Bus Custom Code Interface

Bus Custom Code Interface Handling

Release 2021-A – May 2021

dSPACE

How to Contact dSPACE

Mail:	dSPACE GmbH Rathenaustraße 26 33102 Paderborn Germany
Tel.:	+49 5251 1638-0
Fax:	+49 5251 16198-0
E-mail:	info@dspace.de
Web:	http://www.dspace.com

How to Contact dSPACE Support

If you encounter a problem when using dSPACE products, contact your local dSPACE representative:

- Local dSPACE companies and distributors: http://www.dspace.com/go/locations
- For countries not listed, contact dSPACE GmbH in Paderborn, Germany.
  Tel.: +49 5251 1638-941 or e-mail: support@dspace.de

You can also use the support request form: http://www.dspace.com/go/supportrequest. If you are logged on to mydSPACE, you are automatically identified and do not need to add your contact details manually.

If possible, always provide the relevant dSPACE License ID or the serial number of the CmContainer in your support request.

Software Updates and Patches

dSPACE strongly recommends that you download and install the most recent patches for your current dSPACE installation. Visit http://www.dspace.com/go/patches for software updates and patches.

Important Notice

Contents

About This Document

Content

Introduces you to the Bus Custom Code interface. The Bus Custom Code interface is a common API interface provided by dSPACE that is supported by bus implementation software, such as the Bus Manager or the FlexRay Configuration Package. By using the Bus Custom Code interface, you can integrate user code implementations that provide user-specific bus functionality in bus implementation software.

This document introduces you to the fields of application and to the basic principles of working with the Bus Custom Code interface. Additionally, it provides reference information on the handles and functions of the Bus Custom Code interface.

Target group

This document is primarily targeted at engineers who implement user-specific bus functionality via user code implementations in bus implementation software.

Required knowledge

You must be familiar with the C programming language.

Symbols

dSPACE user documentation uses the following symbols:

Symbol	Description
DANGER	Indicates a hazardous situation that, if not avoided, will result in death or serious injury.
WARNING	Indicates a hazardous situation that, if not avoided, could result in death or serious injury.
CAUTION	Indicates a hazardous situation that, if not avoided, could result in minor or moderate injury.
NOTICE	Indicates a hazard that, if not avoided, could result in property damage.
Note	Indicates important information that you should take into account to avoid malfunctions.

Symbol	Description
Tip	Indicates tips that can make your work easier.
	Indicates a link that refers to a definition in the glossary, which you can find at the end of the document unless stated otherwise.
	Precedes the document title in a link that refers to another document.

Naming conventions

dSPACE user documentation uses the following naming conventions:

%name% Names enclosed in percent signs refer to environment variables for file and path names.

< > Angle brackets contain wildcard characters or placeholders for variable file and path names, etc.

Special folders

Some software products use the following special folders:

Common Program Data folder A standard folder for application-specific configuration data that is used by all users.

```
%PROGRAMDATA%\dSPACE\<InstallationGUID>\<ProductName>
```

or

```
%PROGRAMDATA%\dSPACE\<ProductName>\<VersionNumber>
```

Documents folder A standard folder for user-specific documents.

```
%USERPROFILE%\Documents\dSPACE\<ProductName>\
<VersionNumber>
```

Local Program Data folder A standard folder for application-specific configuration data that is used by the current, non-roaming user.

```
%USERPROFILE%\AppData\Local\dSPACE\<InstallationGUID>\
<ProductName>
```

Accessing dSPACE Help and PDF Files

After you install and decrypt dSPACE software, the documentation for the installed products is available in dSPACE Help and as PDF files.

dSPACE Help (local) You can open your local installation of dSPACE Help:

- On its home page via Windows Start Menu
- On specific content using context-sensitive help via **F1**

dSPACE Help (Web) You can access the Web version of dSPACE Help at www.dspace.com/go/help.

To access the Web version, you must have a *mydSPACE* account.

PDF files You can access PDF files via the icon in dSPACE Help. The PDF opens on the first page.

Basics on the Bus Custom Code Interface

Where to go from here

Information in this section

Introduction to the Bus Custom Code Interface

Overview

The Bus Custom Code interface is a dSPACE API interface that is supported by bus implementation software, such as the Bus Manager or the FlexRay Configuration Package. By using the Bus Custom Code interface, you can implement user code in executable applications, i.e., in real-time applications and/or offline simulation applications. Via user code, you can provide additional functionality to the bus implementation software, such as user-specific algorithms for calculating authentication information in secure onboard communication scenarios. However, the additionally supported functionality depends on the specific bus implementation software.

Because the Bus Custom Code interface is supported by various dSPACE software tools, a user code implementation can be used by all of these dSPACE

software tools and on various dSPACE platforms, such as SCALEXIO, MicroAutoBox II/III, PHS-bus-based systems, or VEOS.

Bus implementation software supporting the Bus Custom Code interface

The Bus Custom Code interface is supported by bus implementation software, i.e., by the following dSPACE software tools:

- ConfigurationDesk
- Bus Manager (stand-alone)
- RTI CAN MultiMessage Blockset
- FlexRay Configuration Package
- Ethernet Configuration Package

Depending on the dSPACE software tool, not all of the functionalities provided by the Bus Custom Code interface might be supported.

User code and the Bus Custom Code interface

The Bus Custom Code interface lets you implement user code and its functionalities in executable applications, i.e., in real-time applications and/or offline simulation applications.

User code User code is C code or C++ code that contains user-specific algorithms. You can use user-specific algorithms to provide additional functionality to the bus implementation software, for example, for calculating checksum or counter signal values or generating authentication information in secure onboard communication (SecOC) scenarios. A user code implementation consists of one or more source files (*.c, *.cpp) and optional include files (*.h, *.hpp).

Bus Custom Code interface The Bus Custom Code interface is a C-based API interface provided by dSPACE. The interface is required to implement user code and its functionalities in executable applications. In general, the Bus Custom Code interface lets you do the following:

- Specify a user code ID for each user code implementation. This ID is required to unambiguously reference specific user code implementations in the bus implementation software.
- Initialize the functionalities provided by user code in the executable application.
- Exchange data between the user code and the bus implementation software, for example, to access properties of secured IPDUs or write calculated checksum values to ISignals of a PDU.

For more information, refer to Basic Principles of the Bus Custom Code Interface on page 11.

Implementing user code in executable applications

To implement user code in executable applications, you must extend the user code by functions of the Bus Custom Code interface, e.g., to write data to or read data from the user code. For more information, refer to Integrating Functions of the Bus Custom Code Interface in the User Code on page 17.

Additionally, you must add the user code implementation (e.g., source files and include files) to the bus implementation software. For more information, refer to:

Bus Implementation Software	Refer to
ConfigurationDesk: Bus Manager in ConfigurationDesk	Working with User Code (ConfigurationDesk Bus Manager Implementation Guide 🕮)
Bus Manager (stand-alone)	Working with User Code (Bus Manager (Stand-Alone) Implementation Guide 🕮)
RTI CAN MultiMessage Blockset	Secure Onboard Communication Page (RTICANMM MainBlock) (RTI CAN MultiMessage Blockset Reference 🕮)
FlexRay Configuration Package	General Page (FlexRay Configuration Tool Reference 🕮)
Ethernet Configuration Package	Contact dSPACE Support (www.dspace.com/go/supportrequest).

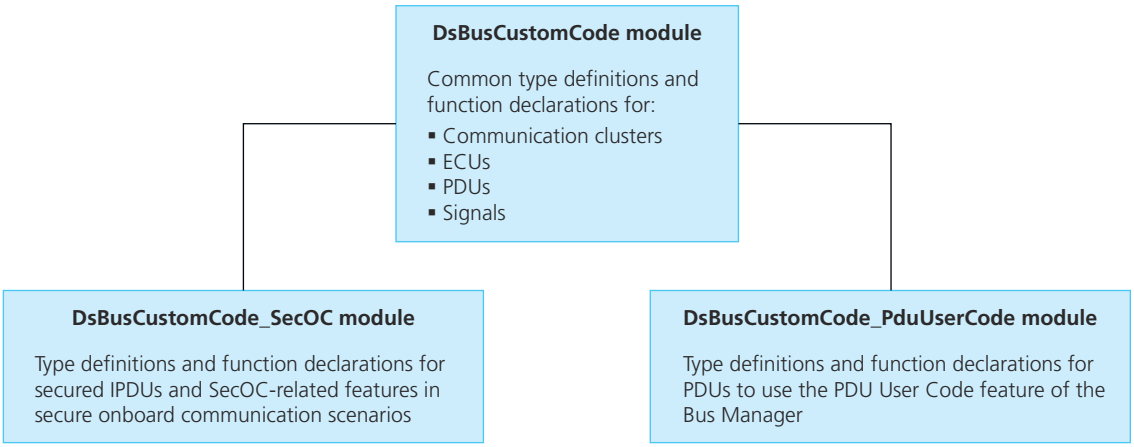
Basic Principles of the Bus Custom Code Interface

Modules of the Bus Custom Code interface

The Bus Custom Code interface is a C-based API interface that consists of three modules:

Module	Purpose
`DsBusCustomCode`	Provides common type definitions and function declarations that are independent of a specific user scenario.
`DsBusCustomCode_SecOC`	Provides type definitions and function declarations that are required if you want to implement secure onboard communication (SecOC) via user code in executable applications.
`DsBusCustomCode_PduUserCode`	Provides type definitions and function declarations that are required if you want to implement additional PDU-based functionalities via user code in executable applications, such as user-specific algorithms for calculating checksum signal values. **Note** The `DsBusCustomCode_PduUserCode` module can be used only with the Bus Manager, i.e., only the Bus Manager can parameterize the functions of this module. You cannot use this module with any other bus implementation software.

The following illustration provides an overview of the three modules.



Addressing elements via handles

The Bus Custom Code interface uses handles to address elements such as PDUs and access their data at run time. For each element that is affected by the user code, a unique handle is generated during the initialization phase of the executable application (i.e., real-time application or offline simulation application). Via these handles, functions of the Bus Custom Code interface can get and set data, even across subsequent function calls. The generated handles are persistent at run time.

Syntax of Bus Custom Code functions

Almost all functions provided by the Bus Custom Code interface use the same syntax, as shown in the following example:

```
/* Get the length of a PDU */
DsBusCustomCodePdu_getSduLength(pduHandle, &IPdu_SduLength);
|      1      |2|3|   4   |    5    |        6        |
```

```
/* Set a pointer to a user-defined data structure related to a PDU feature */
DsBusCustomCodePduFeature_setFeatureDataPtr(PduFeatureHandle, featureDataPtr);
|      1      |   2   |3|    4    |        5        |        6        |
```

Position	Purpose
1	Constant prefix Used by all functions of the Bus Custom Code interface.
2	Handle identification Determines the handle type that is addressed by the function and expected as the function's first parameter.
3	Get/set definition Determines whether data is read from (`get`) or written to (`set`) the element that is addressed by the function.
4	Addressed element Determines the element that is addressed by the function and expected as the function's second parameter.

Position	Purpose
5	First function parameter Handle type that is expected according to position 2.
6	Second function parameter Element that is addressed according to position 4.

Working with return values

The Bus Custom Code interface provides return values that indicate whether functions of the Bus Custom Code interface are executed correctly. For example, the return values can indicate that a required parameter is not specified or an addressed handle is invalid.

However, the return values of the Bus Custom Code interface do not provide the results of user code functionalities, e.g., whether a received counter value matches the expected counter value.

For this purpose, you can specify user-defined return values. You can pass such user-defined return values to suitable `set` functions of the Bus Custom Code interface, e.g., to the `DsBusCustomCodeUserCodePduFeature_setResult` function of the `DsBusCustomCode_PduUserCode` module.

Parameter data of Bus Custom Code functions

In general, the functions of the Bus Custom Code interface can be used for the following purposes:

- Exchange data between the code that is generated by the bus implementation software and the user code.
- Store run-time data and provide it to the user code during subsequent function calls. In this case, the data is handled in the user code and not directly exchanged with the bus implementation software code. For example, this applies to data that is accessed by the user code via a data pointer.

The data that is exchanged via the bus implementation software code and the user code can have different data sources, as shown in the following table.

Data Source	Description
Communication matrix, e.g., AUTOSAR-specific attributes	The data is provided to the user code via the bus implementation software code if the following conditions are fulfilled: ▪ The required attributes and their values are specified in the communication matrix. ▪ The related bus implementation software evaluates the required attributes.
Settings that are specified by the user in the bus implementation software, e.g., constant property values or enabled run-time parameters	Depending on the settings and the bus implementation software, the bus implementation software code can provide the data to the user code as follows: ▪ Constant value ▪ TRC file variable ▪ Model variable

Data Source	Description
Run-time data that is received via the bus, e.g., current PDU data	The bus implementation software code can pass this data directly to the user code.
User code, e.g., authentication information that is calculated in the user code	Depending on the bus implementation software, the bus implementation software code can provide the received data via TRC file variables and/or model variables to the executable application.

If a Bus Custom Code get function is called and no data source provides data to a function parameter, the function returns `PARAMETER_NOT_SET` and no data is written to the function's output parameter.

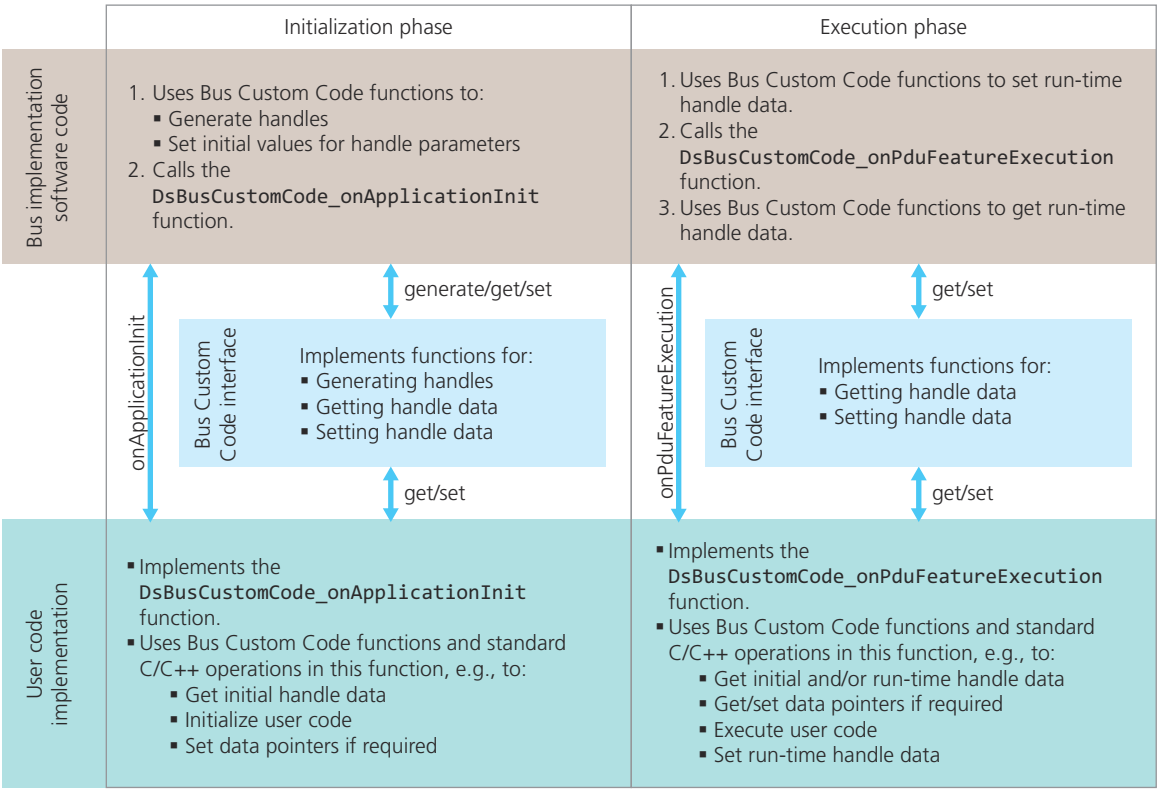
Tip

Depending on the relevance of individual Bus Custom Code functions in your use scenario, it might be useful to specify a default value beforehand in case a function returns `PARAMETER_NOT_SET`. This increases the fault tolerance of the user code, for example, if an addressed communication matrix attribute is not specified in the communication matrix.

Run-Time Behavior of Code That Uses Functions of the Bus Custom Code Interface

Initialization phase and execution phase

In general, an initialization phase and an execution phase can be distinguished to characterize the run-time behavior of code that uses functions of the Bus Custom Code interface, as shown in the following illustration.



Initialization phase During the initialization phase of the executable application (i.e., real-time application or offline simulation application), the code that is generated by the bus implementation software uses functions of the Bus Custom Code interface to generate and initialize handles and their parameters. Then, the code calls the `DsBusCustomCode_onApplicationInit` function.

The `DsBusCustomCode_onApplicationInit` function is the main function for initializing the user code for each PDU to which the user code is applied to. It must be implemented in the user code implementation. Within this function, Bus Custom Code functions and common C/C++ operations can be used to get initial handle data, specify user-defined data structures, and set data pointers to access the data structures, for example.

Tip

The `DsBusCustomCode_onApplicationInit` function does not influence the real-time performance of the user code during the execution phase. For optimum performance, it is therefore recommended to implement time-consuming operations such as `malloc` or non-deterministic function calls in the `DsBusCustomCode_onApplicationInit` function.

Execution phase When the executable application is running, the execution of the code that is generated by the bus implementation software is PDU-based: When a PDU that is affected by the user code is received or to be transmitted via the bus, the code uses Bus Custom Code functions to write the current run-time

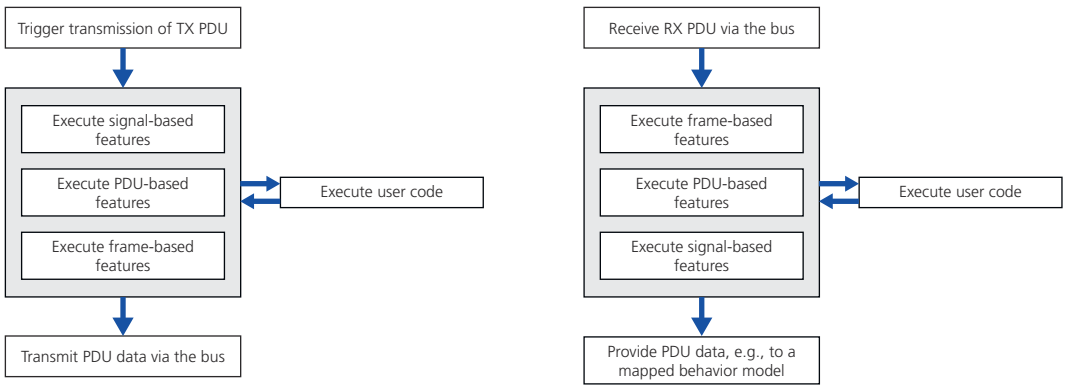
data to the related handles. Then, the code calls the `DsBusCustomCode_onPduFeatureExecution` function.

The `DsBusCustomCode_onPduFeatureExecution` function is the main function for executing the user code. It must be implemented in the user code implementation. Within this function, Bus Custom Code functions and common C/C++ operations can be used to access handle data, provide the data to algorithms of the user code, and execute the user code. If required, you can get or set data pointers to access user-defined data structures, for example. The results of the executed user code can be written to the related handles as new data.

After the `DsBusCustomCode_onPduFeatureExecution` function was executed, the code that is generated by the bus implementation software reads the new handle data and provides it to the executable application, e.g., to transmit the data via the bus.

Execution sequence of user code and other features of bus implementation software

Even though the code that is generated by the bus implementation software is PDU-based, user code can also affect signals and frames of the related PDUs. Additionally, signals, PDUs, and frames can be affected by other features of the bus implementation software, e.g., by bus configuration features of the Bus Manager. Basically, the execution sequence of the user code and other features of the bus implementation software depends on whether the PDU is a TX or an RX PDU, as shown in the following example.



However, the exact behavior depends on the following factors:

- The used bus implementation software and the software-specific architecture to configure bus communication.
- The configured bus communication.

  For example, there can be several calls of user code functions while other features are executed between the calls.
- The PDU type to which the user code is applied.

  For example, if user code for secure onboard communication (SecOC) is applied to a contained IPDU, the execution sequence can be as follows:
  - Before transmission, the user code is applied to the contained IPDU, e.g., the contained IPDU is secured. Then, the contained IPDU is included in the

container IPDU. Before transmission, other features might be applied to the container IPDU and/or its related frame.

- When the container IPDU is received, other features might be applied to the container IPDU and/or its related frame before the contained IPDU is extracted. After the contained IPDU is extracted, the user code is applied to the contained IPDU, e.g., the received authentication information is verified.

Integrating Functions of the Bus Custom Code Interface in the User Code

Overview

To implement user code and its functionalities in executable applications, you must extend the user code by functions of the Bus Custom Code interface. Some functions of the Bus Custom Code interface are mandatory, regardless of the specific use scenario. Additionally, there are mandatory and optional functions depending on the use scenario.

Mandatory user code ID

For each user code implementation, one unique user code ID must be specified. The user code ID is required to unambiguously reference specific user code implementations in the bus implementation software.

To specify the user code ID, exactly one source file of a user code implementation must start with the following definition:

```
#define DS_BUS_CUSTOM_FEATURE_NAME <user code ID>
```

The user code ID must be a unique C-compliant string, i.e., only letters, numbers, and '_' are allowed and the string must be at least one character long.

Note

- There must not be any definitions, includes, function calls, etc. before the definition of the user code ID.
- If a user code implementation consists of multiple source files, only one source file must contain the definition of the user code ID.
- To use functions of the Bus Custom Code interface at run time, their function definitions must be included in the source file that contains the definition of the user code ID.

Mandatory functions

For each user code implementation, the `DsBusCustomCode_onApplicationInit` and `DsBusCustomCode_onPduFeatureExecution` functions must be implemented exactly once. The following example shows the required syntax for implementing the functions.

```
Std_ReturnType DsBusCustomCode_onApplicationInit(DsBusCustomCodePduFeatureHandle PduFeatureHandle)
{
   /* Initialization of the user code */
```

```
    return E_OK;
}

Std_ReturnType DsBusCustomCode_onPduFeatureExecution(DsBusCustomCodePduFeatureHandle PduFeatureHandle)
{
    /* Execution of the user code (PDU-based) */
    return E_OK;
}
```

Note

Both functions must be implemented exactly once in the source file that contains the definition of the user code ID.

The specific initialization and execution of the user code depends on your use scenario.

Tip

Especially if you work with complex user code implementations, it might be useful to implement these functions as empty functions at an early development stage. When you do this and you generate the executable application, source file settings, such as path specifications, includes, etc. are tested. This lets you check at an early development stage whether settings of the source file are implemented correctly.

Use-scenario-specific content

Depending on your use scenario, you must include one of the following include files in your user code implementation:

Use Scenario	Include
Implementing user-specific PDU features in executable applications (e.g., checksum or counter signal value calculation). **Note** This use scenario is supported only by the Bus Manager and not by any other bus implementation software.	`#include <DsBusCustomCode_PduUserCode.h>`
Implementing secure onboard communication in executable applications.	`#include <DsBusCustomCode_SecOC.h>`

Note

The include file must be included in the source file that contains the definition of the user code ID.

