MICROSAR EXI

Technical Reference

Efficient XML Interchange Generator and Parser

Version 2.00.00

Authors	Patrick Sommer
Status	Released

Document Information

History

Author	Date	Version	Remarks
Daniel Dausend	2012-07-20	1.00.00	Creation of document
Patrick Sommer	2014-08-25	1.00.01	Add VersionInfo encoding information
Fabian Eisele	2015-05-19	2.00.00	Added MSR4 information

Reference Documents

No.	Source	Title	Version
[1]	W3C	http://www.w3.org/TR/exi/	1.0
[2]	AUTOSAR	AUTOSAR_SWS_DET.pdf	2.2.1
[3]	AUTOSAR	AUTOSAR_BasicSoftwareModules.pdf	1.0.0
[4]	DIN	DIN 70121: Electromobility — Digital communication between a d.c. EV charging station and an electric vehicle for control of d.c. charging in the Combined Charging System	

Scope of the Document

This technical reference describes the general use of the Exi basic software.

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

Caution
This symbol calls your attention to warnings.

Contents

Illustrations

Tables

1 Component History

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

Component Version	New Features
1.00.xx	First beta version for DIN 70121 RC1 draft schema version
1.01.xx	Full DIN 70121 schema support
1.02.xx	Reduce code size
1.03.xx	Add XML security support
2.00.xx	Add ISO 15118-2 FDIS support
2.01.xx	Add Customer specific extensions
2.02.xx	Add Customer specific extensions
3.00.xx	Changes for Vector internal version management
3.01.xx	Added support for Tx streaming
3.02.xx	Added configurable optional padding for the EXI structs
3.03.xx	Changes for Vector internal version management

Table 1-1 Component history

2 Introduction

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

Supported AUTOSAR Release*:	3, 4	
Supported Configuration Variants:	Pre-compile	
Vendor ID:	EXI_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	EXI_MODULE_ID	255 decimal (according to ref. [3])

* For the precise AUTOSAR Release 3.x please see the release specific documentation.

Exi module is used to generate and parse DIN 70121 & ISO 15118 XML schema-conform EXI streams. Exi is a W3C recommendation to process XML-based message data on a binary level. DIN 70121 & ISO 15118 messages are encoded using the Exi format for data transmission.

