Elektrobit

EB tresos® glossary

version 2.1.0

Elektrobit Automotive GmbH
Am Wolfsmantel 46
91058 Erlangen, Germany
Phone: +49 9131 7701 0
Fax: +49 9131 7701 6333
Email: info.automotive@elektrobit.com

Technical support

https://www.elektrobit.com/support

Legal disclaimer

Confidential information.

Table of Contents

Modification history

Version	Description
2.1.0	Added new glossary terms *PIL* and *TAL*
2.0.0	Initial new version

Glossary

.bss section	The .bss section is a data section of the memory that contains uninitialized data. It is located in the RAM.
.data section	The .data section is a data section of the memory that contains initialized data. It is located in the RAM.
.init section	The .init section is a data section of the memory that contains the initialization data for the elements of the .data section. It is usually located in the ROM.
.rodata section	The .rodata section is the data section of a memory that contains read-only data. It is usually located in the ROM.
.text section	The .text section is a section of the memory that contains executable code. It is usually located in the ROM.

A

`Adc`	`ADC Driver` module: The `ADC Driver` initializes and controls the internal analog-to-digital converter unit of the microcontroller. The driver is equipped with a set of basic functionalities with single value result access mode. Enhanced functionalities with stream access are available, too. Typical applications in which this sort of specific analog conversion is requested are knocking detection or valve control position via spectrum analysis of the current on the coil. The `ADC Driver` is part of the microcontroller abstraction layer.
ADC	analog-to-digital converter
alive supervision	The alive supervision is a feature of the `WdgM` module. The `Supervised Entities` which are parts of the SWCs trigger the `WdgM` according to a configured timing schema. If there occurs a timeout on such a trigger cycle, the alive supervision fails and the `WdgM` itself stops triggering the Watchdog hardware.
alive timeout	Alive timeout (or also known as communication timeout) is a feature, which can be enabled in the unqueued receiver COM specification of a data element prototype. In case of inter-ECU sender/receiver communication, the `Rte` then monitors the timely reception of the data, i.e. if no new value has been received within a certain time period, an error is reported to the application. For intra-ECU communication, timeout monitoring is not supported.
API	Application programming interface

application error	An application-specific error that is returned by a server to the client. Application errors are defined in the software component description as a part of the client/server interface.
application header file	A software component's application header file contains the definition of the `Rte`'s API and associated data structures that are generated for the software component. One application header file is generated for each atomic software component type encountered in the input. Its name is `Rte_<appl>.h` whereas `<appl>` is the name of the atomic software component type. The application header files are both generated in the contract and in the generation phase and must be included by the software component's code.
argument prototype	Argument prototypes describe the arguments for an operation prototype in a client/server interface. The arguments are passed to the operation prototype in the sequence in which they are defined in the input. Each argument prototype has a type and a direction. The direction indicates if the argument prototype is an ► input parameter (`direction = IN`, read but not written by the operation prototype) ► input/output parameter (`direction = INOUT`, read and written by the operation prototype ► output parameter (`direction = OUT`, not read but written by the operation prototype)
asynchronous server call point	Property of a runnable entity which indicates that the runnable entity asynchronously invokes an operation prototype in the provide port of a client/server interface by using the appropriate Rte API call. Asynchronous invocation of an operation prototype implies that the Rte API call does not block until the execution of the server has terminated and that a separate Rte API call is necessary to retrieve the result.
asynchronous server call returns event	An asynchronous server call returns event is an *RTE event* that is issued by the `Rte` to notify the client in a client/server connection that either the server execution has terminated or a timeout or communication error has occurred. Only one asynchronous server call returns event is issued by the `Rte` for each asynchronous invocation of an operation prototype. An asynchronous server call returns event can trigger both a runnable entity and resolve a wait point.
atomic software component	An *atomic software component* encloses application software, realizing a particular functionality that cannot be divided any further. Therefore, atomic software components are the smallest software components that are executed on a particular ECU. The atomic software component type describes the communication interface (external view) of the atomic software component, i.e.

	the provide and require ports with their provided or required interfaces. Each atomic software component type must have exactly one internal behavior that specifies its functionality in the form of runnable entities and RTE events that trigger the runnable entities (internal view). An implementation has to be selected for the internal behavior as part of the Rte configuration process.
atomic software component instance	An *atomic software component instance* of an *atomic software component type* is created by specifying a component prototype of the atomic software component type within a composition type. If the internal behavior of the atomic software component type supports multiple instantiation, more than one component prototype of the atomic software component type can be created. Since the composition type can again be instantiated, an atomic software component instance does not necessarily correspond to a component prototype. For the generation of the `Rte`, only atomic software component instances are relevant. The system is flattened and reduced to atomic software component instances, all compositions are removed.
AUTOSAR	*Aut*omotive *O*pen *S*ystem *AR*chitecture
AUTOSAR types header file	Is a header file generated by the Rte Generator that contains ► the definition of all data types and enumeration constants encountered in the input as well as ► the definition of all data types that are required by the Rte implementation ► except the software component specific type definitions. The latter are defined in the software component's application header file.

B

basic storage object	A *basic storage object* is a sub-structure which is used to administrate and to store non-volatile data. A basic storage object can reside in different memory locations, such as RAM, ROM, or NVRAM.
block management type	The (configurable) individual composition of an *NVRAM block* in chunks of mandatory/optional *basic storage objects* is called a *block management type*.
BSP	board support package: A board support package is a set of files that defines the characteristics of a particular hardware platform (board) that is used by the kernel.
BSW	*B*asic *sw*are; defined by AUTOSAR the basic software includes the following software modules ► microcontroller abstraction driver

- ECU abstraction
- communication stack
- AUTOSAR services
- operating system
- complex device drivers

C

calibration parameter

Calibration parameters are parameters which influence the ECU behavior. Examples for such parameters are variant-dependent data, run-time software switches, process controlling data, interpolation curves or interpolation fields. Calibration parameters are typically determined during an calibration process and then stored into ROM or non-volatile RAM. In AUTOSAR, calibration parameters can be defined within one atomic software component or within a calibration software component. Calibration parameters defined for an atomic software component can only be used within the atomic software component. Several instances of the same atomic software component type can either share the same instance of a calibration parameter or use their own instances. The `Rte` provides the API `Rte_CData` to access the calibration parameters.

calibration software component

A calibration software component is a software component which provides calibration parameters to other software components. Calibration components are used when a calibration parameter is to be shared among different application components. A calibration software component provides ports with a calibration interface. The calibration interface defines calibration parameters which are provided to other atomic software components. Ports of atomic software components can be connected to the calibration components via assembly connectors. The `Rte` provides the `Rte_Calprm` API to read the calibration parameters.

`Can`

`CAN Driver` module: The `CAN Driver` performs hardware access and offers a hardware independent API to the upper layer. Only the `CAN Interface` has access to the `CAN Driver`. It provides services to control the behavior and state of the CAN controllers that belong to the same CAN hardware unit. The `CAN Driver` furthermore offers the functionality for performing communication on the bus (e.g. sending messages). The `CAN Driver` is part of the microcontroller abstraction layer (MCAL). It is the lowest layer of the CAN bundle.