Additional required extensions of the user code depend on your use scenario. For examples, refer to:

- Example of Implementing User-Specific PDU Functionalities via the Bus Custom Code Interface on page 25

- Example of Implementing Secure Onboard Communication via the Bus Custom Code Interface on page 32

Overview of the Handles and Their Functions

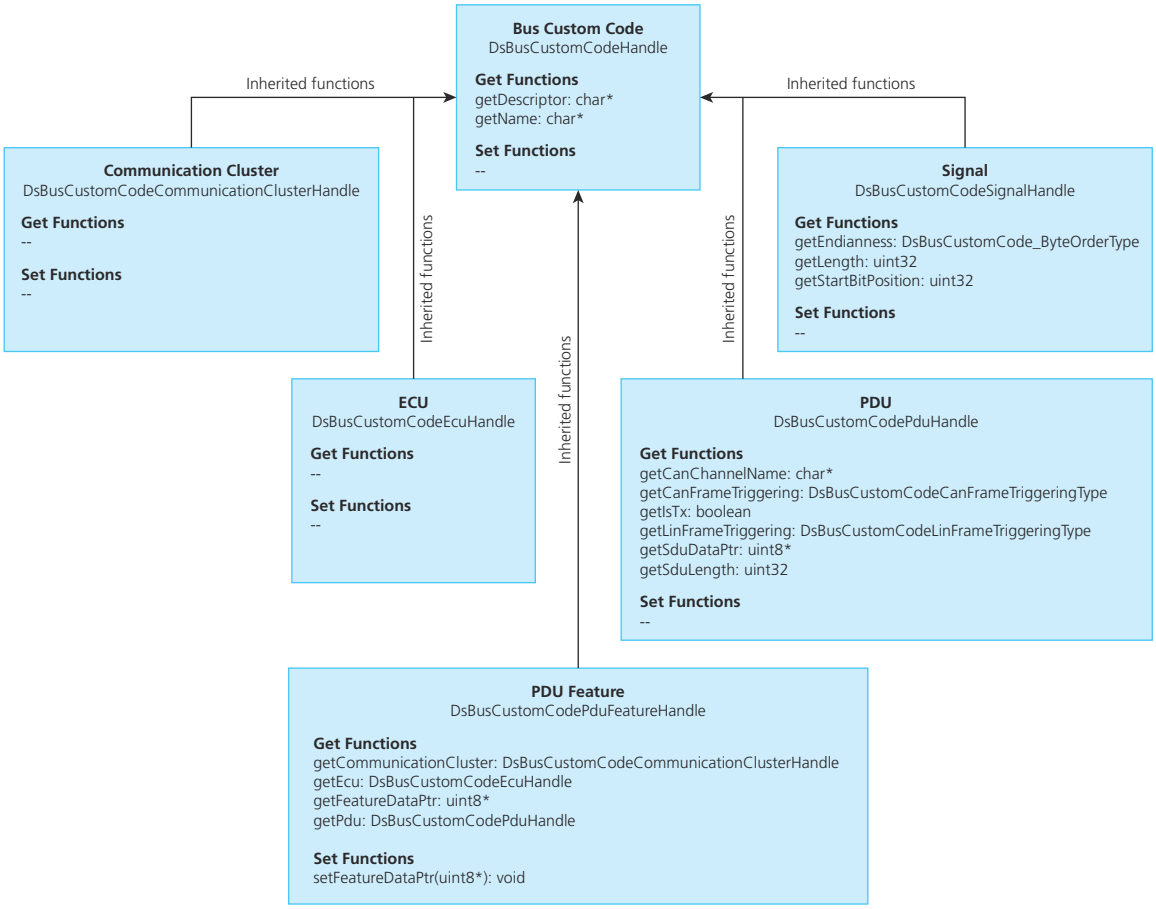
Overview

The Bus Custom Code interface provides a set of handles that let you access elements such as PDUs or PDU features. For specific handles, functions are available that let you access element properties: Get functions provide a read access to properties, set functions provide write access.

The following illustrations provide a graphical overview of the public handles and functions that are provided by the modules of the Bus Custom Code interface.

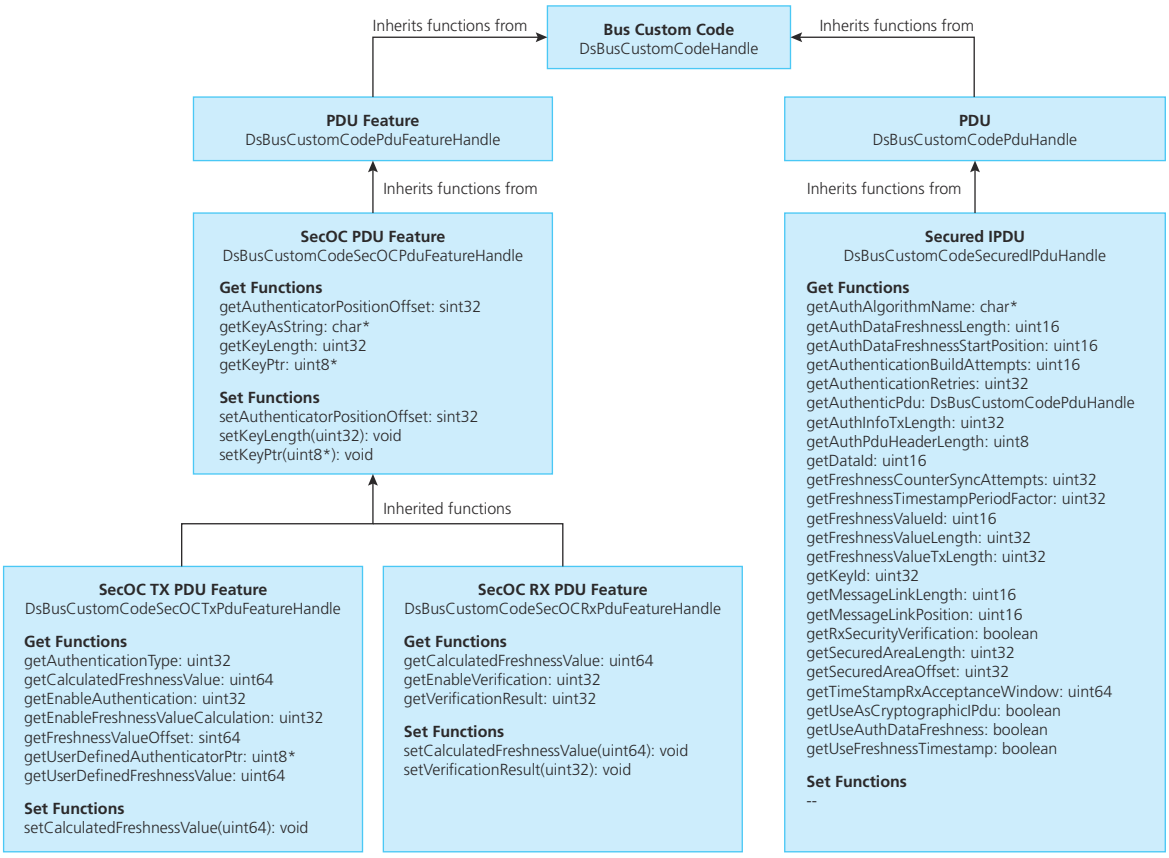
Handles and functions provided by the common DsBusCustomCode module

The following illustration provides an overview of the handles and functions of the common `DsBusCustomCode` module, and the inheritance of the functions.



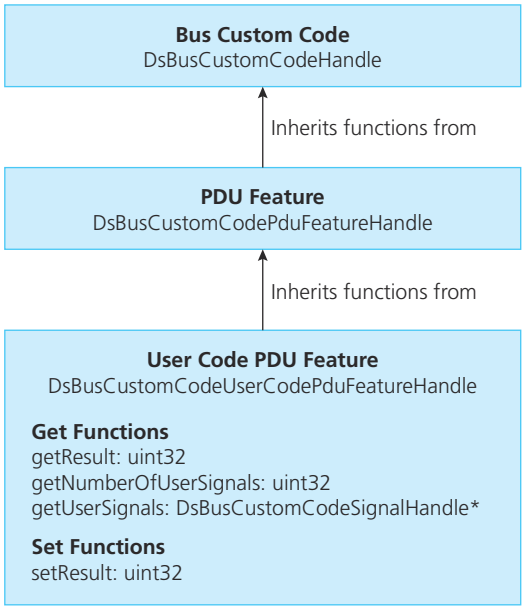
Handles and functions provided by the DsBusCustomCode_SecOC module

The following illustration provides an overview of the handles and functions of the `DsBusCustomCode_SecOC` module, and the inheritance of the functions.



Handles and functions provided by the DsBusCustomCode_ PduUserCode module

The following illustration provides an overview of the handles and functions of the `DsBusCustomCode_PduUserCode` module, and the inheritance of the functions.



Using Bus Custom Code Functions with Bus Implementation Software

Using Bus Custom Code functions with different bus implementation software tools

In general, user code that uses functions of the Bus Custom Code interface can be used with all bus implementation software tools. However, not all functions of the Bus Custom Code interface can be used with each bus implementation software tool. For example, because of the following reasons:

- A function requires data of a specific communication matrix attribute but this attribute is not evaluated by a bus implementation software tool.
- A function requires data from a property of the bus implementation software but this property is not available for a bus implementation software tool.

If a function cannot be used with a bus implementation software tool, the behavior is one of the following:

- The tool sets an internal constant value, which is often `0`. If the user code calls the function, the user code gets this constant value.
- The tool does not parameterize the function parameters at all. If the user code calls the function, the function returns `E_PARAMETER_NOT_SET`.

For more information on function parameters and return values, refer to Basic Principles of the Bus Custom Code Interface on page 11.

Mapping of Bus Custom Code functions to bus implementation software

The following table provides an overview of the public Bus Custom Code functions and with which bus implementation software tool they can be used.

Bus Custom Code Function	Usable with			
	Bus Manager	RTI CAN MultiMessage Blockset	FlexRay Configuration Package	Ethernet Configuration Package
DsBusCustomCode_getDescriptor	✓	✓	✓	✓
DsBusCustomCode_getName	✓	✓	✓	✓
DsBusCustomCodePdu_getCanChannelName	✓	✓	-	-
DsBusCustomCodePdu_getCanFrameTriggering	✓	✓	-	-
DsBusCustomCodePdu_getIsTx	✓	✓	✓	✓
DsBusCustomCodePdu_getLinFrameTriggering	✓	-	-	-
DsBusCustomCodePdu_getSduDataPtr	✓	✓	✓	✓
DsBusCustomCodePdu_getSduLength	✓	✓	✓	✓
DsBusCustomCodePduFeature_getFeatureDataPtr	✓	✓	✓	✓
DsBusCustomCodePduFeature_getPdu	✓	✓	✓	✓
DsBusCustomCodePduFeature_setFeatureDataPtr	✓	✓	✓	✓
DsBusCustomCodeSecOCPduFeature_getAuthenticatorPositionOffset	✓	✓	-	-
DsBusCustomCodeSecOCPduFeature_getKeyAsString	✓	✓	✓	✓
DsBusCustomCodeSecOCPduFeature_getKeyLength	✓	✓	✓	✓
DsBusCustomCodeSecOCPduFeature_getKeyPtr	✓	✓	✓	✓
DsBusCustomCodeSecOCPduFeature_setAuthenticatorPositionOffset	✓	✓	-	-
DsBusCustomCodeSecOCPduFeature_setKeyLength	✓	✓	✓	✓
DsBusCustomCodeSecOCPduFeature_setKeyPtr	✓	✓	✓	✓
DsBusCustomCodeSecOCRxPduFeature_getCalculatedFreshnessValue	✓	✓	✓	✓
DsBusCustomCodeSecOCRxPduFeature_getEnableVerification	✓	✓	✓	✓
DsBusCustomCodeSecOCRxPduFeature_getVerificationResult	✓	✓	✓	✓
DsBusCustomCodeSecOCRxPduFeature_setCalculatedFreshnessValue	✓	✓	✓	✓
DsBusCustomCodeSecOCRxPduFeature_setVerificationResult	✓	✓	✓	✓
DsBusCustomCodeSecOCTxPduFeature_getAuthenticationType	✓	✓	✓	✓
DsBusCustomCodeSecOCTxPduFeature_getEnableAuthentication	✓	✓	✓	✓
DsBusCustomCodeSecOCTxPduFeature_getEnableFreshnessValueCalculation	✓	✓	✓	✓
DsBusCustomCodeSecOCTxPduFeature_getFreshnessValueOffset	✓	✓	✓	✓
DsBusCustomCodeSecOCTxPduFeature_getUserDefinedAuthenticatorPtr	✓	✓	✓	✓
DsBusCustomCodeSecOCTxPduFeature_getUserDefinedFreshnessValue	✓	✓	✓	✓
DsBusCustomCodeSecuredIPdu_getAuthAlgorithmName	✓	-	-	-
DsBusCustomCodeSecuredIPdu_getAuthDataFreshnessLength	✓	-	-	-
DsBusCustomCodeSecuredIPdu_getAuthDataFreshnessStartPosition	✓	-	-	-
DsBusCustomCodeSecuredIPdu_getAuthInfoTxLength	✓	✓	✓	✓

Bus Custom Code Function	Usable with			
	Bus Manager	RTI CAN MultiMessage Blockset	FlexRay Configuration Package	Ethernet Configuration Package
DsBusCustomCodeSecuredIPdu_getAuthPduHeaderLength	✓	✓	-	-
DsBusCustomCodeSecuredIPdu_getAuthenticPdu	✓	✓	✓	✓
DsBusCustomCodeSecuredIPdu_getAuthenticationBuildAttempts	✓	-	✓	-
DsBusCustomCodeSecuredIPdu_getAuthenticationRetries	✓	-	✓	-
DsBusCustomCodeSecuredIPdu_getDataId	✓	✓	✓	✓
DsBusCustomCodeSecuredIPdu_getFreshnessCounterSyncAttempts	✓	-	-	-
DsBusCustomCodeSecuredIPdu_getFreshnessTimestampPeriodFactor	✓	-	✓	-
DsBusCustomCodeSecuredIPdu_getFreshnessValueId	✓	✓	✓	-
DsBusCustomCodeSecuredIPdu_getFreshnessValueLength	✓	✓	✓	✓
DsBusCustomCodeSecuredIPdu_getFreshnessValueTxLength	✓	✓	✓	✓
DsBusCustomCodeSecuredIPdu_getKeyId	✓	✓	✓	✓
DsBusCustomCodeSecuredIPdu_getMessageLinkLength	✓	✓	✓	-
DsBusCustomCodeSecuredIPdu_getMessageLinkPosition	✓	✓	✓	-
DsBusCustomCodeSecuredIPdu_getRxSecurityVerification	✓	-	-	-
DsBusCustomCodeSecuredIPdu_getSecuredAreaLength	✓	-	-	-
DsBusCustomCodeSecuredIPdu_getSecuredAreaOffset	✓	-	-	-
DsBusCustomCodeSecuredIPdu_getTimeStampRxAcceptanceWindow	✓	-	✓	-
DsBusCustomCodeSecuredIPdu_getUseAsCryptographicPdu	✓	✓	✓	-
DsBusCustomCodeSecuredIPdu_getUseAuthDataFreshness	✓	-	-	-
DsBusCustomCodeSecuredIPdu_getUseFreshnessTimestamp	✓	✓	✓	-
DsBusCustomCodeSignal_getEndianness	✓	-	-	-
DsBusCustomCodeSignal_getLength	✓	-	-	-
DsBusCustomCodeSignal_getStartBitPosition	✓	-	-	-
DsBusCustomCodeUserCodePduFeature_getNumberOfUserSignals	✓	-	-	-
DsBusCustomCodeUserCodePduFeature_getResult	✓	-	-	-
DsBusCustomCodeUserCodePduFeature_getUserSignals	✓	-	-	-
DsBusCustomCodeUserCodePduFeature_setResult	✓	-	-	-

Restrictions for Working with the Bus Custom Code Interface

Overview

If you work with the Bus Custom Code interface, the following restrictions apply.

Restrictions for working with the files of the Bus Custom Code interface

The Bus Custom Code interface is specified in the `<bus custom code module name>.c`, `<bus custom code module name>.h`, and `<bus custom code module name>_Internal.h` files. These files are installed with the bus implementation software.

The following restrictions apply to the files:

- Do not make any manual changes to the files.
- Do not change the installation path of the files.
- Do not work with local copies of the files.
- Do not explicitly add the files to the bus implementation software.

  You must add only the files of the user code implementation to the bus implementation software. The files of the Bus Custom Code interface are automatically accessed by the bus implementation software.

Restrictions for using reserved functions

The Bus Custom Code interface provides functions that are reserved for internal use only. These functions are specified in the `<bus custom code module name>_Internal.h` files. Do not use any of these functions in your user code.

Restrictions for using the DsBusCustomCode_PduUserCode module

The `DsBusCustomCode_PduUserCode` module can be used only with the Bus Manager. You cannot use this interface with any other bus implementation software.

Examples of Implementing User-Specific Functionalities via the Bus Custom Code Interface

Where to go from here

Information in this section

Example of Implementing User-Specific PDU Functionalities via the Bus Custom Code Interface

Overview

Via the `DsBusCustomCode_PduUserCode` module of the Bus Custom Code interface, you can implement user-specific PDU functionalities in executable applications.

Note

The `DsBusCustomCode_PduUserCode` module can be used only with the Bus Manager, i.e., only the Bus Manager can parameterize the functions of this module. You cannot use this module with any other bus implementation software.

The following is an example of a source file of a user code implementation that implements user-specific algorithms for checksum and counter value calculation.

At run time, the algorithms apply to all PDUs for which the **PDU User Code** feature of the Bus Manager is added and that reference the user code ID of this source file.

The source file of this example consists of the following parts:

- Defining the user code ID on page 26
- Including required include files on page 26
- Defining return values on page 26
- Saving current counter data on page 27
- Initializing the user code for each PDU on page 27
- Executing the user code for each PDU on page 28

Defining the user code ID

The first element in the source file is the definition of a user code ID, which is named `UserCRCDemo` in this example:

```
/* Start of user code source file */
/* Define user code ID */
#define DS_BUS_CUSTOM_FEATURE_NAME UserCRCDemo
```

For each user code implementation, one unique user code ID must be specified. To apply the algorithms of the user code implementation to a PDU, you must reference the specified user code ID via the **PDU User Code** feature of this PDU in the ConfigurationDesk application.

Including required include files

After the definition of the user code ID, required include files are included:

```
/* Mandatory include file */
#include <DsBusCustomCode_PduUserCode.h>

/* Further required include files */
#include <stdio.h>
#include <stdlib.h>
```

Including the `DsBusCustomCode_PduUserCode.h` file is mandatory. The file provides type definitions and function declarations of the `DsBusCustomCode_PduUserCode` module of the Bus Custom Code interface. Additionally, it includes type definitions and function declarations of the common `DsBusCustomCode` module.

The `stdio.h` and `stdlib.h` files are header files of the standard C library, providing definitions and declarations of the C library. It depends on your user code whether including these files and/or other include files is required.

Defining return values

In this example, the following return values are defined to provide status information on the received counter and checksum values at run time:

```
#define CUSTOM_CRC_OK                  0x00  /* Expected checksum value received */
#define CUSTOM_CRC_FAILED              0x01  /* Unexpected checksum value received */
#define CUSTOM_CRC_COUNTER_FAILED      0x02  /* Unexpected counter value received */
#define CUSTOM_CRC_COUNTER_INITIALIZED 0x04  /* Counter initialized */
```

In this example, the definition of these return values is global, i.e., the values can be used by all functions of this source file.

Tip

If you define such return values, it is recommended to define reasonable values, e.g., according to related AUTOSAR definitions.

Saving current counter data

At run time, functions of this user code example require access to the initialization state of the counter and the current counter value. In most cases, the current counter value is the value that was calculated during the previous execution of the PDU User Code feature.

To provide access to the required data, the following structure is specified:

```
/* Struct to store counter data */
struct UserCode_FeatureData
{
    uint32 counter_value;            /* Current counter value */
    uint32 counter_is_initialized;   /* Flag indicating whether the counter is initialized */
};
```

Tip

In the following, this structure is accessed by the `featureDataPtr` pointer. The `featureDataPtr` pointer is used to pass the data of the structure to PDU handles.

Initializing the user code for each PDU

The user code must be initialized for each PDU to which the PDU User Code feature is added. For this purpose, you must implement the `DsBusCustomCode_onApplicationInit` function. During the initialization phase of the executable application, this function is called for each PDU that is affected by the PDU User Code feature.

In this example, the user code is initialized for each PDU as follows:

```
Std_ReturnType DsBusCustomCode_onApplicationInit(DsBusCustomCodePduFeatureHandle PduFeatureHandle)
{
    /* Declare data pointers. */
    uint8* featureDataPtr;
    char* featureDescriptor;
    struct UserCode_FeatureData* featureInstanceData;

    /* Execute the DSBUSCUSTOMCODE_CHECK_NULL macro to check whether the PduFeatureHandle is NULL.
     * If it is, a message is written to the application log and the DsBusCustomCode_onApplicationInit function
     * returns E_NOT_OK. */
    DSBUSCUSTOMCODE_CHECK_NULL(g_DsBusCustomCodeDefaultDescriptor, PduFeatureHandle);
    DsBusCustomCode_getDescriptor(PduFeatureHandle, &featureDescriptor);

    /* Create an instance of the UserCode_FeatureData struct to store PDU-specific feature data.
     * Assign the instantiated struct to the featureInstanceData pointer.
     * Check the pointer via the DSBUSCUSTOMCODE_CHECK_NULL macro. */
```

```
13      featureInstanceData = (struct UserCode_FeatureData*)malloc(sizeof(struct UserCode_FeatureData));
14      DSBUSCUSTOMCODE_CHECK_NULL(g_DsBusCustomCodeDefaultDescriptor, featureInstanceData);
15
16      /* Initialize the counter of the PDU with 0. */
17      featureInstanceData->counter_value = 0;
18      featureInstanceData->counter_is_initialized = 0;
19
20      /* Assign the instantiated UserCode_FeatureData struct via its featureInstanceData pointer to
         * the featureDataPtr pointer.
         * Assign the featureDataPtr pointer to the PDUFeatureHandle.
         * Execute the DSBUSCUSTOMCODE_TRY macro to check whether the DsBusCustomCodePduFeature_setFeatureDataPtr
         * function is called successfully.
         * If it is not, a message is written to the application log and the DsBusCustomCode_onApplicationInit
         * function returns E_NOT_OK. */
21      featureDataPtr = (uint8*)featureInstanceData;
22      DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCodePduFeature_setFeatureDataPtr(PduFeatureHandle,
         featureDataPtr));
23
24      /* Return whether DsBusCustomCode_onApplicationInit function is implemented correctly. */
25      return E_OK;
26  }
```

Tip

- The `DsBusCustomCode_onApplicationInit` function does not influence the real-time performance of the user code during the execution phase. For optimum performance, it is therefore recommended to implement time-consuming operations such as `malloc` or non-deterministic function calls in the `DsBusCustomCode_onApplicationInit` function.
- The `DSBUSCUSTOMCODE_CHECK_NULL` and `DSBUSCUSTOMCODE_TRY` macros are defined by the common `DsBusCustomCode` module. For optimum performance, it is recommended to use these macros to check each pointer and function parameter once at an early stage of the code execution, i.e., as early as the `DsBusCustomCode_onApplicationInit` function, if possible.
- By using the `DsBusCustomCodePduFeature_setFeatureDataPtr` function, you can assign a pointer to any handle. At run time, you can access the handle and its current data by calling the pointer at any time in the `DsBusCustomCode_onPduFeatureExecution` function.

Executing the user code for each PDU

To execute the user code for each PDU to which the PDU User Code feature is added, you must implement the `DsBusCustomCode_onPduFeatureExecution` function. This function is called each time the transmission of an affected PDU is triggered or the PDU is received. With each function call, PDU data (e.g., direction, payload length) and data of the PDU User Code feature (e.g., function port values) is provided to the user code.