2.1 Architecture Overview

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

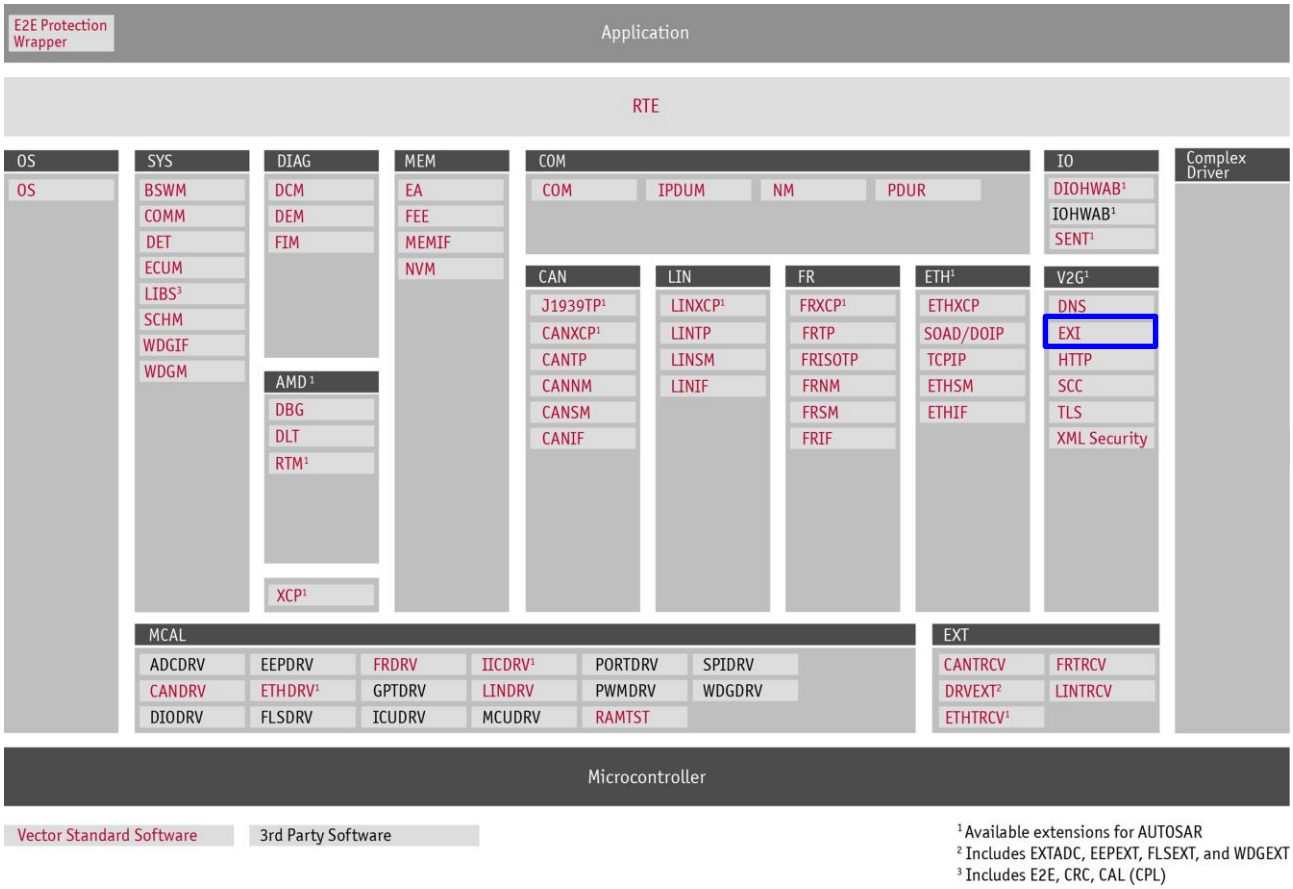

Figure 2-1 AUTOSAR 3.x Architecture Overview

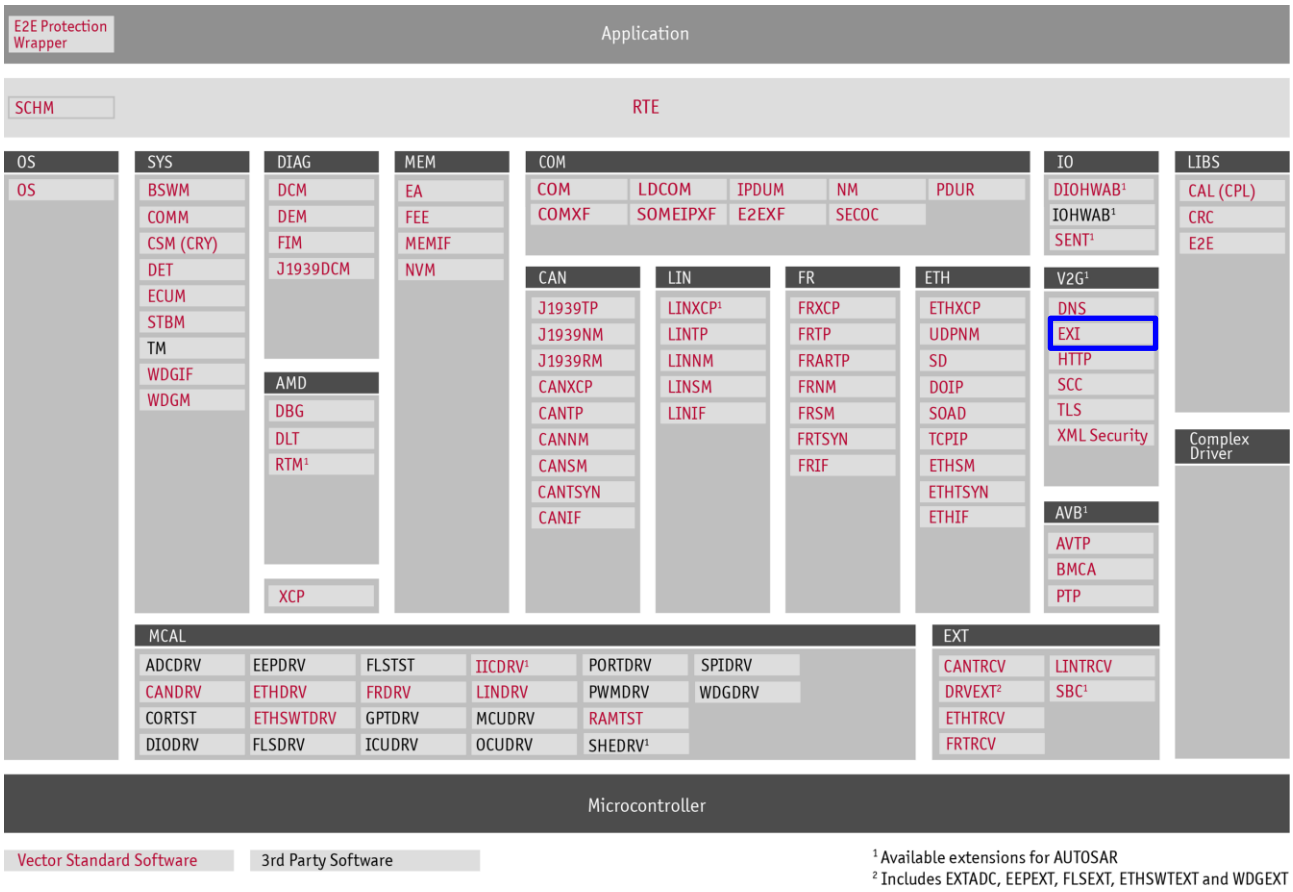

Figure 2-2 AUTOSAR 4.2 Architecture Overview

The next figure shows the interfaces to adjacent modules of the EXI. These interfaces are described in chapter 5.

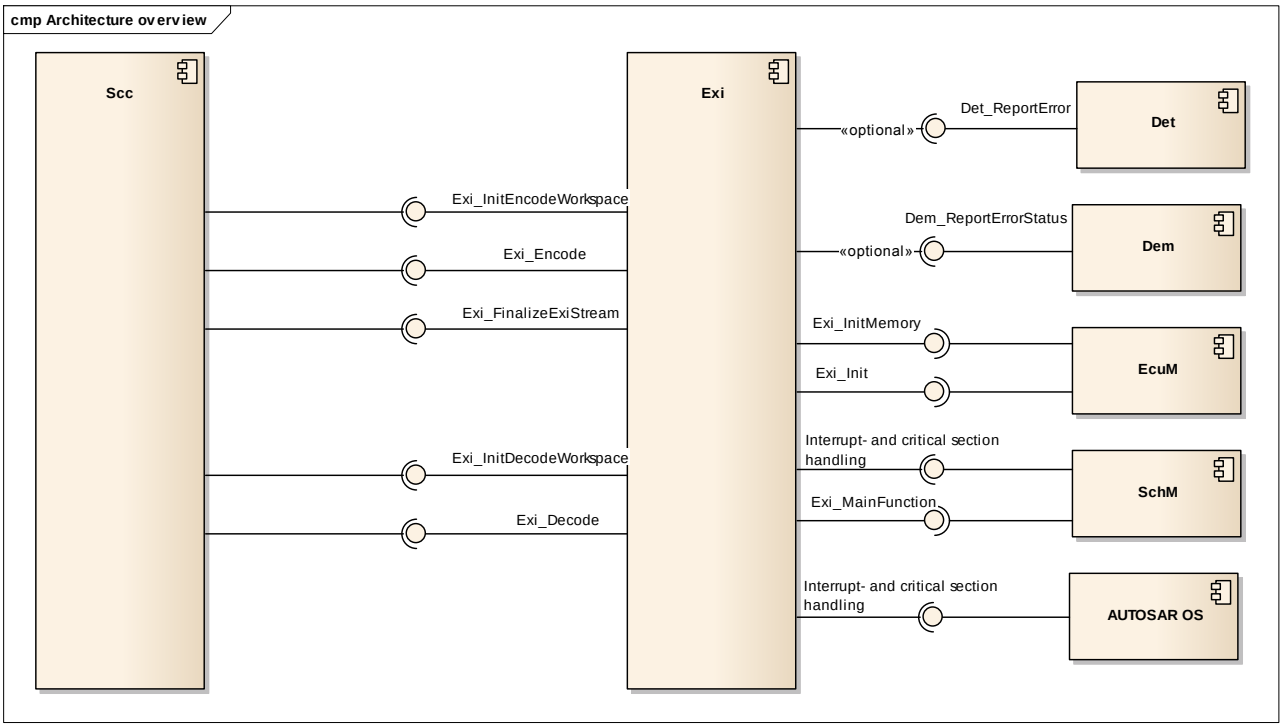

Figure 2-3 Interfaces to adjacent modules of the EXI

3 Functional Description

3.1 Features

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

The W3C standard functionality is specified in [1], functionality required by DIN 70121 and ISO 15118 standards is specified in [4], the corresponding features are listed in the tables

> Table 3-1 Supported W3C and DIN 70121 & ISO 15118 standard conform features

> Table 3-2 Not supported W3C and DIN 70121 & ISO 15118 standard conform features

For further information of not supported features see also chapter 6.4.2.