`CanIf`

`CAN Interface` module: The `CAN Interface` is the hardware independent connection of the `CAN Driver (Can)` and the `CAN Transceiver`

	Driver (CanTrcv) to the upper layer modules which want to communicate on the CAN bus. It provides mode handling (including bus-sleep and wakeup if supported by Can and/or CanTrcv) on a per network base and offers a hardware independent interface for message communication that maps PDU IDs to the corresponding CAN Driver parameters and vice versa. It furthermore provides some extended functionalities, e.g. dynamic transmit CAN IDs, receive buffering and polling, which can be configured during configuration. The CanIf is part of the ECU abstraction layer.
CanNm	CAN Network Managementmodule: The CAN Network Management is responsible for the cyclic transmission of network management messages on the CAN bus. The CAN Network Management is part of the communication services inside the AUTOSAR service layer. It is positioned below the Generic Network Management Interface (Nm) and uses the CAN Interface (CanIf) to send and receive network management messages.
CanSm	The CAN State Management module is the CAN specific state manager, handling the control flow of the associated CAN buses. The CAN Network Management is part of the communication services inside the AUTOSAR service layer. The CanSm implements the CAN specific network mode state machine which handles the states full-, silent- and no- communication.
CanTp	CAN Transport Layermodule: The CAN Transport Layer is part of the service layer. The CAN Transport Layeris responsible for sending and receiving diagnostic messages. It furthermore segments and reassembles messages which do not fit to a single CAN frame.
CanTrcv	CAN Transceiver Driver module: The CAN Transceiver Driver is part of the communication hardware abstraction in the ECU abstraction layer.
CC	Communication controller.
client/server interface	A client/server interface aggregates all operation prototypes that are provided by a server and can be invoked by a client. A client/server interface must contain at least one operation prototype. A client/server interface may define possible application errors.
CNm	CAN Generic Network Management module: Obsolete module in AUTOSAR 3.0 and up. Functionality of this module has been moved into CanNm.
code type	The code type attribute of an implementation specifies whether an atomic software component is delivered as a source code component (code type = SRC) or as an object-code component (code type = OBJ). Optimized access to operation prototypes is only available for atomic software components delivered as source code components.

Com	Communication module: The Com module implements the packing of signals into PDUs and initiates the transmission on sender side. On the receiver side the signals are unpacked from the PDUs, decoded and the applications are informed about the reception via the Rte. The Com module is located in the communication stack above the PduR and below the Rte.
ComM	Communication Manager module: The Communication Manager simplifies the resource management for the EcuM, which in turn simplifies the resource handing of the communication stack. The Communication Manager is located in the system services in the service layer.
Common Published Information	*Common Published Information* are read-only configuration parameters, such as version information, module-ID and vendor-ID.
compatibility mode	Compatibility mode of the Rte Generator: The compatibility mode ensures compatibility of Rte Generators by different vendors between contract phase and generation phase. Do not confuse with OSEK compatibility mode.
component prototype	A component prototype is the instantiation of an atomic software component type or a composition type within a composition type. Since the composition type itself can also be instantiated again within another composition, a component prototype does not necessarily correspond to an atomic software component instance.
composition	Compositions enable a hierarchical software architecture of an AUTOSAR system. The composition type describes the communication interface (external view) of a composition, i.e. the provide and require ports with their provided or required interfaces that are visible outside the composition. Furthermore, the composition type consists of component prototypes, which structure the composition's functionality, and of connectors, which either connect the ports of the composition's component prototypes or make them visible to the outside of the composition type. Unlike atomic software component types, no internal behavior or implementation is specified for a composition type. A system must have at least one composition which is the top level composition.
Com signal	Messages handled by AUTOSAR Com for reception and transmission. They were called Com messages in OSEK Com.
connector	Connectors are part of a composition type. An assembly connector connects one provide port of a software component instance with one require port of another software component instance within the composition type. A delegation connector makes a port of a software component instance within the composition type visible to the outside of the composition type. This is done by connecting a require port of the software component instance with a require port of the composition type or by connecting a provide port of the software

	component instance with a provide port of the composition type. A service connector prototype connects a port of a service component prototype to a port of an application software component instance. In contrast to assembly connectors, service connector prototypes can be used across composition boundaries, i.e. no delegation connectors have to be used.
contract phase	The first of two generation phases distinguished by an Rte Generator. Only the application header files and the AUTOSAR types header file are generated during the contract phase. With the application header file and the AUTOSAR types header file it is possible to compile an atomic software component's code and deliver the software component as an object-code component. If different Rte generators are used for both generation phases, they have to operate in compatibility mode (see operating modes).
control path	The control path of the AUTOSAR communication stack is responsible for controlling the state of the communication stack. The control path of the communication stack does not sent or receive data. The only exception are the network management messages.
CPU	central processing unit
`Crc`	`CRC Routines` module: In the AUTOSAR environment Crc routines are used by the `NVRAM Manager`, for example. The `Crc` is located in the system services in the service layer.
CRC	A cyclic redundancy check (CRC) is a type of function that takes an input of data stream of any length and produces a value of a certain fixed size as output. The term *CRC* is often used to denote either the function or the function's output. A CRC can be used as a checksum to detect alteration of data during transmission or storage.

D

data element invalidation	A feature which can be enabled in the unqueued sender and receiver COM specification of a data element. On the sender side, the `Rte` provides an API to invalidate a data element prototype. On receiver side, the reception of an invalid value is either handled as an error or the data is silently replaced by the invalid value.
data element prototype	Represents the data elements that can be sent/received in a port that provides/requires the sender/receiver interface. A data element prototype is aggregated by a sender/receiver interface. It has a type and is either queued or unqueued.

data path	The real user data are sent and received via the data path of an AUTOSAR communication stack. Software components use the data path of the communication stack to send data from one ECU to another.
data read access	Property of a runnable entity that indicates that the runnable entity has access to the value of a data element in a require port that was received when the runnable entity was activated. That value does not change during the execution of the runnable entity. The `Rte` has to provide a copy of the data element and has to ensure that it does not change while the runnable entity is executed. If alive timeout or invalidation is specified for the data element, the existence of data read access for the data element prototype in the require port also implies that the runnable entity has access to the status information that was returned to the `Rte` when the local copy of the data element prototype was updated.
data receive error event	A data receive error event is issued by the `Rte` to notify the receiver in a sender/receiver connection for a data element prototype with data semantics that either the available value is outdated or it has been invalidated. A data receive error event can only trigger a runnable entity but is not allowed to resolve a wait point.
data receive point	Property of a runnable entity which indicates that the runnable entity explicitly reads a data element prototype in a require port by invoking an Rte API call.
data received event	A data received event is issued by the `Rte` to notify the receiver in a sender/receiver connection for a data element prototype that a new value is available. If the corresponding data element prototype is unqueued, the data received event is only allowed to trigger a runnable entity. If the corresponding data element prototype is queued, the data received event can both trigger a runnable entity and resolve a wait point.
data send completed event	A data send completed event is an RTE event that is associated with a data send point and that is issued by the `Rte` to notify the sender in a sender/receiver connection that the transmission was successful or that a timeout or communication error was detected. For transmission of a data element prototype, only one data send completed event is issued by the `Rte`. A data send completed event can both trigger a runnable entity and resolve a wait point.
data send point	Property of a runnable entity which indicates that the runnable entity explicitly requests transmission of a data element prototype through a provide port by invoking an Rte API call. If `canInvalidate` or `transmission acknowledgment` is specified for the data element prototype in the provide port, the existence of a data send point also implies that the Rte API for invalidating the data element prototype or for collecting the transmission acknowledgment is available for the runnable entity.

