MICROSAR TCPIP

Technical Reference

Version 6.2.0

Authors	Thorsten Albers, Frederik Dornemann, Knut Winkelbach, Jens Bauer, Manisha Jadav
Status	Released

Document Information

History

Author	Date	Version	Remarks
Thorsten Albers Frederik Dornemann	2015-05-18	1.0.0	Creation of document. This document combines the Technical References of the former modules TcpIp, IpV4 and IpV6.
Frederik Dornemann	2015-10-19	1.0.1	Added Known Issues (IETF RFC Conformance)
Frederik Dornemann Thorsten Albers	2016-01-13	2.0.0	Added IPv4 Fragmentation Added API to read the local physical address Added sending the FQDN option via DHCP to identify the ECU Added triggering of IP address assignment
Thorsten Albers	2016-03-15	2.1.0	TLS as TcpIp plug-in
Frederik Dornemann	2016-06-29	3.0.0	Added description of unicast address assignment methods Added description of configuration of static on-link prefixes for IPv6
Frederik Dornemann	2016-09-22	3.1.0	Added description of optional callout for link-local address assignment.
Frederik Dornemann	2016-10-20	4.0.0	Updated description of MainFunction, <Up>_CopyTxData() Callback and Tls Heartbeat.
Frederik Dornemann	2017-01-18	4.1.0	Release of DHCPv4 Server.
Knut Winkelbach	2017-03-17	5.0.0	Added Production Error Reporting and Other Error Reporting and Diagnostic Options
Jens Bauer Knut Winkelbach	2017-04-28 2017-05-03	6.0.0	Added description of TcpIp_ClearARCache() Extended Production Error Reporting, Added 5.5.1.7 TcpIpDuplicateAddressDetectionConfig and 3.6.3.4 IPv6 Duplicate Address Detection Conflict
Jens Bauer	2017-05-23	6.1.0	Added description of TcpIp_GetNdpCacheEntries() according ASR 4.3
Manisha Jadav	2017-05-30	6.1.0	Added description of TcpIp_GetArpCacheEntries() according ASR 4.3
Thorsten Albers	2017-08-10	6.2.0	Add new socket owner callback for TLS validation result

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	AUTOSAR_SWS_TcpIp	V4.2.1
[2]	AUTOSAR	AUTOSAR_SWS_DefaultErrorTracer	V4.2.1
[3]	AUTOSAR	AUTOSAR_SWS_DiagnosticEventManager	V4.2.1
[4]	AUTOSAR	AUTOSAR_TR_BSWModuleList	V4.2.1

[5]	AUTOSAR	AUTOSAR_SWS_EthernetInterface	V4.2.1
[6]	IETF RFC768	User Datagram Protocol	Aug 1980
[7]	IETF RFC791	Internet Protocol	Sep 1981
[8]	IETF RFC792	Internet Control Message Protocol (ICMP)	Sep 1981
[9]	IETF RFC793	Transmission Control Protocol	Sep 1981
[10]	IETF RFC813	Window and Acknowledgement Strategy in TCP	Jul 1982
[11]	IETF RFC826	An Ethernet Address Resolution Protocol (ARP)	Nov 1982
[12]	IETF RFC894	A Standard for the Transmission of IP Datagrams over Ethernet Networks	Apr 1984
[13]	IETF RFC1112	Host Extensions for IP Multicasting	Aug 1989
[14]	IETF RFC1122	Requirements for Internet Hosts -- Communication Layers	Oct 1989
[15]	IETF RFC1323	TCP Extensions for High Performance	May 1992
[16]	IETF RFC1981	Path MTU Discovery for IP version 6	Aug 1996
[17]	IETF RFC2018	TCP Selective Acknowledgment Options	Oct 1996
[18]	IETF RFC2131	Dynamic Host Configuration Protocol (DHCP)	Mar 1997
[19]	IETF RFC2460	Internet Protocol, Version 6 (IPv6) Specification	Dec 1998
[20]	IETF RFC2464	Transmission of IPv6 Packets over Ethernet Networks	Dec 1998
[21]	IETF RFC2711	IPv6 Router Alert Option	Oct 1999
[22]	IETF RFC3122	Extensions to IPv6 Neighbor Discovery for Inverse Discovery Specification	Jun 2001
[23]	IETF RFC3315	Dynamic Host Configuration Protocol for IPv6 (DHCPv6)	Jul 2003
[24]	IETF RFC3810	Multicast Listener Discovery Version 2 (MLDv2) for IPv6	Jun 2004
[25]	IETF RFC3927	Dynamic Configuration of IPv4 Link-Local Addresses	May 2005
[26]	IETF RFC4291	IP Version 6 Addressing Architecture	Feb 2006
[27]	IETF RFC4429	Optimistic Duplicate Address Detection (DAD) for IPv6	Apr 2006
[28]	IETF RFC4443	Internet Control Message Protocol (ICMPv6) for the IPv6 Specification	Mar 2006
[29]	IETF RFC4702	The DHCP Client FQDN option	Oct 2006
[30]	IETF RFC4704	The DHCPv6 Client FQDN option	Oct 2006
[31]	IETF RFC4861	Neighbor Discovery for IP version 6 (IPv6)	Sep 2007
[32]	IETF RFC4862	IPv6 Stateless Address Autoconfiguration	Sep 2007
[33]	IETF RFC4884	Extended ICMP to Support Multi-Part Messages	Apr 2007
[34]	IETF RFC4941	Privacy Extensions for Stateless Address Autoconfiguration in IPv6	Sep 2007
[35]	IETF RFC5095	Deprecation of Type 0 Routing Headers in IPv6	Dec 2007
[36]	IETF RFC5482	TCP User Timeout Option	Mar 2009
[37]	IETF RFC5681	TCP Congestion Control	Sep 2009
[38]	IETF RFC5722	Handling of Overlapping IPv6 Fragments	Dec 2009
[39]	IETF RFC5942	IPv6 Subnet Model: The Relationship between Links and Subnet Prefixes	Jul 2010
[40]	IETF RFC6085	Address Mapping of IPv6 Multicast Packets on Ethernet	Jan 2011

[41]	IETF RFC6106	IPv6 Router Advertisement Options for DNS Configuration	Nov 2010
[42]	IETF RFC6298	Computing TCP's Retransmission Timer	Jun 2011
[43]	IETF RFC6582	The NewReno Modification to TCP's Fast Recovery Algorithm	Apr 2012
[44]	IETF RFC6724	Default Address Selection for Internet Protocol version 6 (IPv6)	Sep 2012
[45]	Vector	TechnicalReference_TIs.pdf	V4.00.00

Scope of the Document

This technical reference describes the general use of the TCPIP basis software. Please refer to your Release Notes to get a detailed description of the platform (host, compiler, controller) your Vector Ethernet Bundle has been configured for.

Caution

We have configured the programs in accordance with your specifications in the questionnaire. Whereas the programs do support other configurations than the one specified in your questionnaire, Vector´s release of the programs delivered to your company is expressly restricted to the configuration you have specified in the questionnaire.

Contents

Illustrations

Tables

1 Component History

The component history gives an overview over the important milestones that are supported in the different versions of the component.

Component Version	New Features
1.0.x	Configuration of the TCP/IP stack according to AR 4.2.1 - Combination of the former modules TcpIp, IpV4 and IpV6 to the new module TcpIp. - Some additional AUTOSAR R4.2.1 APIs
2.0.x	Support of IPv4 Fragmentation and Reassembly. Support of ASR APIs: - TcpIp_GetRemotePhysAddr() - TcpIp_Dhcp(V6)Read/WriteOption() (Support of FQDN option) - TcpIp_RequestIpAddrAssignment
2.1.x	Support of TLS for TCP connections (TcpIp plug-in)
3.0.x	Support any combination of IPv4 address assignment methods according to AR 4.2.1. Support static configuration of on-link prefixes for IPv6.
3.1.x	Support of IPv4 <User>_LinkLocalAddrCandidateCallout().
4.0.0	Split-up of MainFunction into Rx, Tx and State part. Support of TCP Keep-Alives (replaces Idle Timeout). Support alternative CopyTxData callback with variable length. Use incrementing numbers for TCP/UDP ports in dynamic range. Support TLS Heartbeat for sockets using TLS.
4.1.0	Release of DHCPv4 Server.
5.0.x	Support sending gratuitous ARP messages (manually triggered) Added diagnostic interfaces for detection of DHCP, ARP and IP conflicts.
6.0.x	Added TcpIp_ClearARCache() API
6.1.x	Added TcpIp_GetNdpCacheEntries() and TcpIp_GetArpCacheEntries() APIs
6.2.x	Added support of ICMP Destination (Port/Protocol) Unreachable Messages and AR <Up>_IcmpMsgHandler(). Added Tls Validation Callback for socket owner
6.3.x	Added support of ASR4.3 API TcpIp_IcmpV6Transmit() and <Up>_IcmpMsgHandler() for IPv6.

Table 1-1 Component history

2 Introduction

This document describes the functionality, API and configuration of the AUTOSAR BSW module TCPIP as specified in [1].

Supported AUTOSAR Release*:	4	
Supported Configuration Variants:	pre-compile	
Vendor ID:	TCPIP_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	TCPIP_MODULE_ID	170 decimal (according to ref. [4])

* For the detailed functional specification please also refer to the corresponding AUTOSAR SWS.

The TcpIp provides access to a socket interface based on the Internet Protocol (IPv4 and IPv6) over Ethernet.

The TcpIp module consists of 4 submodules:

- TcpIp (TCP, UDP, Socket interface)
- IpV4 (IPv4, ARP, ICMPv4, DHCPv4 client)
- IpV6 (IPv6, NDP, ICMPv6, DHCPv6 client)
- DhcpV4Server (DHCPv4 server)

Note

The submodules **IpV4**, **IpV6** and **DhcpV4Server** must be licensed and explicitly ordered. Your delivery may not contain all submodules.

Some sections of this Technical Reference do not apply if the submodule is not part of your delivery.

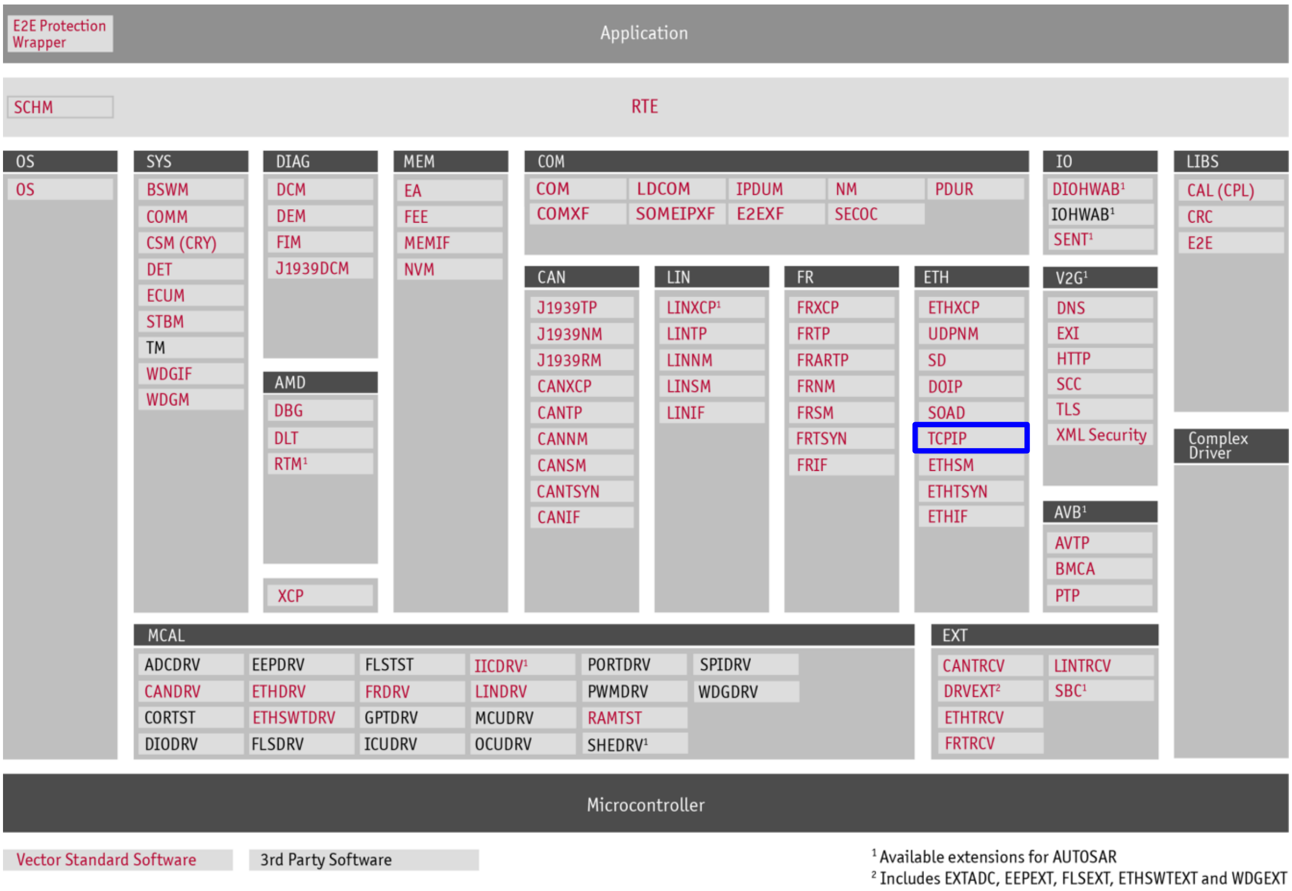
The following combinations of submodules are possible:

Variant	TcpIp	IpV4	IpV6	DhcpV4Server	SC*	Description
1	x	x			1	IPv4 only
2	x		x		2	IPv6 only
3	x	x	x		3	IPv4/IPv6 Dual Stack
4	x	x		x	1	SC1 with DHCPv4 Server
5	x	x	x	x	3	SC3 with DHCPv4 Server

* SC: Scalability Class according to [1] AUTOSAR_SWS_TcpIp

2.1 Architecture Overview

The following figure shows where the TCPIP is located in the AUTOSAR architecture.



[1] Available extensions for AUTOSAR
[2] Includes EXTADC, EEPEXT, FLSEXT, ETHSWTEXT and WDGEXT

Figure 2-1 AUTOSAR 4.2 Architecture Overview

3 Functional Description

3.1 Features

The features listed in the following tables cover the complete functionality specified for the TCPIP.

The AUTOSAR standard functionality is specified in [1], the corresponding features are listed in the tables

- Table 3-1 Supported AUTOSAR standard conform features
- Table 3-2 Not supported AUTOSAR R4.2.1 standard conform features

Vector Informatik provides further TCPIP functionality beyond the AUTOSAR standard. The corresponding features are listed in the table

- Table 3-6 Features provided beyond the AUTOSAR standard

Supported AUTOSAR R4.2.1 Standard Conform Features
Configuration according to AUTOSTAR R4.2.1
Implementation of TCP and UDP
Implementation of IPv4 (incl. ARP, ICMPv4, DHCPv4 client/ server) with multi controller support
Implementation of IPv6 (incl. NDP, ICMPv6, DHCPv6 client) with multi controller support

Table 3-1 Supported AUTOSAR standard conform features

Not supported AUTOSAR R4.2.1 standard conform features	
TcpIp	Production error reporting.
TcpIp	TCP/IP Stack State handling (STARTUP, ONHOLD, SHUTDOWN)
TcpIp	API TcpIp_RequestComMode (empty API implementation only)
TcpIp	API TcpIp_ResetIpAssignment
TcpIp / TCP	API TcpIp_TcpTransmit: ForceRetrieve parameter always assumed as TRUE.
TcpIp / TCP	Simultaneous Open Attempts, Delayed ACKs.
TcpIp / TCP	The following configuration parameters are currently not used: TcpIpTcpReceiveWindowMax, TcpIpTcpMaxRtx, TcpIpTcpSynMaxRtx, TcpIpTcpSynReceivedTimeout, TcpIpTcpFinWait2Timeout
TcpIp / UDP	Forward received multicasts to all matching sockets.
IpV4	Path MTU Discovery for IPv4 (IETF RFC1191).
IpV4 / ARP	Extract physical address information from each received IP packet.
IpV4 / ARP	Send gratuitous ARP packets after new IP address has been assigned.
IpV4 / ICMP	Transmit/Receive ICMP messages other than Echo Request and Echo Reply Configuration of ICMP message handler.
IpV4 / ICMP	API TcpIp_IcmpTransmit

Table 3-2 Not supported AUTOSAR R4.2.1 standard conform features

3.1.1 IETF RFC Conformance

The TcpIp module conforms to the following RFCs published by the Internet Engineering Task Force (IETF) with the noted exceptions.