In this example, the `DsBusCustomCode_onPduFeatureExecution` function is structured in three parts, as shown in the following overview.

```
1  Std_ReturnType DsBusCustomCode_onPduFeatureExecution(DsBusCustomCodePduFeatureHandle PduFeatureHandle)
2  {
```

```
    /* Part 1: Common part of the DsBusCustomCode_onPduFeatureExecution function that applies to each PDU to which
     * the PDU User Code feature is added, regadless of the PDU direction. */

    /* Part 2 and 3: If-Else loop that checks whether the PDU is a TX PDU. */
    if (isTx)
    {
        /* Part 2: Executed only if the PDU is a TX PDU. */
    }
    else
    {
        /* Part 3: Executed only if the PDU is an RX PDU. */
    }
    return E_OK;
}
```

For more information on the individual parts, refer to:

- Part 1: Common part of the DsBusCustomCode_onPduFeatureExecution function on page 29
- Part 2: TX-related part of the DsBusCustomCode_onPduFeatureExecution function on page 30
- Part 3: RX-related part of the DsBusCustomCode_onPduFeatureExecution function on page 31

Part 1: Common part of the DsBusCustomCode_onPduFeatureExecution function The following part of the `DsBusCustomCode_onPduFeatureExecution` function is executed for each PDU to which the PDU User Code feature is added, regardless of whether the PDU is a TX PDU or an RX PDU.

```
Std_ReturnType DsBusCustomCode_onPduFeatureExecution(DsBusCustomCodePduFeatureHandle PduFeatureHandle)
{
    /* Declare required handles, data pointers, and variables, for example, the following: */
    boolean isTx = FALSE;
    struct UserCode_FeatureData* featureDataPtr;
    char* featureDescriptor;
    DsBusCustomCodePduHandle pduHandle;
    uint8* IPdu_SduDataPtr;
    uint32 IPdu_SduLength;
    DsBusCustomCodeSignalHandle *Signals;
    uint32 SignalCount;
    uint32 ISignalCounterPos;
    uint32 ISignalCrcPos;
    uint8 tmpcrcval;
    uint32 tmpPos;
    uint32 tmpCounterPos;
    uint32 tmpCrcPos;
    ...

    /* Access the PduFeatureHandle and its data.
     * Check the PduFeatureHandle via the DSBUSCUSTOMCODE_CHECK_NULL macro. */
    DSBUSCUSTOMCODE_CHECK_NULL(g_DsBusCustomCodeDefaultDescriptor, PduFeatureHandle);
    /* Get the descriptor of the PduFeatureHandle. */
    DSBUSCUSTOMCODE_TRY(g_DsBusCustomCodeDefaultDescriptor, DsBusCustomCode_getDescriptor(PduFeatureHandle,
     &featureDescriptor));
    /* Get the data structure that is addressed via the featureDataPtr pointer, i.e., the data of the
     * related UserCode_FeatureData struct that was created during the execution of the
     * DsBusCustomCode_onApplicationInit function. */
    DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCodePduFeature_getFeatureDataPtr(PduFeatureHandle,
     (uint8**)&featureDataPtr));
```

```
26
27      /* Access the PduHandle and its data.
         * Get the PduHandle of the PDU from its PduFeatureHandle. */
28      DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCodePduFeature_getPdu(PduFeatureHandle, &pduHandle));
29      /* Get properties of the PDU from its PduHandle */
30      DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCodePdu_getSduLength(pduHandle, &IPdu_SduLength));
31      DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCodePdu_getSduDataPtr(pduHandle, &IPdu_SduDataPtr));
32      DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCodePdu_getIsTx(pduHandle, &isTx));
33
34      /* Access the signals of the PDU.
         * Get the user signal array of the PDU from its PduFeatureHandle. */
35      DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCodeUserCodePduFeature_getUserSignals(PduFeatureHandle,
         &Signals));
36      /* Get the number of user signals of the PDU from its PduFeatureHandle. */
37      DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCodeUserCodePduFeature_getNumberOfUserSignals(
        PduFeatureHandle, &SignalCount));
38
39      /* Get the start bit position of the first two signals in the signal array.
         * Assign the ISignalCrcPos and ISignalCounterPos variables to the start bit position. */
40      DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCodeSignal_getStartBitPosition(Signals[0], &ISignalCrcPos));
41      DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCodeSignal_getStartBitPosition(Signals[1],
         &ISignalCounterPos));
42
43      /* Specify initial value for some variables of each PDU feature handle. */
44      tmpcrcval = 0;
45      tmpPos = IPdu_SduLength;
46      tmpCounterPos = ISignalCounterPos >> 3;
47      tmpCrcPos = ISignalCrcPos >> 3;
48
49      /* End of part 1, i.e., end of the common part of the DsBusCustomCode_onPduFeatureExecution function. */
50
51      /* Part 2 and 3: If-Else loop that checks whether the PDU is a TX PDU. */
52      if (isTx)
53      {
54      /* Part 2: Executed only if the PDU is a TX PDU.
         * For details, see below. */
55      }
56      else
57      {
58      /* Part 3: Executed only if the PDU is an RX PDU.
         * For details, see below. */
59      }
60
61      /* Return whether DsBusCustomCode_onPduFeatureExecution function is implemented correctly. */
62      return E_OK;
63      }
```

Part 2: TX-related part of the DsBusCustomCode_onPduFeatureExecution function The following part of the `DsBusCustomCode_onPduFeatureExecution` function is executed only if the PDU is a TX PDU.

```
1  Std_ReturnType DsBusCustomCode_onPduFeatureExecution(DsBusCustomCodePduFeatureHandle PduFeatureHandle)
2  {
3      /* Part 1: Common part of the DsBusCustomCode_onPduFeatureExecution function. */
4      ...
5
6      /* If-Else loop that checks whether the PDU is a TX PDU.
       * Part 2: If the PDU is a TX PDU, the If part of the If-Else loop is executed. */
7      if (isTx)
8      {
```

```
      /* Write counter value to the allocated position of the PDU's payload. */
      if (IPdu_SduLength > tmpCounterPos)
        IPdu_SduDataPtr[tmpCounterPos] = (++(featureDataPtr->counter_value) & 0xFF);

      /* Iterate over the PDU's data buffer and calculate the new checksum value. */
      while (tmpPos)
      {
        tmpPos--;

        /* Exclude the CRC position from the checksum calculation. */
        if (tmpPos != tmpCrcPos)
          tmpcrcval += (0xFF ^ IPdu_SduDataPtr[tmpPos]);
      }

      /* Write the calculated checksum value to the allocated position of the PDU's payload. */
      if (IPdu_SduLength > tmpCrcPos)
        IPdu_SduDataPtr[tmpCrcPos] = tmpcrcval;
    }
    /* End of part 2, i.e., end of the TX-related part of the DsBusCustomCode_onPduFeatureExecution function. */

    /* Part 3: If the PDU is an RX PDU, the Else part of the If-Else loop is executed. */
    else
    {
    /* For details, see below. */
    }

    /* Return whether DsBusCustomCode_onPduFeatureExecution function is implemented correctly. */
    return E_OK;
    }
```

Part 3: RX-related part of the DsBusCustomCode_onPduFeatureExecution function The following part of the `DsBusCustomCode_onPduFeatureExecution` function is executed only if the PDU is an RX PDU.

```
Std_ReturnType DsBusCustomCode_onPduFeatureExecution(DsBusCustomCodePduFeatureHandle PduFeatureHandle)
{
    /* Part 1: Common part of the DsBusCustomCode_onPduFeatureExecution function. */
    ...

    /* If-Else loop that checks whether the PDU is a TX PDU. */
    if (isTx)
    {
    /* Part 2: If the PDU is a TX PDU, the If part of the If-Else loop is executed. */
    ...
    }
    /* Part 3: If the PDU is an RX PDU, the Else part of the If-Else loop is executed. */
    else
    {
      uint32 status = CUSTOM_CRC_OK;
      /* Check whether the RX counter has been initialized. */
      if (IPdu_SduLength > tmpCounterPos)
      {
        /* If the RX counter has not been initialized yet, initialize it with the received counter value. */
        if (!featureDataPtr->counter_is_initialized)
        {
          featureDataPtr->counter_value = IPdu_SduDataPtr[tmpCounterPos];
          status |= CUSTOM_CRC_COUNTER_INITIALIZED;
          featureDataPtr->counter_is_initialized = 1;
        }
```

```
26          /* If the RX counter has already been initialized, increase the local counter value and
             * compare it with the received counter value. */
27          else
28          {
29            featureDataPtr->counter_value++;
30            if ((featureDataPtr->counter_value & 0xFF) != IPdu_SduDataPtr[tmpCounterPos])
31            {
32              featureDataPtr->counter_is_initialized = 0;
33              status |= CUSTOM_CRC_COUNTER_FAILED;
34            }
35          }
36        }
37        /* Iterate over the PDU's data buffer and calculate the expected checksum value. */
38        if (IPdu_SduLength > tmpCrcPos)
39        {
40          while (tmpPos)
41          {
42            tmpPos--;
43            if (tmpPos != tmpCrcPos)
44              tmpcrcval += (0xFF ^ IPdu_SduDataPtr[tmpPos]);
45          }
46          /* Compare the expected checksum value with the received checksum value. */
47          if (tmpcrcval != IPdu_SduDataPtr[tmpCrcPos])
48          {
49            status |= CUSTOM_CRC_FAILED;
50          }
51        }
52        DsBusCustomCodeUserCodePduFeature_setResult(PduFeatureHandle, status);
53      }
54
55      /* Return whether DsBusCustomCode_onPduFeatureExecution function is implemented correctly. */
56      return E_OK;
57  }
```

Related topics

Basics

Working with Bus Configuration Features (ConfigurationDesk Bus Manager Implementation Guide 🕮)
Working with Bus Configuration Features (Bus Manager (Stand-Alone) Implementation Guide 🕮)

Example of Implementing Secure Onboard Communication via the Bus Custom Code Interface

Overview

Via the `DsBusCustomCode_SecOC` module of the Bus Custom Code interface, you can implement secure onboard communication (SecOC) in executable applications.

The following example is an extract of the `UserCode_SecOC.c` source file. The `UserCode_SecOC.c` source file implements user-specific algorithms for

calculating and verifying authentication information by using functions of the `DsBusCustomCode_SecOC` module.

Tip

The `UserCode_SecOC.c` file is part of a demo user code implementation for implementing secure onboard communication. The demo user code implementation is available on demand. For more information, contact dSPACE Support (www.dspace.com/go/supportrequest).

The source file of this example consists of the following parts:

- Defining the user code ID on page 33
- Including required include files on page 33
- Defining return values on page 34
- Saving current data provided by crypto service manager and freshness value manager on page 34
- Initializing the user code for each secured IPDU on page 35
- Executing the user code for each secured IPDU on page 38

Defining the user code ID

The first element in the source file is the definition of a user code ID, which is named `SecOC` in this example:

```
/* Start of the UserCode_SecOC.c source file */
/* Define user code ID */
#define DS_BUS_CUSTOM_FEATURE_NAME SecOC
```

For each user code implementation, one unique user code ID must be specified. To apply the algorithms of the user code implementation to secured IPDUs, the specified user code ID must be referenced in the bus implementation software.

Including required include files

After the definition of the user code ID, required include files are included:

```
/* Mandatory include file */
#include <DsBusCustomCode_SecOC.h>

/* Further required include files */
#include <dsstd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "UserCode_SecOCHelper.h"
#include "UserCode_Csm.h"
#include "UserCode_Fvm.h"
```

Including the `DsBusCustomCode_SecOC.h` file is mandatory. The file provides type definitions and function declarations of the `DsBusCustomCode_SecOC` module of the Bus Custom Code interface. Additionally, it includes type definitions and function declarations of the common `DsBusCustomCode` module.

It depends on your user code whether including further files is required. In this example, the following include files are required:

Include File	Purpose
`dsstd.h`	dSPACE-specific header file, providing platform-dependent type definitions for dSPACE systems.
▪ `stdio.h` ▪ `stdlib.h` ▪ `string.h`	Header files of the standard C library, providing definitions and declarations of the C library.
▪ `UserCode_SecOCHelper.h` ▪ `UserCode_Csm.h` ▪ `UserCode_Fvm.h`	Header files of the dSPACE demo user code implementation. These files are examples on how user code can provide required helper functions and implement functionalities of the AUTOSAR crypto service manager and freshness value manager.

Defining return values

In this example, return values are defined to provide the verification result of received and verified authentication information. Additionally, a return value is defined that indicates whether all required parameters are successfully initialized during the initialization phase.

```
#define SECOC_VERIFICATIONSUCCESS     0x00   /* Verification successful */
#define SECOC_VERIFICATIONFAILURE     0x01   /* Verification failed */
#define SECOC_FRESHNESSFAILURE        0x02   /* Verification failed because of an unexpected freshness value */
#define SECOC_MESSAGELINKERFAILURE    0x03E  /* Verification failed because of an unexpected message linker value */
#define SECOC_VERIFICATIONNOTEXECUTED 0x03F  /* Verification not executed */

#define SECOC_FEATUREDATAINITIALIZED 0x3e0d8212  /* Parameters successfully initialized */
```

In this example, the definition of these return values is global, i.e., the values can be used by all functions of this source file.

Tip

If you define such return values, it is recommended to define reasonable values, e.g., according to related AUTOSAR definitions.

Saving current data provided by crypto service manager and freshness value manager

At run time, functions of the user code require access to the current data of the implemented functionalities of the crypto service manager and freshness value manager. To provide access to this data, the following structure is specified:

```
/* Struct to store current feature data */
struct UserCode_FeatureData
{
    uint8* CsmJobHandle;    /* Handle providing current data of the crypto service manager functionalities */
    uint8* TimeBaseHandle;  /* Handle providing current data of the freshness value manager functionalities */
    uint32 isInitialized;   /* Flag indicating whether the all required parameters are initialized */
};
```

Initializing the user code for each secured IPDU

The user code must be initialized for each secured IPDU. For this purpose, you must implement the `DsBusCustomCode_onApplicationInit` function. During the initialization phase of the executable application, this function is called for each secured IPDU.

The following code is an extract of the `DsBusCustomCode_onApplicationInit` function as it is implemented in the `UserCode_SecOC.c` source file.

```
Std_ReturnType DsBusCustomCode_onApplicationInit(DsBusCustomCodePduFeatureHandle PduFeatureHandle)
{
    /* Declare required handles, data pointers, and variables, for example, the following: */
    DsBusCustomCodePduHandle securedIPdu_PduHandle, authenticIPdu_PduHandle;
    DsBusCustomCodeEcuHandle ecuHandle;
    DsBusCustomCodeCommunicationClusterHandle communicationClusterHandle;
    uint32 authenticIPdu_SduLength;
    char*  keyAsString;
    uint32 keyLength;
    uint8* featureDataPtr;
    static uint8 isInitialized = 0;
    ...

    /* Execute the DSBUSCUSTOMCODE_CHECK_NULL macro to check whether the PduFeatureHandle is NULL.
     * If it is, a message is written to the application log and the DsBusCustomCode_onApplicationInit function
     * returns E_NOT_OK.
     * Get the descriptor of the PduFeatureHandle. */
    DSBUSCUSTOMCODE_CHECK_NULL(g_DsBusCustomCodeDefaultDescriptor, PduFeatureHandle);
    DsBusCustomCode_getDescriptor(PduFeatureHandle, &featureDescriptor);

    /* Create an instance of the UserCode_FeatureData struct to store PDU-specific feature data.
     * Assign the instantiated struct to the featureInstanceData pointer.
     * Check the pointer via the DSBUSCUSTOMCODE_CHECK_NULL macro. */
    struct UserCode_FeatureData* featureInstanceData = (struct UserCode_FeatureData*)rtlib_malloc(sizeof(struct
     UserCode_FeatureData));
    DSBUSCUSTOMCODE_CHECK_NULL(g_DsBusCustomCodeDefaultDescriptor, featureInstanceData);
    memset(featureInstanceData, 0, sizeof(struct UserCode_FeatureData));

    /* Assign the instantiated UserCode_FeatureData struct via its featureInstanceData pointer to
     * the featureDataPtr pointer.
     * Assign the featureDataPtr pointer to the PDUFeatureHandle.
     * Execute the DSBUSCUSTOMCODE_TRY macro to check whether the DsBusCustomCodePduFeature_setFeatureDataPtr
     * function is called successfully.
     * If it is not, a message is written to the application log and the DsBusCustomCode_onApplicationInit
     * function returns E_NOT_OK. */
    featureDataPtr = (uint8*)featureInstanceData;
    DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCodePduFeature_setFeatureDataPtr(PduFeatureHandle,
     featureDataPtr));

    /* Call the UserCode_SecOC_Init function of the demo user code implementation to initialize required
     * user code functions etc. */
    UserCode_SecOC_Init();

    /* Initialize the time base. Depending on the bus implementation software, the bus communication is either
     * communication-cluster-based or ECU-based. Therefore, there can be one time base per communication cluster
     * or ECU. By initialzing the time base as follows, no software-specific adaptions are required:
     * For cluster-based bus communication, the time base is initialized for the cluster via the first If-Else loop.
     * For ECU-based bus communcation, the cluster-based initialization of the time base is overwritten by an
     * ECU-based initialization via the second If-Else loop.
     * Check the handles and function calls via the DSBUSCUSTOMCODE_CHECK_NULL and DSBUSCUSTOMCODE_TRY macros. */
```

```
31      returnValue = DsBusCustomCodePduFeature_getCommunicationCluster(PduFeatureHandle, &communicationClusterHandle);
32      if (returnValue == E_OK)
33      {
34          DSBUSCUSTOMCODE_CHECK_NULL(featureDescriptor, communicationClusterHandle);
35          DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCode_getName(communicationClusterHandle,
             &communicationClusterName));
36          USERCODE_DEBUG_PRINT("Communication Cluster: %s\n", communicationClusterName);
37          timeBaseName = communicationClusterName;
38      }
39      else
40      {
41          /* No cluster available. */
42          communicationClusterHandle = NULL_PTR;
43          communicationClusterName = NULL_PTR;
44          USERCODE_DEBUG_PRINT("No Communication Cluster defined.\n");
45          timeBaseName = NULL_PTR;
46      }
47
48      returnValue = DsBusCustomCodePduFeature_getEcu(PduFeatureHandle, &ecuHandle);
49      if (returnValue == E_OK)
50      {
51          DSBUSCUSTOMCODE_CHECK_NULL(featureDescriptor, ecuHandle);
52          DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCode_getName(ecuHandle, &ecuName));
53          USERCODE_DEBUG_PRINT("ECU: %s\n", ecuName);
54          /* If ECU available, initialize time base ECU-based. */
55          timeBaseName = ecuName;
56      }
57      else
58      {
59          /* No ECU available. */
60          ecuHandle = NULL_PTR;
61          ecuName = NULL_PTR;
62          USERCODE_DEBUG_PRINT("No ECU defined.\n");
63          timeBaseName = communicationClusterName;
64      }
65      USERCODE_DEBUG_PRINT("Timebase name: %s\n", timeBaseName);
66
67      /* Get the secured IPDU and its related authentic IPDU via their handles.
         * Check the handles and function calls via the DSBUSCUSTOMCODE_CHECK_NULL and DSBUSCUSTOMCODE_TRY macros. */
68      DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCodePduFeature_getPdu(PduFeatureHandle,
         &securedIPdu_PduHandle));
69      DSBUSCUSTOMCODE_CHECK_NULL(featureDescriptor, securedIPdu_PduHandle);
70      DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCodeSecuredIPdu_getAuthenticPdu(securedIPdu_PduHandle,
         &authenticIPdu_PduHandle));
71      DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCode_getName(securedIPdu_PduHandle, &securedIPduName));
72      USERCODE_DEBUG_PRINT("Secured IPDU: %s\n", securedIPduName);
73
74      /* Get parameters that are required by the bus real-time code, such as the authentic IPDU payload length or
         * the key ID. */
75      DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCodePdu_getSduLength(authenticIPdu_PduHandle,
         &authenticIPdu_SduLength));
76      DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCodeSecuredIPdu_getKeyId(securedIPdu_PduHandle, &keyId));
77      DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCode_getName(authenticIPdu_PduHandle, &authenticPduName));
78      USERCODE_DEBUG_PRINT("Authentic IPDU: %s\n", authenticPduName);
79
80      /* Get the complete length of the freshness value.
         * Calculate the data length that is to be authenticated by taking the related AUTOSAR calculation method
         * into account. */
81      DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCodeSecuredIPdu_getFreshnessValueLength(securedIPdu_PduHandle,
         &freshnessValueLength));
82      dataToAuthentictatorLength = sizeof(uint16) + authenticIPdu_SduLength + (freshnessValueLength + 7) / 8;
```

```
83
84      /* Call the demo user code function that creates a CMAC/AES128-compliant crypto service manager handle
         * (CsmJobHandle) for the calculated authenticated length. */
85      DSBUSCUSTOMCODE_TRY(featureDescriptor, UserCode_SecOCHelper_Create_CryptoJob_For_CMAC_AES_128_Authentication(
        &featureInstanceData->CsmJobHandle, dataToAuthentictatorLength));
86
87      /* Initialize the access to the OEM-specific key that is used for calculating the authenticator.
         * To access the key, it must be provided by a user-specific function, e.g., getKeyFromKeyID. */
88      DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCodeSecOCPduFeature_setKeyPtr(PduFeatureHandle, keyPtr));
89      DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCodeSecOCPduFeature_setKeyLength(PduFeatureHandle,
         keyLength));
90      DSBUSCUSTOMCODE_TRY(featureDescriptor, UserCode_Csm_KeyElementSet(featureInstanceData->CsmJobHandle, keyId,
         CRYPTO_KE_MAC_KEY, keyPtr, keyLength));
91
92      /* Call the demo user code function that creates a freshness value manager instance. */
93      DSBUSCUSTOMCODE_TRY(featureDescriptor, UserCode_SecOC_Fvm_Init(timeBaseName, &featureInstanceData->
        TimeBaseHandle));
94
95      /* Get the required secured IPDU parameters and check whether they are set by using the
         * DSBUSCUSTOMCODE_TRY macro */
96      DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCodePdu_getSduLength(securedIPdu_PduHandle,
         &securedIPdu_SduLength));
97      DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCodePdu_getIsTx(securedIPdu_PduHandle, &isTx));
98      DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCodeSecuredIPdu_getDataId(securedIPdu_PduHandle, &dataId));
99      DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCodeSecuredIPdu_getFreshnessValueTxLength(
        securedIPdu_PduHandle, &freshnessValueTxLength));
100     DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCodeSecuredIPdu_getAuthInfoTxLength(securedIPdu_PduHandle,
         &authInfoTxLength));
101
102     /* Set the SECOC_FEATUREDATAINITIALIZED return value after initializing all parameters successfully. */
103     featureInstanceData->isInitialized = SECOC_FEATUREDATAINITIALIZED;
104
105     /* Return whether DsBusCustomCode_onApplicationInit function is implemented correctly. */
106     return E_OK;
107 }
```

Tip

- The `DsBusCustomCode_onApplicationInit` function does not influence the real-time performance of the user code during the execution phase. For optimum performance, it is therefore recommended to implement time-consuming operations such as `malloc` or non-deterministic function calls in the `DsBusCustomCode_onApplicationInit` function.
- The `DSBUSCUSTOMCODE_CHECK_NULL` and `DSBUSCUSTOMCODE_TRY` macros are defined by the common `DsBusCustomCode` module. For optimum performance, it is recommended to use these macros to check each pointer and function parameter once at an early stage of the code execution, i.e., as early as the `DsBusCustomCode_onApplicationInit` function, if possible.
- By using the `DsBusCustomCodePduFeature_setFeatureDataPtr` function, you can assign a pointer to any handle. At run time, you can access the handle and its current data by calling the pointer at any time in the `DsBusCustomCode_onPduFeatureExecution` function.

Executing the user code for each secured IPDU

To execute the user code for each secured IPDU, you must implement the `DsBusCustomCode_onPduFeatureExecution` function. This function is called each time the transmission of a secured IPDU is triggered or a secured IPDU is received. With each function call, PDU data (e.g., direction, payload length) is provided to the user code.

In the `UserCode_SecOC.c` source file, the `DsBusCustomCode_onPduFeatureExecution` function is structured in three parts, as shown in the following overview.

```
Std_ReturnType DsBusCustomCode_onPduFeatureExecution(DsBusCustomCodePduFeatureHandle PduFeatureHandle)
{
    /* Part 1: Common part of the DsBusCustomCode_onPduFeatureExecution function that applies to each secured IPDU,
     * regadless of its direction. */

    /* Part 2 and 3: If-Else loop that checks whether the secured IPDU is a TX IPDU. */
    if (isTx)
    {
        /* Part 2: Executed only if the secured IPDU is a TX secured IPDU. */
    }
    else
    {
        /* Part 3: Executed only if the secured IPDU is an RX secured IPDU. */
    }
    return E_OK;
}
```

For more information on the individual parts, refer to:

- Part 1: Common part of the DsBusCustomCode_onPduFeatureExecution function on page 38
- Part 2: TX-related part of the DsBusCustomCode_onPduFeatureExecution function on page 42
- Part 3: RX-related part of the DsBusCustomCode_onPduFeatureExecution function on page 45

Part 1: Common part of the DsBusCustomCode_onPduFeatureExecution function The following part of the `DsBusCustomCode_onPduFeatureExecution` function is executed for each secured IPDU, regardless of whether it is a TX secured IPDU or an RX secured IPDU.

```
Std_ReturnType DsBusCustomCode_onPduFeatureExecution(DsBusCustomCodePduFeatureHandle PduFeatureHandle)
{
    /* Declare required handles, data pointers, and variables. Among others, the following: */
    DsBusCustomCodePduHandle securedIPdu_PduHandle;
    uint32 securedIPdu_SduLength;
    uint8 *securedIPdu_SduDataPtr;
    DsBusCustomCodePduHandle authenticIPdu_PduHandle;
    uint8* featureDataPtr;
    Crypto_JobType *Crypto_Job;
    uint8* csmJobHandle;
    uint16 messageLinkLength = 0;
    uint16 messageLinkPosition = 0;
    char* featureDescriptor;
    ...