dataset NV block	The dataset NV block is a configurable type of an NV block.
data write access	Property of a runnable entity which indicates that the runnable entity provides a new value for transmission of a data element prototype through a provide port after the runnable entity has terminated. The `Rte` has to ensure transmission after the termination of the runnable entity. If `canInvalidate` is specified for the data element prototype in the provide port, the existence of data write access also implies that the invalid value can be set during execution of the runnable entity and is then transmitted after the runnable entity terminates.
DBC	The DBC format is a legacy format used to describe CAN communication data. It is a proprietary format by Vector Informatik.
`Dbg`	Short form of the AUTOSAR `Debugging` module.
`Dcm`	`Diagnostic Communication Manager` module: The `Diagnostic Communication Manager` manages diagnostic protocols (UDS, OBD) inside the ECU. It monitors ► protocol communication states, ► timings, ► life sign signals, etc. and dispatches diagnostic requests to the responsible software components. The `Diagnostic Communication Manager` is located in the communication services in the service layer.
debouncing	Debouncing is a kind of filtering that the `Dem` module can apply to unsteady diagnostic events.
`Dem`	`Diagnostic Event Manager` module: The `Diagnostic Event Manager` processes and stores diagnostic events (errors) and associated environment conditions reported by the application layer or the AUTOSAR BSW. It provides fault information to the `Diagnostic Communication Manager` (`Dcm`) for off-board diagnostics. The `Diagnostic Event Manager` is located in the communication services in the service layer.
`Det`	`Development Error Tracer` module: The `Development Error Tracer` (`Det`) is a debugging API to detect software bugs during the software development and integration phase. The `Development Error Tracer` is located in the system services in the service layer.
diagnostic event	Diagnostics are used to determine the proper functionality of a component. The state of each diagnostic functionality is stored by a diagnostic event within the `Dem` module. Unlike an error, a diagnostic event is reported as failed or

	passed. The `Dem` maps each event to a diagnostic trouble code to allow to identify a diagnostic event unambiguously within a whole car. See also DTC See also monitor function
diagnostic service	The requests from a *diagnostic tester* towards an ECU are called diagnostic service. The service `ReadMemoryByAddress`, for example, is issued by the diagnostic tester and triggers the ECU to read the current value of a memory range.
diagnostic session	A diagnostic session is a state within the `Dcm` module. A *diagnostic tester* can initiate the switch to another session. In each session, specific diagnostic services are available.
diagnostic tester	See tester
diagnostic tool	See tester
diagnostic trouble code	See DTC
DID	debugging identifier: The debugging core module identifies data by debugging identifiers (DIDs) which are of type uint8. To properly communicate between host and target, the DIDs need to be known to the debugging module and to the host. There are two kinds of DIDs, which can be used: ► Standard debugging identifiers with address/size information. ► Predefined debugging identifiers without address/size information.
`Dio`	`Digital IO Driver` module: The `Dio` module allows level read/write of single port pins (channel), a port pin group (channel group) or a complete port. The `Dio Driver` is part of the microcontroller abstraction layer.
direct (function) call	Optimization for synchronous server calls. For a direct function call, no context switch is performed, the server runnable entity is executed within the client's task context. This is realized by directly mapping the Rte API to the C function of the server runnable entity. Certain criteria have to be fulfilled so that a direct function call can be made.
DTC	A diagnostic trouble code is an Id which is unique within a whole vehicle. It is used to identify a diagnostic event unambiguously. For service in garages, a tester device could read the DTCs of a vehicle for troubleshooting. There are standardized DTCs for emission related events and manufacturer specific DTCs which could vary on each vehicle platform. The standardized

DTCs allow to read emission related events of all vehicles of all car manufactures.

See also OBD

E

Ea	EEPROM Abstraction module: The EEPROM Interface provides equal mechanism to access microcontroller internal and external EEPROM devices. It abstracts from the location of peripheral EEPROM devices (internal or external), the ECU hardware layout and the number of EEPROM devices. The EEPROM is part of the memory hardware abstraction in the abstraction layer.
EABI	embedded application binary interface: An embedded application binary interface is a specification of the programming interfaces (for example C calling conventions) that are used.
EB tresos	The Elektrobit tresos product line
EB tresos AutoCore	Elektrobit's implementation of an AUTOSAR basic software stack (BSW).
Eclipse	Eclipse is an open-source framework. Eclipse is a multi-language software development platform comprising an IDE (integrated development environment) and a plug-in system to extend it. It is written primarily in Java and is used to develop applications in this language and, by means of the various plug-ins, in other languages as well C/C++, Cobol, Python, Perl, PHP and others.
ECU	electronic control unit
EcuC	ECU configuration module; virtual module to collect ECU configuration-specific/global configuration information. The virtual ECU configuration module is defined by AUTOSAR to hold the global PDUs. To be able to define a PDU flowing through the COM stack two modules need to be able to refer to the same PDU object. Therefore a generic PDU container has been defined which does not belong to any module but to the whole COM stack. All COM stack modules dealing with PDUs have a reference to such a global PDU.
ECU instance	ECU instance denotes the presence of an ECU within an automotive network, whereas an ECU can be seen as the common prototype of multiple ECU instances of the same kind. AUTOSAR defines ECU instances in the AUTOSAR_SystemTemplate.pdf.
EcuM	ECU State Manager module: Basic software module that manages all aspects of the ECU related OFF, RUN and SLEEP states and the transitions

	between these states such as STARTUP and SHUTDOWN. Management of all wakeup events and configuration of the ECU for SLEEP when requested The `Ecu State Manager` is part of the system services in the service layer.
Editor view	The **Editor** view is a view of EB tresos Studio, which is in the center of the EB tresos Studio main window. It enables editing the configuration data of a project, e.g. the parameters of an AUTOSAR module.
`Eep`	`EEPROM Driver` module: The `EEPROM Driver` provides services for reading, writing, erasing to/from an EEPROM. It also provides a service for comparing a data block in the EEPROM with a data block in the memory. The `Eep` is part of the memory driver in the microcontroller abstraction layer.
EEPROM	*E*lectrically *e*rasable *p*rogrammable *r*ead *o*nly *m*emory; is a type of non-volatile memory used in computers and other electronic devices to store small amounts of data that must be saved when power is removed, e.g., calibration tables or device configuration. When larger amounts of static data are to be stored (such as in USB flash drives) a specific type of EEPROM such as flash memory is more economical than traditional EEPROM devices. EEPROMs are realized as arrays of floating-gate transistors.
entry point	The entry point (attribute `Symbol`) of a runnable entity specifies the name of the C function that implements the runnable entity. The names of the entry points of all runnable entities on one ECU have to be unique. The same entry point cannot be shared by two or more runnable entities.
Error Log view	The **Error Log** view is a EB tresos Studio view in which all errors are logged. The **Error Log** receives error messages from EB tresos Studio components and from other Eclipse components. The list of messages in the **Error Log** is persistently stored. The default location of the Error Log view is on the bottom of the EB tresos Studio main window.
exclusive area	A mechanism of the `Rte` to specify critical sections. A runnable entity can request access to an exclusive area to prevent preemption by other runnable entities that have access to the same exclusive area in order to ensure data consistency when accessing shared data. A runnable entity can either access an exclusive area implicitly (runs in exclusive area, i.e. the whole execution of the runnable entity is protected by the exclusive area) or explicitly (can enter exclusive area, i.e. Rte API calls are provided to enter/leave the exclusive area). For an exclusive area, different implementation mechanisms can be configured: ► enabling/disabling of interrupts ► usage of `Os` resources

- usage of non-preemptive tasks
- cooperative runnable placement

extended data record — For each diagnostic event, environmental data can be stored in an extended data record. This could be, for example, an occurrence counter. Extended data records are overwritten with each report of a diagnostic event.

F

`Fee` — `Flash EEPROM Emulation` module: Emulates EEPROM functionality using the flash memory. The `Fee` is part of the memory services in the microcontroller abstraction layer.

Fibex — Fibex is an XML-based format for the description of complex and message-based communication systems. The Fibex format is especially used to describe FlexRay networks. Besides using the LDF description file format, it is possible to define LIN networks with the Fibex format, too.

filter — Value-based filters can be applied to primitive unqueued data element prototypes as part of a unqueued receiver COM specification in a require port to filter the data on receiver side, i.e. the data is not passed to the application if the filter does not match. For inter-ECU communication, filtering is not provided by the `Rte` but is handled by AUTOSAR Com. In order to use the filter mechanism, the appropriate reception filters have to be specified in the AUTOSAR `Com` module for the Com signal(s) that is used to receive the data element prototype. In case of intra-ECU communication, filtering is provided by the `Rte`. In this case, the filters have to be defined in the unqeueud receiver COM specification.

filter mask — A filter mask is used to configure a CAN receive buffer (HRH) for the reception of multiple CAN-IDs. It specifies the significant bits that must be equal in the CAN-IDs of an incoming CAN L-SDU and in the HRH so that the HRH will receive the L-SDU.