- RFC referenced by the AUTOSAR SWS TcpIp R4.2.1
- RFC supported beyond the AUTOSAR standard

Note
Unless otherwise noted the following parts of the IETF RFCs are not implemented:

- Sections that specify server/router specific functionality
- Sections that are excluded by the AUTOSAR specification

IETF RFC	Title
768	User Datagram Protocol
793	Transmission Control Protocol
813	Window and Acknowledgement Strategy in TCP
896	Congestion Control in IP/TCP Internetworks (Nagle)
1122	Requirements for Internet Hosts -- Communication Layers
1323	TCP Extensions for High Performance
2018	TCP Selective Acknowledgment Options
5482	TCP User Timeout Option
5681	TCP Congestion Control
6298	Computing TCP's Retransmission Timer
6582	The NewReno Modification to TCP's Fast Recovery Algorithm

Table 3-3 List of supported TcpIp IETF RFCs

IETF RFC	Title
791	Internet Protocol
826	An Ethernet Address Resolution Protocol (ARP)
792	Internet Control Message Protocol (ICMP)
815	IP Datagram Reassembly Algorithms
894	A Standard for the Transmission of IP Datagrams over Ethernet Networks
1112	Host Extensions for IP Multicasting
2131	Dynamic Host Configuration Protocol (DHCP) / client and server
3927	Dynamic Configuration of IPv4 Link-Local Addresses

Table 3-4 List of supported IPv4 IETF RFCs

IETF RFC	Title
1981	Path MTU Discovery for IP version 6
2460	Internet Protocol, Version 6 (IPv6) Specification
2464	Transmission of IPv6 Packets over Ethernet Networks
2711	IPv6 Router Alert Option
3122	Extensions to IPv6 Neighbor Discovery for Inverse Discovery Specification
3315	Dynamic Host Configuration Protocol for IPv6 (DHCPv6) / client
3810	Multicast Listener Discovery Version 2 (MLDv2) for IPv6
4291	IP Version 6 Addressing Architecture
4429	Optimistic Duplicate Address Detection (DAD) for IPv6
4443	Internet Control Message Protocol (ICMPv6) for the IPv6 Specification
4861	Neighbor Discovery for IP version 6 (IPv6)
4862	IPv6 Stateless Address Autoconfiguration
4884	Extended ICMP to Support Multi-Part Messages
4941	Privacy Extensions for Stateless Address Autoconfiguration in IPv6
5095	Deprecation of Type 0 Routing Headers in IPv6
5722	Handling of Overlapping IPv6 Fragments
5942	IPv6 Subnet Model: The Relationship between Links and Subnet Prefixes
6085	Address Mapping of IPv6 Multicast Packets on Ethernet
6106	IPv6 Router Advertisement Options for DNS Configuration
6724	Default Address Selection for Internet Protocol version 6 (IPv6)

Table 3-5 List of supported IPv6 IETF RFCs

3.1.2 Additions / Extensions

The following features are provided beyond the AUTOSAR standard:

Features Provided Beyond The AUTOSAR Standard	
API TcpIp_GetLocNetAddr: Get the local IPv4 address	5.2.25
API TcpIp_GetLocNetMask: Get the local IPv4 network mask	5.2.26
API TcpIp_GetIpAddrCfgSrc: Get the address configuration source of a local address	5.2.35
API TcpIp_DiagDataReadAccess: Read access to some internal variables	5.2.39
API TcpIp_GetDhcpTimeoutInfo: Get the information if a DHCP timeout has occurred	5.2.34
API TcpIp_GetLocSockAddr: Get the current local socket address for one socket	5.2.38
API TcpIp_GetRemNetAddr: Get the current remote socket address for one socket	5.2.28
IPv4 Manual override of any IPv4 local unicast address during runtime	6.1.1
IpV4 Configuration of static ARP Tables (disable dynamic address resolution)	6.2.2
IpV4: Static Link Layer Address Resolution	6.4
IpV4/IpV6: Persistence of static IPv4/IPv6 local unicast address	6.1.1

Features Provided Beyond The AUTOSAR Standard	
IpV6: Transmit and reassemble fragmented packets	6.1.1
IpV6: Inverse Neighbor Discovery	
TcpIp / TCP Reception of out-of-order segments	3.1.2.1.1
TcpIp / TCP Selective Acknowledgement	3.1.2.1.2
TcpIp / TCP: Use TLS as a TCP plug-in to encrypt and authenticate a TCP connection	5.2.21 5.2.22 5.2.23 5.2.24
TcpIp: Separate MainFunctions for reception/transmission of user data and state handling.	3.5
TcpIp: Extended <Up>_CopyTxData-Callback for transmission of data with dynamic length.	6.1.1
API TcpIp_ClearARCache: Clear the address resolution cache (currently only IPv6/NDP is supported)	5.2.40

Table 3-6 Features provided beyond the AUTOSAR standard

3.1.2.1 TCP Extensions

3.1.2.1.1 TCP Out Of Order Support

For TCP a support for messages received out of order (OOO) can be enabled. If this feature is enabled TCP stores incoming segments even if previous messages are missing. As soon as the missing segments are received the TCP forwards all stored data to the socket owner.

3.1.2.1.2 TCP Selective Acknowledgement Support

In the standard configuration TCP won’t notify the communication partner that it has a gap in its received data sequence. This feature called ‘selective acknowledgement’ (SACK) can be enabled separately (in addition to OOO). Before SACK can be used it has to be agreed during TCP connection setup (using TCP header options).

3.1.2.2 Parameter Checking

3.1.2.2.1 Parameter checking for submodule TcpIp

The following development errors and error codes are reported additionally to the errors defined in [1].

Development errors:

- TCPIP_API_ID_SHUTDOWN
- TCPIP_API_ID_PROVIDE_TX_BUFFER
- TCPIP_API_ID_TRANSMIT_TO
- TCPIP_API_ID_GET_LOC_NET_ADDR
- TCPIP_API_ID_GET_LOC_NET_MASK
- TCPIP_API_ID_SET_DHCP_HOST_NAME
- TCPIP_API_ID_SET_DHCP_V4_TX_OPT
- TCPIP_API_ID_SET_DHCP_V6_TX_OPT
- TCPIP_API_ID_CBK_RX_INDICATION
- TCPIP_API_ID_CBK_TX_CONFIRMATION
- TCPIP_API_ID_CBK_TCP_ACCEPTED
- TCPIP_API_ID_CBK_TCP_CONNECTED
- TCPIP_API_ID_CBK_TCP_IP_EVENT
- TCPIP_API_ID_RX_SOCK_IDENT
- TCPIP_API_ID_TCP_STATE_CHG
- TCPIP_API_ID_LOC_IP_ASS_CHG
- TCPIP_API_ID_PREPARE_SHUTDOWN
- TCPIP_API_ID_TRCV_LINK_STATE_CHG
- TCPIP_API_ID_ADDR_RES_TIMEOUT
- TCPIP_API_ID_SOCK_ADDR_IS_EQ
- TCPIP_API_ID_SOCK_ADDR_IS_EQ_OR_UNSPEC
- TCPIP_API_ID_PATH_MTU_CHG
- TCPIP_API_ID_GET_IP_ADDR_CFG_SRC
- TCPIP_API_ID_GET_IP_ADDR
- TCPIP_API_ID_GET_REM_NET_ADDR
- TCPIP_API_ID_CLEAR_AR_CACHE

Error codes:

- TCPIP_E_INV_SOCK_ID

3.1.3 Limitations

3.1.3.1 General Limitations

- The number of sockets is limited to 254.

3.1.3.2 Limitations for AUTOSAR APIs

3.1.3.2.1 DHCP write and read option

The APIs TcpIp_DhcpWriteOption and TcpIp_DhcpV6WriteOption only support setting the FQDN option. There is no generic option handling implemented.

The APIs TcpIp_DhcpReadOption and TcpIp_DhcpV6ReadOption only support reading the FQDN option. Only the data that was previously set via the WriteOption API can be read, no received option data will be evaluated. There is no generic option handling implemented.

3.1.3.2.2 Triggering IP address request

The request of IP address assignment is configurable, but only a configuration subset is supported. All address assignments for one IpAddrId need to have the same configuration.

3.1.3.3 RFC specific deviations of submodule IpV4

3.1.3.3.1 RFC791 Internet Protocol [7]

Section in Document	Limitation
Page 15 / IPv4 Header Options	IP options may appear in received packets but will be silently ignored.

Table 3-7 Deviations from IETF RFC791

3.1.3.3.2 RFC792 Internet Control Message Protocol (ICMP) [8]

Section in Document	Limitation
ICMP Messages	ICMP message support limited to Echo and Echo Reply. The length of the data mirrored in the Echo Reply message is limited (configurable).

Table 3-8 Deviations from IETF RFC792

3.1.3.3.3 RFC1323 TCP Extensions for High Performance [15]

Section in Document	Limitation
2. “TCP Window Scale Option”	Not supported.
4. “PAWS -- Protect Against Wrapped Sequence Numbers”	Not supported.

Table 3-9 Deviations from IETF RFC1323

3.1.3.3.4 RFC2131 Dynamic Host Configuration Protocol (DHCP) [18]

Section in Document	Limitation
General	Basic DHCP client and server support implemented.
Probing of Addresses	IP addresses assigned by DHCP are not probed by the IP-Module to be unique.
4.3.5 “DHCPINFORM message”	DHCPINFORM messages are not send by the client and are ignored by the DHCP server.

Table 3-10 Deviations from IETF RFC2131

3.1.3.3.5 RFC4702 The DHCP Client FQDN option [29]

Section in Document	Limitation
General	The ‚Flags‘ field is filled with fixed values. No DNS update handling is implemented in any way.

Table 3-11 Deviations from IETF RFC4702

3.1.3.4 RFC specific deviations of submodule IpV6

3.1.3.4.1 RFC2460 Internet Protocol, Version 6 (IPv6) Specification [19]

Section in Document	Limitation
4. “IPv6 Extension Headers”	IPsec / “Authentication” and “Encapsulating Security Payload” (ESP) Headers are not supported.

Table 3-12 Deviations from IETF RFC2460

3.1.3.4.2 RFC3315 Dynamic Host Configuration Protocol for IPv6 (DHCPv6) [23]

Section in Document	Limitation
22.4. “Identity Association for Non-temporary Addresses Option”	At most one DHCPv6 address can be configured on each controller.

Table 3-13 Deviations from IETF RFC3315

3.1.3.4.3 RFC3484 Default Address Selection for Internet Protocol version 6 (IPv6) [44]

Section in Document	Limitation
5. “Source Address Selection”	Rule 4: Prefer home addresses. (no Mobile IP support) Rule 6: Prefer matching label.

Table 3-14 Deviations from IETF RFC3484

3.1.3.4.4 RFC4291 IP Version 6 Addressing Architecture [26]

Section in Document	Limitation
2.5.1. “Interface Identifiers”	Only 64 bit interface identifiers are supported: “For all unicast addresses, except those that start with the binary value 000, Interface IDs are required to be 64 bits long and

Section in Document	Limitation
	to be constructed in Modified EUI-64 format.”

Table 3-15 Deviations from IETF RFC4291

3.1.3.4.5 RFC4443 Internet Control Message Protocol (ICMPv6) for the IPv6 Specification [28]

Section in Document	Limitation
3.1. “Destination Unreachable Message”	The IpV6 will not send Destination Unreachable Messages. (ICMP Message Type 1)

Table 3-16 Deviations from IETF RFC4443

3.1.3.4.6 RFC4704 The DHCPv6 Client FQDN option [30]

Section in Document	Limitation
General	The ‚Flags‘ field is filled with fixed values. No DNS update handling is implemented in any way.

Table 3-17 Deviations from IETF RFC4443

3.1.3.4.7 RFC4941 Privacy Extensions for Stateless Address Autoconfiguration in IPv6 [34]

RFC4941 “Privacy Extensions for Stateless Address Autoconfiguration” is currently not supported in MICROSAR4 Releases.

3.1.3.4.8 RFC3810 Multicast Listener Discovery Version 2 (MLDv2) for IPv6 [24]

RFC3810 “Multicast Listener Discovery Version 2 (MLDv2) for IPv6” is currently not supported in MICROSAR4 Releases.

3.1.4 Known Issues (low priority)

This section lists known issues that are not reported by the Vector issue reporting because they have low priority.

Compiler Warning "conditional expression is constant"

Some compilers may report warnings about conditional expressions that always evaluate to TRUE. These warnings are configuration dependent. If a TCPIP feature is enabled but not used by all IP instances/controllers a runtime check is required in order to decide whether the feature is enabled on a specific controller. If the feature is enabled on all controllers the condition is replaced by the constant value TRUE, so the compiler can eliminate the runtime check.

If a feature is disabled on all controllers, the code will be completely excluded from compilation.

3.1.5 Known Issues (IETF RFC Conformance)

The following items describe behavior of the Vector TCP/IP stack that slightly deviates from the behavior defined by the considered IETF RFCs.

These deviations have been identified using IETF RFC conformance tests.

Although this behavior may reduce TCP/IP performance in a few cases, it should not cause operational issues in currently known use cases.

IPv6 sends ICMP error messages in response to ICMP error messages

According to RFC4443 the TCPIP must not send an ICMPv6 error message as a result of receiving an ICMPv6 error message. But the IpV6 will do this if the received ICMPv6 error message is corrupt before the ICMPv6 header of the received error message.

ESCAN00071983

IPv6 stops reassembly of packet if the same fragment is received twice

According to IETF RFC 5722 IPv6 shall stop reassembly of a packet if two fragments overlap. But the RFC requirements are unclear about what shall be done if the same fragment is received twice. The IPv6 will also stop reassembly of a packet if a fragment is received twice.

ESCAN00084714

TCP backs off retransmission timer on fast retransmit

When a "Fast Retransmit" according to IETF RFC5681 has been sent, TCP will back-off the timer for the next regular retransmission.

ESCAN00084732

TCP uses too big cwnd if SYN or SYNACK is lost during active open

If a SYN/ACK segment gets lost during connection establishment, TCP assumes a bigger congestion window (CWND) than specified in IETF RFC5681. TCP will still respect the receive window of the remote host.

ESCAN00084734

TCP retransmits more than one segment after retransmit timeout expired

If the retransmit timer expires TCP may send more than one unacknowledged segment. The congestion window (CWND) is reduced as specified by IETF RFC6298.

ESCAN00084735

TCP does not echo timestamp option value of received SYN

If the TCP timestamp option according to IETF RFC1323 is enabled, the TCP will not send a timestamp echo value in a SYN segment. TCP sends timestamp echo values in other segments as specified by the RFC.

ESCAN00084729

TCP does not reply timestamp of earliest unacknowledged segment

If multiple TCP segments with different timestamp values are ACKed by the TCP the most recent timestamp value instead of the oldest value will be replied.

ESCAN00084731

Limitations of TCP User Timeout Option (RFC5482)

The user timeout value for a TCP socket will only be used if it is set after connection establishment.

Retransmitted segments do not contain a user timeout option.

Received user timeout values will only be considered while the connection is in an established state.