```

```
16      /* Access the PduFeatureHandle and its data.
         * Check the handle by using the DSBUSCUSTOMCODE_CHECK_NULL macro. */
17      DSBUSCUSTOMCODE_CHECK_NULL(g_DsBusCustomCodeDefaultDescriptor, PduFeatureHandle);
18      DsBusCustomCode_getDescriptor(PduFeatureHandle, &featureDescriptor);
19
20      /* Get the data structure that is addressed via the featureDataPtr pointer, i.e., the data of the related
         * UserCode_FeatureData struct that was created during the execution of the
         * DsBusCustomCode_onApplicationInit function. */
21      DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCodePduFeature_getFeatureDataPtr(PduFeatureHandle,
         (uint8**)&featureDataPtr));
22
23      /* Check whether the initialization of the PduFeatureHandle was successfull.
         * If not, return E_NOT_OK and exit DsBusCustomCode_onPduFeatureExecution. */
24      if (SECOC_FEATUREDATAINITIALIZED != ((struct UserCode_FeatureData*)featureDataPtr)->isInitialized)
25      {
26          return E_NOT_OK;
27      }
28
29      /* Get the secured IPDU via its securedIPdu_PduHandle.
         * Check the handle by using the DSBUSCUSTOMCODE_CHECK_NULL macro.
         * Get required secured IPDU parameters. */
30      DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCodePduFeature_getPdu(PduFeatureHandle,
         &securedIPdu_PduHandle));
31      DSBUSCUSTOMCODE_CHECK_NULL(featureDescriptor, securedIPdu_PduHandle);
32
33      DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCodePdu_getSduLength(securedIPdu_PduHandle,
         &securedIPdu_SduLength));
34      DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCodePdu_getSduDataPtr(securedIPdu_PduHandle,
         &securedIPdu_SduDataPtr));
35      DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCodePdu_getIsTx(securedIPdu_PduHandle, &isTx));
36
37      DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCodeSecuredIPdu_getDataId(securedIPdu_PduHandle, &dataId));
38      DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCodeSecuredIPdu_getFreshnessValueLength(securedIPdu_PduHandle,
         &freshnessValueLength));
39      DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCodeSecuredIPdu_getFreshnessValueTxLength(
        securedIPdu_PduHandle, &freshnessValueTxLength));
40      DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCodeSecuredIPdu_getKeyId(securedIPdu_PduHandle, &keyId));
41      DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCodeSecuredIPdu_getAuthInfoTxLength(securedIPdu_PduHandle,
         &authInfoTxLength));
42
43      /* Get the length of the secured header of the authentic IPDU in bytes.
         * If the parameter is not set, set the parameter value to 0.
         * If a logical error occurs (e.g., an invalid handle), return E_NOT_OK and
         * exit DsBusCustomCode_onPduFeatureExecution. */
44      returnValue = DsBusCustomCodeSecuredIPdu_getAuthPduHeaderLength(securedIPdu_PduHandle, &authPduHeaderLength);
45      if (returnValue == E_PARAMETER_NOT_SET)
46      {
47          authPduHeaderLength = 0;
48      }
49      else if (returnValue != E_OK)
50      {
51          return E_NOT_OK;
52      }
53      /* Convert the byte-based header length in a bit-based value. */
54      securedIPduAuthenticIPduPosition = authPduHeaderLength * 8;
55
56      /* Get the authentic IPDU via its authenticIPdu_PduHandle.
         * Check the handle by using the DSBUSCUSTOMCODE_CHECK_NULL macro.
         * Get required authentic IPDU parameters. */
57      DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCodeSecuredIPdu_getAuthenticPdu(securedIPdu_PduHandle,
         &authenticIPdu_PduHandle));
```

```
58      DSBUSCUSTOMCODE_CHECK_NULL(featureDescriptor, authenticIPdu_PduHandle);
59
60      DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCodePdu_getSduLength(authenticIPdu_PduHandle,
         &authenticIPdu_SduLength));
61      DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCodePdu_getSduDataPtr(authenticIPdu_PduHandle,
         &authenticIPdu_SduDataPtr));
62
63      /* Determine the acutal length of the authentic IPDU according to AUTOSAR:
         * If no secured header is specified, the actual length is derived from the authenticIPdu_SduLength parameter.
         * If a secured header is specified, the acutal length is read from the header by using the related demo user
         * code function.
         * The actual length can be equal or smaller than the SDU length. If the actual length is bigger than the SDU
         * length, this indicates a logical error in determining the actual length. In this case, return E_NOT_OK and
         * exit DsBusCustomCode_onPduFeatureExecution. */
64      if (authPduHeaderLength == 0)
65      {
66          authenticIPdu_ActualLength = authenticIPdu_SduLength;
67      }
68      else if (authPduHeaderLength == 1)
69      {
70          uint8* header = securedIPdu_SduDataPtr;
71          authenticIPdu_ActualLength = USER_CODE_INT8_FROM_BE(*header);
72      }
73      else if (authPduHeaderLength == 2)
74      {
75          uint16* header = (uint16*) securedIPdu_SduDataPtr;
76          authenticIPdu_ActualLength = USER_CODE_INT16_FROM_BE(*header);
77      }
78      else if (authPduHeaderLength == 4)
79      {
80          uint32* header = (uint32*) securedIPdu_SduDataPtr;
81          authenticIPdu_ActualLength = USER_CODE_INT32_FROM_BE(*header);
82      }
83      else
84      {
85          return E_NOT_OK;
86      }
87
88      if (authenticIPdu_ActualLength > authenticIPdu_SduLength)
89      {
90          return E_NOT_OK;
91      }
92
93      /* Get whether the secured IPDU is configured as cryptographic IPDU.
         * If the parameter is not set, set the parameter value to FALSE, i.e., the secured IPDU is not configured as
         * cryptographic IPDU.
         * If a logical error occurs (e.g., an invalid handle), return E_NOT_OK and
         * exit DsBusCustomCode_onPduFeatureExecution. */
94      returnValue = DsBusCustomCodeSecuredIPdu_getUseAsCryptographicPdu(securedIPdu_PduHandle,
         &useAsCryptographicPdu);
95      if (returnValue == E_PARAMETER_NOT_SET)
96      {
97          useAsCryptographicPdu = FALSE;
98      }
99      else if (returnValue != E_OK)
100     {
101         return E_NOT_OK;
102     }
103
```

```
104     /* If the secured IPDU is configured as cryptographic IPDU, get the message link length.
         * If the function is executed without logical errors and a message link length is specified, get the message
         * link position. However, regardless of whether a message linker is specified, the authentic IPDU is not
         * included in the cryptographic IPDU.
         * If the secured IPDU is not configured as cryptographic IPDU, include the authentic IPDU with its acutal
         * length in the secured IPDU.
         * Use the result of the If-Else loop to calculate the authenticator position and message linker position in
         * the secured IPDU. */
105     if (useAsCryptographicPdu)
106     {
107         uint16 linkLength;
108         returnValue = DsBusCustomCodeSecuredIPdu_getMessageLinkLength(securedIPdu_PduHandle, &linkLength);
109         if (returnValue == E_OK && linkLength > 0)
110         {
111             messageLinkLength = linkLength;
112             DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCodeSecuredIPdu_getMessageLinkPosition(
                securedIPdu_PduHandle, &messageLinkPosition));
113         }
114
115         securedIPduFreshnessValuePosition = securedIPduAuthenticIPduPosition;
116     }
117     else
118     {
119         securedIPduFreshnessValuePosition = securedIPduAuthenticIPduPosition + authenticIPdu_ActualLength * 8;
120     }
121
122     securedIPduAuthenticatorPosition = securedIPduFreshnessValuePosition + freshnessValueTxLength;
123     securedIPduMessageLinkerPosition = securedIPduAuthenticatorPosition + authInfoTxLength;
124
125     /* Get the length of the secured area, i.e., the lenght of the authentic IPDU that is to be secured.
         * If the function is executed without logical errors and a secured area length is specified, get the secured
         * area offset.
         * If the sum of secured area length and secured area offset is bigger than the actual length, this indicates a
         * logical error in getting the secured area length and/or offset. In this case, return E_NOT_OK and
         * exit DsBusCustomCode_onPduFeatureExecution.
         * Pass the accessed and validated values to the actualSecuredAreaLength and
         * actualSecuredAreaOffset variables. */
126     {
127         uint32 areaLength;
128         returnValue = DsBusCustomCodeSecuredIPdu_getSecuredAreaLength(securedIPdu_PduHandle, &areaLength);
129         if (returnValue == E_OK && areaLength > 0)
130         {
131             securedAreaLength = areaLength;
132             DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCodeSecuredIPdu_getSecuredAreaOffset(
                securedIPdu_PduHandle, &securedAreaOffset));
133
134             if (securedAreaOffset + securedAreaLength > authenticIPdu_ActualLength)
135             {
136                 return E_NOT_OK;
137             }
138
139             actualSecuredAreaOffset = securedAreaOffset;
140             actualSecuredAreaLength = securedAreaLength;
141         }
142         /* If no secured area length is specified, set the actual secured area offset to 0 and use the entire
             * actual length of the authentic IPDU as the actual secured area length. */
143         else
144         {
145             actualSecuredAreaOffset = 0;
146             actualSecuredAreaLength = authenticIPdu_ActualLength;
147         }
```

```
148     }
149
150     /* Access the run-time data that is written to the UserCode_FeatureData struct by using the CsmJobHandle.
         * Get the applicable crypto job by calling the related demo user code function.
         * Check the user code function and its Crypto_Job parameter by using the DSBUSCUSTOMCODE_TRY and
         * DSBUSCUSTOMCODE_CHECK_NULL marcros. */
151     csmJobHandle = ((struct UserCode_FeatureData*)featureDataPtr)->CsmJobHandle;
152     DSBUSCUSTOMCODE_TRY(featureDescriptor, UserCode_CryptoJob_Get(csmJobHandle, &Crypto_Job));
153     DSBUSCUSTOMCODE_CHECK_NULL(featureDescriptor, Crypto_Job);
154     /* End of part 1, i.e., end of the common part of the DsBusCustomCode_onPduFeatureExecution function */
155
156     /* Part 2 and 3: If-Else loop that checks whether the secured IPDU is a TX IPDU. */
157     if (isTx)
158     {
159         /* Part 2: Executed only if the secured IPDU is a TX secured IPDU.
             * For details, see below. */
160     }
161     else
162     {
163         /* Part 3: Executed only if the secured IPDU is an RX secured IPDU.
             * For details, see below. */
164     }
165
166     /* Return whether DsBusCustomCode_onPduFeatureExecution function is implemented correctly. */
167     return E_OK;
168 }
```

Part 2: TX-related part of the DsBusCustomCode_onPduFeatureExecution function The following part of the `DsBusCustomCode_onPduFeatureExecution` function is executed only if the secured IPDU is a TX secured IPDU.

```
1  Std_ReturnType DsBusCustomCode_onPduFeatureExecution(DsBusCustomCodePduFeatureHandle PduFeatureHandle)
2  {
3      /* Part 1: Common part of the DsBusCustomCode_onPduFeatureExecution function. */
4      ...
5
6      /* If-Else loop that checks whether the secured IPDU is a TX IPDU.
        * Part 2: If the secured IPDU is a TX IPDU, the If part of the If-Else loop is executed. */
7      if (isTx)
8      {
9          /* Declare local variables and get required TX secured IPDU parameters. */
10         uint32 enableAuthentication;
11         uint32 enableFreshnessValueCalculation;
12         uint32 authenticatonType;
13         sint64 freshnessValueOffset;
14
15         DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCodeSecOCTxPduFeature_getEnableFreshnessValueCalculation(
           PduFeatureHandle, &enableFreshnessValueCalculation));
16         DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCodeSecOCTxPduFeature_getEnableAuthentication(
           PduFeatureHandle, &enableAuthentication));
17         DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCodeSecOCTxPduFeature_getAuthenticationType(
           PduFeatureHandle, &authenticatonType));
18         DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCodeSecOCTxPduFeature_getFreshnessValueOffset(
           PduFeatureHandle, &freshnessValueOffset));
19
20         /* If the secured IPDU is configured as cryptographic IPDU and a message link length is specified, call the
            * related demo user code function to copy the required authentication information to the
            * message link length. */
21         if (useAsCryptographicPdu)
22         {
```

```
23            if (messageLinkLength > 0)
24            {
25                UserCode_SecOCHelper_CopyBits(securedIPdu_SduDataPtr, securedIPduMessageLinkerPosition,
                   authenticIPdu_SduDataPtr, messageLinkPosition, messageLinkLength);
26            }
27        }
28        /* If the secured IPDU is not configured as cryptographic IPDU, copy the authentic IPDU to the secured IPDU,
           * if required. Whether copying is required depends on the bus implementation software.
           * Specifying the copying as follows increases the performance and no software-specific adaptions
           * are required. */
29        else
30        {
31            if (securedIPdu_SduDataPtr != authenticIPdu_SduDataPtr)
32            {
33                memcpy(securedIPdu_SduDataPtr + authPduHeaderLength, authenticIPdu_SduDataPtr,
                   authenticIPdu_ActualLength);
34            }
35        }
36
37        /* If calculating the freshness value in the user code is disabled, get the user-defined freshness value. */
38        if(enableFreshnessValueCalculation == 0)
39        {
40            DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCodeSecOCTxPduFeature_getUserDefinedFreshnessValue(
              PduFeatureHandle, &freshnessValue));
41        }
42        /* If calculating the freshness value in the user code is enabled, copy the calculated freshness value and
           * the specified TX length to local variables.
           * Copying the data to local variables with defined data types avoids problems, e.g., if the freshness value
           * manager specifies different data types for the accessed variables. */
43        else
44        {
45            uint32 txFreshnessLength = freshnessValueTxLength;
46            uint64 txFreshnessValue;
47            UserCode_SecOC_GetTxFreshness(((struct UserCode_FeatureData*)featureDataPtr)->TimeBaseHandle, 0,
               (uint8*)&txFreshnessValue, &txFreshnessLength);
48            freshnessValue = txFreshnessValue;
49        }
50        USERCODE_DEBUG_PRINT("TX Freshness Value 0x%llX\n", freshnessValue);
51
52        /* Write the applicable freshness value (i.e., user-defined or calculated in user code) to the
           * freshnessValue parameter.
           * If specified, add the feshness offset value to the freshness value. */
53        DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCodeSecOCTxPduFeature_setCalculatedFreshnessValue(
          PduFeatureHandle, freshnessValue));
54        freshnessValue += freshnessValueOffset;
55
56        /* If caluclating the authenticator in the user code is disabled, get the user-defined authenticator value
           * via the AuthenticatorPtr. */
57        if (enableAuthentication == 0)
58        {
59            DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCodeSecOCTxPduFeature_getUserDefinedAuthenticatorPtr(
              PduFeatureHandle, &AuthenticatorPtr));
60            DSBUSCUSTOMCODE_CHECK_NULL(featureDescriptor, AuthenticatorPtr);
61            authenticatorLength = (authInfoTxLength + 7) / 8;
62        }
63        /* If caluclating the authenticator in the user code is enabled, call the related demo user code function
           * for calculating the authenticator. */
64        else
65        {
```

```
66              DSBUSCUSTOMCODE_TRY(featureDescriptor, UserCode_SecOCHelper_PrepareDataToAuthenticator(dataId,
                 authenticIPdu_SduDataPtr + actualSecuredAreaOffset, actualSecuredAreaLength, freshnessValue,
                 freshnessValueLength, Crypto_Job->PrimitiveInputOutput->inputPtr, &Crypto_Job->PrimitiveInputOutput->
                inputLength));
67              USERCODE_DEBUG_PRINTDATA("TX Data to Auth", Crypto_Job->PrimitiveInputOutput->inputPtr, Crypto_Job->
                PrimitiveInputOutput->inputLength);
68
69              AuthenticatorPtr = Crypto_Job->PrimitiveInputOutput->outputPtr;
70              authenticatorLength = Crypto_Job->jobPrimitiveInfo->primitiveInfo->resultLength;
71
72              /* According to AUTOSAR, the crypto service manager expects a pre-set outputLengthPtr. Therefore, set
                 * the outputLengthPtr to the expected authenticator length. */
73              *(Crypto_Job->PrimitiveInputOutput->outputLengthPtr) = authenticatorLength;
74
75              DSBUSCUSTOMCODE_TRY(featureDescriptor, UserCode_Csm_MacGenerate(csmJobHandle, Crypto_Job->
                PrimitiveInputOutput->mode, Crypto_Job->PrimitiveInputOutput->inputPtr, Crypto_Job->
                PrimitiveInputOutput->inputLength, Crypto_Job->PrimitiveInputOutput->outputPtr, Crypto_Job->
                PrimitiveInputOutput->outputLengthPtr));
76          }
77          USERCODE_DEBUG_PRINTDATA("TX MAC", AuthenticatorPtr, authenticatorLength);
78
79          /* If invalidating the authentication value is enabled, invert the most significant bit of the
             * authenticator. */
80          if (authenticatonType != 0)
81          {
82              USERCODE_DEBUG_PRINT("====== *** MAC MANIPULATION *** ======\n");
83              AuthenticatorPtr[0] ^= 0x80;
84              USERCODE_DEBUG_PRINTDATA("TX Manipulated MAC", AuthenticatorPtr, authenticatorLength);
85          }
86
87          /* Copy the TX freshness value (i.e., the truncated freshness value) to the secured IPDU by calling the
             * related demo user code function. */
88          uint64 freshnessValue_BigEndian = USER_CODE_INT64_TO_BE(freshnessValue);
89          uint8* freshnessPtr = (uint8*)&freshnessValue_BigEndian;
90          uint32 startpos = 64 - freshnessValueTxLength;
91          UserCode_SecOCHelper_CopyBits(securedIPdu_SduDataPtr, securedIPduFreshnessValuePosition, freshnessPtr,
             startpos, freshnessValueTxLength);
92
93          /* Copy the TX authenticator value (i.e., the truncated authenticator value) to the secured IPDU by calling
             * the related demo user code function. */
94          UserCode_SecOCHelper_CopyBits(securedIPdu_SduDataPtr, securedIPduAuthenticatorPosition, AuthenticatorPtr, 0,
             authInfoTxLength);
95          USERCODE_DEBUG_PRINTDATA("TX AuthenticIPdu", authenticIPdu_SduDataPtr, authenticIPdu_ActualLength);
96          USERCODE_DEBUG_PRINTDATA("TX SecuredIPdu", securedIPdu_SduDataPtr, securedIPdu_SduLength);
97      }
98      /* End of part 2, i.e., end of the TX-related part of the DsBusCustomCode_onPduFeatureExecution function */
99
100     /* Part 3: If the secured IPDU is an RX IPDU, the Else part of the If-Else loop is executed. */
101     else
102     {
103         /* For details, see below. */
104     }
105
106     /* Return whether DsBusCustomCode_onPduFeatureExecution function is implemented correctly. */
107     return E_OK;
108 }
```

Part 3: RX-related part of the DsBusCustomCode_onPduFeatureExecution function The following part of the `DsBusCustomCode_onPduFeatureExecution` function is executed only if the secured IPDU is an RX secured IPDU.

```
Std_ReturnType DsBusCustomCode_onPduFeatureExecution(DsBusCustomCodePduFeatureHandle PduFeatureHandle)
{
    /* Part 1: Common part of the DsBusCustomCode_onPduFeatureExecution function. */
    ...

    /* If-Else loop that checks whether the secured IPDU is a TX IPDU. */
    if (isTx)
    {
        /* Part 2: If the secured IPDU is a TX IPDU, the If part of the If-Else loop is executed. */
        ...
    }
    /* Part 3: If the secured IPDU is an RX IPDU, the Else part of the If-Else loop is executed. */
    else
    {
        /* Declare and initialize local variables. */
        boolean useAuthDataFreshness = FALSE;
        uint16 authDataFreshnessLength = 0;
        uint16 authDataFreshnessStartPosition = 0;
        uint32 verificationEnable;

        /* If the secured IPDU is not configured as cryptographic IPDU, copy the data bytes of the secured IPDU to
         * the authentic IPDU, if required. Whether copying is required depends on the bus implementation software.
         * Specifying the copying as follows increases the performance and no software-specific adaptions
         * are required. */
        if (!useAsCryptographicPdu)
        {
            if (authenticIPdu_SduDataPtr != securedIPdu_SduDataPtr)
            {
                memcpy(authenticIPdu_SduDataPtr, securedIPdu_SduDataPtr + authPduHeaderLength,
                 authenticIPdu_ActualLength);
            }
        }
        USERCODE_DEBUG_PRINTDATA("RX AuthenticIPdu", authenticIPdu_SduDataPtr, authenticIPdu_ActualLength);
        USERCODE_DEBUG_PRINTDATA("RX SecuredIPdu", securedIPdu_SduDataPtr, securedIPdu_SduLength);

        /* Get the enable state for verifying received authentication information.
         * If disabled, set the verification result to SECOC_VERIFICATIONNOTEXECUTED. Then, exit
         * DsBusCustomCode_onPduFeatureExecution with E_OK. */
        DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCodeSecOCRxPduFeature_getEnableVerification(
        PduFeatureHandle, &verificationEnable));
        if (verificationEnable == 0)
        {
            DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCodeSecOCRxPduFeature_setVerificationResult(
            PduFeatureHandle, SECOC_VERIFICATIONNOTEXECUTED));
            USERCODE_DEBUG_PRINT("Verification not executed\n");
            return E_OK;
        }

        /* If verification is enabled, perform the verification in the following sequence to increase the
         * performance. */

        /* Check if a message linker is specified. If it is, first verify the message linker by calling the
         * related demo user code function.
         * If the user code function indicates an error, set the verification result to SECOC_MESSAGELINKERFAILURE.
         * Then, exit DsBusCustomCode_onPduFeatureExecution with E_OK. */
        if (messageLinkLength > 0 && authenticIPdu_ActualLength * 8 >= messageLinkLength)
```

```
45          {
46              Std_ReturnType result = UserCode_SecOCHelper_CompareBits(securedIPdu_SduDataPtr,
                 securedIPduMessageLinkerPosition, authenticIPdu_SduDataPtr, messageLinkPosition, messageLinkLength);
47              if (result != E_OK)
48              {
49                  DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCodeSecOCRxPduFeature_setVerificationResult(
                    PduFeatureHandle, SECOC_MESSAGELINKERFAILURE));
50                  USERCODE_DEBUG_PRINT("Verification aborted due to not matching message linker\n");
51                  return E_OK;
52              }
53          }
54
55          /* Get whether a timestamp is used to calculate the freshness value.
             * If the parameter is not set, set the parameter value to FALSE, i.e., no timestamp is used.
             * If a logical error occurs (e.g., an invalid handle), return E_NOT_OK and
             * exit DsBusCustomCode_onPduFeatureExecution. */
56          {
57              boolean useFreshnessTimestamp;
58              returnValue = DsBusCustomCodeSecuredIPdu_getUseFreshnessTimestamp(securedIPdu_PduHandle,
                 &useFreshnessTimestamp);
59              if (returnValue == E_PARAMETER_NOT_SET)
60              {
61                  useFreshnessTimestamp = FALSE;
62              }
63              else if (returnValue != E_OK)
64              {
65                  return E_NOT_OK;
66              }
67
68              /* If a timestamp is used, call the related demo user code function to access the received
                 * freshness value and its length, and copy the values to local variables.
                 * Copying the data to local variables with defined data types avoids problems, e.g., if the freshness
                 * value manager specifies different data types for the accessed variables. */
69              uint32 freshnessVerifyValueLength = freshnessValueLength;
70              uint64 freshnessVerifyValue;
71              DSBUSCUSTOMCODE_TRY(featureDescriptor, UserCode_SecOC_GetRxFreshness(((struct UserCode_FeatureData*)
                featureDataPtr)->TimeBaseHandle, 0, securedIPdu_SduDataPtr + (securedIPduFreshnessValuePosition / 8),
                 freshnessValueTxLength, 0, (uint8*)&freshnessVerifyValue, &freshnessVerifyValueLength,
                 useFreshnessTimestamp));
72              freshnessValue = freshnessVerifyValue;
73          }
74          USERCODE_DEBUG_PRINT("RX Local Freshness Value 0x%llX\n", freshnessValue);
75
76         /* Get whether a part of the payload of the authentic IPDU is included in the freshness value.
             * If the parameter is not set, set the parameter value to FALSE, i.e., no payload is included in the
             * freshness value.
             * If a logical error occurs (e.g., an invalid handle), return E_NOT_OK and
             * exit DsBusCustomCode_onPduFeatureExecution. */
77         returnValue = DsBusCustomCodeSecuredIPdu_getUseAuthDataFreshness(securedIPdu_PduHandle,
            &useAuthDataFreshness);
78          if (returnValue == E_PARAMETER_NOT_SET)
79          {
80              useAuthDataFreshness = FALSE;
81          }
82          else if (returnValue != E_OK)
83          {
84              return E_NOT_OK;
85          }
86
```

```
87          /* If a part of the payload is included in the freshness value, get the length of the included payload.
             * If the function is executed without logical errors and a length is specified, get the position of the
             * start bit.
             * Call the related demo user code function to access the related payload bits in the received
             * freshness value */
88          if (useAuthDataFreshness)
89          {
90              uint16 length;
91              uint64 authDataFreshnessBE = 0;
92              uint64 authDataFreshness = 0;
93
94              returnValue = DsBusCustomCodeSecuredIPdu_getAuthDataFreshnessLength(securedIPdu_PduHandle, &length);
95              if (returnValue == E_OK && length > 0)
96              {
97                  authDataFreshnessLength = length;
98                  DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCodeSecuredIPdu_
                    getAuthDataFreshnessStartPosition(securedIPdu_PduHandle, &authDataFreshnessStartPosition));
99              }
100
101             UserCode_SecOCHelper_CopyBits((uint8*) &authDataFreshnessBE, 64 - authDataFreshnessLength,
                 authenticIPdu_SduDataPtr, authDataFreshnessStartPosition, authDataFreshnessLength);
102             authDataFreshness = USER_CODE_INT64_FROM_BE(authDataFreshnessBE);
103             USERCODE_DEBUG_PRINT("RX AuthDataFreshness 0x%llX\n", authDataFreshness);
104         }
105
106         /* Now you can call the user code functions for calculating the expected freshness value.
             * These functions are not part of the demo user code. */
107
108         /* Set the freshness value that was calculated in the user code and is expected to be received. */
109         DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCodeSecOCRxPduFeature_setCalculatedFreshnessValue(
            PduFeatureHandle, freshnessValue));
110
111         /* Call the demo user code function that collects the received data which is required for verifying
             * the received authentication information. */
112         DSBUSCUSTOMCODE_TRY(featureDescriptor, UserCode_SecOCHelper_PrepareDataToAuthenticator(dataId,
             authenticIPdu_SduDataPtr + actualSecuredAreaOffset, actualSecuredAreaLength, freshnessValue,
             freshnessValueLength, Crypto_Job->PrimitiveInputOutput->inputPtr, &Crypto_Job->PrimitiveInputOutput->
            inputLength));
113         USERCODE_DEBUG_PRINTDATA("RX Data to Auth", Crypto_Job->PrimitiveInputOutput->inputPtr, Crypto_Job->
            PrimitiveInputOutput->inputLength);
114
115         authenticatorLength = Crypto_Job->jobPrimitiveInfo->primitiveInfo->resultLength;
116
117         /* According to AUTOSAR, the crypto service manager expects a pre-set outputLengthPtr. Therefore, set the
             * outputLengthPtr to the expected authenticator length. */
118         *(Crypto_Job->PrimitiveInputOutput->outputLengthPtr) = authenticatorLength;
119
120         /* Call the demo user code function that calculates the received authentication information. */
121         DSBUSCUSTOMCODE_TRY(featureDescriptor, UserCode_Csm_MacGenerate(csmJobHandle, Crypto_Job->
            PrimitiveInputOutput->mode, Crypto_Job->PrimitiveInputOutput->inputPtr, Crypto_Job->
            PrimitiveInputOutput->inputLength, Crypto_Job->PrimitiveInputOutput->outputPtr, Crypto_Job->
            PrimitiveInputOutput->outputLengthPtr));
122         USERCODE_DEBUG_PRINTDATA("MAC received", securedIPdu_SduDataPtr + (securedIPduAuthenticatorPosition / 8),
             (authInfoTxLength + 7) / 8);
123
124         /* Call the demo user code function that compares the received authentication information with the
             * expected authentication information.
             * Set the related return value, i.e., SECOC_VERIFICATIONFAILURE or SECOC_VERIFICATIONSUCCESS. */
125         Std_ReturnType result = UserCode_SecOCHelper_CompareBits(securedIPdu_SduDataPtr + (
            securedIPduAuthenticatorPosition / 8), freshnessValueTxLength % 8, Crypto_Job->PrimitiveInputOutput->
            outputPtr, 0, authInfoTxLength);
```

```
126
127         if (result != E_OK)
128         {
129             DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCodeSecOCRxPduFeature_setVerificationResult(
                PduFeatureHandle, SECOC_VERIFICATIONFAILURE));
130             USERCODE_DEBUG_PRINT("Verification failure\n");
131         }
132         else
133         {
134             DSBUSCUSTOMCODE_TRY(featureDescriptor, DsBusCustomCodeSecOCRxPduFeature_setVerificationResult(
                PduFeatureHandle, SECOC_VERIFICATIONSUCCESS));
135             USERCODE_DEBUG_PRINT("Verification success\n");
136         }
137     }
138     /* Return whether DsBusCustomCode_onPduFeatureExecution function is implemented correctly. */
139     return E_OK;
140 }
```