Vector Informatik provides further EXI functionality beyond the W3C and DIN 70121 & ISO 15118 standards. The corresponding features are listed in the table

> Table 3-3 Features provided beyond the W3C and DIN 70121 & ISO 15118 standard

The following features specified in [1] and [4] are supported:

Supported W3C and DIN 70121 & ISO 15118 Standard Conform Features
EXI header
EXI schema-informed grammar for DIN 70121 defined schema
EXI strict mode = false
Build-in data type representation for DIN 70121 required data types
Basic schema deviation support during decoding process

Table 3-1 Supported W3C and DIN 70121 & ISO 15118 standard conform features

The following features specified in [1] and [4] are not supported:

Not Supported W3C and DIN 70121 & ISO 15118 Standard Conform Features
EXI cookie
EXI options header field
Fidelity options
Evolving build-in XML grammar
Compression
Data type representation maps

Table 3-2 Not supported W3C and DIN 70121 & ISO 15118 standard conform features

The following features are provided beyond the AUTOSAR standard:

Features Provided Beyond The W3C and DIN 70121 & ISO 15118 Standards
Optimization: remove unneeded schema elements (e.g. AC messages for DC charging ECUs)

Features Provided Beyond The W3C and DIN 70121 & ISO 15118 Standards
Limitation of unbounded schema elements
Limited schema deviation support for DIN 70121 and automotive use case
Development error detection for internal encoding and decoding functions

Table 3-3 Features provided beyond the W3C and DIN 70121 & ISO 15118 standard

3.2 Initialization

The Exi component gets initialized by call of `Exi_InitMemory` and then `Exi_Init` out of EcuM.

If EcuM is not used the application has to call `Exi_InitMemory` and `Exi_Init`.

3.3 States

The Exi component is operational after initialization.

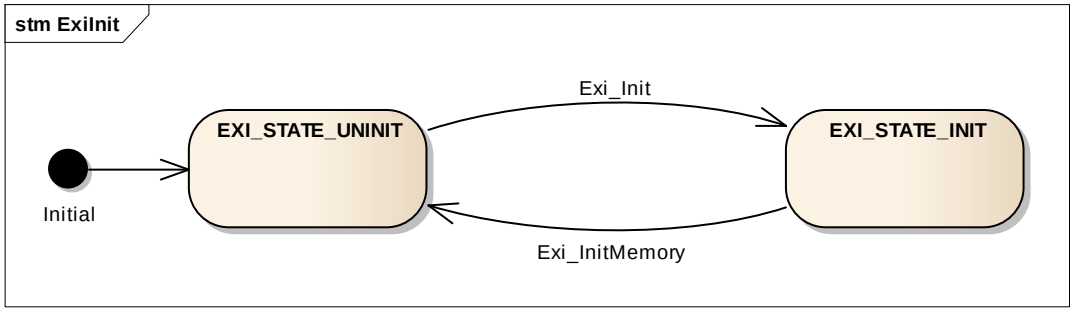

Figure 3-1 Exi component state

3.4 Main Function

The Exi Main Function shall be called by SchM.

3.5 Error Handling

3.5.1 Development Error Reporting

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

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

The reported EXI ID is 255. Instance ID is used to differ between EXI and other CDD modules. Instance ID 107 is used for EXI Core, 108 for Encoder and 109 for Decoder development errors.

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

Service ID	Service
0x01	`Exi_Init`
0x02	`Exi_MainFunction`
0x03	`Exi_InitDecodeWorkspace`
0x04	`Exi_InitEncodeWorkspace`
0x05	`Exi_Decode`
0x06	`Exi_Encode`
0x07	`Exi_FilanizeExiStream`
0x08	`Exi_GetVersionInfo`
0x09	`Used for EXI internal functions for basic encoding/decoding`

Table 3-4 Service IDs

Because of the schema dependent number and naming of internal encoding and decoding functions, instance ID 108 and 109 are introduced to report development errors that occur in these functions. Reported Service ID starts with 0x01 and the range depends in the schema. See `Exi_SchemaEncoder.h` and `Exi_SchemaDecoder.h` in case these errors are reported.