ESCAN00084733

TCP zero window probe intervals

Zero window probes are sent with an interval of the configured default retransmission timer until a window update is received.

ESCAN00084741

TCP sends window update with a value less than one MSS

TCP does not wait until the window can hold at least one MSS before sending a window update. This may affect TCP performance if very small chunks of data are received.

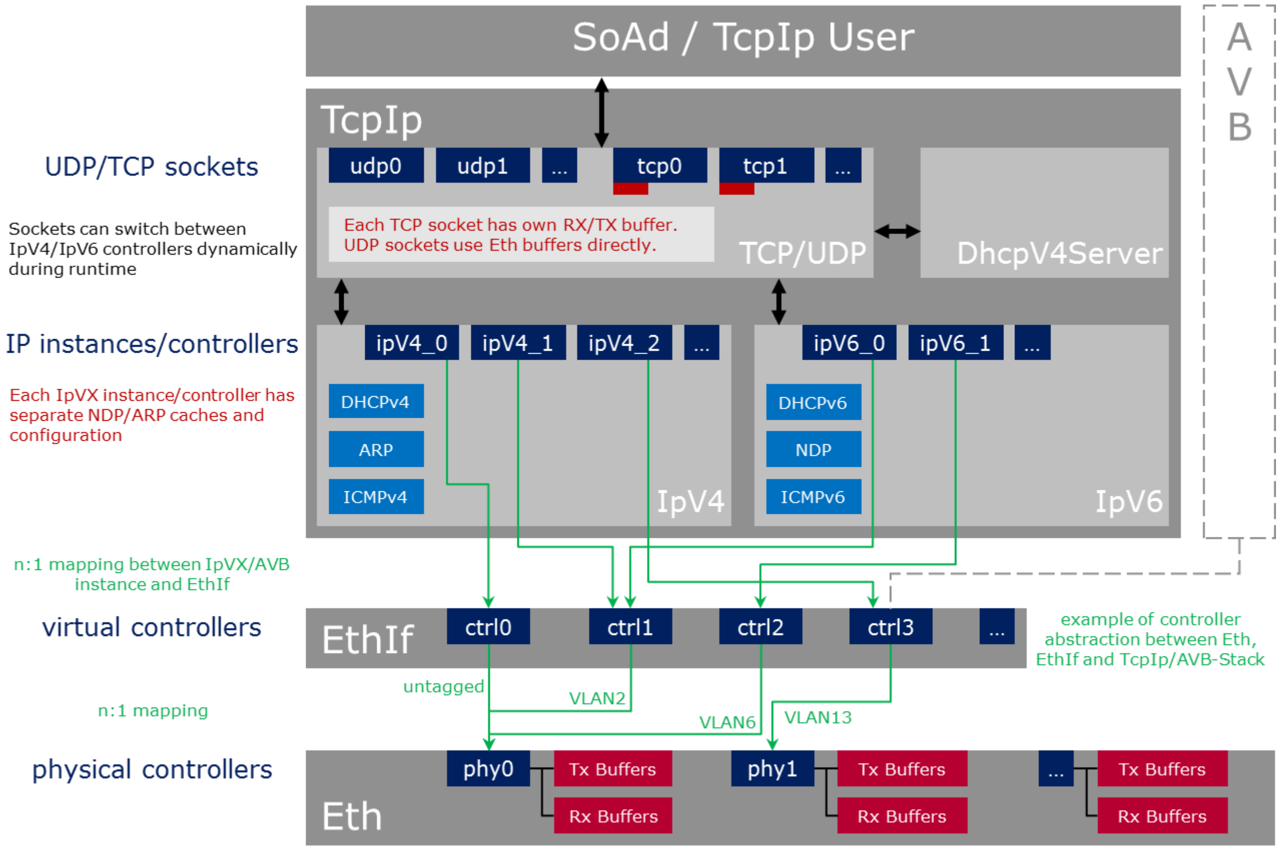
ESCAN00084744

TCP does not support simultaneous open of connection

The TCP specification allows that two TCP peers actively open a TCP connection simultaneously. This case is not used in practice and is therefore not supported.

ESCAN00084723

3.2 Structure of the TcpIp Stack



The TCPIP module provides UDP and TCP sockets to the upper layer (user).

A socket can be opened by the user during runtime via the API TcpIp_<Up>GetSocket by specifying the socket protocol (UDP/TCP) and the Internet Protocol version (IPv4/INET or IPv6/INET6).

After a socket has been opened it must be bound to a local IP address and port via the API TcpIp_Bind(). Local IP addresses are represented by identifiers (LocalAddrIds). A LocalAddrId represents an address of a specific IP instance/controller and is not shared between IP instances. IP instances operate independent from each other and use their own data structures like ARP/NDP caches and addresses.

Each IP instance/controller operates on exactly one virtual controller provided by the EthIf module.

Expert Knowledge
A virtual controller of the EthIf represents a physical Eth controller or a VLAN on a physical Eth controller.

3.2.1 IpV6

Each IPv6 instance has at least an auto configured link-local unicast address (see [32] 5.3. Creation of Link-Local Addresses) and an all-nodes multicast address. Additional unicast and multicast addresses may be configured.

3.2.2 IpV4

Each IPv4 instance has exactly one unicast and one broadcast address. Additionally one or more multicast addresses may be configured.

3.2.3 TLS

The TLS module has been adapted to work as a TcpIp plug-in. After requesting a TCP socket, the socket owner can call additional functions to request the usage of TLS on this socket.

- TcpIp_ChangeParameter(sock, TCPIP_PARAMID_V_USE_TLS, TRUE)
- TcpIp_ChangeParameter(sock, TCPIP_PARAMID_V_TLS_TXBUFSIZE, size)
- TcpIp_ChangeParameter(sock, TCPIP_PARAMID_V_TLS_RXBUFSIZE, size)
- TcpIp_Tls_SetClientCertInfo(sock, …) if client authentication shall be used
- TcpIp_ChangeParameter(sock, TCPIP_PARAMID_V_TLS_HEARTBEAT_MODE, TLS_HB_PEER_NOT_ALLOWED_TO_SEND)
- TcpIp_ChangeParameter(sock, TCPIP_PARAMID_V_TLS_HEARTBEAT_PERIOD_S, time)

The socket owner uses the normal TcpIp API to transmit and receive data.

Tls implements an additional callback to the TcpIp socket owner to exchange information on the validation of the server certificate chain. The callback 'Tls Validation Result' is an optional callback that is only used for sockets using TLS, and even for those TLS sockets the callback is optional.

For further information on TLS refer to [45].

3.2.4 DHCPv4 Server

In addition to a DHCPv4 client, the TcpIp optionally supports a DHCPv4 server according to IETF RFC2131 [18].

Independent DHCPv4 server instances can be configured for each IpV4 controller/VLAN.

According to AUTOSAR a DHCPv4 server instance optionally supports the assignment of IPv4 addresses based on the Ethernet Switch Port information. In this case the address assignment configuration is selected depending on the EthIf Switch Port on which the client DISCOVER/REQUEST message was received.

If no matching Ethernet Switch Port dependent address assignment configuration is found or configured a default address assignment configuration can be used.

Configuration in DaVinci Configurator Pro
The **TcpIpDhcpServerConfig** must be referenced by the **TcpIpDhcpServerConfigRef** parameter of the corresponding **TcpIp/TcpIpConfig/TcpIpCtrl**.

3.3 Initialization

The TcpIp is initialized by calling the TcpIp_InitMemory service, and by calling the TcpIp_Init service with the configuration as parameter. Currently only the configuration variant ‘pre-compile’ is supported, so the configuration pointer used for TcpIp_Init can be a null-pointer.

3.4 States

The TcpIp is operational after the initialization. Sockets can be requested immediately, but binding a socket to a local address is possible only after a local address could be assigned to the node previously.

IP addresses can be assigned / get assigned after a physical link to the network was detected.

3.5 Main Functions

The TcpIp can be configured to use either a single or three MainFunctions:

Single MainFunction (default)	Multiple MainFunctions
TcpIp_MainFunction	TcpIp_MainFunctionRx TcpIp_MainFunctionState TcpIp_MainFunctionTx

Table 3-18 MainFunctions of the TcpIp module

By default only a single MainFunction needs to be called. This MainFunction processes state changes, transmission and reception of data for all submodules (IPv4, IPv6, UDP, TCP and DHCPv4 Server). The TcpIp_MainFunction wraps the calls to the Rx, Tx and State MainFunctions.

In order to optimize the response behavior for certain use cases it may be reasonable to split-up the Rx, Tx and state processing parts of the MainFunction. These separate MainFunctions may then be arranged individually inside the task.

There are some restrictions that apply to the mapping of the MainFunctions:

> All three MainFunctions must be called cyclically with the same period. Otherwise the timing of the protocols will be inaccurate.

> If the MainFunctions are mapped to different tasks these tasks must not interrupt each other.

Configuration in DaVinci Configurator Pro
See configuration parameters **TcpIpMainFunctionPeriod** and **TcpIpMainFunctionSplitEnabled**.

Depending on the value of TcpIpMainFunctionSplitEnabled one or three **Schedulable Entities** need to be mapped in the **Task Mapping** view of the Rte.

3.6 Error Handling

3.6.1 Development Error Reporting

By default, development errors are reported to the DET using the service `Det_ReportError()` as specified in [2], if development error reporting is enabled (i.e. pre-compile parameter `TCPIP_DEV_ERROR_DETECT==STD_ON`).

If another module is used for development error reporting, the function prototype for reporting the error can be configured by the integrator, but must have the same signature as the service `Det_ReportError()`.

The reported TCPIP ID is 170.

The submodules use the following instance IDs:

>	TcpIp:	1
>	IpV4:	2
>	IpV6:	3
>	DhcpV4Server:	4

The reported service IDs identify the services which are described in 5.2. The following table presents the service IDs and the related services:

Service ID	Service
0x01	TcpIp_Init
0x02	TcpIp_GetVersionInfo
0x03	TcpIp_GetSocketForUser (indirectly called by each configured socket owner)
0x04	TcpIp_Close
0x05	TcpIp_Bind
0x06	TcpIp_TcpConnect
0x07	TcpIp_TcpListen
0x08	TcpIp_TcpReceived
0x0A	TcpIp_RequestIpAddrAssignment
0x0B	TcpIp_ReleaseIpAddrAssignment
0x0D	TcpIp_DhcpWriteOption
0x0E	TcpIp_DhcpReadOption
0x0F	TcpIp_ChangeParameter
0x11	TcpIp_GetPhysAddr
0x12	TcpIp_UdpTransmit
0x13	TcpIp_TcpTransmit
0x14	TcpIp_RxIndication
0x15	TcpIp_MainFunction
0x16	TcpIp_GetRemotePhysAddr
0x17	TcpIp_GetCtrlIdx
0x19	TcpIp_DhcpV6ReadOption
0x1A	TcpIp_DhcpV6WriteOption
0x1C	TcpIp_GetNdpCacheEntries
0x1D	TcpIp_GetArpCacheEntries
0x81	TcpIp_ProvideTxBufferInt
0x82	TcpIp_ProvideTxBuffer
0x83	TcpIp_TransmitTo
0x84	TcpIp_TransmitToInt
0x85	TcpIp_GetLocNetAddr
0x86	TcpIp_GetLocNetMask
0x87	TcpIp_SetDhcpHostNameOption

Service ID	Service
0x88	TcpIp_Tls_SetClientCertInfo
0x89	TcpIp_Tls_GetNvmBlockIdForUsedRootCert
0x8A	TcpIp_Tls_GetNvmBlockIdForUsedRootCert
0x8B	TcpIp_ClearARCache
Services of the IpV4 submodule	
0x00	IpV4_Init
0x01	IpV4_Ip_MainFunction
0x03	IpV4_GetVersionInfo
0x04	IpV4_Ip_GetLocNetAddrCfgSrc
0x05	IpV4_Ip_SetNetAddr
0x07	IpV4_Ip_IsAddrIdAcceptable
0x20	IpV4_Ip_ProvideTxBuffer
0x21	IpV4_Ip_Transmit
0x22	IpV4_Ip_GetLocNetAddr
0x23	IpV4_Ip_GetLocNetMask
0x24	IpV4_Ip_AddrLossInd
0x25	IpV4_Ip_Dhcp_AddrInd
0x26	IpV4_Ip_ProvideNextTxBuffer
0x28	IpV4_Ip_RxIndication
0x29	IpV4_Ip_TxConfirmation
0x2A	IpV4_Ip_Cbk_TrcvLinkStateChg
0x30	IpV4_Ip_Init
0x31	IpV4_Ip_MainFunction
0x32	IpV4_Ip_Reset
0x33	IpV4_Ip_AddrConflictInd
0x34	IpV4_Ip_ResetSocket
0x35	IpV4_Ip_SetTimeToLive
0x36	IpV4_Ip_SetTypeOfService
0x37	IpV4_Ip_SetEthIfFramePrio
0x38	IpV4_Ip_RequestIpAddrAssignment
0x39	IpV4_GetLastDuplicateDhcpAddrDid
0x40	IpV4_Arp_RxIndication

Service ID	Service
0x50	IpV4_Arp_Init
0x51	IpV4_Arp_Reset
0x52	IpV4_Arp_MainFunction
0x53	IpV4_Arp_GetPhysicalAddress
0x54	IpV4_Arp_SendArpRequest
0x55	IpV4_Arp_MapIpToPhysMulticastAddr
0x56	IpV4_Arp_SendArpProbe
0x57	IpV4_Arp_SendArpAnnouncement
0x58	IpV4_Arp_SendGratuitousArpReq
0x60	IpV4_Icmp_SendEcho
0x70	IpV4_Icmp_Init
0x71	IpV4_Icmp_MainFunction
0x72	IpV4_Icmp_RxIndication
Services of the IpV6 submodule	
0x00	IpV6_GetVersionInfo
0x01	IpV6_InitMemory
0x02	IpV6_Init
0x03	IpV6_MainFunction
0x04	IpV6_ProvideTxBuffer
0x05	IpV6_Transmit
0x06	IpV6_RxIndication
0x07	IpV6_TxConfirmation
0x08	IpV6_Cbk_TrcvLinkStateChg
0x09	IpV6_SetAddress
0x0A	IpV6_GetSrcAddrPtr
0x0B	IpV6_GetDstAddrPtr
0x0C	IpV6_SetTrafficClass
0x0D	IpV6_SetFlowLabel
0x0E	IpV6_IsValidDestinationAddress
0x0F	IpV6_GetLocalAddressCfgSrc
0x10	IpV6_GetCurrentHopLimit
0x11	IpV6_GetPhysAddr
0x12	IpV6_GetAsBcAddrId
0x13	IpV6_IsAddrIdAcceptable
0x14	IpV6_SetHopLimit

Service ID	Service
0x15	IpV6_SetEthIfFramePrio
0x16	IpV6_ResetSocket
0x17	IpV6_CancelTransmit
0x18	IpV6_GetLocalAddress
0x19	IpV6_GetDefaultRouterAddress
0x20	IpV6_Icmp_ProvideTxBuffer
0x21	IpV6_Icmp_TxEchoRequest
0x23	IpV6_Icmp_Transmit
0x24	IpV6_Icmp_MainFunction
0x30	IpV6_Ndp_MainFunction
0x40	IpV6_Mld_MainFunction
0x41	IpV6_Mld_Init

Table 3-19 Service IDs

The errors reported to DET are described in the following table:

Error Code	Description
0x01	TCPIP_E_NOTINIT
0x02	TCPIP_E_PARAM_POINTER
0x03	TCPIP_E_INV_ARG
0x04	TCPIP_E_NOBUFS
0x07	TCPIP_E_MSGSIZE
0x08	TCPIP_E_PROTOTYPE
0x09	TCPIP_E_ADDRINUSE
0x0A	TCPIP_E_ADDRNOTAVAIL
0x0B	TCPIP_E_ISCONN
0x0C	TCPIP_E_NOTCONN
0x0D	TCPIP_E_NOPROTOOPT
0x0E	TCPIP_E_AFNOSUPPORT
0x0F	TCPIP_E_INIT_FAILED
Services of the IpV4 submodule	
0x01	IPV4_E_NOT_INITIALIZED
0x02	IPV4_E_INV_POINTER
0x03	IPV4_E_INV_PARAM
0x04	IPV4_E_INV_CTRL_IDX
0x05	IPV4_E_INV_SOCK_IDX
0x06	IPV4_E_INV_CONFIG
0x07	IPV4_E_INV_ADDR_ID

Error Code	Description
Services of the IpV6 submodule	
0x01	IPV6_E_NOT_INITIALIZED
0x02	IPV6_E_INV_POINTER
0x03	IPV6_E_INV_SOCK_IDX
0x04	IPV6_E_INV_BUFF_IDX
0x05	IPV6_E_INV_CTRL_IDX
0x06	IPV6_E_INV_DATA_IDX
0x07	IPV6_E_INV_PARAM
0x08	IPV6_E_DLIST_BUFFER_EMPTY

Table 3-20 Errors reported to DET

3.6.2 Production Code Error Reporting

By default, production code related errors are reported to the DEM using the service `Dem_ReportErrorStatus()` as specified in [3], if production error reporting is enabled (i.e. at least one reference to a DEM event is configured in the TcpIp by referencing a DEM event using this EcuC path: `/ActiveEcuC/Dem/DemConfigSet/DemEventParameter`).