API Reference of the Bus Custom Code Interface

Where to go from here

Information in this section

Handles of the Bus Custom Code Interface

Where to go from here

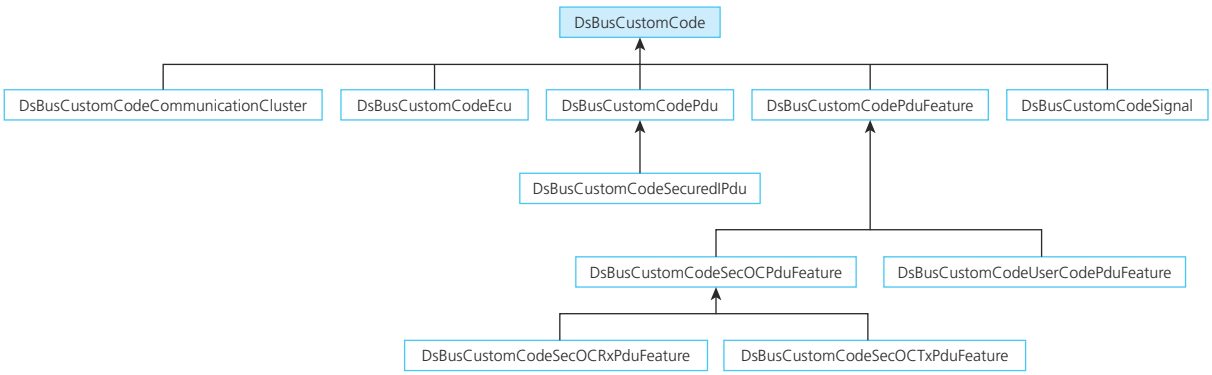
Information in this section

DsBusCustomCodeHandle

Purpose

This handle represents a DsBusCustomCode element.

Inheritance diagram



Include file `DsBusCustomCode.h`

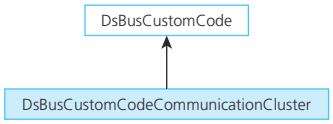
Functions of the handle This handle provides the following functions:

Function	Purpose
`DsBusCustomCode_getDescriptor` on page 67	To get the descriptor of a DsBusCustomCode handle.
`DsBusCustomCode_getName` on page 68	To get the name of an element that is addressed by a DsBusCustomCode handle.

DsBusCustomCodeCommunicationClusterHandle

Purpose This handle represents a communication cluster.

Inheritance diagram



Include file `DsBusCustomCode.h`

Functions of the handle This handle does not provide handle-specific functions.

Functions inherited from DsBusCustomCodeHandle This handle inherits the following functions from `DsBusCustomCodeHandle` on page 50:

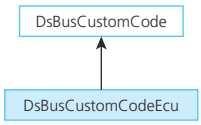
Function	Purpose
`DsBusCustomCode_getDescriptor` on page 67	To get the descriptor of a DsBusCustomCode handle.
`DsBusCustomCode_getName` on page 68	To get the name of an element that is addressed by a DsBusCustomCode handle.

DsBusCustomCodeEcuHandle

Purpose

This handle represents an ECU.

Inheritance diagram



Include file

`DsBusCustomCode.h`

Functions of the handle

This handle does not provide handle-specific functions.

Functions inherited from DsBusCustomCodeHandle

This handle inherits the following functions from `DsBusCustomCodeHandle` on page 50:

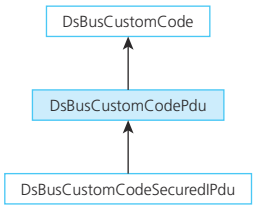
Function	Purpose
`DsBusCustomCode_getDescriptor` on page 67	To get the descriptor of a DsBusCustomCode handle.
`DsBusCustomCode_getName` on page 68	To get the name of an element that is addressed by a DsBusCustomCode handle.

DsBusCustomCodePduHandle

Purpose

This handle represents a PDU.

Inheritance diagram



Include file

`DsBusCustomCode.h`

Functions of the handle

This handle provides the following functions:

Function	Purpose
`DsBusCustomCodePdu_getCanChannelName` on page 70	To get the name of a CAN channel.
`DsBusCustomCodePdu_getCanFrameTriggering` on page 71	To get the frame triggering of a CAN frame.
`DsBusCustomCodePdu_getIsTx` on page 73	To get the direction of a PDU.
`DsBusCustomCodePdu_getLinFrameTriggering` on page 74	To get the frame triggering of a LIN frame.
`DsBusCustomCodePdu_getSduDataPtr` on page 76	To get the `SduDataPtr` pointer of a PDU.
`DsBusCustomCodePdu_getSduLength` on page 77	To get the SDU length of a PDU in bytes.

Functions inherited from DsBusCustomCodeHandle

This handle inherits the following functions from `DsBusCustomCodeHandle` on page 50:

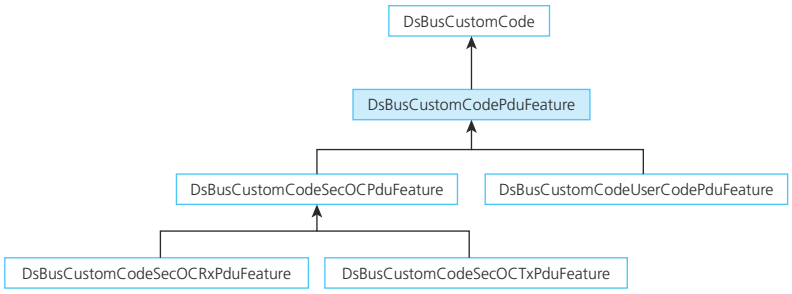
Function	Purpose
`DsBusCustomCode_getDescriptor` on page 67	To get the descriptor of a DsBusCustomCode handle.
`DsBusCustomCode_getName` on page 68	To get the name of an element that is addressed by a DsBusCustomCode handle.

DsBusCustomCodePduFeatureHandle

Purpose

This handle represents a PDU feature.

Inheritance diagram



Include file

`DsBusCustomCode.h`

Functions of the handle

This handle provides the following functions:

Function	Purpose
`DsBusCustomCodePduFeature_getFeatureDataPtr` on page 78	To get the `FeatureDataPtr` pointer of a PDU feature handle.
`DsBusCustomCodePduFeature_getPdu` on page 80	To get the PDU that is accessed by a PDU feature handle.
`DsBusCustomCodePduFeature_setFeatureDataPtr` on page 81	To set the `FeatureDataPtr` pointer of a PDU feature handle.

Functions inherited from DsBusCustomCodeHandle

This handle inherits the following functions from `DsBusCustomCodeHandle` on page 50:

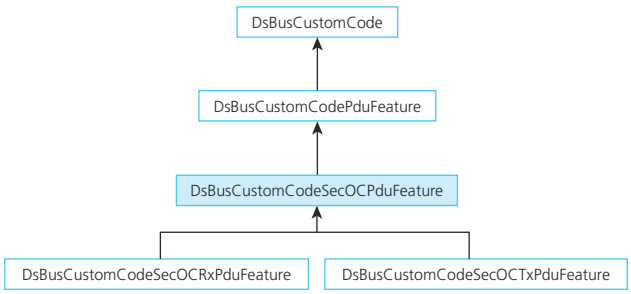
Function	Purpose
`DsBusCustomCode_getDescriptor` on page 67	To get the descriptor of a DsBusCustomCode handle.
`DsBusCustomCode_getName` on page 68	To get the name of an element that is addressed by a DsBusCustomCode handle.

DsBusCustomCodeSecOCPduFeatureHandle

Purpose

This handle represents a SecOC PDU feature.

Inheritance diagram



Include file

`DsBusCustomCode_SecOC.h`

Functions of the handle

This handle provides the following functions:

Function	Purpose
`DsBusCustomCodeSecOCPduFeature_getAuthenticatorPositionOffset` on page 83	To get the offset value of the position of the authentication information in a non-cryptographic secured IPDU.

Function	Purpose
DsBusCustomCodeSecOCPduFeature_getKeyAsString on page 84	To get the SecOC key of a secured IPDU as a string value.
DsBusCustomCodeSecOCPduFeature_getKeyLength on page 86	To get the length of a SecOC key byte array of a secured IPDU.
DsBusCustomCodeSecOCPduFeature_getKeyPtr on page 87	To get the KeyPtr pointer of a SecOC PDU feature handle.
DsBusCustomCodeSecOCPduFeature_setAuthenticatorPositionOffset on page 89	To set an offset value to the position of the authentication information in a non-cryptographic secured IPDU.
DsBusCustomCodeSecOCPduFeature_setKeyLength on page 91	To set the length of a SecOC key byte array of a secured IPDU.
DsBusCustomCodeSecOCPduFeature_setKeyPtr on page 92	To set the KeyPtr pointer of a SecOC PDU feature handle.

Functions inherited from DsBusCustomCodePduFeature Handle

This handle inherits the following functions from DsBusCustomCodePduFeatureHandle on page 53:

Function	Purpose
DsBusCustomCodePduFeature_getFeatureDataPtr on page 78	To get the FeatureDataPtr pointer of a PDU feature handle.
DsBusCustomCodePduFeature_getPdu on page 80	To get the PDU that is accessed by a PDU feature handle.
DsBusCustomCodePduFeature_setFeatureDataPtr on page 81	To set the FeatureDataPtr pointer of a PDU feature handle.

Functions inherited from DsBusCustomCodeHandle

This handle inherits the following functions from DsBusCustomCodeHandle on page 50:

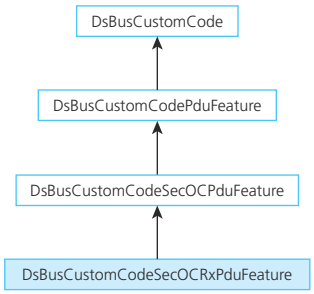
Function	Purpose
DsBusCustomCode_getDescriptor on page 67	To get the descriptor of a DsBusCustomCode handle.
DsBusCustomCode_getName on page 68	To get the name of an element that is addressed by a DsBusCustomCode handle.

DsBusCustomCodeSecOCRxPduFeatureHandle

Purpose

This handle represents a SecOC RX PDU feature.

Inheritance diagram



Include file

`DsBusCustomCode_SecOC.h`

Functions of the handle

This handle provides the following functions:

Function	Purpose
`DsBusCustomCodeSecOCRxPduFeature_getCalculatedFreshnessValue` on page 94	To get the data of the user code that is provided to an RX secured IPDU via the `Calculated Freshness Value` variable.
`DsBusCustomCodeSecOCRxPduFeature_getEnableVerification` on page 95	To get the enable state for verifying the authentication information of an RX secured IPDU.
`DsBusCustomCodeSecOCRxPduFeature_getVerificationResult` on page 97	To get data of the user code that is provided to an RX secured IPDU via the `Verification Result` variable.
`DsBusCustomCodeSecOCRxPduFeature_setCalculatedFreshnessValue` on page 98	To set data of the user code to an RX secured IPDU via the `Calculated Freshness Value` variable.
`DsBusCustomCodeSecOCRxPduFeature_setVerificationResult` on page 100	To set data of the user code to an RX secured IPDU via the `Verification Result` variable.

Functions inherited from DsBusCustomCodeSecOCPduFeatureHandle

This handle inherits the following functions from `DsBusCustomCodeSecOCPduFeatureHandle` on page 54:

Function	Purpose
`DsBusCustomCodeSecOCPduFeature_getAuthenticatorPositionOffset` on page 83	To get the offset value of the position of the authentication information in a non-cryptographic secured IPDU.

Function	Purpose
DsBusCustomCodeSecOCPduFeature_getKeyAsString on page 84	To get the SecOC key of a secured IPDU as a string value.
DsBusCustomCodeSecOCPduFeature_getKeyLength on page 86	To get the length of a SecOC key byte array of a secured IPDU.
DsBusCustomCodeSecOCPduFeature_getKeyPtr on page 87	To get the KeyPtr pointer of a SecOC PDU feature handle.
DsBusCustomCodeSecOCPduFeature_setAuthenticatorPositionOffset on page 89	To set an offset value to the position of the authentication information in a non-cryptographic secured IPDU.
DsBusCustomCodeSecOCPduFeature_setKeyLength on page 91	To set the length of a SecOC key byte array of a secured IPDU.
DsBusCustomCodeSecOCPduFeature_setKeyPtr on page 92	To set the KeyPtr pointer of a SecOC PDU feature handle.

Functions inherited from DsBusCustomCodePduFeature Handle

This handle inherits the following functions from DsBusCustomCodePduFeatureHandle on page 53:

Function	Purpose
DsBusCustomCodePduFeature_getFeatureDataPtr on page 78	To get the FeatureDataPtr pointer of a PDU feature handle.
DsBusCustomCodePduFeature_getPdu on page 80	To get the PDU that is accessed by a PDU feature handle.
DsBusCustomCodePduFeature_setFeatureDataPtr on page 81	To set the FeatureDataPtr pointer of a PDU feature handle.

Functions inherited from DsBusCustomCodeHandle

This handle inherits the following functions from DsBusCustomCodeHandle on page 50:

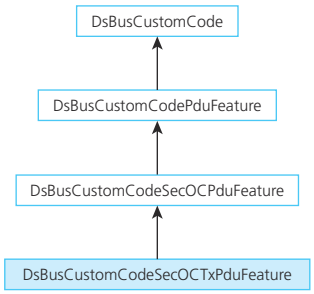
Function	Purpose
DsBusCustomCode_getDescriptor on page 67	To get the descriptor of a DsBusCustomCode handle.
DsBusCustomCode_getName on page 68	To get the name of an element that is addressed by a DsBusCustomCode handle.

DsBusCustomCodeSecOCTxPduFeatureHandle

Purpose

This handle represents a SecOC TX PDU feature.

Inheritance diagram



Include file

`DsBusCustomCode_SecOC.h`

Functions of the handle

This handle provides the following functions:

Function	Purpose
`DsBusCustomCodeSecOCTxPduFeature_getAuthenticationType` on page 101	To get the enable state of the authentication type manipulation that is specified for a TX secured IPDU.
`DsBusCustomCodeSecOCTxPduFeature_getEnableAuthentication` on page 103	To get the enable state of the authentication value calculation that is specified for a TX secured IPDU.
`DsBusCustomCodeSecOCTxPduFeature_getEnableFreshnessValueCalculation` on page 105	To get the enable state of the freshness value calculation that is specified for a TX secured IPDU.
`DsBusCustomCodeSecOCTxPduFeature_getFreshnessValueOffset` on page 106	To get the offset value that is specified for the freshness value of a TX secured IPDU.
`DsBusCustomCodeSecOCTxPduFeature_getUserDefinedAuthenticatorPtr` on page 108	To get the pointer to the byte array that provides the user-defined authentication value, which is specified for a TX secured IPDU.
`DsBusCustomCodeSecOCTxPduFeature_getUserDefinedFreshnessValue` on page 109	To get the user-defined freshness value that is specified for a TX secured IPDU.

Functions inherited from DsBusCustomCodeSecOCPduFeatureHandle

This handle inherits the following functions from `DsBusCustomCodeSecOCPduFeatureHandle` on page 54:

Function	Purpose
`DsBusCustomCodeSecOCPduFeature_getAuthenticatorPositionOffset` on page 83	To get the offset value of the position of the authentication information in a non-cryptographic secured IPDU.
`DsBusCustomCodeSecOCPduFeature_getKeyAsString` on page 84	To get the SecOC key of a secured IPDU as a string value.
`DsBusCustomCodeSecOCPduFeature_getKeyLength` on page 86	To get the length of a SecOC key byte array of a secured IPDU.
`DsBusCustomCodeSecOCPduFeature_getKeyPtr` on page 87	To get the `KeyPtr` pointer of a SecOC PDU feature handle.
`DsBusCustomCodeSecOCPduFeature_setAuthenticatorPositionOffset` on page 89	To set an offset value to the position of the authentication information in a non-cryptographic secured IPDU.
`DsBusCustomCodeSecOCPduFeature_setKeyLength` on page 91	To set the length of a SecOC key byte array of a secured IPDU.
`DsBusCustomCodeSecOCPduFeature_setKeyPtr` on page 92	To set the `KeyPtr` pointer of a SecOC PDU feature handle.

Functions inherited from DsBusCustomCodePduFeatureHandle

This handle inherits the following functions from `DsBusCustomCodePduFeatureHandle` on page 53:

Function	Purpose
`DsBusCustomCodePduFeature_getFeatureDataPtr` on page 78	To get the `FeatureDataPtr` pointer of a PDU feature handle.
`DsBusCustomCodePduFeature_getPdu` on page 80	To get the PDU that is accessed by a PDU feature handle.
`DsBusCustomCodePduFeature_setFeatureDataPtr` on page 81	To set the `FeatureDataPtr` pointer of a PDU feature handle.

Functions inherited from DsBusCustomCodeHandle

This handle inherits the following functions from `DsBusCustomCodeHandle` on page 50:

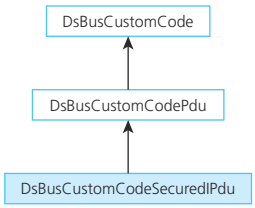
Function	Purpose
`DsBusCustomCode_getDescriptor` on page 67	To get the descriptor of a DsBusCustomCode handle.
`DsBusCustomCode_getName` on page 68	To get the name of an element that is addressed by a DsBusCustomCode handle.

DsBusCustomCodeSecuredIPduHandle

Purpose

This handle represents a secured IPDU.

Inheritance diagram



Include file

`DsBusCustomCode_SecOC.h`

Functions of the handle

This handle provides the following functions:

Function	Purpose
`DsBusCustomCodeSecuredIPdu_getAuthAlgorithmName` on page 111	To get the value of the AUTOSAR `AuthAlgorithm` attribute of a secured IPDU.
`DsBusCustomCodeSecuredIPdu_getAuthDataFreshnessLength` on page 112	To get the value of the AUTOSAR `AuthDataFreshnessLength` attribute of a secured IPDU.
`DsBusCustomCodeSecuredIPdu_getAuthDataFreshnessStartPosition` on page 113	To get the value of the AUTOSAR `AuthDataFreshnessStartPosition` attribute of a secured IPDU.
`DsBusCustomCodeSecuredIPdu_getAuthInfoTxLength` on page 115	To get the value of the AUTOSAR `AuthInfoTxLength` attribute of a secured IPDU.
`DsBusCustomCodeSecuredIPdu_getAuthPduHeaderLength` on page 116	To get the value of the AUTOSAR `SecOCAuthPduHeaderLength` attribute of a secured IPDU.
`DsBusCustomCodeSecuredIPdu_getAuthenticPdu` on page 118	To get the authentic IPDU that is secured by a secured IPDU.
`DsBusCustomCodeSecuredIPdu_getAuthenticationBuildAttempts` on page 119	To get the value of the AUTOSAR `AuthenticationBuildAttempts` attribute of a secured IPDU.
`DsBusCustomCodeSecuredIPdu_getAuthenticationRetries` on page 121	To get the value of the AUTOSAR `AuthenticationRetries` attribute of a secured IPDU.
`DsBusCustomCodeSecuredIPdu_getDataId` on page 122	To get the value of the AUTOSAR `DataId` attribute of a secured IPDU.
`DsBusCustomCodeSecuredIPdu_getFreshnessCounterSyncAttempts` on page 123	To get the value of the AUTOSAR `FreshnessCounterSyncAttempts` attribute of a secured IPDU.

Function	Purpose
DsBusCustomCodeSecuredIPdu_getFreshnessTimestampPeriodFactor on page 125	To get the value of the AUTOSAR FreshnessTimestampTimePeriodFactor attribute of a secured IPDU.
DsBusCustomCodeSecuredIPdu_getFreshnessValueId on page 127	To get the value of the AUTOSAR FreshnessValueId attribute of a secured IPDU.
DsBusCustomCodeSecuredIPdu_getFreshnessValueLength on page 128	To get the value of the AUTOSAR FreshnessValueLength attribute of a secured IPDU.
DsBusCustomCodeSecuredIPdu_getFreshnessValueTxLength on page 129	To get the value of the AUTOSAR FreshnessValueTxLength attribute of a secured IPDU.
DsBusCustomCodeSecuredIPdu_getKeyId on page 131	To get the value of the AUTOSAR KeyID attribute of a secured IPDU.
DsBusCustomCodeSecuredIPdu_getMessageLinkLength on page 132	To get the value of the AUTOSAR MessageLinkLength attribute of a secured IPDU.
DsBusCustomCodeSecuredIPdu_getMessageLinkPosition on page 133	To get the value of the AUTOSAR MessageLinkPosition attribute of a secured IPDU.
DsBusCustomCodeSecuredIPdu_getRxSecurityVerification on page 135	To get the value of the AUTOSAR RxSecurityVerification attribute of a secured IPDU.
DsBusCustomCodeSecuredIPdu_getSecuredAreaLength on page 136	To get the value of the AUTOSAR SecuredAreaLength attribute of a secured IPDU.
DsBusCustomCodeSecuredIPdu_getSecuredAreaOffset on page 138	To get the value of the AUTOSAR SecuredAreaOffset attribute of a secured IPDU.
DsBusCustomCodeSecuredIPdu_getTimeStampRxAcceptanceWindow on page 139	To get the value of the AUTOSAR TimestampRxAcceptanceWindow attribute of a secured IPDU.
DsBusCustomCodeSecuredIPdu_getUseAsCryptographicPdu on page 141	To get the value of the AUTOSAR UseAsCryptographicIPdu attribute of a secured IPDU.
DsBusCustomCodeSecuredIPdu_getUseAuthDataFreshness on page 142	To get the value of the AUTOSAR UseAuthDataFreshness attribute of a secured IPDU.
DsBusCustomCodeSecuredIPdu_getUseFreshnessTimestamp on page 144	To get the value of the AUTOSAR UseFreshnessTimestamp attribute of a secured IPDU.

Functions inherited from DsBusCustomCodePduHandle

This handle inherits the following functions from `DsBusCustomCodePduHandle` on page 52:

Function	Purpose
`DsBusCustomCodePdu_getCanChannelName` on page 70	To get the name of a CAN channel.
`DsBusCustomCodePdu_getCanFrameTriggering` on page 71	To get the frame triggering of a CAN frame.
`DsBusCustomCodePdu_getIsTx` on page 73	To get the direction of a PDU.
`DsBusCustomCodePdu_getLinFrameTriggering` on page 74	To get the frame triggering of a LIN frame.
`DsBusCustomCodePdu_getSduDataPtr` on page 76	To get the `SduDataPtr` pointer of a PDU.
`DsBusCustomCodePdu_getSduLength` on page 77	To get the SDU length of a PDU in bytes.