`FiM` — `Function Inhibition Manager` module: The `Function Inhibition Manager` is responsible for providing a control mechanism for software components and the functionality therein.

`Fls` — `Flash Driver` module: The `Flash Driver` provides services for reading, writing and erasing flash memory and a configuration interface for setting/resetting the write/erase protection if supported by the underlying hardware. The `Flash Driver` is part of the microcontroller abstraction layer. It is the lowest layer of the memory stack.

FPU	floating-point unit: A floating-point unit is a part of the CPU (or sometimes a separate unit) that performs calculations of floating point numbers.
Fr	FlexRay Driver module: The FlexRay Driver hides the peculiarities of a specific FlexRay controller by providing a device independent (but FlexRay controller-specific) API to the upper layers. The FlexRay Driver does not have an own execution context (e.g. MainFunction) nor a state machine. The FlexRay Driver is part of the communication drivers in the microcontroller abstraction layer.
freeze frame	A freeze frame holds environmental data such as a time stamp or the mileage of a diagnostic event. This environmental data provides additional information for troubleshooting. Freeze frames are also called *snapshot data*. With each report of a diagnostic event, a freeze frame is appended.
FrIf	FlexRay Interface module: The FlexRay Interface provides equal mechanisms to access a FlexRay bus channel regardless of its location (microcontroller-internal or -external). It abstracts from the location of CAN controllers (on-chip or on-board), the ECU hardware layout and the number CAN drivers. The FlexRay Interface is part of the communication hardware in the AUTOSAR ECU abstraction layer
FrNm	FlexRay Network Management module: The FlexRay Network Management is responsible for the cyclic transmission of network management messages on the FlexRay bus. The FlexRay Network Management is part of the communication services inside the AUTOSAR service layer. It is positioned below the generic Network Management (Nm) and uses the FlexRay Interface (FrIf) to send and receive network management messages.
FrTp	FlexRay Transport Layer module: The FlexRay Transport Layer segments and reassembles messages that do not fit in one of the assigned Fr N-PDUs. The FlexRay Transport Layer is located between the PDU Router and the FlexRay Interface module in the communication services.
FrTrcv	FlexRay Transceiver Driver module: The FlexRay Transceiver is a hardware device, which mainly transforms the logical 1/0 signals of the microcontroller ports to the bus-compliant electrical levels, currents and timings. Within an automotive environment there is currently only one single physical layer specification for FlexRay. In addition, the transceivers are often able to detect electrical malfunctions such as wiring issues, ground offsets or collisions. Depending on the interface they flag, the detected error is summarized by a single port pin or very detailed via SPI. The FlexRay Transceiver Driver is located in the communication hardware abstraction in the abstraction layer.

function elidation	Function elidation is a vendor-specific optimization of the EB tresos AutoCore `Rte`. If function elidation is enabled, some parts of the Rte API will be realized as macros instead of functions under certain conditions.

G

GCC	GNU compiler collection; originally stood for GNU C compiler (analog to the Unix command cc for C compiler), but has taken on the meaning compiler collection since GCC is able to compile languages other than just C by now.
GNU	supposedly stands for *G*NU is *n*ot *U*nix; the GNU General Public License is a free, copyleft license for software and other kinds of works. Developers who use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.
`Gpt`	`General Purpose Timer Driver` module: Simple driver for creating one-shot/periodic time intervals using a hardware timer. Generates a clock signal. The `Gpt` is part of the microcontroller abstraction layer.

H

healing	Unlearning/deleting of a no longer failed event after a configurable number of operation cycles from event memory. Diagnostic events can be reported as failed or passed. After an event has been reported as failed, it is stored as failed inside the `Dem`'s event memory. If this event is now reported as passed over a configurable amount of operation cycles, it is removed from the event memory. The removal of a previously failed event from the event memory is called *healing* of an event.
HOH	Hardware object handle. An HOH is a configuration item of a `Can` module configuration that represents one or more CAN hardware buffers for receiving or sending CAN L-SDUs.
HRH	Hardware receive handle. An HRH is a configuration item of a `CanIf` module configuration that represents one or more CAN hardware buffers for receiving CAN L-SDUs.
HTH	Hardware transmit handle. An HTH is a configuration item of a `CanIf` module configuration thatrepresents one or more CAN hardware buffers for sending CAN L-SDUs.

I

I-PDU	Interaction layer protocol data unit. PDU used in the Presentation (interaction) Layer (layer 6) of the ISO/OSI-layer model.
`Icu`	`Input Capture Unit Driver` module: The `Icu` is a module that uses the input capture unit (ICU) for demodulation of a pulse-width modulated (PWM) signal. It is counting pulses, measuring frequency and duty cycle, generating simple interrupts and wake-up interruptions. It provides services for: signal edge notification, controlling wakeup interrupts, periodic signal time measurement, edge time stamping (for the acquisition of non-periodic signals) and edge counting. The `Icu` is part of the microcontroller abstraction layer.
implementation	The implementation that refers to an internal behavior describes the properties of the particular implementation such as code type, resource consumption, etc. During configuration of the `Rte`, an implementation has to be selected for each atomic software component type.
init value	For an unqueued data element prototype, the presence of an init value ensures that valid data is accessed even if no value has been received. An init value can be specified both on receiver and on sender side using an unqueued receiver COM specification or an unqueued sender COM specification. If no init value is specified for each data element prototype, the init value of the connected data element prototypes is used. If the connected data element prototypes also not have an init value, a default init value of 0 is used. If Com signals are used for transmission or reception of a data element prototype, the init value that is specified for the data element prototype and the init value that is specified for the corresponding Com signal(s) have to be consistent. The Rte Generator checks whether they match and reports a warning if there is an inconsistency. Moreover, if several data element prototypes share a single Com signal or are connected to the same sender, they all have to use the same init value.
internal behavior	The internal behavior of an atomic software component type describes the component's functionality and internal structure. Only one internal behavior can be specified for each atomic software component type.
inter runnable variable	Inter runnable variables offer a sender/receiver-like communication mechanism for runnable entities within an atomic software component. The `Rte` ensures data consistency for accesses to the inter runnable variables. Inter runnable variables are specified as parts of a software component's internal behavior. Each inter runnable variable has a type and a communication approach (implicit or explicit). Each runnable entity of the same internal behavior can specify read- or write access to any of the inter runnable variables.