If another module is used for production code error reporting, the function prototype for reporting the error can be configured by the integrator, but must have the same signature as the service `Dem_ReportErrorStatus()`.

The errors reported to DEM are described in the following table:

Error Code	Description
TCPIP_E_DUPLICATE_DHCP_ADDR	An ECU's IPv4 address was used by another network node, i.e., the ECU received a frame which uses the same source IPv4 address but a different source MAC address. The DEM event is configured by creating a reference to a DEM event named `E DUPLICATE DHCP ADDR` in the EcuC-container: `/ActiveEcuC/TcpIp/TcpIpConfig/TcpIpCtrl_0_Fix/TcpIpCtrlDemEventParameterRefs` Refer to IpV4_GetLastDuplicateDhcpAddrDid for details on how to retrieve the related DID information.

Table 3-21 Errors reported to DEM

Note
From all references contained in the EcuC container `/ActiveEcuC/TcpIp/TcpIpConfig/TcpIpCtrl_0_Fix/TcpIpCtrlDemEventParameterRefs` the TcpIp currently only supports:
TCPIP_E_DUPLICATE_DHCP_ADDR

3.6.3 Other Error Reporting and Diagnostic Options

The TcpIp offers the possibility to track the following OEM-specific error conditions:

3.6.3.1 Tracking discarded ARP entries

TcpIp ARP (IPv4) offers the possibility to report an error in case an existing ARP entry was replaced by a new entry or a new entry could not be added to the ARP table because the maximum number of ARP entries has been reached.

Implementing this scenario is supported by the IPv4-subcomponent as follows:

- Turning on EcuC switch TcpIpArpDiscardedEntryHandling leads to the handling of the ARP-cache as ARP-table, which is never updated once it is full.
- Configuring the callback TcpIpPhysAddrChgHandler offers the possibility to track all new entries that are inserted into the ARP table.
- Configuring the callback TcpIpPhysAddrChgDiscardedHandler offers the possibility to track all IP addresses which are discarded because the ARP table is already full. (At the time when the address should have been resolved.)

3.6.3.2 Triggering the publishing of IP-address via diagnostic services

Some diagnostic services may demand that the ECU sends a gratuitous ARP request packet to publish its IP-address. This possibility is offered by the function IpV4_Arp_SendGratuitousArpReq.

3.6.3.3 Get DHCP Status for IPv4

TcpIp DHCPv4 offers the possibility to retrieve the status of the DHCP handshaking for an IPv4 address that has DHCP configured as address assignment method.
This possibility is offered by the function TcpIp_DhcpV4_GetStatus.

3.6.3.4 IPv6 Duplicate Address Detection Conflict

TcpIp supports the implementation of a DEM event that is equivalent to the runtime error TCPIP_E_DADCONFLICT described in [1].

Setting the DEM event to ‘active’ can be done by configuring the callback DADAddressConflict() (Refer to section TcpIpDuplicateAddressDetectionConfig).

In order to ‘heal’ the DEM event, the LocalAddrId parameter of the latter callback has to be stored and the state-change of the affected address back to a valid IP address has to be tracked using the configurable callback LocalIpAddrAssignmentChg()-callback (Refer to

EcuC item: /MICROSAR/TcpIp/TcpIpConfig/TcpIpSocketOwnerConfig/TcpIpSocketOwner/TcpIpSocketOwnerLocalIpAddrAssignmentChgName).

4 Integration

This chapter gives necessary information for the integration of the MICROSAR TCPIP into an application environment of an ECU.

4.1 Scope of Delivery

The delivery of the TCPIP contains the files which are described in the chapters 4.1.1 and 4.1.2:

4.1.1 Static Files

File Name	Description
Static Files (contained in each Delivery)	
TcpIp.(c|h)	Implementation/Declaration of public TcpIp APIs
TcpIp_Cbk.h	Declaration of public call-back APIs
TcpIp_Types.h	Types header for ASR API of TcpIp
TcpIp_Tcp.(c|h)	Implementation of TCP (internal functions)
TcpIp_Tcp_Cbk.h	Implementation of TCP (call-back function declarations)
TcpIp_Udp.(c|h)	Implementation of UDP (internal functions)
TcpIp_Udp_Cbk.h	Implementation of UDP (call-back function declarations)
TcpIp_Priv.(c|h)	Implementation of internal/private TcpIp functions
TcpIp_Priv_Types.h	TcpIp internal Type definitions
Optional Static Files (contained in Delivery if IPv4 support was ordered)	
TcpIp_IpV4.(c|h)	Implementation of IPv4 (internal functions)
TcpIp_IpV4_Cbk.h	Implementation of IPv4 (call-back function declarations)
TcpIp_IpV4_Types.h	Internal type definitions of the IpV4 submodule
TcpIp_Arp.(c|h)	Implementation of ARP for IPv4 (internal functions)
TcpIp_Arp_Cbk.h	Implementation of ARP for IPv4 (call-back function declarations)
TcpIp_DhcpV4.(c|h)	Implementation of DHCP Client for IPv4 (internal functions)
TcpIp_DhcpV4_Cbk.h	Implementation of DHCP Client for IPv4 (call-back function declarations)
TcpIp_IcmpV4.(c|h)	Implementation of ICMP for IPv4 (internal functions)
TcpIp_IcmpV4_Cbk.h	Implementation of ICMP for IPv4 (call-back function declarations)
TcpIp_IpV4_Priv.(c|h)	Implementation of internal/private functions of the IpV4 submodule
Optional Static Files (contained in Delivery if DHCPv4 Server support was ordered)	
TcpIp_DhcpV4Server.(c|h)	Implementation of DHCP Server for IPv4
Optional Static Files (contained in Delivery if IPv6 support was ordered)	

File Name	Description
TcpIp_IpV6.(c|h)	Implementation of IPv6 (internal functions)
TcpIp_IpV6_Cbk.h	Implementation of IPv6 (call-back function declarations)
TcpIp_IpV6_Types.h	Internal type definitions of the IpV6 submodule
TcpIp_IcmpV6.(c|h)	Implementation of ICMP for IPv6 (internal functions)
TcpIp_Ndp.(c|h)	Implementation of NDP for IPv6 (internal functions)
TcpIp_Mld.(c|h)	Implementation of MLDv2 for IPv6 (internal functions)
TcpIp_DhcpV6.(c|h)	Implementation of DHCP Client for IPv6 (internal functions)
TcpIp_DhcpV6_Cbk.h	Implementation of DHCP Client for IPv6 (call-back function declarations)
TcpIp_IpV6_Priv.(c|h)	Implementation of internal/private functions of the IpV6 submodule

Table 4-1 Static files

4.1.2 Dynamic Files

The dynamic files are generated by the configuration tool DaVinci Configurator Pro. The files for the submodules IpV4, IpV6 and DhcpV4Server are only generated if the submodules are enabled.

File Name	Description
Dynamic Files (always generated if TcpIp module exists in configuration)	
TcpIp_Cfg.h	TcpIp Pre-compile time parameter configuration
TcpIp_Lcfg.(c|h)	TcpIp Link-time parameter configuration
Optional Dynamic Files (generated if DHCPv4 Server support was ordered and enabled)	
TcpIp_DhcpV4Server_Cfg.h	DhcpV4Server Pre-compile time parameter configuration
TcpIp_DhcpV4Server_Lcfg.(c|h)	DhcpV4Server Link-time parameter configuration
Optional Dynamic Files (generated if IPv4 support was ordered and enabled)	
TcpIp_IpV4_Cfg.h	IpV4 Pre-compile time parameter configuration
TcpIp_IpV4_Lcfg.(c|h)	IpV4 Link-time parameter configuration
Optional Dynamic Files (generated if IPv6 support was ordered and enabled)	
TcpIp_IpV6_Cfg.h	IpV6 Pre-compile time parameter configuration
TcpIp_IpV6_Lcfg.(c|h)	IpV6 Link-time parameter configuration

Table 4-2 Generated files

4.2 Critical Sections

The TCPIP calls service functions of upper layers in order to prevent interruption of critical sections.

These service functions have to be provided (normally by the Schedule Manager) and configured accordingly.

To ensure data consistency and a correct function of the TCPIP module the exclusive areas `TCPIP_EXCLUSIVE_AREA_0`, `IPV4_EXCLUSIVE_AREA_0` and `IPV6_EXCLUSIVE_AREA_0` have to be provided during the integration.

Considering the timing behavior of your system (e.g. depending on the CPU load of your system, priorities and interruptibility of interrupts and OS tasks and their jitter and delay times) the integrator has to choose and configure a critical section solution in such way that it is ensured that the API functions do not interrupt each other.

It is recommended to use the functions `SuspendAllInterrupts()` and `ResumeAllInterrupts`() (global interrupt locks) for the used exclusive areas to ensure data consistency.

5 API Description

5.1 Type Definitions

The types defined by the TCPIP are described in this chapter.

Type Name	C-Type	Description	Value Range
TcpIp_ConfigType		Configuration data structure of the TcpIp module.	Configuration data structure of the TcpIp module.
TcpIp_DomainType	uint16	TcpIp address families.	`TCPIP_AF_INET (0x02)` Use IPv4
			`TCPIP_AF_INET6 (0x1c)` Use IPv6
TcpIp_ProtocolType	Enum	Protocol type used by a socket.	`TCPIP_IPPROTO_TCP (0x06)` Use TCP
			`TCPIP_IPPROTO_UDP (0x11)` Use UDP
TcpIp_LocalAddrIdType	uint8	Address identification type for unique identification of a local IP address and EthIf Controller configured in the TcpIp module.	
TcpIp_SocketIdType	uint8, uint16	socket identifier type for unique identification of a TcpIp stack socket. TCPIP_SOCKETID_INVALID shall specify an invalid socket handle	
TcpIp_StateType	Enum	Specifies the TcpIp state for a specific EthIf controller.	`TCPIP_STATE_ONLINE` TCP/IP stack state for a specific EthIf controller is ONLINE, i.e. communication via at least one IP address is possible.
			`TCPIP_STATE_ONHOLD`

Type Name	C-Type	Description	Value Range
			TCP/IP stack state for a specific EthIf controller is ONHOLD, i.e. no communication is currently possible (e.g. link down).
			`TCPIP_STATE_OFFLINE` TCP/IP stack state for a specific EthIf controller is OFFLINE, i.e. no communication is possible.
			`TCPIP_STATE_STARTUP` TCP/IP stack state for a specific EthIf controller is STARTUP, i.e. IP address assignment in progress or ready for manual start, communication is currently not possible.
			`TCPIP_STATE_SHUTDOWN` TCP/IP stack state for a specific EthIf controller is SHUTDOWN, i.e. release of resources using the EthIf controller, release of IP address assignment.
TcpIp_IpAddrStateType	Enum	Specifies the state of local IP address assignment	`TCPIP_IPADDR_STATE_ASSIGNED` local IP address is assigned
			`TCPIP_IPADDR_STATE_ONHOLD` local IP address is assigned, but cannot be used as the network is not active
			`TCPIP_IPADDR_STATE_UNASSIGNED` local IP address is unassigned
TcpIp_EventType	Enum	Events reported by TcpIp.	`TCPIP_TCP_RESET` TCP connection was reset, TCP socket and all related resources have been released.
			`TCPIP_TCP_CLOSED` TCP connection was closed successfully, TCP socket and all related resources have been released.
			`TCPIP_TCP_FIN_RECEIVED` A FIN signal was received on the TCP connection, TCP socket is

Type Name	C-Type	Description	Value Range
			still valid.
			TCPIP_UDP_CLOSED UDP socket and all related resources have been released.
TcpIp_IpAddrAssignmentType	Enum	Specification of IPv4/IPv6 address assignment policy.	TCPIP_IPADDR_ASSIGNMENT_STATIC Static configured IPv4/IPv6 address.
			TCPIP_IPADDR_ASSIGNMENT_LINKLOCAL_DOIP Linklocal IPv4/IPv6 address assignment using DoIP parameters.
			TCPIP_IPADDR_ASSIGNMENT_DHCP Dynamic configured IPv4/IPv6 address by DHCP.
			TCPIP_IPADDR_ASSIGNMENT_LINKLOCAL Linklocal IPv4/IPv6 address assignment.
			TCPIP_IPADDR_ASSIGNMENT_IPV6_ROUTER Dynamic configured IPv4/IPv6 address by Router Advertisement.
TcpIp_ReturnType	Enum	TcpIp specific return type.	TCPIP_OK Operation completed successfully.
			TCPIP_E_NOT_OK Operation failed.
			TCPIP_E_PHYS_ADDR_MISS Operation failed because of an ARP/NDP cache miss.
TcpIp_ParamIdType	uint8	Type for the specification of all supported Parameter IDs.	TCPIP_PARAMID_TCP_RXWND_MAX 0x00 Specifies the maximum TCP receive window for the socket.
			TCPIP_PARAMID_FRAMEPRIO (0x01) Specifies the frame priority for outgoing frames on the socket.
			TCPIP_PARAMID_TCP_NAGLE (0x02) Specifies if the Nagle Algorithm according to IETF RFC 896 is enabled or not.

Type Name	C-Type	Description	Value Range
			TCPIP_PARAMID_TCP_KEEPALIVE (0x03) Specifies if TCP Keep Alive Probes are sent on the socket connection.
			TCPIP_PARAMID_TTL (0x04) Specifies the time to live value for outgoing frames on the socket. For IPv6 this parameter specifies the value of the HopLimit field used in the IPv6 header.
			TCPIP_PARAMID_TCP_KEEPALIVE_TIME (0x05) Specifies the time in [s] between the last data packet sent (simple ACKs are not considered data) and the first keepalive probe.
			TCPIP_PARAMID_TCP_KEEPALIVE_PROBES_MAX (0x06) Specifies the maximum number of times that a keepalive probe is retransmitted.
			TCPIP_PARAMID_TCP_KEEPALIVE_INTERVAL (0x07) Specifies the interval in [s] between subsequent keepalive probes.
			TCPIP_PARAMID_VENDOR_SPECIFIC (0x80) Start of vendor specific range of parameter IDs.
TcpIpIpAddrWildcardType	uint32	IP address wildcard.	TCPIP_IPADDR_ANY defines the value used as wildcard
TcpIpIp6AddrWildcardType	uint32	IP6 address wildcard.	TCPIP_IP6ADDR_ANY defines the value used as wildcard for all IP6 address parts
TcpIpPortWildcardType	uint16	Port wildcard.	TCPIP_PORT_ANY defines the value used as wildcard
TcpIpLocalAddrIdWildcardType	uint8	LocalAddrId wildcard.	TCPIP_LOCALADDRID_ANY defines the value used as wildcard
TcpIp_ArpCacheEntryType	struct	Represents an entry inside the	TCPIP_(NDP|ARP)_ENTRY_STATIC (0x00) defines the value used for static NDP/ARP cache entries used in

Type Name	C-Type	Description	Value Range
TcpIp_NdpCacheEntryType		ARP/NDP cache	TcpIp_Get(Ndp|Arp)CacheEntries() API
			`TCPIP_(NDP|ARP)_ENTRY_VALID (0x01)` defines the value used for valid NDP/ARP cache entries used in TcpIp_Get(Ndp|Arp)CacheEntries() API
			`TCPIP_(NDP|ARP)_ENTRY_STALE (0x02)` defines the value used for stale NDP/ARP cache entries used in TcpIp_Get(Ndp|Arp)CacheEntries() API

Table 5-1 Type definitions

TcpIp_SockAddrType

Generic structure used by APIs to specify an IP address. (A specific address type can be derived from this structure via a cast to the specific struct type.)