Functions inherited from DsBusCustomCodeHandle

This handle inherits the following functions from `DsBusCustomCodeHandle` on page 50:

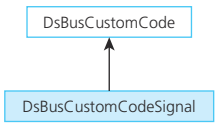
Function	Purpose
`DsBusCustomCode_getDescriptor` on page 67	To get the descriptor of a DsBusCustomCode handle.
`DsBusCustomCode_getName` on page 68	To get the name of an element that is addressed by a DsBusCustomCode handle.

DsBusCustomCodeSignalHandle

Purpose

This handle represents a signal of a PDU.

Inheritance diagram



Include file

`DsBusCustomCode.h`

Functions of the handle

This handle provides the following functions:

Function	Purpose
`DsBusCustomCodeSignal_getEndianness` on page 145	To get the endianness of a signal.
`DsBusCustomCodeSignal_getLength` on page 147	To get the length of a signal.
`DsBusCustomCodeSignal_getStartBitPosition` on page 148	To get the start bit position of a signal.

Functions inherited from DsBusCustomCodeHandle

This handle inherits the following functions from `DsBusCustomCodeHandle` on page 50:

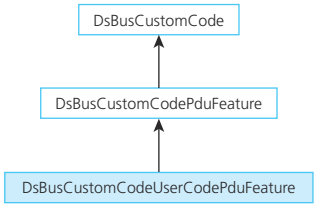
Function	Purpose
`DsBusCustomCode_getDescriptor` on page 67	To get the descriptor of a DsBusCustomCode handle.
`DsBusCustomCode_getName` on page 68	To get the name of an element that is addressed by a DsBusCustomCode handle.

DsBusCustomCodeUserCodePduFeatureHandle

Purpose

This handle represents the PDU User Code feature of a PDU.

Inheritance diagram



Include file

`DsBusCustomCode_PduUserCode.h`

Functions of the handle

This handle provides the following functions:

Function	Purpose
`DsBusCustomCodeUserCodePduFeature_getNumberOfUserSignals` on page 149	To get the number of user signals that are specified for a PDU via the PDU User Code feature.
`DsBusCustomCodeUserCodePduFeature_getResult` on page 150	To get the data of the user code that is provided to an RX PDU via the PDU User Code feature.
`DsBusCustomCodeUserCodePduFeature_getUserSignals` on page 152	To get the array of the user signals that are specified for a PDU via the PDU User Code feature.
`DsBusCustomCodeUserCodePduFeature_setResult` on page 153	To set data of the user code to an RX PDU via the PDU User Code feature.

Functions inherited from DsBusCustomCodePduFeature Handle

This handle inherits the following functions from `DsBusCustomCodePduFeatureHandle` on page 53:

Function	Purpose
`DsBusCustomCodePduFeature_getFeatureDataPtr` on page 78	To get the `FeatureDataPtr` pointer of a PDU feature handle.
`DsBusCustomCodePduFeature_getPdu` on page 80	To get the PDU that is accessed by a PDU feature handle.
`DsBusCustomCodePduFeature_setFeatureDataPtr` on page 81	To set the `FeatureDataPtr` pointer of a PDU feature handle.

Functions inherited from DsBusCustomCodeHandle

This handle inherits the following functions from `DsBusCustomCodeHandle` on page 50:

Function	Purpose
`DsBusCustomCode_getDescriptor` on page 67	To get the descriptor of a DsBusCustomCode handle.
`DsBusCustomCode_getName` on page 68	To get the name of an element that is addressed by a DsBusCustomCode handle.

Functions of the Bus Custom Code Interface

Where to go from here

Information in this section

DsBusCustomCode_getDescriptor

Purpose

To get the descriptor of a DsBusCustomCode handle.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	User code
Parameter data recipient	User code

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function gets the descriptor of a DsBusCustomCode handle.

Syntax

```
Std_ReturnType DsBusCustomCode_getDescriptor (DsBusCustomCodeHandle Handle, char **Descriptor)
```

Parent handle

This function is provided by `DsBusCustomCodeHandle` on page 50.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During run time
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`Handle`	A DsBusCustomCode handle.
`Descriptor`	The descriptor of the handle.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

```
// Declare a pointer for the descriptor
char* featureDescriptor;
// Let the pointer point to the descriptor
DsBusCustomCode_getDescriptor (PduFeatureHandle, &featureDescriptor);

```

DsBusCustomCode_getName

Purpose

To get the name of an element that is addressed by a DsBusCustomCode handle.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	Communication matrix (via bus implementation software)
Parameter data recipient	User code

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function gets the name of an element that is addressed by a DsBusCustomCode handle.

Syntax

```
Std_ReturnType DsBusCustomCode_getName (DsBusCustomCodeHandle Handle, char **Name)
```

Parent handle

This function is provided by `DsBusCustomCodeHandle` on page 50.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During run time
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`Handle`	A DsBusCustomCode handle.
`Name`	The name of the element that is addressed by the DsBusCustomCode handle.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

```
// Declare pointer for name
char* name;
// Let the pointer point to the name
DsBusCustomCode_getName(PduFeatureHandle, &name);
```

DsBusCustomCodePdu_getCanChannelName

Purpose

To get the name of a CAN channel.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	Communication matrix (via bus implementation software)
Parameter data recipient	User code

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function gets the the name of a CAN channel.

Syntax

```
Std_ReturnType DsBusCustomCodePdu_getCanChannelName (DsBusCustomCodePduHandle PduHandle, char **Name)
```

Parent handle

This function is provided by `DsBusCustomCodePduHandle` on page 52.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During run time
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduHandle`	A DsBusCustomCode PDU handle.
`Name`	The name of the CAN channel.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.

Value	Description
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

```
// Declare pointer for CAN channel name
char* My_CanChannelName;

// Let the pointer point to the CAN channel name
DsBusCustomCodePdu_getCanChannelName(pduHandle, &My_CanChannelName);
```

DsBusCustomCodePdu_getCanFrameTriggering

Purpose

To get the frame triggering of a CAN frame.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	Communication matrix (via bus implementation software)
Parameter data recipient	User code

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function gets the following information on the frame triggering of a CAN frame:

- CAN frame identifier
- Identifier format, i.e., standard identifier format (11-bit) or extended identifier format (29-bit)
- CAN frame type, i.e., classic CAN frame or CAN FD frame

The information is provided as a `struct`.

To get the frame triggering, this function has to access a PDU that is included in the related frame. However, this function can get the frame triggering only if the accessed PDU is included in exactly one CAN frame, i.e., there is only one CAN frame triggering.

Syntax

```
Std_ReturnType DsBusCustomCodePdu_getCanFrameTriggering(DsBusCustomCodePduHandle PduHandle,
DsBusCustomCodeCanFrameTriggeringType *DsBusCustomCodeCanFrameTriggering)
```

Parent handle

This function is provided by `DsBusCustomCodePduHandle` on page 52.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During run time
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduHandle`	A DsBusCustomCode PDU handle.
`DsBusCustomCodeCanFrameTriggering`	A `struct` of three elements providing the CAN frame triggering: ▪ `(.Identifier)`: Providing the CAN frame identifier. ▪ `(.AddressingMode)`: Providing the identifier format: ▪ `1` = standard identifier format (11-bit) ▪ `2` = extended identifier format (29-bit) ▪ `(.FrameBehavior)`: Providing the CAN frame type: ▪ `1` = classic CAN frame ▪ `2` = CAN FD frame

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

```
// Declare CAN frame triggering variables (frame identifier, identifier format, frame type)
DsBusCustomCodeCanFrameTriggeringType My_CanFrameTriggering;
uint32 My_ID;
uint8 My_AddressingMode;
```

```
uint8 My_FrameBehavior;
...
// Let the pointer point to the CAN frame triggering structure and extract frame triggering variables
DsBusCustomCodePdu_getCanFrameTriggering(pduHandle, &My_CanFrameTriggering);
My_ID = (uint32)My_CanFrameTriggering.Identifier;
My_AddressingMode = (uint8)My_CanFrameTriggering.AddresingMode;
My_FrameBehavior = (uint8)My_CanFrameTriggering.FrameBehavior;
```

DsBusCustomCodePdu_getIsTx

Purpose

To get the direction of a PDU.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	Communication matrix (via bus implementation software)
Parameter data recipient	User code

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function gets the direction of a PDU, i.e, whether a PDU is transmitted (TX PDU) or received (RX PDU).

Syntax

```
Std_ReturnType DsBusCustomCodePdu_getIsTx (DsBusCustomCodePduHandle PduHandle, boolean *IsTx)
```

Parent handle

This function is provided by `DsBusCustomCodePduHandle` on page 52.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During run time
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduHandle`	A DsBusCustomCode PDU handle.
`IsTx`	The direction of the PDU: ▪ `TRUE`: TX PDU ▪ `FALSE`: RX PDU

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

-

DsBusCustomCodePdu_getLinFrameTriggering

Purpose

To get the frame triggering of a LIN frame.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	Communication matrix (via bus implementation software)
Parameter data recipient	User code

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function gets the following information on the frame triggering of a LIN frame:

- LIN frame identifier (ID), which is part of the protected identifier (PID) of the LIN frame header
- Checksum type, i.e., classic or enhanced checksum calculation type

The information is provided as a `struct`.

To get the frame triggering, this function has to access a PDU that is included in the related frame. However, this function can get the frame triggering only if the accessed PDU is included in exactly one LIN frame, i.e., there is only one LIN frame triggering.

Syntax

```
Std_ReturnType DsBusCustomCodePdu_getLinFrameTriggering(DsBusCustomCodePduHandle PduHandle,
DsBusCustomCodeLinFrameTriggeringType *DsBusCustomCodeLinFrameTriggering)
```

Parent handle

This function is provided by `DsBusCustomCodePduHandle` on page 52.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During run time
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduHandle`	A DsBusCustomCode PDU handle.
`DsBusCustomCodeLinFrameTriggering`	A `struct` of two elements providing the LIN frame triggering: ▪ `(.Identifier)`: Providing the LIN frame identifier. ▪ `(.ChecksumType)`: Providing the checksum type: ▪ `1` = classic checksum calculation type ▪ `2` = enhanced checksum calculation type

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

```
// Declare LIN frame triggering variables (frame identifier, checksum type)
DsBusCustomCodeLinFrameTriggeringType My_LinFrameTriggering;
uint32 My_ID;
uint8 My_ChecksumType;
...
// Let the pointer point to the LIN frame triggering structure and extract frame triggering variables
DsBusCustomCodePdu_getLinFrameTriggering(pduHandle, &My_LinFrameTriggering);
My_ID = (uint32)My_LinFrameTriggering.Identifier;
My_ChecksumType = (uint8)My_LinFrameTriggering.ChecksumType;
```

DsBusCustomCodePdu_getSduDataPtr

Purpose

To get the `SduDataPtr` pointer of a PDU.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	Communication matrix (via bus implementation software)
Parameter data recipient	User code

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function gets the `SduDataPtr` pointer of a PDU. The pointer points to the SDU bytes, i.e., the data bytes of the PDU.

Syntax

```
Std_ReturnType DsBusCustomCodePdu_getSduDataPtr (DsBusCustomCodePduHandle PduHandle, uint8 **SduDataPtr)
```

Parent handle

This function is provided by `DsBusCustomCodePduHandle` on page 52.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During run time
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduHandle`	A DsBusCustomCode PDU handle.
`SduDataPtr`	The pointer to the SDU bytes.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

-

DsBusCustomCodePdu_getSduLength

Purpose

To get the SDU length of a PDU in bytes.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	Communication matrix (via bus implementation software)
Parameter data recipient	User code

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function gets the SDU length, i.e., the payload length of a PDU in bytes.

Syntax

```
Std_ReturnType DsBusCustomCodePdu_getSduLength (DsBusCustomCodePduHandle PduHandle, uint32 *SduLength)
```

Parent handle

This function is provided by `DsBusCustomCodePduHandle` on page 52.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During run time
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduHandle`	A DsBusCustomCode PDU handle.
`SduLength`	The SDU length in bytes.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

-

DsBusCustomCodePduFeature_getFeatureDataPtr

Purpose

To get the `FeatureDataPtr` pointer of a PDU feature handle.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	User code
Parameter data recipient	User code

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function gets the `FeatureDataPtr` pointer of a PDU feature handle. To get the pointer, it must be set beforehand by using the `DsBusCustomCodePduFeature_setFeatureDataPtr` function. Refer to `DsBusCustomCodePduFeature_setFeatureDataPtr` on page 81.

In the user code, you can let the pointer point to an arbitrary, user-defined data structure, e.g., a `struct`. By using this function, you can access the data of this data structure.

For example, you use the data structure to store data that alternates during subsequent PDU feature execution calls, such as counter values or state information. In this case, you can use this function to access the data that was written to the data structure during the previous PDU feature execution call.

Syntax

```
Std_ReturnType DsBusCustomCodePduFeature_getFeatureDataPtr (DsBusCustomCodePduFeatureHandle PduFeatureHandle, uint8
**FeatureDataPtr)
```

Parent handle

This function is provided by `DsBusCustomCodePduFeatureHandle` on page 53.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During run time
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduFeatureHandle`	A DsBusCustomCode PDU feature handle.
`FeatureDataPtr`	The `FeatureDataPtr` pointer of the PDU feature handle. The data type of the pointer is `uint8`. However, in the user code you can cast this data type to the data type of the data structure to which the pointer points.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

-

DsBusCustomCodePduFeature_getPdu

Purpose

To get the PDU that is accessed by a PDU feature handle.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	Communication matrix (via bus implementation software)
Parameter data recipient	User code

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function gets the PDU that is accessed by a PDU feature handle. The function gets the PDU via its PDU handle.

Syntax

```
Std_ReturnType DsBusCustomCodePduFeature_getPdu (DsBusCustomCodePduFeatureHandle PduFeatureHandle,
DsBusCustomCodePduHandle *PduHandle)
```

Parent handle

This function is provided by `DsBusCustomCodePduFeatureHandle` on page 53.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During run time
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduFeatureHandle`	A DsBusCustomCode PDU feature handle.
`PduHandle`	The PDU handle that addresses the PDU which is accessed by the PDU feature handle.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

-

DsBusCustomCodePduFeature_setFeatureDataPtr

Purpose

To set the `FeatureDataPtr` pointer of a PDU feature handle.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	User code
Parameter data recipient	User code

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function sets the `FeatureDataPtr` pointer of a PDU feature handle.

In the user code, you can let the pointer point to an arbitrary, user-defined data structure, e.g., a `struct`. When you set the `FeatureDataPtr` pointer by using this function during the initialization phase of an executable application (i.e., real-time application or offline simulation application), you can access the data of the data structure by calling the `DsBusCustomCodePduFeature_getFeatureDataPtr` function during PDU feature execution calls. Refer to `DsBusCustomCodePduFeature_getFeatureDataPtr` on page 78.

For example, you can use a user-defined data structure to store data that alternates during different feature execution calls such as counter values and state information. By using the `FeatureDataPtr` pointer, you can provide this alternating data to the user code.

Syntax

```
Std_ReturnType DsBusCustomCodePduFeature_setFeatureDataPtr (DsBusCustomCodePduFeatureHandle PduFeatureHandle, uint8
*FeatureDataPtr)
```

Parent handle

This function is provided by `DsBusCustomCodePduFeatureHandle` on page 53.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During initialization
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduFeatureHandle`	A DsBusCustomCode PDU feature handle.
`FeatureDataPtr`	The `FeatureDataPtr` pointer of the PDU feature handle. The data type of the pointer is `uint8`. However, in the user code you can cast this data type to the data type of the data structure to which the pointer points.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.

Value	Description
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

-

DsBusCustomCodeSecOCPduFeature_getAuthenticatorPositionOffset

Purpose

To get the offset value of the position of the authentication information in a non-cryptographic secured IPDU.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	User code
Parameter data recipient	User code

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function gets the offset value of the position of the authentication information (i.e., the freshness value and the authenticator) in a non-cryptographic secured IPDU. The offset value indicates the deviation in bits of the current position of the authentication information from the position that is specified in the communication matrix. If the offset value is `0`, there is no deviation and the current position of the authentication information is the position that is specified in the communication matrix. You can specify a user-defined offset value by using the `DsBusCustomCodeSecOCPduFeature_setAuthenticatorPositionOffset` function. Refer to `DsBusCustomCodeSecOCPduFeature_setAuthenticatorPositionOffset` on page 89.

Syntax

```
Std_ReturnType DsBusCustomCodeSecOCPduFeature_getAuthenticatorPositionOffset(DsBusCustomCodePduFeatureHandle
PduFeatureHandle, sint32 *AuthenticatorPositionOffset)
```

Parent handle

This function is provided by `DsBusCustomCodeSecOCPduFeatureHandle` on page 54.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During run time
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduFeatureHandle`	A DsBusCustomCode PDU feature handle.
`AuthenticatorPositionOffset`	The deviation in bits of the current position of the authentication information in the secured IPDU from the position that is specified in the communication matrix.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

-

DsBusCustomCodeSecOCPduFeature_getKeyAsString

Purpose

To get the SecOC key of a secured IPDU as a string value.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	Communication matrix (via bus implementation software)
Parameter data recipient	User code

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function can get the SecOC key of a secured IPDU as a string value. To get the SecOC key, it must be specified in the communication matrix. The implementation of the SecOC key in the communication matrix is customer-specific.

Via a user-defined C-function, you can parse the key value in a byte array, for example. When you do this, you can use the `DsBusCustomCodeSecOCPduFeature_setKeyLength` function to specify the length of the byte array. If you set the `KeyPtr` pointer of the SecOC PDU feature handle by using the `DsBusCustomCodeSecOCPduFeature_setKeyPtr` function, you can access the data of the byte array by using the `KeyPtr` pointer. Refer to `DsBusCustomCodeSecOCPduFeature_setKeyLength` on page 91 and `DsBusCustomCodeSecOCPduFeature_setKeyPtr` on page 92.

Syntax

```
Std_ReturnType DsBusCustomCodeSecOCPduFeature_getKeyAsString (DsBusCustomCodePduFeatureHandle PduFeatureHandle, char
**KeyAsString)
```

Parent handle

This function is provided by `DsBusCustomCodeSecOCPduFeatureHandle` on page 54.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During initialization
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduFeatureHandle`	A DsBusCustomCode PDU feature handle.
`KeyAsString`	The SecOC key that is specified in the communication matrix. The key is provided as a string value.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

-

DsBusCustomCodeSecOCPduFeature_getKeyLength

Purpose

To get the length of a SecOC key byte array of a secured IPDU.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	User code
Parameter data recipient	User code

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function gets the length of a SecOC key byte array of a secured IPDU. To get the length, it must be set beforehand by using the `DsBusCustomCodeSecOCPduFeature_setKeyLength` function. Refer to `DsBusCustomCodeSecOCPduFeature_setKeyLength` on page 91.

Syntax

```
Std_ReturnType DsBusCustomCodeSecOCPduFeature_getKeyLength (DsBusCustomCodePduFeatureHandle PduFeatureHandle, uint32
*KeyLength)
```

Parent handle

This function is provided by `DsBusCustomCodeSecOCPduFeatureHandle` on page 54.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During run time
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduFeatureHandle`	A DsBusCustomCode PDU feature handle.
`KeyLength`	The length of the SecOC key byte array in bytes.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

-

DsBusCustomCodeSecOCPduFeature_getKeyPtr

Purpose

To get the `KeyPtr` pointer of a SecOC PDU feature handle.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	User code
Parameter data recipient	User code

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function gets the `KeyPtr` pointer of a SecOC PDU feature handle. To get the pointer, it must be set beforehand by using the `DsBusCustomCodeSecOCPduFeature_setKeyPtr` function. Refer to `DsBusCustomCodeSecOCPduFeature_setKeyPtr` on page 92.

For example, you can use the `KeyPtr` pointer to access the SecOC key of a secured IPDU: In the user code, you can specify a byte array that contains the SecOC key of a secured IPDU and let the pointer point to this array. By using this function, you can access the data of this array during subsequent PDU feature execution calls.

Syntax

```
Std_ReturnType DsBusCustomCodeSecOCPduFeature_getKeyPtr (DsBusCustomCodePduFeatureHandle PduFeatureHandle, uint8
**KeyPtr)
```

Parent handle

This function is provided by `DsBusCustomCodeSecOCPduFeatureHandle` on page 54.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During run time
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduFeatureHandle`	A DsBusCustomCode PDU feature handle.
`KeyPtr`	The `KeyPtr` pointer of the SecOC PDU feature handle. The data type of the pointer is `uint8`. However, in the user code you can cast this data type to the data type of the byte array to which the pointer points.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example -

DsBusCustomCodeSecOCPduFeature_setAuthenticatorPositionOffset

Purpose

To set an offset value to the position of the authentication information in a non-cryptographic secured IPDU.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	User code
Parameter data recipient	Bus implementation software (only for internal use)

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function sets an offset value to the position of the authentication information in a non-cryptographic secured IPDU, i.e., in a secured IPDU whose `UseAsCryptographicIPdu` attribute is unspecified or set to `False`.

The offset value applies to the position of the authentication information, i.e., to the freshness value and the authenticator, that is included in the non-cryptographic secured IPDU. The offset value determines the deviation in bits of the current position of the authentication information to the position that is specified in the communication matrix. By default, the offset value is `0`, i.e., there is no deviation and the current position of the authentication information is the position that is specified in the communication matrix. You can specify a positive deviation to the originally specified position (i.e., offset value `> 0`) or a negative deviation (i.e., offset value `< 0`).

For example, you can use this function to specify the position of the authentication information if the secured IPDU has a dynamic payload length. To get the payload length in the current PDU feature execution call (i.e., the current sampling step), you can use the `DsBusCustomCodePdu_getSduLength` function. Refer to `DsBusCustomCodePdu_getSduLength` on page 77.

The bus implementation software internally uses the specified offset value to determine the start position of the authentication information in the secured IPDU, e.g., for inspecting the truncated freshness and authenticator values of a received secured IPDU.

The specified offset value applies only to the current PDU feature execution call: The bus implementation software resets the offset value to `0` before the next PDU feature execution call (i.e., before the next sampling step).

At run time, you can access the offset value by using the `DsBusCustomCodeSecOCPduFeature_getAuthenticatorPositionOffset` function. Refer to `DsBusCustomCodeSecOCPduFeature_getAuthenticatorPositionOffset` on page 83.

Syntax

```
Std_ReturnType DsBusCustomCodeSecOCPduFeature_setAuthenticatorPositionOffset(DsBusCustomCodePduFeatureHandle
PduFeatureHandle, sint32 AuthenticatorPositionOffset)
```

Parent handle

This function is provided by `DsBusCustomCodeSecOCPduFeatureHandle` on page 54.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During run time
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduFeatureHandle`	A DsBusCustomCode PDU feature handle.
`AuthenticatorPositionOffset`	The offset value in bits that applies to the position of the authentication information that it is specified in the communication matrix for a non-cryptographic secured IPDU.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

-

DsBusCustomCodeSecOCPduFeature_setKeyLength

Purpose

To set the length of a SecOC key byte array of a secured IPDU.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	User code
Parameter data recipient	User code

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function sets the length of a SecOC key byte array of a secured IPDU.

In the user code, you can use a byte array to store the SecOC key of the secured IPDU. The SecOC key can directly be specified in the user code. The SecOC key can also be specified in the communication matrix but this requires a customer-specific implementation of the SecOC key. Depending on the customer-specific implementation, you can get the SecOC key by using the `DsBusCustomCodeSecOCPduFeature_getKeyAsString` function. Refer to `DsBusCustomCodeSecOCPduFeature_getKeyAsString` on page 84.

At run time, you can access the length of the SecOC key byte array by using the `DsBusCustomCodeSecOCPduFeature_getKeyLength` function. Refer to `DsBusCustomCodeSecOCPduFeature_getKeyLength` on page 86.

Syntax

```
Std_ReturnType DsBusCustomCodeSecOCPduFeature_setKeyLength (DsBusCustomCodePduFeatureHandle PduFeatureHandle, uint32
KeyLength)
```

Parent handle

This function is provided by `DsBusCustomCodeSecOCPduFeatureHandle` on page 54.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During initialization
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduFeatureHandle`	A DsBusCustomCode PDU feature handle.
`KeyLength`	The length of the SecOC key byte array in bytes.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

-

DsBusCustomCodeSecOCPduFeature_setKeyPtr

Purpose

To set the `KeyPtr` pointer of a SecOC PDU feature handle.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	User code
Parameter data recipient	User code

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function sets the `KeyPtr` pointer of the SecOC PDU feature handle.

For example, you can use the `KeyPtr` pointer to access the SecOC key of a secured IPDU: In the user code, you can specify a byte array that contains the SecOC key of a secured IPDU and let the pointer point to this array. When you set the `KeyPtr` pointer by using this function during the initialization phase of an executable application (i.e., real-time application or offline simulation application), you can access the data of the array by calling the

`DsBusCustomCodeSecOCPduFeature_getKeyPtr` function during PDU feature execution calls. Refer to `DsBusCustomCodeSecOCPduFeature_getKeyPtr` on page 87.