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

Error Code		Description
0x01	`EXI_E_NOT_INITIALIZED`	Exi not initialized
0x02	`EXI_E_INV_POINTER`	Invalid pointer
0x03	`EXI_E_INV_PARAM`	Invalid parameter
0x04	`EXI_E_INVALID_PB_CONFIG`	Invalid Post-Build configuration
0x05	`EXI_E_VALUE_OUT_OF_RANGE`	Invalid value

Table 3-5 Errors reported to DET

3.6 Exi Workspace Concept

To be able to avoid large memory usage EXI uses a storage concept. All memory required for encoding and decoding of EXI data must be provided by the application. Therefor a so called workspace is created. Workspaces consist of input and output data buffers. One buffer is used to store the C-structure application data the other to store the corresponding EXI stream.

For EXI streams EXI supports AUTOSAR defined `SoAd_TcpIpPbufType` buffers or linear buffers. `SoAd_TcpIpPbufType` buffers are defined as `IpBase_PbufType` in the MICROSAR stack and enabled the possibility for the application to directly forward via an AUTOSAR TcpIp Stack received EXI data to the EXI component or transmit the EXI stream without an additional copy step.

During encoding the application data can be located at any memory position because they are linked using pointers. Decoding EXI streams require a storage block size that is able to store the worst case C-structure data representation of the largest EXI message that should be decoded. All decoded structures will be placed in that memory section and linked by pointers starting at memory position zero.

C-structures that are included in the storage section are identified using so called Root Element ID defined in the Exi_SchemaTypes.h. This Root Element ID is used to be able to cast the uint8 storage pointer to the corresponding C-structure type. The Root Element ID is also used to be able to identify XSD substitution groups.

3.7 Exi Encoding

EXI provides the possibility to generate ISO 15118 and DIN 7012 schema conform EXI streams. Supported XML namespaces are urn:iso:15118:2:2010:AppProtocol and urn:din:70121:2012:MsgDef as specified in DIN 70121. EXI provides a structure based data representation of the XSD data types. For a detailed description how XSD data types are represented in the Exi_SchemaTypes.h see chapter 5.1.

3.7.1 Initialize Encoding Workspace

The `Exi_EncodeWorkspaceType` is used for EXI encoding. This type consists of two elements, the basic stream encoding workspace (`Exi_BSEncodeWorkspaceType`) for output data and the encoder input data (`Exi_EncoderInputDataType`).

Caution

`Exi_InitEncodeWorkspace` shall be used to set up the workspace before encoding the data. This is required before every new encoding process.

For initialization of the encoding workspace a pointer to the root C-structure of the data that should be encoded and a pointer to the output buffer where to store the EXI stream are required. The offset parameter can be used to define an offset in the output buffer where the EXI stream should start. This could be helpful when working with `IpBase_PbufType` to avoid an additional copy step.

After initialization the input data `RootElementId` must be set to the corresponding root element ID for the root C-structure. Now all preconditions are fulfilled to be able to encode the data.

Example: Create and initialize encoding workspace

```
/* variable declaration */
Exi_EncodeWorkspaceType Appl_EncWs;
IpBase_PbufType ExiPbuf;
Exi_SAP_supportedAppProtocolReqType SAPReq;
uint8 StreamBuffer[APPL_STREAM_BUFFER_LENGTH];
uint8 StructBuffer[APPL_STRUCT_BUFFER_SIZE];
const uint8 DinNamespace[26] = "urn:din:70121:2012:MsgDef";
Exi_SAP_protocolNamespaceType DinProtocolNamespace;
Exi_SAP_supportedAppProtocolReqType SAPReq;
Exi_SAP_AppProtocolType AppProtocol;
Exi_ReturnType RetValue = EXI_E_NOT_OK;
uint8 index;
/* setup PBuf (a normal application will get this buffer from lower layer TcpIp
component */
ExiPbuf.payload = &StreamBuffer[0];
ExiPbuf.totLen = sizeof(StreamBuffer);
ExiPbuf.len = sizeof(StreamBuffer);

/* initialize workspace */
Exi_InitEncodeWorkspace(&Appl_EncWs, &StructBuffer[0], &ExiPbuf, 0);
/* we want to encode a supportedAppProtocolReq message */
Appl_EncWs.InputData.RootElementId = EXI_SAP_SUPPORTED_APP_PROTOCOL_REQ_TYPE;
```