Struct Element Name	C-Type	Description
domain	TcpIp_DomainType (uint16)	This is the code for the address format of this address

Table 5-2 TcpIp_SockAddrType

TcpIp_SockAddrInetType

This structure defines an IPv4 address type which can be derived from the generic address structure via cast.

Struct Element Name	C-Type	Description
domain	TcpIp_DomainType (uint16)	This is the code for the address format of this address
port	uint16	port number
addr	uint32[1]	IPv4 address in network byte order

Table 5-3 TcpIp_SockAddrInetType

TcpIp_SockAddrInet6Type

This structure defines an IPv6 address type which can be derived from the generic address structure via cast.

Struct Element Name	C-Type	Description
domain	TcpIp_DomainType (uint16)	This is the code for the address format of this address
port	uint16	port number
addr	uint32[4]	IPv6 address in network byte order

Table 5-4 TcpIp_SockAddrInet6Type

5.2 Services provided by TCPIP

5.2.1 TcpIp_InitMemory

Prototype	
void **TcpIp_InitMemory** (void)	
Parameter	
void [in]	none
Return code	
void	void
Functional Description	
Initialize the TcpIp-internal global module state variable.	
Particularities and Limitations	
Call context	
> initialization	

Table 5-5 TcpIp_InitMemory

5.2.2 TcpIp_Init

Prototype	
void **TcpIp_Init** (const TcpIp_ConfigType *ConfigPtr)	
Parameter	
ConfigPtr [in]	pointer to module configuration
Return code	
void	void
Functional Description	
Initialize the TcpIp module. Calls all TcpIp internal init functions and sets state to 'initialized'.	
Particularities and Limitations	
No configuration variant support implemented, so only TCPIP_CONFIG_VARIANT_1 (compile time) is available.	
Call context	
> initialization	

Table 5-6 TcpIp_Init

5.2.3 TcpIp_Shutdown

Prototype	
`void TcpIp_Shutdown (void)`	
Parameter	
void [in]	none
Return code	
void	void
Functional Description	
Shutdown the module TcpIp.	
Particularities and Limitations	
Call context	
> task level	

Table 5-7 TcpIp_Shutdown

5.2.4 TcpIp_GetSocketForUser

Prototype	
`Std_ReturnType TcpIp_GetSocketForUser (TcpIp_DomainType Domain, TcpIp_ProtocolType Protocol, TcpIp_SocketIdType *SocketIdPtr, uint8 UserIndex)`	
Parameter	
Domain [in]	IP address family (IPv4 or IPv6)
Protocol [in]	Socket protocol as sub-family of parameter type (TCP or UDP)
out [in]	SocketIdPtr Pointer to socket identifier representing the requested socket. This socket identifier must be provided for all further API calls which requires a SocketId.
UserIndex [in]	Identifier for the socket user (e.g. SoAd or HTTP)
Return code	
Std_ReturnType	> E_OK The request has been accepted > E_NOT_OK The request has not been accepted: no free socket
Functional Description	
Request a socket (TCP or UDP). By this API service the TCP/IP stack is requested to allocate a new socket.	
Particularities and Limitations	
SocketIdPtr is only valid if return value is E_OK. Each accepted incoming TCP connection also allocates a socket resource. Server sockets are special because they do not need any rx and tx buffer and a lot less socket description parameters.	
Call context	

> task level

Table 5-8 TcpIp_GetSocketForUser

5.2.5 TcpIp_TcpListen

Prototype	
`Std_ReturnType` **`TcpIp_TcpListen`** `(TcpIp_SocketIdType SocketId, uint16 MaxChannels)`	
Parameter	
SocketId [in]	Socket identifier of the related local socket resource.
MaxChannels [in]	Maximum number of new parallel connections established on this listen connection.
Return code	
Std_ReturnType	> E_OK The request has been accepted > E_NOT_OK The request has not been accepted
Functional Description	
Set TCP socket into listen mode. By this API service the TCP/IP stack is requested to listen on the TCP socket specified by the socket identifier.	
Particularities and Limitations	
Call context	
> task level	

Table 5-9 TcpIp_TcpListen

5.2.6 TcpIp_TcpConnect

Prototype	
`Std_ReturnType` **`TcpIp_TcpConnect`** `(TcpIp_SocketIdType SocketId, const TcpIp_SockAddrType *RemoteAddrPtr)`	
Parameter	
SocketId [in]	Socket identifier of the related local socket resource
RemoteAddrPtr [in]	IP address and port of the remote host to connect to
Return code	
Std_ReturnType	> E_OK Connect could be triggered > E_NOT_OK Connect could be triggered
Functional Description	
By this API service the TCP/IP stack is requested to establish a TCP connection to the configured peer.	

Particularities and Limitations
Call context
> task level

Table 5-10 TcpIp_TcpConnect

5.2.7 TcpIp_TcpTransmit

Prototype	
`Std_ReturnType` **`TcpIp_TcpTransmit`** `(TcpIp_SocketIdType SocketId, uint8 *DataPtr, uint32 AvailableLength, boolean ForceRetrieve)`	
Parameter	
SocketId [in]	socket index
DataPtr [in]	Pointer to a linear buffer of AvailableLength bytes containing the data to be transmitted. In case DataPtr is a NULL_PTR, TcpIp shall retrieve data from SoAd via callback SoAd_CopyTxData().
AvailableLength [in]	Available data for transmission in bytes
ForceRetrieve [in]	This parameter is only valid if DataPtr is a NULL_PTR. Indicates how the TCP/IP stack retrieves data from SoAd if DataPtr is a NULL_PTR. TRUE: the whole data indicated by availableLength shall be retrieved from the upper layer via one or multiple SoAd_CopyTxData() calls within the context of this transmit function. FALSE: The TCP/IP stack may retrieve up to availableLength data from the upper layer. It is allowed to retrieve less than availableLength bytes. Note: Not retrieved data will be provided by SoAd with the next call to TcpIp_TcpTransmit (along with new data if available).
Return code	
Std_ReturnType	> E_OK The request has been accepted > E_NOT_OK The request has not been accepted
Functional Description	
Transmit TCP message This service requests transmission of data via TCP to a remote node. The transmission of the data is decoupled.	
Particularities and Limitations	
The TCP segment(s) are sent dependent on runtime factors (e.g. receive window) and configuration parameter (e.g. Nagle algorithm).	
Call context	
> task level	

Table 5-11 TcpIp_TcpTransmit

5.2.8 TcpIp_UdpTransmit

Prototype	
`TcpIp_ReturnType` **`TcpIp_UdpTransmit`** `(TcpIp_SocketIdType SocketId, uint8 *DataPtr, TcpIp_SockAddrType *RemoteAddrPtr, uint16 TotalLength)`	
Parameter	
SocketId [in]	socket index
DataPtr [in]	Pointer to a linear buffer of TotalLength bytes containing the data to be transmitted. In case DataPtr is a NULL_PTR, TcpIp shall retrieve data from SoAd via callback SoAd_CopyTxData().
RemoteAddrPtr [in]	IP address and port of the remote host to transmit to.
TotalLength [in]	indicates the payload size of the UDP datagram.
Return code	
TcpIp_ReturnType	> TCPIP_OK UDP message has been forwarded to EthIf for transmission. > TCPIP_E_NOT_OK UDP message could not be sent because of a permanent error, e.g. message is too long. > TCPIP_E_PHYS_ADDR_MISS UDP message could not be sent because of an ARP cache miss, ARP request has been sent and upper layer may retry transmission by calling this function later again.
Functional Description	
Transmit UDP message This service transmits data via UDP to a remote node. The transmission of the data is immediately performed with this function call by forwarding it to EthIf.	
Particularities and Limitations	
This function will return TCPIP_E_PHYS_ADDR_MISS in case the physical address could not be resolved or no Ethernet TxBuffer was available. Differing between 'phy addr resolution is missing' and 'no buffer available' does not imply any useful information for the user. This function may not be called reentrant from one socket owner.	
Call context	
> task level	

Table 5-12 TcpIp_UdpTransmit

5.2.9 TcpIp_IcmpV6Transmit

Prototype	
`Std_ReturnType` **`TcpIp_IcmpV6Transmit`**`(TcpIp_LocalAddrIdType LocalIpAddrId, const TcpIp_SockAddrType *RemoteAddrPtr, uint8 HopLimit, uint8 Type, uint8 Code, uint16 DataLength, const uint8 *DataPtr)`	
Parameter	
LocalIpAddrId [in]	Local address identifier. (must reference an IPv6 address)
RemoteAddrPtr [in]	Destination address. (must be an IPv6 address)
HopLimit [in]	HopLimit value of the outgoing IPv6 packet.
Type [in]	Value of the Type field in the ICMPv6 message header.
Code [in]	Value of the Code field in the ICMPv6 message header.
DataLength [in]	Length of the message payload.

DataPtr [in]	Message payload.
Return code	
TcpIp_ReturnType	> E_OK Message was transmitted. > E_NOT_OK Message was not transmitted due to pending link-layer address resolution.
Functional Description	
Sends a raw ICMPv6 message of specified Type and Code.	
Particularities and Limitations	
This function may also be used to send ICMPv6 Echo Requests. Replies to send ICMPv6 messages can be received via the <Up>_IcmpMsgHandler callback.	
Call context	
> task level	

Table 5-13 TcpIp_IcmpV6Transmit

5.2.10 TcpIp_ProvideTxBuffer

Prototype	
`BufReq_ReturnType` **`TcpIp_ProvideTxBuffer`** `(TcpIp_SockHndType SockHnd, const TcpIp_SockAddrType *DestinationPtr, TcpIp_PbufType **PbufPtr, uint32 Length)`	
Parameter	
SockHnd [in]	socket handle
Destination [in]	destination network address and port
PbufPtr [out]	pointer for the data
Length [in]	data length in bytes
Return code	
BufReq_ReturnType	> BUFREQ_OK buffer provided > BUFREQ_E_NOT_OK provide buffer failed
Functional Description	
Request a tx buffer for sending data.	
Particularities and Limitations	
This API is deprecated and will be removed in future revisions of this module. This is the old API according to AUTOSAR 4.0.3, do not use it any longer.	
Call context	
> task level	

Table 5-14 TcpIp_ProvideTxBuffer

5.2.11 TcpIp_TransmitTo

Prototype	
`Std_ReturnType` **`TcpIp_TransmitTo`** `(TcpIp_SockHndType SockHnd, const TcpIp_PbufType *PbufPtr, const TcpIp_SockAddrType *DestinationPtr)`	
Parameter	
SockHnd [in]	socket handle
PbufPtr [out]	pointer for the data
Destination [in]	destination network address and port
Return code	
Std_ReturnType	> E_OK data transmit requested > E_NOT_OK data transmission request failed
Functional Description	
Trigger sending of data.	
Particularities and Limitations	
This API is deprecated and will be removed in future revisions of this module. This is the old API according to AUTOSAR 4.0.3, do not use it any longer.	
Call context	
> task level	

Table 5-15 TcpIp_TransmitTo

5.2.12 TcpIp_TcpReceived

Prototype	
`Std_ReturnType` **`TcpIp_TcpReceived`** `(TcpIp_SocketIdType SocketId, uint32 Length)`	
Parameter	
SocketId [in]	Socket identifier of the related local socket resource
Length [in]	Number of bytes finally consumed by the upper layer
Return code	
Std_ReturnType	> E_OK The request has been accepted > E_NOT_OK The request has not been accepted
Functional Description	
By this API service the reception of socket data is confirmed to the TCP/IP stack.	
Particularities and Limitations	
Increase the TCP receive window of the socket specified by SocketId considering the number of finally consumed bytes specified by Length.	
Call context	
> interrupt or task level	

Table 5-16 TcpIp_TcpReceived

5.2.13 TcpIp_Close

Prototype	
`Std_ReturnType TcpIp_Close (TcpIp_SocketIdType SocketId, boolean Abort)`	
Parameter	
SocketId [in]	Socket handle identifying the local socket resource.
Abort [in]	TRUE: connection will immediately be terminated by sending a RST-Segment and releasing all related resources. FALSE: connection will be terminated after performing a regular connection termination handshake and releasing all related resources.
Return code	
Std_ReturnType	> E_OK The request has been accepted. > E_NOT_OK The request has not been accepted.
Functional Description	
Close the socket. By this API service the TCP/IP stack is requested to close the socket and release all related resources. The service TcpIp_Close() shall perform the following actions for the socket specified by SocketId in case it is a TCP socket: (a) if the connection is active and (a1) abort = FALSE: the connection shall be terminated after performing a regular connection termination handshake and releasing all related resources. (a2) abort = TRUE: connection shall immediately be terminated by sending a RST-Segment and releasing all related resources. (b) if the socket is in the Listen state, the Listen state shall be left immediately and related resources shall be released.	
Particularities and Limitations	
Call context	
> task level	

Table 5-17 TcpIp_Close

5.2.14 TcpIp_Bind

Prototype	
`Std_ReturnType TcpIp_Bind (TcpIp_SocketIdType SocketId, TcpIp_LocalAddrIdType LocalAddrId, uint16 *PortPtr)`	
Parameter	
SocketId [in]	Socket identifier of the related local socket resource.
LocalAddrId [in]	IP address identifier representing the local IP address and EthIf controller to bind the socket to.
inout] [in]	PortPtr Pointer to memory where the local port to which the socket shall be bound is specified. In case the parameter is specified as TCPIP_PORT_ANY,

	the TCP/IP stack shall choose the local port automatically from the range 49152 to 65535 and shall update the parameter to the chosen value.
Return code	
Std_ReturnType	> E_OK The request has been accepted > E_NOT_OK The request has not been accepted (e.g. address in use)
Functional Description	
By this API service the TCP/IP stack is requested to bind a UDP or TCP socket to a local resource.	
Particularities and Limitations	
Call context	
> task level	

Table 5-18 TcpIp_Bind

5.2.15 TcpIp_RequestIpAddrAssignment

Prototype	
`Std_ReturnType` **`TcpIp_RequestIpAddrAssignment`** `(TcpIp_LocalAddrIdType LocalAddrId, TcpIp_IpAddrAssignmentType Type, TcpIp_SockAddrType *LocalIpAddrPtr, uint8 Netmask, TcpIp_SockAddrType *DefaultRouterPtr)`	
Parameter	
LocalAddrId [in]	IP address index specifying the IP address for which an assignment shall be initiated.
Type [in]	type of IP address assignment which shall be initiated
LocalIpAddrPtr [in]	pointer to structure containing the IP address which shall be assigned to the EthIf controller indirectly specified via LocalAddrId. Note: This parameter is only used in case the parameters Type is set to TCPIP_IPADDR_ASSIGNMENT_STATIC.
Netmask [in]	Network mask of IPv4 address or address prefix of IPv6 address in CIDR Notation.
DefaultRouterPtr [in]	Pointer to struct where the IP address of the default router (gateway) is stored (struct member 'port' is not used and of arbitrary value) (IPv4 only)
Return code	
Std_ReturnType	> E_OK The request has been accepted > E_NOT_OK The request has not been accepted
Functional Description	
Initiate the local IP address assignment for the IP address specified by LocalAddrId.	
Particularities and Limitations	
Call context	
> task level	

