, it is dropped.
- When the same ConsecutiveFrame should be doubled and delayed, it will be sent two times after the delay has expired.
- When the same FlowControl frame should be doubled and delayed, it will be sent two times where the distance between the frames will be the specified delay. But if a ConsecutiveFrame arrives in the meantime, the second frame may be sent directly after it.

- Stopping the transmission after the specified number of bytes and delaying the last frame may happen together.
- The `PreSend` CAPL-callback function is called *before* a message is delayed. It is possible to retrieve the delay the message will experience, and set the delay to use. Note that this is not possible for doubled frames: the first FC frame will be sent immediately, the second one will be delayed. For CF frames, they will be sent back to back after the delay has passed.
- The delay a message will experience will be computed as follows:
  1. The delay resulting from the protocol settings is computed, e.g. for CF, STmin is used, or for FC frames the value specified via `OSEKTL_SetFCDelay`.
  2. If fault injection is activated for a frame, the delay given there will overwrite the value computed.
  3. If the `PreSend`-callback can be found, it is called and the resulting delay is used.

7.5 Examples

This sections collects some examples for fault injection using the OSEK TP DLL. As a reference, this is the trace of an unmodified data transmission:

```
Time diff     ID  Type    Data
1.500000      601 FF      10 20 30 31 32 33 34 35       // First Frame, normal mode
0.001930      602 FC.CTS  30 02 18 00 00 00 00 00       // Flow Control, BS 2, ST 24ms
0.024280      601 CF      21 36 37 38 39 3a 3b 3c       // Continuous Frame #1
0.024260      601 CF      22 3d 3e 3f 40 41 42 43       // Continuous Frame #2
0.001810      602 FC.CTS  30 02 18 00 00 00 00 00       // Flow Control, BS 2, ST 24ms
0.024278      601 CF      23 44 45 46 47 48 49 4a       // Continuous Frame #3
0.024293      601 CF      24 4b 4c 4d 4e 4f 00 00       // Continuous Frame #4, last
```

In the examples, the resulting modified traces will be given.

7.5.1 Basic fault injection functions

The following code configures the *receiver* node:

```
OSEKTL_SetFCDelay( 20);          // set the FC delay to 20 ms
OSEKTL_FI_Enable();              // allow fault injection
OSEKTL_FI_DelayFC( 1, 10);       // concrete value overwrites general one

Time diff     ID  Type    Data
1.500000      601 FF      10 20 30 31 32 33 34 35
0.021930      602 FC.CTS  30 02 18 00 00 00 00 00       // FC 0 uses FC delay
0.024284      601 CF      21 36 37 38 39 3a 3b 3c
0.024269      601 CF      22 3d 3e 3f 40 41 42 43
0.011810      602 FC.CTS  30 02 18 00 00 00 00 00       // FC 1 delay via FI-function
0.024278      601 CF      23 44 45 46 47 48 49 4a
0.024293      601 CF      24 4b 4c 4d 4e 4f 00 00
```

7.5.2 Change the delay in the Presend-callback

After activating the fault injection features, the following presend function can be used in the sender node. In this example, the CF #2 will be delayed for 100 ms and its id is made extended. Note that the data transfer is aborted with error indications.

```
OSEKTL_FI_PreSend( word msgDlc[], byte data[])     // Sender (callback)
{
   Dword PCIoffset;
   If( !OSEKTL_FI_IsFC())  // patch only CF
   {
      PCIoffset = OSEKTL_FI_GetPCIOff();     // protocol byte position
      If( (data[ PCIoffset] & 0xF) == 2)     // sequence number 2 only
      {
```

```
            OSEKTL_FI_SetDelay( 100000);        // 100 ms = 100000 µs
            OSEKTL_FI_SetId( OSEKTL_FI_GetId() | 0x80000000);
                                                // make extended id
        }
    }
}
```

Resulting trace:

```
    Time diff     ID  Type    Data
    1.500000      601 FF      10 20 30 31 32 33 34 35       // First Frame, normal mode
    0.001930      602 FC.CTS  30 02 18 00 00 00 00 00       // Flow Control, BS 2, ST 24ms
    0.024280      601 CF      21 36 37 38 39 3a 3b 3c       // Continuous Frame #1
    0.100260      601x CF     22 3d 3e 3f 40 41 42 43       // Continuous Frame #2
Receiver (callback): OSEKTL_ErrorInd( 1);    // Timeout waiting for CF
Sender   (callback): OSEKTL_ErrorInd( 2);    // Timeout waiting for FC
```

- this page is intentionally left blank -