	If access to an implicit inter runnable variable is specified, the `Rte` provides access to a runnable-specific copy of the Inter runnable variable. If access to an explicit Inter runnable variable is specified, the `Rte` provides access to the shared buffer that represents the inter runnable variable. The access is protected against preemption by other runnable entities that have access to the same inter runnable variable.
interrupt request	The interrupt request (IRQ) signals the CPU to stop the program execution at the current point and to continue the execution at the address which is associated with the specific interrupt request. The CPU provides means to save the interrupted context and to restore it again after the interrupt service routine that handles the interrupt request has finished.
interrupt service routine	The interrupt service routine (ISR) is the function which handles an interrupt request.
invalid value	The invalid value is specified as part of a data element prototype's data type and is used for the data element invalidation feature. Only primitive types allow to specify an invalid value (Char type, Integer type, Boolean type, Real type or Opaque type).
IOC	Inter-OS-Application Communicator: The IOC is an AUTOSAR module, which is included in the AUTOSAR Os. The IOC provides functions that allow OS applications to communicate across memory boundaries (if memory protection is used) or across different CPU cores on a multi-core system.
`IoHwAb`	`I/O Hardware Abstraction` module.
`IpduM`	`I-PDU Multiplexer` module: The `I-PDU Multiplexer` module (IpduM) allows the transmission of several PDUs with the same CAN-ID but with different layout. Therefore one byte of the data segment is used as multiplexer byte. On sender side, the `I-PDU Multiplexer` module is responsible for combining appropriate I-PDUs from `Com` to new, multiplexed I-PDUs and sends them back to the `PDU Router`. On the receiver side, it is responsible for interpreting the content of multiplexed I-PDUs taking into account the value of the selector field.
IRQ	See interrupt request
`isQueued`	The `isQueued` attribute describes the semantics of a data element prototype. If the `isQueued` attribute is set to `FALSE` , the data element prototype has *data semantics*, which implies buffered reception with *last-is-best* semantics and a limitation to non-blocking read accesses. If the `isQueued` attribute is set to `TRUE` , the data element prototype is said to have *event semantics*, which implies queued reception. Blockages of read accesses are possible.

ISR	See interrupt service routine
`isService`	The `isService` attribute of a sender/receiver or client/server interface indicates that the interface is defined by an AUTOSAR service and therefore belongs to a standardized AUTOSAR interface.

K

kernel	The kernel is the core part of the operating system. It provides the task and interrupt service routine (interrupt service routine) scheduling mechanisms.

L

LDF	LIN description file; the LIN description file LDF describes all signals, data packages, and their time order on a LIN network. If several LIN slaves are connected to a LIN network, all signals and frames are described in the LIN description file. Nodes can identify if a frame is relevant for them from the identifier field of the LIN data frame.
lifecycle header file	A static header file of the `Rte`. It defines the `Rte`'s lifecycle API, i.e. the API functions for initializing and finalizing the `Rte`. It is included by BSW modules which use the `Rte`'s lifecycle API.
`Lin`	`LIN Driver` module: The `LIN Driver` performs hardware access and offers a hardware-independent API to the upper layer. Only the `LIN Interface` has access to the `LIN Driver`. It provides services to control the behavior and state of the UARTs serving as LIN hardware unit. The `LIN Driver` is part of the `Com Driver` in the microcontroller abstraction layer. It is the lowest layer of the LIN bundle.
LIN	Local Interconnect Network: Serial network based on time-scheduling on low-speed networks (up to 19.2 kbit/s).
`LinIf`	`LIN Interface` module: The `Lin Interface` implements the core functionality of a LIN master. This includes the processing of LIN schedule tables and therefore the control over which and when frames are transmitted on the bus. It is part of the communication hardware abstraction in the ECU abstraction layer.
linker script	A linker script is the configuration file for the linker component of a toolchain that defines where the linker program locates the code and data sections of a program in the memory.
L-PDU	PDU used in the Data Link Layer (layer 2) of the ISO/OSI layer model.

LSB	least significant bit: the bit position in a binary number with the smallest value.

M

µC	Microcontroller
`make`	A `make` file is a software program intended to automate and optimize the construction of programs or files. `Make` is one of the original Unix tools for software engineering. There are public domain versions (e. g. GNU) and versions for other systems (e. g. Vax/VMS).
makefile	A makefile provides instructions for a `make` program (this program is called target), and the rules how to implement these instructions. Every target may depend upon other targets or files.
MCAL	See: microcontroller abstraction layer
`Mcu`	`MCU Driver` module: The `MCU Driver` provides services for basic microcontroller initialization, power down functionality, reset and microcontroller-specific functions required for other MCAL software modules. The `MCU Driver` is part of the microcontroller abstraction layer.
MCU	Microcontroller unit; is a single chip that contains a processor, RAM, ROM, clock and I/O control unit.
`MemIf`	`Memory Abstraction Interface` module: The `Memory Abstraction Interface` abstracts how the memory is stored. This enables the `NVRAM Manager` to manage multiple EEPROM drivers via the `Ea` module, and/or multiple flash drivers via the `Fee` module.
memory mapping	Memory mapping defines which data is used in which part of the memory.
memory protection	Memory protection is a way to control memory access rights on a computer. The main purpose of memory protection is to prevent a process from accessing memory that has not been allocated to it. Memory protection must be supported by the CPU either via a memory protection unit (MPU) or a memory management unit (MMU) as well as by the operating system.
microcontroller abstraction layer	Microcontroller abstraction layer (MCAL); abstracts µC periphery and memory from the upper layer software layers of the AUTOSAR. The MCAL provides functions for configuring the processor periphery, writing and reading of special function registers, i.e. Dio, Pwm. The functions for the modules within the MCAL are typically called *drivers*.
MMU	memory management unit: a memory management unit (MMU), is a computer hardware component responsible for handling accesses to memory requested

	by the central processing unit (CPU). Its main functions include translation of virtual addresses to physical addresses (i.e., virtual memory management) and memory protection.
mode declaration	A mode declaration specifies a single mode by assigning a name to the mode.
mode declaration group	A mode declaration group aggregates several mode declarations. A mode declaration group prototype is aggregated by a sender/receiver interface and is typed by a mode declaration group.
mode disabling dependency	The mode disabling dependency is a property of an RTE event that triggers a runnable entity. If the RTE event that triggers a runnable entity has a mode disabling dependency for a certain mode declaration, the runnable entity is not triggered by the RTE event as long as the mode specified by the mode declaration is active.
mode manager	The mode manager for a mode declaration group prototype is a software component with a provide port for the sender/receiver interface that aggregates the mode declaration group prototype. This implies that the mode manager can switch between the modes that belong to the mode declaration group prototype.
mode port	A mode port is a provide or require port that is categorized by a sender/receiver interface that aggregates at least one mode declaration group prototype.
mode switch event	A mode switch event is issued by the `Rte` to notify the receiver in a sender/receiver connection for a mode declaration group prototype that a mode switch has occurred. A mode switch event is only allowed to trigger a runnable entity. It is not allowed to resolve a wait point. Two different kinds of activation for mode switch events are distinguished, depending on the mode switch event's activation attribute. If `activation = exit` is specified, the mode switch event triggers a runnable entity when leaving the old mode. If `activation = entry` is specified, the mode switch event triggers a runnable entity when entering the new mode.
mode switch interface	A sender/receiver interface that aggregates at least one mode declaration group prototype is also called a mode switch interface.
mode user	The mode user for a mode declaration group prototype is a software component with a require port for the sender/receiver interface that aggregates the mode declaration group prototype. This implies that the mode user is notified about mode switches by the `Rte`, can use mode disabling dependencies on RTE events and can specify mode switch events that trigger runnable entities.
model transformation code	See MTC

monitor function	A monitor function is used in diagnostics to determine the proper functionality of a component. A monitor function identifies a specific fault type (e.g. short to ground, open load) and reports the result as a diagnostic event to the `Dem`. See also diagnostic event
MPU	memory protection unit: The MPU is a part of the CPU that allows to give access permissions on memory regions, i.e. to grant or deny read, write or execution rights. The operating system uses the MPU to implement the memory protection.
MSB	most significant bit: the bit position in a binary number with the greatest value.
MTC	model transformation code: A model transformation code (MTC) is a software component that provides an arbitrary number of input ports and one output port. When executed, the MTC reads the values from its input ports, applies its function, and propagates the result to its output port.
multiple configuration container	The basic software (BSW) code and several sets of configuration data are flashed onto an ECU. During run-time one of the configuration data sets is used to initialize the ECU. During configuration, several sets of a configuration have to be stored. This is done with the multiple configuration container.
multiple instantiation	An atomic software component type is said to have multiple instances if more than one instances of the atomic software component type exists within the system. Multiple instances of an atomic software component type can be created by specifying more than one component prototype of the atomic software component type within a composition type or by specifying more than one component prototype of the composition type that contains instance(s) of the atomic software component type. Multiple instantiation of an atomic software component prototype is only possible if its internal behavior supports the `MultipleInstantiation` attribute. If that attribute is set, the different instances are distinguished by the instance handle. The instance handle is then always present as first parameter of all Rte API calls that are generated for this atomic software component type, for all `trace hook` functions for those API calls and for all runnable entity prototypes of the atomic software component type's internal behavior. If the `MultipleInstantiation` attribute is not set, the instance handle is omitted as first parameter of all Rte API calls that are generated for this atomic software component type, for all trace hook functions for those API calls and for all runnable entity prototypes of the atomic software component type's internal behavior.