Table 5-19 TcpIp_RequestIpAddrAssignment

5.2.16 TcpIp_ReleaseIpAddrAssignment

Prototype	
Std_ReturnType **TcpIp_ReleaseIpAddrAssignment** (TcpIp_LocalAddrIdType LocalAddrId)	
Parameter	
LocalAddrId [in]	IP address index specifying the IP address for which an assignment shall be released.
Return code	
Std_ReturnType	> E_OK The request has been accepted > E_NOT_OK The request has not been accepted
Functional Description	
Release the local IP address assignment for the IP address specified by LocalAddrId.	
Particularities and Limitations	
Call context	
> task level	

Table 5-20 TcpIp_ReleaseIpAddrAssignment

5.2.17 TcpIp_GetPhysAddr

Prototype	
Std_ReturnType **TcpIp_GetPhysAddr** (TcpIp_LocalAddrIdType LocalAddrId, uint8 *PhysAddrPtr)	
Parameter	
LocalAddrId [in]	Local address identifier implicitly specifying the EthIf controller for which the physical address shall be obtained.
PhysAddrPtr [out]	Pointer to the memory where the physical source address (MAC address) in network byte order is stored
Return code	
Std_ReturnType	> E_OK The request was successful > E_NOT_OK The request was not successful, e.g. no unique Ctrl specified via IpAddrId.
Functional Description	
Obtains the physical source address used by the EthIf controller implicitly specified via LocalAddrId.	
Particularities and Limitations	
Call context	
> task level	

Table 5-21 TcpIp_GetPhysAddr

5.2.18 TcpIp_GetRemotePhysAddr

Prototype	
`TcpIp_ReturnType` **`TcpIp_GetRemotePhysAddr`** `(uint8 CtrlIdx, TcpIp_SockAddrType *IpAddrPtr, uint8 *PhysAddrPtr, boolean initRes)`	
Parameter	
CtrlIdx [in]	EthIf controller index to identify the related ARP/NDP table.
IpAddrPtr [in]	specifies the IP address for which the physical address shall be retrieved
initRes [in]	specifies if the address resolution shall be initiated (TRUE) or not (FALSE) in case the physical address related to the specified IP address is currently unknown.
PhysAddrPtr [out]	Pointer to the memory where the physical address (MAC address) related to the specified IP address is stored in network byte order.
Return code	
TcpIp_ReturnType	> TCPIP_E_OK specified IP address resolved, physical address provided via PhysAddrPtr > TCPIP_E_PHYS_ADDR_MISS physical address currently unknown (address resolution initiated if initRes set to TRUE)
Functional Description	
Lookup the physical address for a remote network address. TcpIp_GetRemotePhysAddr queries the IP/physical address translation table specified by CtrlIdx and returns the physical address related to the IP address specified by IpAddrPtr. In case no physical address can be retrieved and parameter initRes is TRUE, address resolution for the specified IP address is initiated on the local network.	
Particularities and Limitations	
Call context	
> task level	

Table 5-22 TcpIp_GetRemotePhysAddr

5.2.19 TcpIp_GetCtrlIdx

Prototype	
`Std_ReturnType` **`TcpIp_GetCtrlIdx`** `(TcpIp_LocalAddrIdType LocalAddrId, uint8 *CtrlIdxPtr)`	
Parameter	
LocalAddrId [in]	Local address identifier implicitely specifing the EthIf controller that shall be returned
CtrlIdxPtr [out]	Pointer to the memory where the index of the EthIf controller related to LocalAddrId is stored

Return code	
Std_ReturnType	> E_OK The request was successful > E_NOT_OK The request was not successful
Functional Description	
TcpIp_GetCtrlIdx returns the index of the EthIf controller related to LocalAddrId.	
Particularities and Limitations	
Call context	
> task level	

Table 5-23 TcpIp_GetCtrlIdx

5.2.20 TcpIp_MainFunction

Prototype	
`void TcpIp_MainFunction (void)`	
Parameter	
void [in]	none
Return code	
void	void
Functional Description	
The TcpIp MainFunction handles the module internal state handling. Calls all TcpIp-internal MainFunction functions. This function has to be called cyclically.	
Particularities and Limitations	
Init has to be called before	
Call context	
> task level	

Table 5-24 TcpIp_MainFunction

5.2.21 TcpIp_ChangeParameter

Prototype	
`Std_ReturnType TcpIp_ChangeParameter (TcpIp_SocketIdType SocketId, TcpIp_ParamIdType ParameterId, uint8 *ParameterValue)`	
Parameter	
SockHnd [in]	socket handle
ParameterId [in]	parameter identification
ParameterValue [in]	parameter value

Return code	
Std_ReturnType	> E_OK parameter changed > E_NOT_OK parameter change failed
Functional Description	
change socket parameter configuration	
Particularities and Limitations	
Call context	
> task level	

Table 5-25 TcpIp_ChangeParameter

5.2.22 TcpIp_Tls_SetClientCertInfo

Prototype	
`Std_ReturnType` **`TcpIp_Tls_SetClientCertInfo`** `(TcpIp_SocketIdType SocketId, uint8 *CertPtr, uint8 *KeyPtr, uint16 CertLen, uint16 KeyLen)`	
Parameter	
SocketId [in]	socket handle
CertPtr [in]	pointer to client certificate (RAM block)
KeyPtr [in]	pointer to client private key (RAM block)
CertLen [in]	length of client certigicate
KeyLen [in]	length of client private key
Return code	
Std_ReturnType	> E_OK cert info could be set > E_NOT_OK cert info could NOT be set
Functional Description	
Set the client certificate and key block ids for a TLS connection (necessary for client authentication).	
Particularities and Limitations	
Call context	
> task level	

Table 5-26 TcpIp_Tls_SetClientCertInfo

5.2.23 TcpIp_Tls_GetNvmBlockIdForUsedRootCert

Prototype
`Std_ReturnType` **`TcpIp_Tls_GetNvmBlockIdForUsedRootCert`** `(TcpIp_SocketIdType SocketId, NvM_BlockIdType *RootCertBlockIdPtr)`

Parameter	
SocketId [in]	Socket id
RootCertBlockIdPtr [out]	Pointer to the block ID
Return code	
Std_ReturnType	> E_OK Block ID can be provided > E_NOT_OK request failed
Functional Description	
Get the NVM block ID of the root certificate used for a TLS connection.	
Particularities and Limitations	
Call context	
> task level	

Table 5-27 TcpIp_Tls_GetNvmBlockIdForUsedRootCert

5.2.24 TcpIp_Tls_RootCertWasModified

Prototype	
`Std_ReturnType` **`TcpIp_Tls_RootCertWasModified`** `(NvM_BlockIdType NvmBlockId)`	
Parameter	
NvmBlockId [in]	BlockId of the NVM block that has changed
Return code	
Std_ReturnType	> E_OK Block is used as a root certificate > E_NOT_OK request failed, block is not used as a root certificate
Functional Description	
Inform the TLS that a root cert has been updaten in NVM.	
Particularities and Limitations	
Call context	
> task level	

Table 5-28 TcpIp_Tls_RootCertWasModified

5.2.25 TcpIp_GetLocNetAddr

Prototype

`Std_ReturnType` **`TcpIp_GetLocNetAddr`** `(TcpIp_LocalAddrIdType LocalAddrId,`
`TcpIp_NetAddrType *NetAddrPtr)`

Parameter	
LocalAddrId [in]	local address identifier
NetAddrPtr [out]	pointer for the local network address
Return code	
Std_ReturnType	> E_OK local network address returned > E_NOT_OK local network address access failed
Functional Description	
This function returns the current IP address for a given controller.	
Particularities and Limitations	
This API is deprecated and will be removed in future revisions of this module.	
Call context	
> task level	

Table 5-29 TcpIp_GetLocNetAddr

5.2.26 TcpIp_GetLocNetMask

Prototype	
Std_ReturnType **TcpIp_GetLocNetMask** (TcpIp_LocalAddrIdType LocalAddrId, TcpIp_NetAddrType *NetMaskPtr)	
Parameter	
LocalAddrId [in]	local address identifier
NetMaskPtr [out]	pointer for the local network mask
Return code	
Std_ReturnType	> E_OK local network mask returned > E_NOT_OK local network mask access failed
Functional Description	
This function returns the current network mask for a given controller.	
Particularities and Limitations	
This API is deprecated and will be removed in future revisions of this module.	
Call context	
> task level	

Table 5-30 TcpIp_GetLocNetMask

5.2.27 TcpIp_GetIpAddr

Prototype
Std_ReturnType **TcpIp_GetIpAddr** (TcpIp_LocalAddrIdType LocalAddrId, TCPIP_P2V(TcpIp_SockAddrType) IpAddrPtr, TCPIP_P2V(uint8) NetmaskPtr,

`TCPIP_P2V(TcpIp_SockAddrType) DefaultRouterPtr)`	
Parameter	
LocalAddrId [in]	local address identifier
IpAddrPtr [out]	Pointer to a struct where the IP address is stored. (only struct members family and address will be updated)
NetmaskPtr [out]	Pointer to memory where Network mask of IPv4 address or address prefix of IPv6 address in CIDR notation is stored.
DefaultRouterPtr [out]	Pointer to struct where the IP address of the default router (gateway) is stored (only struct members family and address will be updated)
Return code	
Std_ReturnType	> E_OK The request was successful > E_NOT_OK The request was not successful.
Functional Description	
Obtains the local IP address actually used by LocalAddrId, the netmask and default router.	
Particularities and Limitations	
The output parameters IpAddrPtr, NetmaskPtr, DefaultRouterPtr may be set to NULL_PTR if the information is not required.	
Call context	
> task level	

Table 5-31 TcpIp_GetIpAddr

5.2.28 TcpIp_GetRemNetAddr

Prototype	
`Std_ReturnType` **`TcpIp_GetRemNetAddr`** `(TcpIp_SockHndType SockHnd, TCPIP_P2V(TcpIp_SockAddrType)*SockAddrPtr)`	
Parameter	
SockHnd [in]	socket handle
SockAddrPtr [out]	Pointer to a socket address element
Return code	
Std_ReturnType	> E_OK reading the socket address was successful > E_NOT_OK remote IP address could not be obtained
Functional Description	
The API provides the remote address of an established TCP socket connection.	
Particularities and Limitations	
SockAddrPtr is a pointer to a TcpIp memory area where the remote address is stored. The pointer is valid until the socket is closed.	
Call context	
> task level	

Table 5-32 TcpIp_GetRemNetAddr

5.2.29 TcpIp_GetVersionInfo

Prototype	
`void TcpIp_GetVersionInfo (Std_VersionInfoType *versioninfo)`	
Parameter	
versioninfo [in]	pointer for version information
Return code	
void	void
Functional Description	
Get Tcp/Ip Version.	
Particularities and Limitations	
The module version is given decimal coded.	
Call context	
> task level	

Table 5-33 TcpIp_GetVersionInfo

5.2.30 TcpIp_DhcpReadOption

Prototype	
`Std_ReturnType TcpIp_DhcpReadOption (TcpIp_LocalAddrIdType LocalIpAddrId, uint8 Option, uint8 *DataLength, uint8 *DataPtr)`	
Parameter	
LocalIpAddrId [in]	IP address identifier representing the local IP address and EthIf controller for which the DHCP option shall be read.
Option [in]	DHCP option, e.g. Host Name
DataLength [in]	length of DHCP option data
DataPtr [in,out]	As input parameter, contains the length of the provided data buffer. Will be overwritten with the length of the actual data.
Return code	
Std_ReturnType	> E_OK requested data retrieved successfully. > E_NOT_OK requested data could not be retrieved.
Functional Description	
Read user option data for outgoing DHCPv4 messages. By this API service the TCP/IP stack retrieves DHCP option data identified by parameter option for already received DHCP options.	
Particularities and Limitations	
Currently only the option 'TCPIP_DHCPV4_OPT_ID_CLIENT_FQDN' is supported.	
Call context	

> interrupt or task level

Table 5-34 TcpIp_DhcpReadOption

5.2.31 TcpIp_DhcpWriteOption

Prototype	
`Std_ReturnType` **`TcpIp_DhcpWriteOption`** `(TcpIp_LocalAddrIdType LocalIpAddrId, uint8 Option, uint8 DataLength, const uint8 *DataPtr)`	
Parameter	
LocalIpAddrId [in]	IP address identifier representing the local IP address and EthIf controller for which the DHCP option shall be written.
Option [in]	DHCP option, e.g. Host Name
DataLength [in]	length of DHCP option data
DataPtr [in]	Pointer to memory containing DHCP option data
Return code	
Std_ReturnType	> E_OK no error occured. > E_NOT_OK DHCP option data could not be written.
Functional Description	
Set user option data for outgoing DHCPv4 messages. By this API service the TCP/IP stack writes the DHCP option data identified by parameter option.	
Particularities and Limitations	
Old similar Vector API was TcpIp_SetDhcpV4TxOption. Currently only the option 'TCPIP_DHCPV4_OPT_ID_CLIENT_FQDN' is supported.	
Call context	
> interrupt or task level	

Table 5-35 TcpIp_DhcpWriteOption

5.2.32 TcpIp_DhcpV6ReadOption

Prototype	
`Std_ReturnType` **`TcpIp_DhcpV6ReadOption`** `(TcpIp_LocalAddrIdType LocalIpAddrId, uint16 Option, uint16 *DataLength, uint8 *DataPtr)`	
Parameter	
LocalIpAddrId [in]	IP address identifier representing the local IP address and EthIf controller for which the DHCP option shall be read.
Option [in]	DHCP option, e.g. Host Name
DataLength [in]	length of DHCP option data
DataPtr [in,out]	As input parameter, contains the length of the provided data buffer. Will be overwritten with the length of the actual data.