Syntax

```
Std_ReturnType DsBusCustomCodeSecOCPduFeature_setKeyPtr (DsBusCustomCodePduFeatureHandle PduFeatureHandle, uint8 *KeyPtr)
```

Parent handle

This function is provided by `DsBusCustomCodeSecOCPduFeatureHandle` on page 54.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During initialization
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduFeatureHandle`	A DsBusCustomCode PDU feature handle.
`KeyPtr`	The `KeyPtr` pointer of the SecOC PDU feature handle. The data type of the pointer is `uint8`. However, in the user code you can cast this data type to the data type of the byte array the pointer points to.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

-

DsBusCustomCodeSecOCRxPduFeature_getCalculatedFreshnessValue

Purpose

To get the data of the user code that is provided to an RX secured IPDU via the `Calculated Freshness Value` variable.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	User code
Parameter data recipient	Bus implementation software, provides data via TRC and/or model variable

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function gets the data of the user code that is provided to an RX secured IPDU via the `Calculated Freshness Value` variable.

To get the data of the `Calculated Freshness Value` variable, it must be set beforehand by using the `DsBusCustomCodeSecOCRxPduFeature_setCalculatedFreshnessValue` function. Refer to `DsBusCustomCodeSecOCRxPduFeature_setCalculatedFreshnessValue` on page 98.

Syntax

```
Std_ReturnType DsBusCustomCodeSecOCRxPduFeature_getCalculatedFreshnessValue (DsBusCustomCodePduFeatureHandle
PduFeatureHandle, uint64 *CalculatedFreshnessValue)
```

Parent handle

This function is provided by `DsBusCustomCodeSecOCRxPduFeatureHandle` on page 56.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During run time
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduFeatureHandle`	A DsBusCustomCode PDU feature handle.
`CalculatedFreshnessValue`	The data that is provided to the `Calculated Freshness Value` variable by the user code.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

-

DsBusCustomCodeSecOCRxPduFeature_getEnableVerification

Purpose

To get the enable state for verifying the authentication information of an RX secured IPDU.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	TRC and/or model variable, or constant value provided by bus implementation software
Parameter data recipient	User code

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function gets the enable state for verifying the authentication information of an RX secured IPDU.

In the user code, you can use the provided state value to enable or disable the verification of the authentication information of the RX secured IPDU. The specifications in the user code determine how the state value is evaluated. To

ensure full compatibility with all bus implementation software tools, it is recommended to use the following values:

Value	Use For
0x00	Disable verification.
0x01	Enable verification.

Syntax

```
Std_ReturnType DsBusCustomCodeSecOCRxPduFeature_getEnableVerification (DsBusCustomCodePduFeatureHandle PduFeatureHandle,
uint32 *EnableVerification)
```

Parent handle

This function is provided by `DsBusCustomCodeSecOCRxPduFeatureHandle` on page 56.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During run time
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduFeatureHandle`	A DsBusCustomCode PDU feature handle.
`EnableVerification`	The enable state of the authentication information verification that is provided by bus implementation software.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

-

DsBusCustomCodeSecOCRxPduFeature_getVerificationResult

Purpose

To get data of the user code that is provided to an RX secured IPDU via the `Verification Result` variable.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	User code
Parameter data recipient	Bus implementation software, provides data via TRC and/or model variable

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function gets the data of the user code that is provided to an RX secured IPDU via the `Verification Result` variable.

To get the data of the `Verification Result` variable, it must be set beforehand by using the `DsBusCustomCodeSecOCRxPduFeature_setVerificationResult` function. Refer to `DsBusCustomCodeSecOCRxPduFeature_setVerificationResult` on page 100.

Syntax

```
Std_ReturnType DsBusCustomCodeSecOCRxPduFeature_getVerificationResult (DsBusCustomCodePduFeatureHandle PduFeatureHandle,
uint32 *VerificationResult)
```

Parent handle

This function is provided by `DsBusCustomCodeSecOCRxPduFeatureHandle` on page 56.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During run time
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduFeatureHandle`	A DsBusCustomCode PDU feature handle.
`VerificationResult`	The data that is provided to the `Verification Result` variable by the user code.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

-

DsBusCustomCodeSecOCRxPduFeature_setCalculatedFreshnessValue

Purpose

To set data of the user code to an RX secured IPDU via the `Calculated Freshness Value` variable.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	User code
Parameter data recipient	Bus implementation software, provides data via TRC and/or model variable

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function sets data of the user code to an RX secured IPDU via the `Calculated Freshness Value` variable.

The specifications in the user code determine which data is set to the `Calculated Freshness Value` variable. For example, you can use this

function to set the freshness value that was calculated for the RX secured IPDU in the user code by using user-specific algorithms.

You can provide the data of the `Calculated Freshness Value` variable to the executable application (i.e., real-time application or offline simulation application) by calling the `DsBusCustomCodeSecOCRxPduFeature_getCalculatedFreshnessValue` function. Refer to `DsBusCustomCodeSecOCRxPduFeature_getCalculatedFreshnessValue` on page 94.

Syntax

```
Std_ReturnType DsBusCustomCodeSecOCRxPduFeature_setCalculatedFreshnessValue (DsBusCustomCodePduFeatureHandle
PduFeatureHandle, uint64 CalculatedFreshnessValue)
```

Parent handle

This function is provided by `DsBusCustomCodeSecOCRxPduFeatureHandle` on page 56.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During run time
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduFeatureHandle`	A DsBusCustomCode PDU feature handle.
`CalculatedFreshnessValues`	The data that is provided to the `Calculated Freshness Value` variable by the user code.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

-

DsBusCustomCodeSecOCRxPduFeature_setVerificationResult

Purpose

To set data of the user code to an RX secured IPDU via the `Verification Result` variable.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	User code
Parameter data recipient	Bus implementation software, provides data via TRC and/or model variable

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function sets data of the user code to an RX secured IPDU via the `Verification Result` variable.

The specifications in the user code determine which data is set to the `Verification Result` variable. For example, you can use this function to set the verification result that was calculated for the RX secured IPDU in the user code by using user-specific algorithms.

If you do this, the specifications in the user code determine which verification results can be available and by which value they are represented. To ensure full compatibility with all bus implementation software tools, it is recommended to use the following mapping of verification results and values:

Value	Use For
0x00	SECOC_VERIFICATIONSUCCESS: Verification was successful.
0x01	SECOC_VERIFICATIONFAILURE: Verification failed, e.g., because the related verification function of the user code indicates an error.
0x02	SECOC_FRESHNESSFAILURE: Verification failed because of an unexpected freshness value.
0x3E	SECOC_MESSAGELINKERFAILURE: Verification failed because of an unexpected message linker value.
0x3F	SECOC_VERIFICATIONNOTEXECUTED: Verification is not executed because it is disabled.

You can provide the data of the `Verification Result` variable to the executable application (i.e., real-time application or offline simulation application) by calling the `DsBusCustomCodeSecOCRxPduFeature_getVerificationResult` function. Refer to `DsBusCustomCodeSecOCRxPduFeature_getVerificationResult` on page 97.

Syntax

```
Std_ReturnType DsBusCustomCodeSecOCRxPduFeature_setVerificationResult (DsBusCustomCodePduFeatureHandle PduFeatureHandle,
uint32 VerificationResult)
```

Parent handle

This function is provided by `DsBusCustomCodeSecOCRxPduFeatureHandle` on page 56.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During run time
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduFeatureHandle`	A DsBusCustomCode PDU feature handle.
`VerificationResult`	The data that is provided to the `Verification Result` variable by the user code.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

-

DsBusCustomCodeSecOCTxPduFeature_getAuthenticationType

Purpose

To get the enable state of the authentication type manipulation that is specified for a TX secured IPDU.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	TRC and/or model variable, or constant value provided by bus implementation software
Parameter data recipient	User code

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function gets the enable state of the authentication type manipulation that is specified for a TX secured IPDU.

In the user code, you can use the provided state value to enable or disable the manipulation of the authentication type of the TX secured IPDU. The specifications in the user code determine how the state value is evaluated. To ensure full compatibility with all bus implementation software tools, it is recommended to use the following values:

Value	Use For
0x00	Disable the manipulation of the authentication type, i.e, use the correct algorithms to calculate the authenticator in the user code.
0x01	Enable the manipulation of the authentication type, i.e., invalidate the calculation of the authenticator in the user code.

Syntax

```
Std_ReturnType DsBusCustomCodeSecOCTxPduFeature_getAuthenticationType (DsBusCustomCodePduFeatureHandle PduFeatureHandle,
uint32 *AuthenticationType)
```

Parent handle

This function is provided by `DsBusCustomCodeSecOCTxPduFeatureHandle` on page 58.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During run time
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduFeatureHandle`	A DsBusCustomCode PDU feature handle.
`AuthenticationType`	The enable state of the authentication type manipulation that is provided by bus implementation software.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

-

DsBusCustomCodeSecOCTxPduFeature_getEnableAuthentication

Purpose

To get the enable state of the authentication value calculation that is specified for a TX secured IPDU.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	TRC and/or model variable, or constant value provided by bus implementation software
Parameter data recipient	User code

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function gets the enable state of the authentication value calculation that is specified for a TX secured IPDU.

In the user code, you can use the provided state value to enable or disable the calculation of the authentication value of the TX secured IPDU. The specifications in the user code determine how the state value is evaluated. To ensure full

compatibility with all bus implementation software tools, it is recommended to use the following values:

Value	Use For
0x00	Disable the calculation of the authentication value, i.e, the authentication value is not calculated in the user code. Instead, a user-defined authentication value can be used. You can access the user-defined authentication value by using the `DsBusCustomCodeSecOCTxPduFeature_getUserDefinedAuthenticatorPtr` function. Refer to `DsBusCustomCodeSecOCTxPduFeature_getUserDefinedAuthenticatorPtr` on page 108.
0x01	Enable the calculation of the authentication value, i.e, the authentication value is calculated in the user code.

Syntax

```
Std_ReturnType DsBusCustomCodeSecOCTxPduFeature_getEnableAuthentication (DsBusCustomCodePduFeatureHandle
PduFeatureHandle, uint32 *EnableAuthentication)
```

Parent handle

This function is provided by `DsBusCustomCodeSecOCTxPduFeatureHandle` on page 58.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During run time
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduFeatureHandle`	A DsBusCustomCode PDU feature handle.
`EnableAuthentication`	The enable state of the authentication value calculation that is provided by bus implementation software.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example -

DsBusCustomCodeSecOCTxPduFeature_getEnableFreshnessValueCalculation

Purpose

To get the enable state of the freshness value calculation that is specified for a TX secured IPDU.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	TRC and/or model variable, or constant value provided by bus implementation software
Parameter data recipient	User code

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function gets the enable state of the freshness value calculation that is specified for a TX secured IPDU.

In the user code, you can use the provided state value to enable or disable the calculation of the freshness value of the TX secured IPDU. The specifications in the user code determine how the state value is evaluated. To ensure full compatibility with all bus implementation software tools, it is recommended to use the following values:

Value	Use For
0x00	Disable the calculation of the freshness value, i.e, the freshness value is not calculated in the user code. Instead, a user-defined freshness value can be used. You can access the user-defined freshness value by using the `DsBusCustomCodeSecOCTxPduFeature_getUserDefinedFreshnessValue` function. Refer to `DsBusCustomCodeSecOCTxPduFeature_getUserDefinedFreshnessValue` on page 109.
0x01	Enable the calculation of the freshness value, i.e, the freshness value is calculated in the user code.

Syntax

```
Std_ReturnType DsBusCustomCodeSecOCTxPduFeature_getEnableFreshnessValueCalculation (DsBusCustomCodePduFeatureHandle
PduFeatureHandle, uint32 *EnableFreshnessValueCalculation)
```

Parent handle

This function is provided by `DsBusCustomCodeSecOCTxPduFeatureHandle` on page 58.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During run time
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduFeatureHandle`	A DsBusCustomCode PDU feature handle.
`EnableFreshnessValueCalculation`	The enable state of the freshness value calculation that is provided by bus implementation software.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

-

DsBusCustomCodeSecOCTxPduFeature_getFreshnessValueOffset

Purpose

To get the offset value that is specified for the freshness value of a TX secured IPDU.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	TRC and/or model variable, or constant value provided by bus implementation software
Parameter data recipient	User code

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function gets the offset value that is specified for the freshness value of a TX secured IPDU.

The specified value applies to the freshness value of the TX secured IPDU, regardless of whether the freshness value is calculated in the user code or a user-defined freshness value is used. The specified offset can be a positive or negative value, or `0`.

Syntax

```
Std_ReturnType DsBusCustomCodeSecOCTxPduFeature_getFreshnessValueOffset (DsBusCustomCodePduFeatureHandle
PduFeatureHandle, sint64 *FreshnessValueOffset)
```

Parent handle

This function is provided by `DsBusCustomCodeSecOCTxPduFeatureHandle` on page 58.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During run time
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduFeatureHandle`	A DsBusCustomCode PDU feature handle.
`FreshnessValueOffset`	The offset value that applies to the freshness value and which is provided by bus implementation software.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

-

DsBusCustomCodeSecOCTxPduFeature_getUserDefinedAuthenticatorPtr

Purpose

To get the pointer to the byte array that provides the user-defined authentication value, which is specified for a TX secured IPDU.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	TRC and/or model variable, or constant value provided by bus implementation software
Parameter data recipient	User code

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function gets the pointer to the byte array that provides the user-defined authentication value, which is specified for a TX secured IPDU.

The user-defined authentication value applies to the TX secured IPDU only if calculating the authentication value in the user code is disabled via the `DsBusCustomCodeSecOCTxPduFeature_getEnableAuthentication` function. Refer to `DsBusCustomCodeSecOCTxPduFeature_getEnableAuthentication` on page 103.

Syntax

```
Std_ReturnType DsBusCustomCodeSecOCTxPduFeature_getUserDefinedAuthenticatorPtr (DsBusCustomCodePduFeatureHandle
PduFeatureHandle, uint8 **UserDefinedAuthenticatorPtr)
```

Parent handle

This function is provided by `DsBusCustomCodeSecOCTxPduFeatureHandle` on page 58.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During run time
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduFeatureHandle`	A DsBusCustomCode PDU feature handle.
`UserDefinedAuthenticatorPtr`	The pointer to the byte array that provides the user-defined authentication value.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

-

DsBusCustomCodeSecOCTxPduFeature_getUserDefinedFreshnessValue

Purpose

To get the user-defined freshness value that is specified for a TX secured IPDU.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	TRC and/or model variable, or constant value provided by bus implementation software
Parameter data recipient	User code

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function gets the user-defined freshness value that is specified for a TX secured IPDU.

The user-defined freshness value applies to the TX secured IPDU only if calculating the freshness value in the user code is disabled via the `DsBusCustomCodeSecOCTxPduFeature_getEnableFreshnessValueCalculation` function. Refer to `DsBusCustomCodeSecOCTxPduFeature_getEnableFreshnessValueCalculation` on page 105.

Syntax

```
Std_ReturnType DsBusCustomCodeSecOCTxPduFeature_getUserDefinedFreshnessValue (DsBusCustomCodePduFeatureHandle
PduFeatureHandle, uint64 *UserDefinedFreshnessValue)
```

Parent handle

This function is provided by `DsBusCustomCodeSecOCTxPduFeatureHandle` on page 58.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During run time
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduFeatureHandle`	A DsBusCustomCode PDU feature handle.
`UserDefinedFreshnessValue`	The user-defined freshness value that is provided by bus implementation software.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example -

DsBusCustomCodeSecuredIPdu_getAuthAlgorithmName

Purpose

To get the value of the AUTOSAR `AuthAlgorithm` attribute of a secured IPDU.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	Communication matrix (via bus implementation software)
Parameter data recipient	User code

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function gets the value of the AUTOSAR `AuthAlgorithm` attribute of a secured IPDU.

According to AUTOSAR, the value of the `AuthAlgorithm` attribute specifies the name of the authentication algorithm of a secured IPDU.

Syntax

```
Std_ReturnType DsBusCustomCodeSecuredIPdu_getAuthAlgorithmName (DsBusCustomCodePduHandle PduHandle, char
**AuthAlgorithmNameAsString)
```

Parent handle

This function is provided by `DsBusCustomCodeSecuredIPduHandle` on page 60.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During run time
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduHandle`	A DsBusCustomCode PDU handle.
`AuthAlgorithmNameAsString`	The name of the authentication algorithm.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

-

DsBusCustomCodeSecuredIPdu_getAuthDataFreshnessLength

Purpose

To get the value of the AUTOSAR `AuthDataFreshnessLength` attribute of a secured IPDU.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	Communication matrix (via bus implementation software)
Parameter data recipient	User code

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function gets the value of the AUTOSAR `AuthDataFreshnessLength` attribute of a secured IPDU.

According to AUTOSAR, a part of the payload of an authentic IPDU can be included in the freshness value. The value of the `AuthDataFreshnessLength` attribute determines the length in bits of the authentic IPDU's payload that is used for this purpose.

Syntax

```
Std_ReturnType DsBusCustomCodeSecuredIPdu_getAuthDataFreshnessLength (DsBusCustomCodePduHandle PduHandle, uint16
*AuthDataFreshnessLength)
```

Parent handle

This function is provided by `DsBusCustomCodeSecuredIPduHandle` on page 60.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During run time
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduHandle`	A DsBusCustomCode PDU handle.
`AuthDataFreshnessLength`	The length in bits of the authentic IPDU's payload that is included in the freshness value.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

-

DsBusCustomCodeSecuredIPdu_getAuthDataFreshnessStartPosition

Purpose

To get the value of the AUTOSAR `AuthDataFreshnessStartPosition` attribute of a secured IPDU.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	Communication matrix (via bus implementation software)
Parameter data recipient	User code

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function gets the value of the AUTOSAR `AuthDataFreshnessStartPosition` attribute of a secured IPDU.

According to AUTOSAR, a part of the payload of an authentic IPDU can be included in the freshness value. The value of the `AuthDataFreshnessStartPosition` attribute determines the position of the first bit in the authentic IPDU's payload that is used for this purpose. To determine the bit position, the counting starts from the most significant bit (MSB) of the first byte of the payload.

Syntax

```
Std_ReturnType DsBusCustomCodeSecuredIPdu_getAuthDataFreshnessStartPosition (DsBusCustomCodePduHandle PduHandle, uint16
*AuthDataFreshnessStartPosition)
```

Parent handle

This function is provided by `DsBusCustomCodeSecuredIPduHandle` on page 60.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During run time
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduHandle`	A DsBusCustomCode PDU handle.
`AuthDataFreshnessStartPosition`	The position of the first bit in the authentic IPDU's payload that is included in the freshness value.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

-

DsBusCustomCodeSecuredIPdu_getAuthInfoTxLength

Purpose

To get the value of the AUTOSAR `AuthInfoTxLength` attribute of a secured IPDU.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	Communication matrix (via bus implementation software)
Parameter data recipient	User code

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function gets the value of the AUTOSAR `AuthInfoTxLength` attribute of a secured IPDU.

According to AUTOSAR, the value of the `AuthInfoTxLength` attribute determines the length in bits of the calculated authenticator that is included in the payload of a secured IPDU.

Syntax

```
Std_ReturnType DsBusCustomCodeSecuredIPdu_getAuthInfoTxLength (DsBusCustomCodePduHandle PduHandle, uint32
*AuthInfoTxLength)
```

Parent handle

This function is provided by `DsBusCustomCodeSecuredIPduHandle` on page 60.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During run time
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduHandle`	A DsBusCustomCode PDU handle.
`AuthInfoTxLength`	The length in bits of the calculated authenticator that is included in the payload of the secured IPDU.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

```
// Copy authInfoTxLength number of bits of authenticator to secured IPDU
uint32 authInfoTxLength;
DsBusCustomCodeSecuredIPdu_getAuthInfoTxLength(securedIPdu_PduHandle, &authInfoTxLength);
```

DsBusCustomCodeSecuredIPdu_getAuthPduHeaderLength

Purpose

To get the value of the AUTOSAR `SecOCAuthPduHeaderLength` attribute of a secured IPDU.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	Communication matrix (via bus implementation software)
Parameter data recipient	User code

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function gets the value of the AUTOSAR `SecOCAuthPduHeaderLength` attribute of a secured IPDU.

According to AUTOSAR, the value of the `SecOCAuthPduHeaderLength` attribute indicates the length of the secured PDU header that is used in a secured IPDU. The length is provided in bytes, e.g., `0` indicates that no secured PDU header is used and `4` indicates a 32-bit secured PDU header.

Syntax

```
Std_ReturnType DsBusCustomCodeSecuredIPdu_getAuthPduHeaderLength (DsBusCustomCodePduHandle PduHandle, uint8
*AuthPduHeaderLength)
```

Parent handle

This function is provided by `DsBusCustomCodeSecuredIPduHandle` on page 60.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During run time
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduHandle`	A DsBusCustomCode PDU handle.
`AuthPduHeaderLength`	The length in bytes of the secured PDU header that is used in the secured IPDU.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example -

DsBusCustomCodeSecuredIPdu_getAuthenticPdu

Purpose

To get the authentic IPDU that is secured by a secured IPDU.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	Communication matrix (via bus implementation software)
Parameter data recipient	User code

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function gets the authentic IPDU that is secured by a secured IPDU. The function gets the authentic IPDU via its handle.

Syntax

```
Std_ReturnType DsBusCustomCodeSecuredIPdu_getAuthenticPdu (DsBusCustomCodePduHandle PduHandle, DsBusCustomCodePduHandle
*AuthenticPduHandle)
```

Parent handle

This function is provided by `DsBusCustomCodeSecuredIPduHandle` on page 60.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During run time
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduHandle`	A DsBusCustomCode PDU handle.
`AuthenticPduHandle`	The handle of the authentic IPDU that is secured by the secured IPDU.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

```
// The PduFeatureHandle is typically provided as a parameter of an interface
// call, e.g. DsBusCustomCode_onApplicationInit
DsBusCustomCodePduHandle securedIPdu_PduHandle, authenticIPdu_PduHandle;
DsBusCustomCodePduFeature_getPdu(PduFeatureHandle, &securedIPdu_PduHandle);
DsBusCustomCodeSecuredIPdu_getAuthenticPdu(securedIPdu_PduHandle, &authenticIPdu_PduHandle);

```

DsBusCustomCodeSecuredIPdu_getAuthenticationBuildAttempts

Purpose

To get the value of the AUTOSAR `AuthenticationBuildAttempts` attribute of a secured IPDU.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	Communication matrix (via bus implementation software)
Parameter data recipient	User code

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function gets the value of the AUTOSAR `AuthenticationBuildAttempts` attribute of a secured IPDU.

According to AUTOSAR, the value of the `AuthenticationBuildAttempts` attribute determines how often the sender of a secured IPDU tries to generate the authentication information.

Syntax

```
Std_ReturnType DsBusCustomCodeSecuredIPdu_getAuthenticationBuildAttempts (DsBusCustomCodePduHandle PduHandle, uint16
*AuthenticationBuildAttempts)
```

Parent handle

This function is provided by `DsBusCustomCodeSecuredIPduHandle` on page 60.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During run time
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduHandle`	A DsBusCustomCode PDU handle.
`AuthenticationBuildAttempts`	The number of the authentication build attempts.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

-

DsBusCustomCodeSecuredIPdu_getAuthenticationRetries

Purpose

To get the value of the AUTOSAR `AuthenticationRetries` attribute of a secured IPDU.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	Communication matrix (via bus implementation software)
Parameter data recipient	User code

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function gets the value of the AUTOSAR `AuthenticationRetries` attribute of a secured IPDU.

According to AUTOSAR, the value of the `AuthenticationRetries` attribute determines how often the receiver of a secured IPDU tries to verify the received authentication information if the first attempt failed. A value of `0` indicates that there will be no additional authentication attempts, i.e., the receiver tries to verify received authentication information only once.

Syntax

```
Std_ReturnType DsBusCustomCodeSecuredIPdu_getAuthenticationRetries (DsBusCustomCodePduHandle PduHandle, uint16
*AuthenticationRetries)
```

Parent handle

This function is provided by `DsBusCustomCodeSecuredIPduHandle` on page 60.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During run time
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduHandle`	A DsBusCustomCode PDU handle.
`AuthenticationRetries`	The number of the additional authentication attempts.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

-

DsBusCustomCodeSecuredIPdu_getDataId

Purpose

To get the value of the AUTOSAR `DataId` attribute of a secured IPDU.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	Communication matrix (via bus implementation software)
Parameter data recipient	User code

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function gets the value of the AUTOSAR `DataId` attribute of a secured IPDU.

According to AUTOSAR, the data ID is a unique numerical identifier for the secured IPDU. Typically, the data ID is included in the calculation of the authenticator.

Syntax

```
Std_ReturnType DsBusCustomCodeSecuredIPdu_getDataId (DsBusCustomCodePduHandle PduHandle, uint16 *DataId)
```

Parent handle

This function is provided by `DsBusCustomCodeSecuredIPduHandle` on page 60.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During run time
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduHandle`	A DsBusCustomCode PDU handle.
`DataId`	The data ID of the secured IPDU.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

```
DsBusCustomCodePduHandle securedIPdu_PduHandle;
DsBusCustomCodePduFeature_getPdu(PduFeatureHandle, &securedIPdu_PduHandle);
DsBusCustomCodeSecuredIPdu_getDataId(securedIPdu_PduHandle, &dataId);
// ... prepare authentication data including data ID and generate MAC

```

DsBusCustomCodeSecuredIPdu_getFreshnessCounterSyncAttempts

Purpose

To get the value of the AUTOSAR `FreshnessCounterSyncAttempts` attribute of a secured IPDU.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	Communication matrix (via bus implementation software)
Parameter data recipient	User code

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function gets the value of the AUTOSAR `FreshnessCounterSyncAttempts` attribute of a secured IPDU.