N

native NV block	The native NV block is a configurable type of an NV block.

N-PDU	PDU used in the Network Layer (layer 3) of the ISO/OSI layer model.
Nm	Generic Network Management Interface module: The Generic Network Management Interface module (Nm) implements a wrapper layer above the CanNm and the FrNm, so that the ComM only sees one homogeneous Nm layer. It features the forwarding of calls. The Generic Network Management Interface is an adaption layer between the AUTOSAR communication manager and the AUTOSAR bus-specific network management modules (e.g. CanNm, FrNm).
Nm-PDU	N-PDU which is processed by the AUTOSAR bus-specific network management modules.
NV	non-volatile
NvM	NVRAM Manager module: The NVRAM Manager provides services to ensure the data storage and maintenance of non-volatile data according to their individual requirements in an automotive environment. The NVRAM Manager administrates the non-volatile data of an EEPROM and/or flash EEPROM emulation device. It provides the required synchronous or asynchronous services for the management and the maintenance of non-volatile data. It is part of the memory services in the service layer.
NVRAM	Non-volatile random access memory.
NVRAM block	The NVRAM block is the entire structure, which is needed to administrate and to store a block of non-volatile data. The NVRAM block is composed of basic storage objects.

O

OBD	On-board diagnostics. OBD is used to capture emission-relevant vehicle information. This comprises two areas: ► All emission-related information are gathered within the affected ECUs. ► A standardized interface allows to read this information independently of the car manufacturer.
OIL	OSEK/VDX Implementation Language
on-board diagnostics	See OBD
operating mode	The Rte specification distinguishes two operating modes: *compatibility mode* and *vendor mode*. Compatibility mode guarantees that software components

	compiled against a contract phase application header file of one Rte Generator operating in compatibility mode are compatible with an `Rte` generated by any other Rte Generator operating in compatibility mode. Vendor mode enables vendor-specific optimizations, but the same Rte Generator has to be used in the contract- and in the generation phase.
operation cycle	The processing of diagnostic events within the `Dem` module is done within different periods, so called *operation cycles*, such as *power up* or *driving cycle*.
operation invoked event	An operation invoked event is issued by the `Rte` to notify the server in a client/server connection that an operation prototype has been invoked. An operation invoked event is only allowed to trigger a runnable entity. It is not allowed to resolve a wait point. runnable entities that are triggered by operation invoked events are also called server runnable entities. They might have a different signature than runnable entities triggered by other RTE events, depending on the argument prototypes of the operation prototype corresponding to the operation invoked event.
operation prototype	An operation prototype is aggregated by a client/server interface. It represents the operation prototype that is provided/can be invoked in a port that provides/requires the client/server interface. An operation prototype can use any of the possible application errors that are defined in the client/server interface. The operation prototype can specify argument prototypes that are passed between client and server when the operation prototype in a require port is invoked.
`Os`	`Operating System` module: AUTOSAR `Os` is an OSEK-compatible operating system with several extensions. It features: schedule tables, applications, memory protection (if supported by the processor) and timing protection It is part of the system services.
OS	Operating system
OSEK	Stands for *Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug*, which means *Open Systems and their Interfaces for the Electronics in Motor Vehicles*. OSEK is a standards body that has produced specifications for an embedded operating system, a communications stack, and a network management protocol for automotive embedded systems. It has also produced other related specifications. OSEK was designed to provide a standard software architecture for the various electronic control units (ECUs) throughout a car.
OSEK communication	Uniform communication environment for automotive control unit application software.

P

PCI	Protocol Control Information; the part of a PDU that contains control information.
PDU	Protocol data unit. Signals are packed into PDUs for transmission to save bus bandwidth. A PDU consists of PCI (Protocol Control Information) and SDU (Service Data Unit). Depending on the layer the PDU carries a prefix, i.e. *I* for Interaction Layer, *N* for Network Layer, *L* for Data Link Layer.
`PduR`	`PDU Router` module: The `PDU Router` (`PduR`) is responsible for routing protocol data units (see also PDU) from the lower layer modules (`Can`, `Lin`, `Fr`) towards the upper layer modules `Dcm` and `Com`. It also provides gateway functionality by routing PDUs among the various buses (CAN, FlexRay, LIN etc). The `PDU Router` is par of the communication services in the service layer.
per instance memory	The per instance memory defines an instance-specific block of memory. All runnable entities of a software component's instance can access the per instance memory, but data consistency is not guaranteed for accesses to the per instance memory. The per instance memory is specified as part of an atomic software component's internal behavior. Per instance memory blocks are typed. The data type is part of the per instance memory specification.
perspective	In Eclipse-based tools such as EB tresos Studio and EB tresos Debug & Trace, each Workbench window contains one or more perspectives. A perspective defines the set and layout of views in the Workbench window and controls what appears in certain menus and tool bars. Within the Workbench, each perspective shares the same set of editors. Each perspective provides a set of functionality aimed at accomplishing a specific type of task or works with specific types of resources. Most perspectives in the Workbench are comprised of an editor area and one or more views.
PIL	Public Interface Library
port	port A software component has ports, through which the software component can interact with other software components. A port always belongs to exactly one software component and represents a point of interaction between a software component and other software components.
`Port`	`Port Driver` module: The `Port Driver` initializes all port pins of the CPU. It allows to configure the pin mode (`Dio`, `Adc`, `Pwm`), the pin direction

	(in/out), the pin level (high/low) and whether the direction is changeable at runtime or not. To configure a port/pin the user has to know the pin number of the CPU package (for example in 208-pin QFP or BGA). This pin number can be assigned to a user-specific symbolic name, so that this name can be used in the software. The `Port Driver` is part of the microcontroller abstraction layer.
port-defined argument values	Port-defined argument values provide a mechanism to pass implicit arguments to a server runnable that remain hidden for the client. This mechanism is used extensively in client/server communication between a software component and basic software modules (AUTOSAR `Services`, `IO Hardware Abstraction`). In order to use port-defined argument values a port argument list has to be specified as part of the server software component's internal behavior. The port argument list refers to the port that provides an operation prototype and contains an ordered list of primitive values that are included by the `Rte` between the instance handle (if existent) and the operation-specific parameters (argument prototypes) whenever the runnable entity is invoked. In order to ensure that the port-defined argument values are specific for each client software component that is connected to the server software component, the client/server connections have to be 1:1, i.e. a separate provide port is necessary for each client.
post build loadable	The same basic software code is flashed for several electronic control (ECU) variants. The configuration-dependent data is flashed afterwards and is adapted to a specific ECU variant. This process is called post build loadable. Unlike the post build selectable, the post build loadable configuration data are stored just once but may be overwritten in flash.
post build process	The AUTOSAR post build process allows to flash configuration data independently of the code. The complete basic software (BSW) code can be flashed onto an electronic control unit. After that the configuration-dependent data are flashed. AUTOSAR defines two kinds of post build: *post build loadable* and *post build selectable*. Both of them enable the original equipment manufacturer (OEM) to handle variants for the ECUs.
post build selectable	During ECU configuration, users store several sets of a configuration. They do this with the multiple configuration container. One of these configurations is selected during start-up. This process is called post-build selectable.
Problems view	The **Problems** view is a EB tresos Studio view that displays the messages found in the configuration of a loaded project. The default location of the **Problems** view is on the bottom of the EB tresos Studio main window.
Project Explorer view	The **Project Explorer** view is a view in EB tresos Studio It displays all EB tresos Studio configuration projects in a workspace and allows you to edit

projects and open **Editor** views. The default location of the **Project Explorer** view is on the left of the screen.

provide port prototype — A provide port prototype is part of a software component type (atomic software component type or composition type) and is categorized by an interface. If the interface is a sender/receiver interface, all data element prototypes that belong to the interface can be sent through the provide port. If the interface is a client/server interface, all operations prototypes that belong to the interface are available for calls in the provide port. If the interface is a mode switch interface, mode switches for all the modes declared in the mode declaration group prototypes of the interface, can be notified through the provide port.