Return code	
Std_ReturnType	> E_OK requested data retrieved successfully. > E_NOT_OK requested data could not be retrieved.
Functional Description	
Read user option data for outgoing DHCPv6 messages. By this API service the TCP/IP stack retrieves DHCP option data identified by parameter option for already received DHCP options.	
Particularities and Limitations	
Currently only the option 'TCPIP_DHCPV6_OPT_ID_CLIENT_FQDN' is supported.	
Call context	
> interrupt or task level	

Table 5-36 TcpIp_DhcpV6ReadOption

5.2.33 TcpIp_DhcpV6WriteOption

Prototype	
`Std_ReturnType` **`TcpIp_DhcpV6WriteOption`** `(TcpIp_LocalAddrIdType LocalIpAddrId, uint16 Option, uint16 DataLength, const uint8 *DataPtr)`	
Parameter	
OptSelector [in]	Selector id of configured option. (Not the option id) Use configured access define.
OptLen [in]	Length of option data in bytes. Set to 0 to inhibit sending of the option.
OptPtr [in]	Payload of the option. Pointer must be valid until it is updated by calling this function. (Ignored if OptLen is 0)
MsgFlags [in]	Defines in which messages the option shall be sent. One or more flags of TCPIP_DHCPV6_MSG_FLAG_*
Return code	
Std_ReturnType	> E_OK Success > E_NOT_OK Invalid OptSelector
Functional Description	
Set user option data for outgoing DHCPv6 messages.	
Particularities and Limitations	
Old similar Vector API was TcpIp_SetDhcpV6TxOption. Currently only the option 'TCPIP_DHCPV6_OPT_ID_CLIENT_FQDN' is supported.	
Call context	
> interrupt or task level	

Table 5-37 TcpIp_DhcpV6WriteOption

5.2.34 TcpIp_GetDhcpTimeoutInfo

Prototype	
`boolean TcpIp_GetDhcpTimeoutInfo (uint8 CtrlIdx, uint8 IpVer)`	
Parameter	
CtrlIdx [in]	controller index
IpVer [in]	IP version
Return code	
boolean	> TRUE there was a DHCP timeout > FALSE there was no DHCP timeout
Functional Description	
check if a DHCP address finding timeout occurred	
Particularities and Limitations	
Call context	
> interrupt or task level	

Table 5-38 TcpIp_GetDhcpTimeoutInfo

5.2.35 TcpIp_DhcpV4_GetStatus

Prototype	
`Std_ReturnType TcpIp_DhcpV4_GetStatus( IpV4_AddrIdType IpAddrId, uint8* DhcpStatePtr )`	
Parameter	
IpAddrId [in]	local address identifier
DhcpStatePtr [out]	current, simplified, DHCP status of the IP address
Return code	
Std_ReturnType	> E_OK the DhcpState is valid > E_NOT_OK given address is invalid or has no DHCP configured or the IP controller of the given address is not in state online active,
Functional Description	
Returns the current DCHP state of a given IP address. The following simplified DHCP states are supported: TCPIP_DHCP_SIMPLIFIED_STATE_NOT_ACTIVE: DHCP client not active. TCPIP_DHCP_SIMPLIFIED_STATE_NO_DISCVR_SENT: No discover sent. TCPIP_DHCP_SIMPLIFIED_STATE_DISCVR_SENT_NO_ANSWR_RCVD_YET: Discover sent but no answer received. TCPIP_DHCP_SIMPLIFIED_STATE_OFFER_RCVD_WITHOUT_REQ: DHCP offer received, no request sent. TCPIP_DHCP_SIMPLIFIED_STATE_REQ_SENT_NO_ACK_RCVD_YET: DHCP request sent, no ACK	

received. TCPIP_DHCP_SIMPLIFIED_STATE_ACK_RCVD_DHCP_ADDR_ASSIGND: DHCP address assigned to current interface.
Particularities and Limitations
This is a Vector extension.
Call context
> task level

Table 5-39 TcpIp_DhcpV4_GetStatus

5.2.36 TcpIp_GetIpAddrCfgSrc

Prototype	
`Std_ReturnType` **`TcpIp_GetIpAddrCfgSrc`** `(TcpIp_LocalAddrIdType LocalAddrId, uint8 *CfgSrcPtr)`	
Parameter	
LocalAddrId [in]	local address identifier
AddrPtr [in]	address for which the configuration source shall be returned
CfgSrcPtr [out]	configuration source of the address
Return code	
Std_ReturnType	> E_OK address is valid and *CfgSrcPtr has be set > E_NOT_OK address invalid or not assigned to this node. *CfgSrcPtr not modified.
Functional Description	
Get the configuration source of an IP address.	
Particularities and Limitations	
Call context	
> task level	

Table 5-40 TcpIp_GetIpAddrCfgSrc

5.2.37 TcpIp_RequestComMode

Prototype	
`Std_ReturnType` **`TcpIp_RequestComMode`** `(uint8 CtrlIdx, TcpIp_StateType State)`	
Parameter	
CtrlIdx [in]	EthIf controller index to identify the communication network where the TcpIp state is requested
State [in]	Requested TcpIp state

Return code	
Std_ReturnType	> E_OK request was accepted > E_NOT_OK request was not accepted
Functional Description	
By this API service the TCP/IP stack is requested to change the TcpIp state of the communication network identified by EthIf controller index.	
Particularities and Limitations	
Call context	
> task level	

Table 5-41 TcpIp_RequestComMode

5.2.38 TcpIp_GetLocSockAddr

Prototype	
Std_ReturnType **TcpIp_GetLocSockAddr** (TcpIp_SockHndType SockHnd, TcpIp_SockAddrType *SockPtr)	
Parameter	
SockHnd [in]	socket handle
SockPtr [out]	pointer were the local socket address shall be written to
Return code	
Std_ReturnType	> E_OK socket address was written successfully > E_NOT_OK error, socket address is not updated
Functional Description	
Read the local socket address of one socket.	
Particularities and Limitations	
: This is a Vector extension	
Call context	
> task level	

Table 5-42 TcpIp_GetLocSockAddr

5.2.39 TcpIp_DiagDataReadAccess

Prototype	
Std_ReturnType **TcpIp_DiagDataReadAccess** (TcpIp_SockHndType SockHnd, TcpIp_DiagParamsType DataID, uint8 *DataPtr, uint16 *DataLenPtr)	
Parameter	
SockHnd [in]	socket handle

DataID [in]	data identifier
DataPtr [out]	pointer for diagnostic data
DataLenPtr [in,out]	pointer for maximum / actual length of diagnostic data in bytes
Return code	
Std_ReturnType	> E_OK diagnostic data written > E_NOT_OK invalid parameter (data identifier not found, NULL_PTR parameter, invalid length)
Functional Description	
diagnostic data read access	
Particularities and Limitations	
The memory fragment DataPtr points to should be aligned properly regarding the expected returned type / struct. Data is only written if RetValue is E_OK. This is a Vector extension.	
Call context	
> task level	

Table 5-43 TcpIp_DiagDataReadAccess

5.2.40 TcpIp_ClearARCache

Prototype	
`Std_ReturnType` **`TcpIp_ClearARCache`** `(TcpIp_LocalAddrIdType LocalAddrId)`	
Parameter	
LocalAddrId [in]	Local address identifier implicitely specifing the IPv4/IPv6 controller that shall be cleared (currently only IPv6/NDP is supported)
Return code	
Std_ReturnType	> E_OK The request was successful > E_NOT_OK The request was not successful
Functional Description	
TcpIp_ClearARCache clears the address resolution cache (currently only IPv6/NDP is supported)	
Particularities and Limitations	
Currently only IPv6/NDP is supported	
Call context	
> task level	

Table 5-44 TcpIp_ClearARCache

5.2.41 TcpIp_GetArpCacheEntries

Prototype
`Std_ReturnType` **`TcpIp_GetArpCacheEntries`** `(uint8 ctrlIdx, uint32 *numberOfElements, TcpIp_ArpCacheEntryType *entryListPtr)`

Parameter	
ctrlIdx [in]	EthIf controller index to identify the related ARP table.
numberOfElements[in,out]	In: Maximum number of entries that can be stored in the output buffer entryListPtr. Out: Number of entries that are written into the output buffer entryListPtr (Total number of all entries in the cache if input value is 0).
entryListPtr[out]	Pointer to memory where the list of cache entries shall be stored. if *numberOfElements is 0 , the output may be NULL_PTR
Return code	
Std_ReturnType	> E_OK physical address cache could be read. > E_NOT_OK physical address cache could not be read. (i.e. no IPv4 instance active on this controller)
Functional Description	
Returns entries that are currently stored in the IPv4 address resolution cache. Copies entries from the physical address cache of the IPv4 instance that is active on the EthIf controller specified by ctrlIdx into a user provided buffer. The function will copy all or numberOfElements into the output list. If input value of numberOfElements is 0 the function will not copy any data but only return the number of entries in the cache. entryListPtr may be NULL_PTR in this case.	
Particularities and Limitations	
Call context	
> interrupt or task level	

Table 5-45 TcpIp_GetArpCacheEntries

5.2.42 TcpIp_GetNdpCacheEntries

Prototype	
`Std_ReturnType TcpIp_GetNdpCacheEntries (uint8 ctrlIdx, uint32 *numberOfElements, TcpIp_NdpCacheEntryType *entryListPtr)`	
Parameter	
ctrlIdx [in]	EthIf controller index to identify the related NDP table.
numberOfElements[in,out]	In: Maximum number of entries that can be stored in Parameters output entryListPtr. Out: Number of entries written to output entryListPtr (Number of all entries in the cache if input value is 0).
entryListPtr[out]	Pointer to memory where the list of cache entries shall be stored. May only be NULL_PTR if *numberOfElements is 0.
Return code	
Std_ReturnType	> E_OK physical address cache could be read. > E_NOT_OK physical address cache could not be read. (i.e. no IPv6 instance active on this controller)

Functional Description
Returns entries that are currently stored in the IPv6 link layer address resolution cache. Copies entries from the physical address cache of the IPv6 instance that is active on the EthIf controller specified by ctrlIdx into a user provided buffer. The function will copy all or numberOfElements into the output list. If input value of numberOfElements is 0 the function will not copy any data but only return the number of entries in the cache. EntryListPtr may be NULL_PTR in this case.
Particularities and Limitations
Call context
> interrupt or task level

Table 5-46 TcpIp_GetNdpCacheEntries

5.2.43 IpV4_Arp_SendGratuitousArpReq

Prototype	
`Std_ReturnType` **`IpV4_Arp_SendGratuitousArpReq`**`(IpV4_AddrIdType IpAddrId)`	
Parameter	
IpAddrId [in]	Local address identifier implicitely specifing the EthIf controller on which the gratuitous shall be sent.
Return code	
Std_ReturnType	> E_OK the gratuitous ARP request was sent using the related lower layer APIs. > E_NOT_OK otherwise.
Functional Description	
Sends an gratuitous ARP request packet as specified in [RFC2002 4.6. second indentation]	
Particularities and Limitations	
This is a Vector extension.	
Call context	
> task level	

Table 5-47 IpV4_Arp_SendGratuitousArpReq

5.2.44 IpV4_GetLastDuplicateDhcpAddrDid

Prototype	
`FUNC(Std_ReturnType, IPV4_CODE) IpV4_`**`GetLastDuplicateDhcpAddrDid`**`(uint8 IpCtrlIdx, CONSTP2VAR(IpBase_AddrInType, AUTOMATIC, IPV4_APPL_DATA) IpAddrPtr, CONSTP2VAR(IpBase_EthPhysAddrType, AUTOMATIC, IPV4_APPL_DATA) PhysAddrPtr)`	
Parameter	
IpCtrlIdx [in]	Ip controller index

IpAddrPtr [out]	Specifies the memory where the IP address shall be stored for which a duplicated DHCP address assignment has been detected.
PhysAddrPtr [out]	Specifies the memory where the physical address shall be stored for which a duplicated DHCP address assignment has been detected.
Return code	
Std_ReturnType	> E_OK the given IpCtrlIdx was valid and the DID information was stored in the given data structures. > E_NOT_OK otherwise.
Functional Description	
Returns the DID for DEM event TCPIP_E_DUPLICATE_DHCP_ADDR	
Particularities and Limitations	
This is a Vector extension.	
Call context	
> task level	

Table 5-48 IpV4_Arp_SendGratuitousArpReq

5.3 Services used by TCPIP

In the following table services provided by other components, which are used by the TCPIP are listed. For details about prototype and functionality refer to the documentation of the providing component.

Component	API
DET	Det_ReportError
EthIf	EthIf_ProvideTxBuffer
EthIf	EthIf_Transmit
EthIf	EthIf_GetPhysAddr
IpBase	IpBase_Copy
IpBase	IpBase_Fill
IpBase	IpBase_TcpIpChecksumAdd
IpBase	IpBase_DelSockAddr
IpBase	IpBase_CopySockAddr
IpBase	IpBase_SockPortIsEqual

Table 5-49 Services used by the TCPIP

5.4 Callback Functions

This chapter describes the callback functions that are implemented by the TCPIP and can be invoked by other modules. The prototypes of the callback functions are provided in the header file `TcpIp_Cbk.h` by the TCPIP.

5.4.1 TcpIp_RxIndication

Prototype	
`void **TcpIp_RxIndication** (uint8 CtrlIdx, Eth_FrameType FrameType, boolean IsBroadcast, uint8 *PhysAddrPtr, uint8 *DataPtr, uint16 LenByte)`	
Parameter	
CtrlIdx [in]	Index of the EthIf controller.
FrameType [in]	frame type of received Ethernet frame
IsBroadcast [in]	parameter to indicate a broadcast frame
PhysAddrPtr [in]	pointer to Physical source address (MAC address in network byte order) of received Ethernet frame
DataPtr [in]	Pointer to payload of the received Ethernet frame (i.e. Ethernet header is not provided).
LenByte [in]	Length of received data.
Return code	
void	void
Functional Description	
RxIndication from the EthIf. By this API service the TCP/IP stack gets an indication and the data of a	

received frame.
Particularities and Limitations
This function will simply forward the incoming RxIndication to the corresponding RxIndications in the modules IpV4 and IpV6.
Call context
> interrupt or task level

Table 5-50 TcpIp_RxIndication

5.5 Configurable Interfaces

5.5.1 Notifications and Callouts

At its configurable interfaces the TCPIP defines notifications that can be mapped to callback functions provided by other modules. The mapping is not statically defined by the TCPIP but can be performed at configuration time. The function prototypes that can be used for the configuration have to match the appropriate function prototype signatures, which are described in the following sub-chapters.

5.5.1.1 IPv6 Maximum Payload Length Change Callback

Prototype	
`void` **`<Configurable_Cbk_Name>`**`(uint8 CtrlIdx, const IpBase_SockAddrType *DstAddrPtr, uint16 Mtu)`	
Parameter	
`CtrlIdx`	EthIf controller index
`DstAddrPtr`	IPv6 address of the destination
`Mtu`	Maximum Transmission Unit (MTU) for the destination. This is the maximum payload length of the IPv6 packet in bytes. If `IpV6_ProvideTxBuffer()` is called with a `BufLen` larger than `Mtu` the packet will be automatically fragmented by the IpV6 if possible.
Return code	
`void`	-
Functional Description	
These callbacks will be called if an ICMPv6 Packet Too Big (Type 2) message has been received by the IpV6 and the message indicates that the MTU value is smaller than the currently used MTU value for a specific destination address.	
Particularities and Limitations	
> These callbacks will be called only if the PathMTU feature is enabled in the IpV6 configuration.	
Call context	
> interrupt or task context	

Table 5-51 Maximum Payload Length Change Callback

5.5.1.2 IPv6 Inverse Neighbor Discovery Address List Option Receive Callback

Prototype	
`void` **`<Configurable_Cbk_Name>`**`(uint8 CtrlIdx, Eth_PhysAddrType *RemoteLLAddrPtr, IpV6_AddrType *AddrListPtr, uint8 AddrCount)`	
Parameter	
`CtrlIdx`	EthIf controller index
`RemoteLLAddrPtr`	Link-layer (physical) address of the remote node that send the address list.
`AddrListPtr`	List of IPv6 addresses of the remote node
`AddrCount`	Number of IPv6 addresses in the address list
Return code	
`void`	-
Functional Description	
These Callbacks will be called when an Inverse Neighbor Discovery Solicitation or Advertisement with an Address List Option has been received (see [RFC4443 3.2. Packet Too Big Message).	
Particularities and Limitations	
> These callbacks will be called only if the “Inverse Solicitations“ or “Inverse Advertisements” are enabled in the IpV6 configuration. > The callback will be called with an empty address list (`AddrListPtr == NULL_PTR && AddrCount == 0`) if there was no response to a user invoked Inverse Solicitation (see `IpV6_Ndp_SendInverseSolicitation()`).	
Call context	
> interrupt or task context	

Table 5-52 Inverse Neighbor Discovery Address List Option Receive Callback

5.5.1.3 ICMPv4 Message Handler

Prototype	
`void` **`<Configurable_Cbk_Name>`**`(TcpIp_LocalAddrIdType LocalAddrId, const TcpIp_SockAddrType *RemoteAddrPtr, uint8 Ttl, uint8 Type, uint8 Code, uint16 DataLength, uint8 *DataPtr)`	
Parameter	
`LocalAddrId [in]`	Local address identifier representing the local IP address and EthIf controller where the ICMP message has been received
`RemoteAddrPtr [in]`	Pointer to struct representing the address of the ICMP sender.
`Ttl [in]`	Time to live value of the received ICMPv4 message.
`Type [in]`	Type field value of the received ICMP message (Note: the value of the type field determines the format of the remaining ICMP message data)
`Code [in]`	Code field value of the received ICMP message.
`DataLength [in]`	Length of ICMP message.
`DataPtr [in]`	Pointer to the received ICMP message.