According to AUTOSAR, the `FreshnessCounterSyncAttempts` attribute applies only if the freshness value is calculated according to a freshness counter, i.e., the `UseFreshnessTimestamp` attribute of the secured IPDU is set to `FALSE`.

According to AUTOSAR, the value of the `FreshnessCounterSyncAttempts` attribute determines how often the receiver of a secured IPDU tries to synchronize the freshness counter if the first verification of the authentication information failed. A value of `0` indicates that there will be no additional synchronization attempts, i.e., the receiver tries to synchronize the freshness counter only once.

Syntax

```
Std_ReturnType DsBusCustomCodeSecuredIPdu_getFreshnessCounterSyncAttempts (DsBusCustomCodePduHandle PduHandle, uint32
*FreshnessCounterSyncAttempts)
```

Parent handle

This function is provided by `DsBusCustomCodeSecuredIPduHandle` on page 60.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During run time
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduHandle`	A DsBusCustomCode PDU handle.
`FreshnessCounterSyncAttempts`	The number of the additional synchronization attempts for the freshness counter.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

-

DsBusCustomCodeSecuredIPdu_getFreshnessTimestampPeriodFactor

Purpose

To get the value of the AUTOSAR `FreshnessTimestampTimePeriodFactor` attribute of a secured IPDU.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	Communication matrix (via bus implementation software)
Parameter data recipient	User code

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function gets the value of the AUTOSAR `FreshnessTimestampTimePeriodFactor` attribute of a secured IPDU.

According to AUTOSAR, the `FreshnessTimestampTimePeriodFactor` attribute applies only if the freshness value is calculated according to a freshness time stamp, i.e., the `UseFreshnessTimestamp` attribute of the secured IPDU is set to `TRUE`.

According to AUTOSAR, the value of the `FreshnessTimestampTimePeriodFactor` attribute is the factor of the time period for the freshness time stamp. The specified factor determines the duration in microseconds for one increment of the freshness time stamp.

Syntax

```
Std_ReturnType DsBusCustomCodeSecuredIPdu_getFreshnessTimestampPeriodFactor (DsBusCustomCodePduHandle PduHandle, uint32
*FreshnessTimestampPeriodFactor)
```

Parent handle

This function is provided by `DsBusCustomCodeSecuredIPduHandle` on page 60.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During run time
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduHandle`	A DsBusCustomCode PDU handle.
`FreshnessTimestampPeriodFactor`	The factor in microseconds of the time period for the freshness time stamp.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

-

DsBusCustomCodeSecuredIPdu_getFreshnessValueId

Purpose

To get the value of the AUTOSAR `FreshnessValueId` attribute of a secured IPDU.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	Communication matrix (via bus implementation software)
Parameter data recipient	User code

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function gets the value of the AUTOSAR `FreshnessValueId` attribute of a secured IPDU.

According to AUTOSAR, the `FreshnessValueId` attribute specifies the identifier for the freshness value of the secured IPDU.

Syntax

```
Std_ReturnType DsBusCustomCodeSecuredIPdu_getFreshnessValueId (DsBusCustomCodePduHandle PduHandle, uint16
*FreshnessValueId)
```

Parent handle

This function is provided by `DsBusCustomCodeSecuredIPduHandle` on page 60.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During run time
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduHandle`	A DsBusCustomCode PDU handle.
`FreshnessValueId`	The freshness value ID of the secured IPDU.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

-

DsBusCustomCodeSecuredIPdu_getFreshnessValueLength

Purpose

To get the value of the AUTOSAR `FreshnessValueLength` attribute of a secured IPDU.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	Communication matrix (via bus implementation software)
Parameter data recipient	User code

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function gets the value of the AUTOSAR `FreshnessValueLength` attribute of a secured IPDU.

According to AUTOSAR, the value of the `FreshnessValueLength` attribute determines the length in bits that is used for the complete freshness value. Typically, the specified value is larger than the number of bits that are transmitted.

Syntax

```
Std_ReturnType DsBusCustomCodeSecuredIPdu_getFreshnessValueLength (DsBusCustomCodePduHandle PduHandle, uint32
*FreshnessValueLength)
```

Parent handle

This function is provided by `DsBusCustomCodeSecuredIPduHandle` on page 60.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During run time
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduHandle`	A DsBusCustomCode PDU handle.
`FreshnessValueLength`	The length of the complete freshness value in bits.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

-

DsBusCustomCodeSecuredIPdu_getFreshnessValueTxLength

Purpose

To get the value of the AUTOSAR `FreshnessValueTxLength` attribute of a secured IPDU.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	Communication matrix (via bus implementation software)
Parameter data recipient	User code

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function gets the value of the AUTOSAR `FreshnessValueTxLength` attribute of a secured IPDU.

According to AUTOSAR, the value of the `FreshnessValueTxLength` attribute determines the length in bits of the freshness value that is included in the payload of a secured IPDU.

Syntax

```
Std_ReturnType DsBusCustomCodeSecuredIPdu_getFreshnessValueTxLength (DsBusCustomCodePduHandle PduHandle, uint32
*FreshnessValueTxLength)
```

Parent handle

This function is provided by `DsBusCustomCodeSecuredIPduHandle` on page 60.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During run time
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduHandle`	A DsBusCustomCode PDU handle.
`FreshnessValueTxLength`	The length in bits of the freshness value that is included in the payload of the secured IPDU.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

-

DsBusCustomCodeSecuredIPdu_getKeyId

Purpose

To get the value of the AUTOSAR `KeyID` attribute of a secured IPDU.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	Communication matrix (via bus implementation software)
Parameter data recipient	User code

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function gets the value of the AUTOSAR `KeyID` attribute of a secured IPDU.

According to AUTOSAR, the `KeyID` attribute identifies the key that is used to generate and verify the message authentication code (MAC). The specified key ID value must be unique per ECU.

Syntax

```
Std_ReturnType DsBusCustomCodeSecuredIPdu_getKeyId (DsBusCustomCodePduHandle PduHandle, uint32 *KeyId)
```

Parent handle

This function is provided by `DsBusCustomCodeSecuredIPduHandle` on page 60.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During run time
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduHandle`	A DsBusCustomCode PDU handle.
`KeyId`	The key ID.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

-

DsBusCustomCodeSecuredIPdu_getMessageLinkLength

Purpose

To get the value of the AUTOSAR `MessageLinkLength` attribute of a secured IPDU.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	Communication matrix (via bus implementation software)
Parameter data recipient	User code

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function gets the value of the AUTOSAR `MessageLinkLength` attribute of a secured IPDU.

If a secured IPDU is configured as cryptographic IPDU, a part of the payload of the related authentic IPDU can be included in the cryptographic IPDU. This part is called the message linker.

According to AUTOSAR, the value of the `MessageLinkLength` attribute determines the length in bits of the message linker.

Syntax

```
Std_ReturnType DsBusCustomCodeSecuredIPdu_getMessageLinkLength (DsBusCustomCodePduHandle PduHandle, uint16
*MessageLinkLength)
```

Parent handle

This function is provided by `DsBusCustomCodeSecuredIPduHandle` on page 60.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During run time
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduHandle`	A DsBusCustomCode PDU handle.
`MessageLinkLength`	The length of the message linker in bits.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

-

DsBusCustomCodeSecuredIPdu_getMessageLinkPosition

Purpose

To get the value of the AUTOSAR `MessageLinkPosition` attribute of a secured IPDU.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	Communication matrix (via bus implementation software)
Parameter data recipient	User code

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function gets the value of the AUTOSAR `MessageLinkPosition` attribute of a secured IPDU.

If a secured IPDU is configured as cryptographic IPDU, a part of the payload of the related authentic IPDU can be included in the cryptographic IPDU. This part is called the message linker.

According to AUTOSAR, the value of the `MessageLinkPosition` attribute determines the position of the first bit in the authentic IPDU's payload that is used as the message linker.

Syntax

```
Std_ReturnType DsBusCustomCodeSecuredIPdu_getMessageLinkPosition (DsBusCustomCodePduHandle PduHandle, uint16
*MessageLinkPos)
```

Parent handle

This function is provided by `DsBusCustomCodeSecuredIPduHandle` on page 60.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended usage	During run time
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduHandle`	A DsBusCustomCode PDU handle.
`MessageLinkPos`	The position of the first bit in the authentic IPDU's payload that is used as the message linker.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example -

DsBusCustomCodeSecuredIPdu_getRxSecurityVerification

Purpose

To get the value of the AUTOSAR `RxSecurityVerification` attribute of a secured IPDU.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	Communication matrix (via bus implementation software)
Parameter data recipient	User code

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function gets the value of the AUTOSAR `RxSecurityVerification` attribute of a secured IPDU.

According to AUTOSAR, the value of the `RxSecurityVerification` attribute determines whether the authentication information of a received secured IPDU is verified. If the attribute is not specified or set to `TRUE`, the authentication information is verified. If set to `FALSE`, the secured IPDU will be processed by the receiving ECU without verifying its authentication information.

Note

This function gets the enable state for verifying authentication information as specified in the communication matrix. To specify whether the bus implementation software verifies received authentication information, it is recommended to use the `DsBusCustomCodeSecOCRxPduFeature_getEnableVerification` function instead. Refer to `DsBusCustomCodeSecOCRxPduFeature_getEnableVerification` on page 95.

Syntax

```
Std_ReturnType DsBusCustomCodeSecuredIPdu_getRxSecurityVerification (DsBusCustomCodePduHandle PduHandle, boolean
*RxSecurityVerification)
```

Parent handle

This function is provided by `DsBusCustomCodeSecuredIPduHandle` on page 60.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During run time
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduHandle`	A DsBusCustomCode PDU handle.
`RxSecurityVerification`	The value determining whether the authentication information of a secured IPDU is verified on the receiver side: ▪ `TRUE`: The authentication information is verified. ▪ `FALSE`: The authentication information is not verified.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

-

DsBusCustomCodeSecuredIPdu_getSecuredAreaLength

Purpose

To get the value of the AUTOSAR `SecuredAreaLength` attribute of a secured IPDU.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	Communication matrix (via bus implementation software)
Parameter data recipient	User code

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function gets the value of the AUTOSAR `SecuredAreaLength` attribute of a secured IPDU.

According to AUTOSAR, the value of the `SecuredAreaLength` attribute determines the length in bytes of the authentic IPDU's payload that is secured. For these bytes, authentication information is generated.

If this attribute is specified, the `SecuredAreaOffset` attribute must also be specified. If both attributes are not specified, the complete payload of the authentic IPDU is secured.

Syntax

```
Std_ReturnType DsBusCustomCodeSecuredIPdu_getSecuredAreaLength (DsBusCustomCodePduHandle PduHandle, uint32
*SecuredAreaLength)
```

Parent handle

This function is provided by `DsBusCustomCodeSecuredIPduHandle` on page 60.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During run time
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduHandle`	A DsBusCustomCode PDU handle.
`SecuredAreaLength`	The length in bytes of the authentic IPDU's payload that is secured.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

-

DsBusCustomCodeSecuredIPdu_getSecuredAreaOffset

Purpose

To get the value of the AUTOSAR `SecuredAreaOffset` attribute of a secured IPDU.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	Communication matrix (via bus implementation software)
Parameter data recipient	User code

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function gets the value of the AUTOSAR `SecuredAreaOffset` attribute of a secured IPDU.

According to AUTOSAR, the `SecuredAreaOffset` attribute determines the position of the first byte in the authentic IPDU's payload that is secured. Starting from this byte, authentication information is generated for the number of bytes that are specified by the `SecuredAreaLength` attribute.

If this attribute is specified, the `SecuredAreaLength` attribute must also be specified. If both attributes are not specified, the complete payload of the authentic IPDU is secured.

Syntax

```
Std_ReturnType DsBusCustomCodeSecuredIPdu_getSecuredAreaOffset (DsBusCustomCodePduHandle PduHandle, uint32
*SecuredAreaOffset)
```

Parent handle

This function is provided by `DsBusCustomCodeSecuredIPduHandle` on page 60.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During run time
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduHandle`	A DsBusCustomCode PDU handle.
`AuthPduHeaderLength`	The position of the first byte in the authentic IPDU's payload that is secured.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

-

DsBusCustomCodeSecuredIPdu_getTimeStampRxAcceptanceWindow

Purpose

To get the value of the AUTOSAR `TimestampRxAcceptanceWindow` attribute of a secured IPDU.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	Communication matrix (via bus implementation software)
Parameter data recipient	User code

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function gets the value of the AUTOSAR `TimestampRxAcceptanceWindow` attribute of a secured IPDU.

According to AUTOSAR, the value of the `TimestampRxAcceptanceWindow` attribute determines the maximum allowed deviation of a received time stamp value from the expected time stamp value of a received secured IPDU.

In contrast to AUTOSAR, this function gets the value of the `TimestampRxAcceptanceWindow` attribute in milliseconds.

Syntax

```
Std_ReturnType DsBusCustomCodeSecuredIPdu_getTimeStampRxAcceptanceWindow (DsBusCustomCodePduHandle PduHandle, uint64
*TimeStampRxAcceptanceWindow)
```

Parent handle

This function is provided by `DsBusCustomCodeSecuredIPduHandle` on page 60.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During run time
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduHandle`	A DsBusCustomCode PDU handle.
`TimeStampRxAcceptanceWindow`	The maximum allowed deviation in milliseconds of a received time stamp value from the expected time stamp value.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

-

DsBusCustomCodeSecuredIPdu_getUseAsCryptographicPdu

Purpose

To get the value of the AUTOSAR `UseAsCryptographicIPdu` attribute of a secured IPDU.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	Communication matrix (via bus implementation software)
Parameter data recipient	User code

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function gets the value of the AUTOSAR `UseAsCryptographicIPdu` attribute of a secured IPDU.

According to AUTOSAR, the value of the `UseAsCryptographicIPdu` attribute determines whether a secured IPDU is used as cryptographic IPDU. If the attribute is set to `TRUE`, the secured IPDU is used as cryptographic IPDU. In this case, the secured IPDU contains only the authentication information and the related authentic IPDU is transmitted separately via the bus. If the attribute is set to `FALSE` or not specified, the payload of the authentic IPDU is directly included in the secured IPDU. In this case, only one PDU is exchanged via the bus.

Syntax

```
Std_ReturnType DsBusCustomCodeSecuredIPdu_getUseAsCryptographicPdu (DsBusCustomCodePduHandle PduHandle, boolean
*UseAsCryptographicIPdu)
```

Parent handle

This function is provided by `DsBusCustomCodeSecuredIPduHandle` on page 60.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During run time
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduHandle`	A DsBusCustomCode PDU handle.
`UseAsCryptographicIPdu`	The value determining whether the secured IPDU is used as cryptographic IPDU: ▪ `TRUE`: The secured IPDU is used as cryptographic IPDU. ▪ `FALSE`: The secured IPDU is not used as cryptographic IPDU.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

-

DsBusCustomCodeSecuredIPdu_getUseAuthDataFreshness

Purpose

To get the value of the AUTOSAR `UseAuthDataFreshness` attribute of a secured IPDU.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	Communication matrix (via bus implementation software)
Parameter data recipient	User code

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function gets the value of the AUTOSAR `UseAuthDataFreshness` attribute of a secured IPDU.

According to AUTOSAR, the value of the `UseAuthDataFreshness` attribute determines whether a part of the payload of an authentic IPDU is included in the freshness value. If the attribute is set to `TRUE`, a part of the payload of an authentic IPDU is included in the freshness value. In this case, the `AuthDataFreshnessStartPosition` and `AuthDataFreshnessLength` attributes are required to specify the related payload part.

Syntax

```
Std_ReturnType DsBusCustomCodeSecuredIPdu_getUseAuthDataFreshness (DsBusCustomCodePduHandle PduHandle, boolean
*UseAuthDataFreshness)
```

Parent handle

This function is provided by `DsBusCustomCodeSecuredIPduHandle` on page 60.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During run time
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduHandle`	A DsBusCustomCode PDU handle.
`UseAuthDataFreshness`	The value determining whether a part of the payload of an authentic IPDU is included in the freshness value: ▪ `TRUE`: A part of the payload of an authentic IPDU is included in the freshness value. ▪ `FALSE`: No part of the payload of an authentic IPDU is included in the freshness value.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

-

DsBusCustomCodeSecuredIPdu_getUseFreshnessTimestamp

Purpose

To get the value of the AUTOSAR `UseFreshnessTimestamp` attribute of a secured IPDU.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	Communication matrix (via bus implementation software)
Parameter data recipient	User code

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function gets the value of the AUTOSAR `UseFreshnessTimestamp` attribute of a secured IPDU.

According to AUTOSAR, the value of the `UseFreshnessTimestamp` attribute determines whether a freshness time stamp or a freshness counter is used to generate the freshness value. If the attribute is set to `TRUE`, a freshness time stamp is used.

Syntax

```
Std_ReturnType DsBusCustomCodeSecuredIPdu_getUseFreshnessTimestamp (DsBusCustomCodePduHandle PduHandle, boolean
*UseFreshnessTimestamp)
```

Parent handle

This function is provided by `DsBusCustomCodeSecuredIPduHandle` on page 60.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During run time
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduHandle`	A DsBusCustomCode PDU handle.
`UseFreshnessTimestamp`	The value determining whether a freshness time stamp or a freshness counter is used to generate the freshness value: ▪ `TRUE`: A freshness time stamp is used. ▪ `FALSE`: A freshness counter is used.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

-

DsBusCustomCodeSignal_getEndianness

Purpose

To get the endianness of a signal.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	Communication matrix (via bus implementation software)
Parameter data recipient	User code

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function gets the endianness of a signal in bits.

Syntax

```
Std_ReturnType DsBusCustomCodeSignal_getEndianness (DsBusCustomCodeSignalHandle Handle, DsBusCustomCode_ByteOrderType
*Data)
```

Parent handle

This function is provided by `DsBusCustomCodeSignalHandle` on page 62.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During run time
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`Handle`	A DsBusCustomCode handle.
`Data`	The endianness of the signal in bits. The endianness is provided as `DsBusCustomCode_ByteOrderType` data type.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

-

DsBusCustomCodeSignal_getLength

Purpose

To get the length of a signal.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	Communication matrix (via bus implementation software)
Parameter data recipient	User code

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function gets the length of a signal in bits.

Syntax

```
Std_ReturnType DsBusCustomCodeSignal_getLength (DsBusCustomCodeSignalHandle Handle, uint32 *Data)
```

Parent handle

This function is provided by `DsBusCustomCodeSignalHandle` on page 62.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During run time
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`Handle`	A DsBusCustomCode handle.
`Data`	The signal length in bits.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.

Value	Description
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

-

DsBusCustomCodeSignal_getStartBitPosition

Purpose

To get the start bit position of a signal.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	Communication matrix (via bus implementation software)
Parameter data recipient	User code

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function gets the start bit position of a signal in bits. The start bit position determines the first bit in the payload of a PDU that is used by the signal.

Syntax

```
Std_ReturnType DsBusCustomCodeSignal_getStartBitPosition (DsBusCustomCodeSignalHandle Handle, uint32 *Data)
```

Parent handle

This function is provided by `DsBusCustomCodeSignalHandle` on page 62.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During run time
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`Handle`	A DsBusCustomCode handle.
`Data`	The start bit position of the signal in bits.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

-

DsBusCustomCodeUserCodePduFeature_getNumberOfUserSignals

Purpose

To get the number of user signals that are specified for a PDU via the PDU User Code feature.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	TRC and/or model variable, or constant value set by bus implementation software
Parameter data recipient	User code

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function gets the number of user signals that are specified for a PDU via the PDU User Code feature. The PDU User Code feature is a feature of bus implementation software tools. For more information on the feature, refer to the documentation of the related bus implementation software tool.

Syntax

```
Std_ReturnType DsBusCustomCodeUserCodePduFeature_getNumberOfUserSignals (DsBusCustomCodePduFeatureHandle
PduFeatureHandle, uint32 *Data)
```

Parent handle

This function is provided by `DsBusCustomCodeUserCodePduFeatureHandle` on page 63.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During initialization
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduFeatureHandle`	A DsBusCustomCode PDU feature handle.
`Data`	The number of user signals.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

-

DsBusCustomCodeUserCodePduFeature_getResult

Purpose

To get the data of the user code that is provided to an RX PDU via the PDU User Code feature.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the

function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	User code
Parameter data recipient	Bus implementation software, provides data via TRC and/or model variable

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function gets the data of the user code that is provided to an RX PDU via the PDU User Code feature. The PDU User Code feature is a feature of bus implementation software tools. For more information on the feature, refer to the documentation of the related bus implementation software tool.

To get the data, it must be set beforehand by using the `DsBusCustomCodeUserCodePduFeature_setResult` function. Refer to `DsBusCustomCodeUserCodePduFeature_setResult` on page 153.

Syntax

```
Std_ReturnType DsBusCustomCodeUserCodePduFeature_getResult (DsBusCustomCodePduFeatureHandle PduFeatureHandle, uint32
*Data)
```

Parent handle

This function is provided by `DsBusCustomCodeUserCodePduFeatureHandle` on page 63.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During initialization
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduFeatureHandle`	A DsBusCustomCode PDU feature handle.
`Data`	The data that is provided by the user code.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

-

DsBusCustomCodeUserCodePduFeature_getUserSignals

Purpose

To get the array of the user signals that are specified for a PDU via the PDU User Code feature.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	TRC and/or model variable, or constant value set by bus implementation software
Parameter data recipient	User code

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function gets the array of the user signals that are specified for a PDU via the PDU User Code feature. The PDU User Code feature is a feature of bus implementation software tools. For more information on the feature, refer to the documentation of the related bus implementation software tool.

The array size is determined by the number of user signals that are specified via the PDU User Code feature. You can get the array size by using the `DsBusCustomCodeUserCodePduFeature_getNumberOfUserSignals` function. Refer to `DsBusCustomCodeUserCodePduFeature_getNumberOfUserSignals` on page 149.

The array provides access to the configured user signals, i.e., to the ISignals that are mapped to user signals via the PDU User Code feature. If a user signal is not mapped to an ISignal, the array entries of this user signal are `NULL`.

You can use this function to provide the data of the user signals to the user code. In the user code, you can use the data to verify received checksum values or calculate counter values, for example.

Syntax

```
Std_ReturnType DsBusCustomCodeUserCodePduFeature_getUserSignals (DsBusCustomCodePduFeatureHandle PduFeatureHandle,
DsBusCustomCodeSignalHandle **Data)
```

Parent handle

This function is provided by `DsBusCustomCodeUserCodePduFeatureHandle` on page 63.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During initialization
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduFeatureHandle`	A DsBusCustomCode PDU feature handle.
`Data`	The pointer to get the array. To get the array, the pointer points to the handle of the first user signal of the array.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

-

DsBusCustomCodeUserCodePduFeature_setResult

Purpose

To set data of the user code to an RX PDU via the PDU User Code feature.

Parameter data

The following table provides an overview of the required data source of the function parameters and the recipients of the parameter data. When the function is called at run time but the data source does not provide the required data, the function returns `E_PARAMETER_NOT_SET`.

Parameter data source	User code
Parameter data recipient	Bus implementation software, provides data via TRC and/or model variable

For an overview of the bus implementation software tools with which this function can be used, refer to Using Bus Custom Code Functions with Bus Implementation Software on page 21.

Description

This function sets data of the user code to an RX PDU via the PDU User Code feature. The PDU User Code feature is a feature of bus implementation software tools. For more information on the feature, refer to the documentation of the related bus implementation software tool.

The specifications in the user code determine which data is set to the RX PDU. For example, you can use this function to set the verification result of a checksum value that was received with the PDU and evaluated in the user code.

You can provide the data of the user code to the executable application (i.e., real-time application or offline simulation application) by calling the `DsBusCustomCodeUserCodePduFeature_getResult` function. Refer to DsBusCustomCodeUserCodePduFeature_getResult on page 150.

Syntax

```
Std_ReturnType DsBusCustomCodeUserCodePduFeature_setResult (DsBusCustomCodePduFeatureHandle PduFeatureHandle, uint32
Data)
```

Parent handle

This function is provided by `DsBusCustomCodeUserCodePduFeatureHandle` on page 63.

Characteristics

This function has the following characteristics:

Thread-safe	Yes
Intended use	During initialization
Execution time	Deterministic

Parameters

This function has the following parameters:

Parameter	Description
`PduFeatureHandle`	A DsBusCustomCode PDU feature handle.
`Data`	The data that is provided by the user code.

Returns

This function returns one of the following values:

Value	Description
`E_OK`	The function is correctly executed.
`E_NOT_OK`	The function is not correctly executed.
`E_PARAMETER_NOT_SET`	A function parameter is not set.
`E_INVALID_HANDLE`	A handle is invalid, e.g., because of an assigned NULL pointer.

Example

-