`Pwm` — `PWM Driver` module: The `PWM Driver` provides services for the initialization and control of the microcontroller internal pulse width modulation stage. The `Pwm` module generates pulses with variable pulse width. It allows the selection of the duty cycle and the signal period time. The `PWM Driver` is part of the microcontroller abstraction layer.

Q

queued receiver COM specification — A queued receiver COM specification specifies communication attributes for a queued data element prototype that is received by a component in a require port. For an queued data element prototype, the attribute `queue length` (mandatory) must be specified.

queued sender COM specification — A queued sender COM specification specifies communication attributes for a queued data element prototype that is sent by a component prototype in a provide port. For a queued sender data element prototype, the attribute `transmission acknowledgement` (optional) can be specified.

queue length — A queue length has to be defined for the queued reception of data element prototypes in the queued receiver COM specification and for the queueing of server calls of operation prototypes in the server COM specification.

R

RAM — Random access memory

RAM block — The RAM block is a basic storage object. It represents the part of an NVRAM block which resides in the RAM. The RAM block is a mandatory part of an NVRAM block.

`RamTst` — `RAM Test` module: The `RAM Test` implements a functional test of microcontroller internal RAM cells. It is part of the memory drivers in the microcontroller abstraction layer.

redundant NV block	The redundant NV block is a configurable type of a NV block.
require port prototype	A require port prototype is part of a software component type (atomic software component type or composition type) and is categorized by an interface. If the interface is a sender/receiver interface, all data element prototypes that belong to the interface can be received in the require port. If the interface is a client/server interface, all operation prototypes that belong to the interface can be invoked in the require port. If the interface is a mode switch interface, the software component is notified of mode switches for all the modes declared in the mode declaration group prototypes of the interface in the require port and can react accordingly (through mode switch events or mode disabling dependencies).
resource file	Resource files are contained in the EB tresos AutoCore platform plug-in and contain information used to tailor the EB tresos AutoCore to the peculiarities of the hardware platform it is shipped for.
ROM	Read-only memory.
ROM block	The ROM block is a basic storage object. It represents the part of the NVRAM block which resides in the ROM. The ROM block is an optional part of an NVRAM block.
`Rte`	`Run-Time Environment`: The `Run-Time Environment` (`Rte`) is at the heart of the AUTOSAR architecture. The AUTOSAR Rte provides a hardware-independent communication infrastructure for application software (AUTOSAR software components) in an AUTOSAR system on the one hand, and a run-time environment for the software component's runnable entities on the other hand. The communication infrastructure provided by the Rte enables communication between AUTOSAR software components as well as communication between AUTOSAR software components and certain basic software modules (AUTOSAR `Services` and `IoHwAb`). The `Rte` is generated for each ECU to meet the specific needs of the application software on that ECU while keeping code and run-time overhead at a minimum. In order to provide its functionality, the AUTOSAR `Rte` uses services of the `Os` and of `Com`. Furthermore the `Rte` connects the software components with the service modules of the basic software (`NvM`, `WdgM`, `ComM`, `EcuM`, `Det`, `Dem`, `Dcm`, `FiM`).
`Rte configuration` header file	The `Rte configuration` header file contains the configuration of the VFB tracing (globally enabled or disabled) and the individual configuration of all enabled VFB tracing events.
RTE event	RTE events allow an event-based activation of runnable entities. runnable entities can be started in response to an RTE event or they can be woken up at a wait point (not possible for all RTE events). RTE events are part of a soft-

	ware component's internal behavior. The AUTOSAR meta model distinguishes timing events, data received events, data receive error events, data send completed events, operation invoked events, asynchronous server call return events, and mode switch events.
`Rte` header file	A static header file of the `Rte`. It defines static information such as version information, vendor and module identification and error codes.
RTE phase	The second of the two generation phases distinguished by an Rte Generator. The whole `Rte` is generated in the second phase, including the application header files and AUTOSAR types header files, the `VFB tracing` header file, the Rte configuration header file and the C-code of the generated Rte itself. Atomic software components that are delivered as source-code components are compiled against the application header files that are generated in the generation phase. If atomic software components that are delivered as object-code components were compiled against application header files generated in compatibility mode, the Rte Generator has to use compatibility mode for the generation phase as well. The object code of all software components (object-code and source-code components) is linked to the code of all basic software modules and of the `Rte` to build the ECU software.
runnable entity	A runnable entity is part of a software component's internal behavior and serves to structure the software component's functionality. It can be executed and scheduled independently from other runnable entities of the software component. Each runnable entity should be triggered by at least one RTE event, otherwise it will never be executed by the Rte (an exception are runnable entities of complex drivers, services and ECU abstraction which might be triggered by external events, e.g. interrupts). A runnable entity is implemented by a C-function. The runnable entity's entry point (`symbol` attribute) specifies the name of the C-function. The runnable entities of each software component Instance have to be mapped onto Os tasks, except server runnables that can be invoked as a direct function call.
Rx-	Stands for receive, e.g. in Rx-PDU.

S

`SchM`	`BSW Scheduler` module: `BSW Scheduler` is the central integration entity of the AUTOSAR standard core. Hence, this central module provides means to: embed BSW module implementations into the AUTOSAR `Os` context, trigger main processing functions of the BSW modules and apply data consistency mechanisms for the BSW modules. The `BSW Scheduler` is part of the system services and therefore provides services for the periodic activation of

	functions and locking services of global data for all modules of all layers. The `BSW Scheduler` is part of the system services in the service layer.
SDU	Service Data Unit; the part of a PDU that contains the data.
security access	The security access between a diagnostic tester and an ECU prevents unauthorized access to an ECU. The `Dcm` module handles the security access on the ECU side.
sender/receiver interface	A sender/receiver interface aggregates all data element prototypes that are sent/received through a provide/require port and all mode declaration group prototypes that can be notified in a mode port (mode switch interface).
sensor-actuator software component	A sensor-actuator software component type is a special kind of atomic software component type that interacts with sensors/actuators on the ECU via the `I/O Hardware Abstraction`.
server COM specification	A server COM specification specifies communication attributes for an operation prototype that can be invoked in a provide port. For an operation prototype, the attribute `queue length` (mandatory) has to be specified.
server runnable entity	A runnable entity that implements an operation prototype provided in a provide port with a client/server Interface. A server runnable entity is triggered by an operation invoked event. A single server runnable entity can implement an operation prototype that is provided in several provide ports. Under certain circumstances a server runnable entity does not have to be mapped to a task but can be invoked as a direct function call, which means that it is executed in the task context of the client.
service component	A service component is a special type of an atomic software component. A service component type specifies the external view of an AUTOSAR service. A service component prototype is the instance of a service component type. In contrast to application component prototypes, service component prototypes are not instantiated within a composition type but within a special ECU software composition.
service module	A service is a basic software module which offers services to other BSW modules or to the application such as the `ECU State Manager`, the `NVRAM Manager` , the `Com Manager`, the `Diagnostic Event Manager`, the `Development Error Tracer`, `Diagnostic Communication Manager`, the `Function Inhibition Manager`, or the `Watchdog Manager`. The interfaces of the service modules to other basic software modules are implemented as standard C API, whereas the communication with the application is performed through standardized AUTOSAR interfaces via the `Rte` (client/server or sender/receiver communication). So from the application's point of view, the service module is a software component. This means a software