Return code	
void	-
Functional Description	
By this API service the configured ICMP message handler function is called by the TCP/IP stack on reception of an ICMP message which is not handled by the TCP/IP stack.	
Particularities and Limitations	
> none	
Call context	
> interrupt or task context	

Table 5-53 ICMP Destination Unreachable Message Callback

5.5.1.4 ICMPv6 Message Handler

Prototype	
void **<Configurable_Cbk_Name>**(TcpIp_LocalAddrIdType LocalAddrId, const TcpIp_SockAddrType *RemoteAddrPtr, uint8 Ttl, uint8 Type, uint8 Code, uint16 DataLength, uint8 *DataPtr, uint16 MultiPartDataLength, uint8 *MultiPartDataPtr)	
Parameter	
LocalAddrId [in]	Local address identifier representing the local IP address and EthIf controller where the ICMPv6 message has been received
RemoteAddrPtr [in]	Pointer to struct representing the address of the ICMPv6 sender.
Ttl [in]	IP Hop Limit value of the received ICMPv6 message.
Type [in]	Type field value of the received ICMPv6 message (Note: the value of the type field determines the format of the remaining ICMPv6 message data)
Code [in]	Code field value of the received ICMPv6 message.
DataLength [in]	Length of ICMPv6 message.
DataPtr [in]	Pointer to the received ICMPv6 message.
MultiPartDataLength [in]	Length of multi-part data in bytes.
MultiPartDataPtr [in]	ICMPv6 multi-part data (see [RFC4884 8. ICMP Extension Objects]).
Return code	
void	-

Functional Description
By this API service the configured ICMPv6 message handler function is called by the TCP/IP stack on reception of an ICMPv6 message which is not handled by the TCP/IP stack. This implementation supports ICMPv6 multi-part messages according to IETF RFC 4884. If a message contains multi-part data MultiPartDataLength is > 0 and MultiPartDataPtr points to the multi-part data structure. The following ICMPv6 messages may contain multi-part data: - ICMPv6 Destination Unreachable (Type 1) (see [28] 3.1. Destination Unreachable Message). - ICMPv6 Time Exceeded message has been received (see [28] 3.3. Time Exceeded Message). If the message has the ICMPv6 Multi-Part message format `MultiPartExtPtr` points to the first ICMP Extension Object (see [33] 8. ICMP Extension Objects).
Particularities and Limitations
> `MultiPartExtLen` is the length of all extensions objects in bytes and may be 0.
Call context
> interrupt or task context

Table 5-54 ICMP Destination Unreachable Message Callback

5.5.1.5 IPv4 Link-Local Address Candidate Callout

Prototype	
`void` **`<Configurable_Cbk_Name>`** `(TcpIp_LocalAddrIdType LocalAddrId, uint8 conflictCount, uint32 *addrCandidatePtr)`	
Parameter	
`LocalAddrId`	Identifier of the TcpIpLocalAddr that shall be configured using the LINKLOCAL address assignment method.
`conflictCount`	Number of conflicts that occurred since start of the address assignment for the local address. This value is 0 for the first probed address and is incremented every time a conflict has occurred.
`addrCandidatePtr`	Output parameter for the IP address candidate (IPv4 address in network-byte order). If this parameter is not changed inside the callout the IPv4 automatically generates a valid random IP address candidate.
Return code	
`void`	-
Functional Description	
This callout is called when a local unicast address shall be configured using the LINKLOCAL (Auto-IP) assignment method. The function is called every time a link-local address candidate is required. The callout is not required to modify *addrCandidatePtr if a random address according to IETF RFC 3927 shall be used by the IPv4.	
Particularities and Limitations	
> The returned address candidate must be in the range 169.254.1.0 to 169.254.254.255 (inclusive) in order to conform to IETF RFC 3927. The value is not checked by the IPv4.	

Call context
> interrupt or task context

Table 5-55 ICMP Destination Unreachable Message Callback

5.5.1.6 TcpIpPhysAddrChgHandler

Prototype	
void **<Configurable_Cbk_Name>** (uint8 CtrlIdx, TCPIP_P2V(TcpIp_SockAddrType) IpAddrPtr, TCPIP_P2V(uint8) PhysAddrPtr, boolean Valid);	
Parameter	
uint8 CtrlIdx	EthIf controller index to identify the related ARP/NDP table.
IpAddrPtr	Specifies the memory where the IP address is stored for which the assignment to a physical address has changed.
PhysAddrPtr	Specifies the memory where the physical source address (MAC address) in network byte order is stored.
Valid	Flag, whether the couple of Ip address and physical address is valid or not.
Return code	
void	-
Functional Description	
This callback is called in order to notify upper layers about a physical address change.	
Particularities and Limitations	
None.	
Call context	
> interrupt or task context	

Table 5-56 Physical Address Change Callback

5.5.1.7 TcpIpPhysAddrChgDiscardedHandler

Prototype	
void **<Configurable_Cbk_Name>** (uint8 CtrlIdx, TCPIP_P2V(TcpIp_SockAddrType) IpAddrPtr, TCPIP_P2V(uint8) PhysAddrPtr);	
Parameter	
uint8 CtrlIdx	EthIf controller index to identify the related ARP/NDP table.
IpAddrPtr	Specifies the memory where the IP address is stored for which the assignment to a physical address has changed.
PhysAddrPtr	Specifies the memory where the physical source address (MAC address) in network byte order is stored.
Return code	
void	-

Functional Description
This callback is called in order to notify upper layers about the fact that a physical address change has been discarded because EcuC switch TcpIpArpDiscardedEntryHandling is true. Refer to section Tracking discarded ARP entries for details.
Particularities and Limitations
None.
Call context
> interrupt or task context

Table 5-57 Physical Address Change Discarded Callback

5.5.1.8 TcpIpDuplicateAddressDetectionConfig

Prototype	
`void` **`<Configurable_Cbk_Name>`** `(TcpIp_LocalAddrIdType IpAddrId,const TcpIp_SockAddrType* IpAddrPtr, const uint8* LocalPhysAddrPtr, const uint8* RemotePhysAddrPtr);`	
Parameter	
IpAddrId	local address identifier
IpAddrPtr	Specifies the memory where the IP address is stored for which a duplicate address has been detected.
LocalPhysAddrPtr	Specifies the memory where the local physical source address (MAC address) in network byte order is stored.
RemotePhysAddrPtr	Specifies the memory where the remote node’s physical address (MAC address) in network byte order is stored.
Return code	
`void`	-
Functional Description	
This callback is called in order to notify upper layers about the detection of a duplicate IP address.	
Particularities and Limitations	
This callout is currently issued only in case duplicate address detection via NDP is done when IPv6 is active.	
Call context	
> interrupt or task context	

Table 5-58 Duplicate Address Detection Callback

6 Configuration

The TCPIP is configured with the help of the configuration tool DaVinci Configurator Pro. The description of the parameters can be found in the BSWMD-file or in the properties window in the tool.

The following sections explain some general topics regarding the configuration.

6.1 Socket Owner Configuration

TCPIP allows multiple users above the TcpIp, and AUTOSAR calls them ‘socket owners’. The SocketAdaptor (SoAd) is the standard socket owner, other socket owners can be added easily as CDD (complex device driver). Vector furthermore adds a special socket owner ‘DhcpV4Server’ that is only used TCPIP-internal if the DhcpV4Server is enabled.

For each socket owner a number of callbacks have to be configured.

If the socket owner shall act as a server, a special configuration is needed. On the one hand buffers need to be configured for the number of connections the socket owner wants to handle in parallel. On the other hand we need to configure the parameter ‘Socket Owner Tcp Listen Socket Max’. This value specifies the number of TCP listen sockets this socket owner will use simultaneously. These pure listen sockets will have no rx or tx buffers. So if a socket owner wants to offer a service using one server socket and accept two parallel connections to its offered service, the parameter ‘Socket Owner Tcp Listen Socket Max’ has to be set to ‘1’ and 2 pairs of rx and tx buffers have to be configured.

6.1.1 <Up>_CopyTxData Callback

If a TcpIpSocketOwner uses one of the APIs TcpIp_UdpTransmit() or TcpIp_TcpTransmit() with indirect data provision (parameter DataPtr == NULL_PTR) the CopyTxData-Callback must be provided by the socket owner.

AUTOSAR specifies that CopyTxData is called one or more times until the socket owner has provided the specified amount of bytes during the Transmit call.

Vector optionally supports an extension of this API using an in/out parameter BufLengthPtr instead of the in-only parameter BufLength. In this case the socket owner may update the value at BufLengthPtr with a lower value in order to indicate that it wants to transmit less data than specified by the Transmit call. Once a socket owner does not use the entire provided buffer in the CopyTxData call, the TcpIp will not request more data, transmit only the provided amount of bytes and release the unused buffer.

Configuration in DaVinci Configurator Pro
The type of the <Up>_CopyTxData callback can be configured individually for each TcpIpSocketOwner using the configuration parameter **TcpIpSocketOwnerCopyTxDataDynamicLengthEnabled**.

6.2 Unicast Address Assignment Methods

The configuration of multiple unicast address assignment methods differs between IPv4 and IPv6.

The reason is that the IPv6 can handle multiple unicast addresses on one IP controller at the same time while the IPv4 only supports one unicast address per IP controller.

Table 6-1 shows the supported configuration possibilities for IPv4 and IPv6:

Configuration Possibilities	IPv4	IPv6
Number of unicast addresses per controller/VLAN	1	1…N
Number of assignment methods per unicast address	1…3	1
Supported address assignment methods	STATIC, LINKLOCAL, DHCPv4	STATIC, LINKLOCAL, DHCPv6, ROUTER
Unicast address assignments that can be stored into nonvolatile memory	STATIC, DHCPv4	STATIC

Table 6-1 Configuration possibilities for IPv4 and IPv6 unicast address assignments

6.2.1 Multiple Address Assignment Methods for IPv4 Unicast Addresses

According to AUTOSAR it is possible to configure up to three different IP address assignment methods for the IPv4 unicast address of an IP instance.

Supported address assignment methods are STATIC, LINKLOCAL (AUTO-IP) and DHCP.

The address assignment methods of a unicast address must be configured with different priorities (1…3, where 1 means highest priority). During runtime the value of the active unicast address is always set to the address of the assignment method with the highest priority that is triggered and ready.

Depending on the configuration parameter TcpIpAssignmentTrigger an address assignment method is either triggered automatically after the ethernet link is up or manually during runtime via the API TcpIp_RequestIpAddrAssignment().

After being triggered the two dynamic address assignment methods DHCP and LINKLOCAL need so time to negotiate an address or even may not be able to configure an address (e.g. because there is no DHCP server present in the network). During this time the address assignment methods are not ready and an address of a method with lower priority will be used if possible.

An address with the STATIC address assignment method can be set in the configuration (see configuration parameter TcpIpStaticIpAddress) or during runtime via TcpIp_RequestIpAddrAssignment().

AUTOSAR Extension

In addition to AUTOSAR an address with assignment method STATIC may be changed even if the assignment trigger is configured to AUTOMATIC. In order to use this feature the configuration parameter TcpIpAssignmentLifetime must be set to TCPIP_STORE.

In this case the configured address value is used as a default value which may be persistently overwritten later. It is also possible to restore the default value again by requesting the address value TCPIP_ADDR_ANY.

All manually triggered address assignment can be deactivated again by calling TcpIp_ReleaseIpAddrAssignment().

AUTOSAR Extension

The AUTOSAR API TcpIp_ReleaseIpAddrAssignment() only supports releasing all manually triggered address assignment methods of an address at once.

Vector provides the additional API TcpIp_ReleaseSpecificIpAddrAssignment() to release only a specific assignment method of an address.

The TcpIp socket owners are notified about each IP address assignment change via the LocalIpAddrAssignmentChg()-Callback.

The following two APIs may be used in context of the callback in order to obtain detailed information about the unassigned/assigned address:

- TcpIp_GetIpAddr
- TcpIp_GetIpAddrCfgSrc

6.2.2 Callout for provision of IPv4 link-local address candidates

The IPv4 link-local address assignment according to IETF RFC 3927 [25] uses a randomly generated address that is most likely unique on the local link. The uniqueness of the address is tested by sending ARP probes for an address candidate before it is assigned as unicast address of the node. If a conflict is detected, another address candidate is generated and tested.

The optional configuration of the callout function <User>_LinkLocalAddrCandidateCallout() allows it to provide address candidates for the link-local address configuration.

Examples of usage scenarios for this callout are:

- Provide link-local address candidates that are derived from likely unique values and therefore reduce the probability of address conflicts.
- Provide stored address candidates that have been tested for uniqueness before and therefore are likely to be still unique on the local-link.

In any case the provided address candidates will be probed for uniqueness again before they are used as source address by the node.

The callout is not required to provide all address candidates. It may also provide only the first candidate. If the callout does not provide an address candidate, the IPv4 will automatically choose a random value according to IETF RFC 3927.

Configuration in DaVinci Configurator Pro
The <User>_LinkLocalAddrCandidateCallout is specified by the optional parameter TcpIpAutoIpAddrCandidateCalloutFunction inside the TcpIpAutoIpConfig container.

6.3 Configuration of Static On-Link Prefixes for IPv6

The IPv6 uses the on-link definition according to the IETF RFCs 4861 [31] and 5942 [39].

If an IPv6 packet shall be sent to a unicast destination the IPv6 determines whether or not the destination is on-link. A destination is on-link if the prefix of the destination address matches `fe80::/10` or any entry in the prefix list.

Packets to on-link destinations are sent directly to the target after the link layer address has been resolved (e.g. via an NDP Neighbor Solicitation).

Packets to destinations that are not known to be on-link must be sent to a router. If there is no router in the network, these IP packets are dropped.

According to IETF RFC 5942 the IPv6 does not treat a prefix derived from a statically or automatically configured address as on-link (except for the link-local prefix `fe80::/10`).

A prefix may be explicitly marked as on-link (added to the prefix list) via a received Router Advertisement containing a “Prefix Information” option with the “L”-flag set or via manual configuration of the prefix.

Manual Configuration of On-Link Prefixes

- The 64 bit prefix of a statically configured IPv6 address can be marked as on-link by setting the configuration parameter TcpIpIpV6AddrPrefixIsOnLink to “Enabled”.
- Alternatively any prefix can by statically added to the prefix list by specifying it via the configuration parameter TcpIpNdpOnLinkPrefix.

6.4 Static and Dynamic Link Layer Address Resolution

Every IPv4 instance has an independent dynamic address resolution cache that can hold a configurable number of entries. Each entry maps an IP address to the corresponding physical (MAC) address.

The entries in the cache are automatically added, updated and replaced by the Address Resolution Protocol (ARP, see [17]) during runtime based on received ARP Request and Reply packets.

In addition to the dynamic address resolution cache a static table can be configured for each IPv4 instance. Static ARP tables may increase performance since no ARP packets need to be sent during runtime. Additionally some ARP spoofing attacks can be prevented by using static ARP tables because static entries will not be overridden by information from received ARP packets.

A static ARP table may be shared by multiple IPv4 instances.

Each IPv4 instance can use one of the following address resolution modes:

- Dynamic ARP cache only
- Static ARP table only
- Static ARP table and dynamic ARP cache

Configuration in DaVinci Configurator Pro

The size of the dynamic ARP cache is defined by the parameter **[ARP Table Size]** of the **IpV4CtrlConfig**.

Static ARP Tables are defined in an **IpV4StaticArpTable** container that can referenced by the parameter **[Static ARP Table Ref]** of the **IpV4CtrlConfig**.

7 Glossary and Abbreviations

7.1 Glossary

Term	Description

Table 7-1 Glossary

7.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
DEM	Diagnostic Event Manager
DET	Development Error Tracer
EAD	Embedded Architecture Designer
ECU	Electronic Control Unit
ISR	Interrupt Service Routine
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
SRS	Software Requirement Specification
SWS	Software Specification

Table 7-2 Abbreviations

8 Contact

Visit our website for more information on

- News
- Products
- Demo software
- Support
- Training data
- Addresses

www.vector.com