	component description of the service module must be available. The ports of the application software component instances and the service component prototypes must be connected by service connector prototypes.
Sidebar view	The **Sidebar** view is a EB tresos Studio view that displays assistance dialogs for your current task. Per default the **Sidebar** view is located in the upper right corner of the EB tresos Studio main window, as a tab behind the **Workflows** view.
snapshot data	See freeze frame
software component	This term is often used as a synonym for atomic software components, sensor-actuator software components or composite components. A software component encloses software that realizes a particular functionality. Communication with other software components is possible through ports (external view, described by the software component's type). Its functionality is provided by runnable entities (internal view, described by the software component's internal behavior). AUTOSAR allows a hierarchical structure of software components. Atomic software components are the smallest non-dividable software entities. Sensor-actuator software components are special atomic software components that control a sensor or actuator on the ECU.
software component description	SWC-D; one or more XML files which describe the architecture, e.g. port, of the software components. The format is standardized by AUTOSAR.
`Spi`	`SPI Driver` module: The `SPI Driver` provides services for reading from and writing to devices connected via SPI buses. It provides access to SPI communication for several users. It also provides the required mechanism to configure the on-chip SPI peripheral. The `SPI Driver` is part of the communication drivers in the microcontroller abstraction layer.
SWC	See software component
SWC-D	software component description
synchronous server call point	Property of a runnable entity that indicates that the runnable entity synchronously invokes an operation prototype in the provide port of a client/server interface by using the appropriate Rte API call. Synchronous invocation of an operation prototype implies that the Rte API call blocks until the execution of the server has terminated.
system description	One or more XML files which describe a system. The system description includes ► hardware topology ► communication matrix

- software architecture
- software component to ECU mapping
- data mapping

Typically, the system description contains all software component descriptions of the system. The format is standardized by AUTOSAR.

T

TAL — Target Access Library

task — A task is an entity which is under the control of the AUTOSAR operating system. It provides the entry point for a functional unit of an AUTOSAR program and is scheduled according to the Os configuration.

tester — A tester, also called *diagnostic tester* or *(external) diagnostic tool*, is a device that is temporarily connected to a vehicle network. It allows to establish a diagnostic session to all ECUs that support diagnostics. A tester is used for

- developing
- manifacturing
- service, e.g. in a garage

A tester device offers the following functionalities:

- reads the diagnostic events for troubleshooting and service in a garage,
- reads emission-relevant data,
- reflashes an ECU with a new software version.

thread — A thread is an operating system-internal functional unit. It is executed and scheduled by the OS similar to tasks and interrupt service routines (interrupt service routine s). In the EB tresos Safety OS, tasks and ISRs are mapped to threads.

timing event — A timing event is an RTE event that allows periodic activation of a runnable entity. Its period is specified as an attribute of the timing event. A runnable entity cannot have a wait point for a timing event.

timing protection — In an operating system, timing protection is a mechanism to provide runtime supervision of tasks or interrupt service routines (interrupt service routine s).

TLB — translation look-aside buffer: A translation look-aside buffer (TLB) is a register in the memory management unit (MMU), which provides the mapping be-

	tween virtual and physical memory addresses together with the access permissions.
trace event	Element of the VFB tracing mechanism provided by the `Rte`. Trace hook functions are provided for each trace event. The `Rte` distinguishes the following trace events: - Rte API start (when an Rte API call is made) - Rte API return (before an Rte API call returns) - signal transmission (before Com_SendSignal or Com_UpdateShadowBuffer is invoked) - signal reception (after Com_ReceiveSignal or Com_ReceiveShadowBuffer has returned) - Com callback (when a Com callback function is invoked) - task activate (was invoked before a task containing runnable entities is activated) - task dispatch (invoked when an Rte-generated task begins execution) - set Os event (invoked before Rte code sets an Os event) - wait Os event (invoked before Rte code invokes the WaitEvent Os service) - received Os event (invoked after Rte code had returned from waiting for an Os event) - runnable entity invocation (invoked before runnable entity starts execution) - runnable entity termination (invoked after runnable entity has terminated execution).
trace hook function	Function that is automatically invoked by the `Rte` when the corresponding trace event occurs (if trace event enabled). No implementation is generated by the Rte Generator for the trace hook functions. The user must provide the implementation.
transmission acknowledgment	Transmission acknowledgment is specified as part of the sender COM specification. If transmission acknowledgment is specified for a data element prototype, the sender of the data element prototype is notified whether the new value has been sent successfully or not.
Tx-	Stands for transmit, e.g. Tx-PDU.

U

UDS	Unified diagnostic services. UDS defines the diagnostic services, which are used between a diagnostic tester and an ECU. The UDS requirements are satisfied by the `Dcm` module. The UDS standard is specified by the International Organization for Standardization (ISO) within ISO 14229-1.
unqueued receiver COM specification	An unqueued receiver COM specification specifies communication attributes for an unqueued data element prototype that is received by a component in a require port. For an unqueued data element prototype, the attributes `alive timeout` (optional), `init value` (optional) and `handle invalid` (optional) can be specified. Furthermore, an unqueued receiver COM specification allows to specify reception filters for a data element with data semantics.
unqueued sender COM specification	An unqueued sender COM specification specifies communication attributes for an unqueued data element prototype that is sent by a component prototype in a provide port. For an unqueued data element prototype, the attributes `canInvalidate` (optional), `init value` (optional) and `transmission acknowledgement` (optional) can be specified.

V

VFB	Virtual functional bus; AUTOSAR concept that describes the the top level communication behavior. The `Rte` implements the VFB.
VFB tracing	VFB tracing is a mechanism provided by the `Rte` that allows to monitor the interaction of software components with the VFB.
VFB tracing header file	The `VFB tracing` header file contains the prototypes of the trace hook functions for the trace events that are configured in the `Rte configuration` header file. the `VFB tracing` header file is included by the generated Rte and must be included in the module(s) that define the configured trace hook functions.

W

wait point	Property of a runnable entity that indicates that the runnable entity waits for the occurrence of an RTE event or for a timeout to expire. Not all RTE events can resolve a wait point. Wait points are restricted to asynchronous server call returns events, data received events and data send completed events.

`Wdg`	`Watchdog Driver` module: The `Wdg` provides services for initialization, changing the operation mode and triggering the watchdog. The functional requirements and the functional scope are the same for both internal and external watchdog drivers. Hence the API is semantically identical. The `Watchdog Driver` is part of the microcontroller abstraction layer.
`WdgIf`	`Watchdog Interface` module: The `Watchdog Interface` abstracts from the `Watchdog Driver`(s) to make the `WdgM` access to the lower layers hardware (ECU) independent. The `Watchdog Interface` is an on-board device abstraction.
`WdgM`	`Watchdog Manager` module: The task of the `Watchdog Manager` is to monitor the correct operation of system components (called *entities* in the SWS) and to trigger a hardware watchdog on a periodic basis. Different operating modes can be configured: The `Watchdog Manager` can monitor a single entity, multiple entities or no entity at all. It is executed periodically. The `Watchdog Manager` is part of the service layer.
Workflows view	The **Workflows** view is a view of EB tresos Studio. It displays workflows which guide you to accomplish a certain task, e.g. to configure a Com stack. The **Workflows** view is located in the upper right corner of the EB tresos Studio main window.

X

XML	E*x*tensible *M*arkup *L*anguage