3.7.2 Basic Encoding

Encoding is done using the `Exi_Encode` API. In case the C-structure is a schema ISO 15118 and DIN 70121 valid root element (`EXI_DIN_V2G_MESSAGE_TYPE`, `EXI_SAP_SUPPORTED_APP_PROTOCOL_REQ_TYPE` and `EXI_SAP_SUPPORTED_APP_PROTOCOL_RES_TYPE`) and the EXI stream fits into the output buffer the function will return `EXI_E_OK`. The resulting EXI stream is written into the output buffer.

Caution

Streaming is not supported in this EXI version. The generated EXI stream has to fit completely into the output buffer.

Because EXI is used in Bit-aligned mode it is possible that the stream does not end at the end of a byte. Therefor `Exi_FinalizeExiStream` should be used to add padding bits and be able to determine the stream length in Bytes. The length is stored in `EncWsPtr->EncWs.BytePos`.

Example: Encode supportedAppProtocolReq

```
/*
<n1:supportedAppProtocolReq xmlns:n1="urn:iso:15118:2:2010:AppProtocol”
xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:schemaLocation="urn:iso:15118:2:2010:AppProtocol .\V2G_CI_AppProtocol.xsd">
  <AppProtocol>
    <ProtocolNamespace>urn:din:70121:2012:MsgDef</ProtocolNamespace>
    <VersionNumberMajor>2</VersionNumberMajor>
    <VersionNumberMinor>0</VersionNumberMinor>
    <SchemaID>1</SchemaID>
    <Priority>1</Priority>
  </AppProtocol>
</n1:supportedAppProtocolReq>
*/
/* write data into supportedAppProtocol message struct */
SAPReq.AppProtocol = &AppProtocol;
SAPReq.AppProtocol->ProtocolNamespace = &DinProtocolNamespace;
SAPReq.AppProtocol->VersionNumberMajor = 2;
SAPReq.AppProtocol->VersionNumberMinor = 0;
SAPReq.AppProtocol->Priority = 1;
SAPReq.AppProtocol->SchemaID = 1;
/* only one element -> set next pointer to 0 */
SAPReq.AppProtocol->NextAppProtocolPtr = (Exi_SAP_AppProtocolType*)NULL_PTR;
for(index=0; index<25; index++)
{
  SAPReq.AppProtocol->ProtocolNamespace->Buffer[index] = DinNamespace[index];
}
SAPReq.AppProtocol->ProtocolNamespace->Length = 25;

/* start encoding */
RetValue = Exi_Encode(&Appl_EncWs);
if(EXI_E_OK == RetValue)
{
  /* encoding finished successfully -> finalize stream */
  RetValue = Exi_FinalizeExiStream(&Appl_EncWs);
  if(EXI_E_OK == RetValue)
  {
    /* set output stream length */
    ExiPbuf.len = Appl_EncWs.EncWs.BytePos;
  }
}
```

Optional Elements are encoded using the presence flag parameter included in the structure. It is not required to set an optional element to a valid value if the flag is set to zero. The value will only be encoded into the EXI stream if the presence flag is set.

Example: Usage of the presence flag for optional parameters

```
/* optional element SchemaID will not be present in the encoded EXI stream */
Exi_SAP_supportedAppProtocolResType SAPRes;
SAPRes.ResponseCode = EXI_SAP_RESPONSE_CODE_TYPE_OK_SUCCESSFUL_NEGOTIATION;
SAPRes.SchemaIDFlag = 0;


/* optional element SchemaID will be present in the encoded EXI stream */
Exi_SAP_supportedAppProtocolResType SAPRes;
SAPRes.ResponseCode = EXI_SAP_RESPONSE_CODE_TYPE_OK_SUCCESSFUL_NEGOTIATION;
SAPRes.SchemaIDFlag = 1;
SAPRes.SchemaID = 1;
```

3.8 Exi Decoding

EXI supports the decoding of ISO 15118 and DIN 70121 schema conform EXI streams. Supported XML namespaces are urn:iso:15118:2:2010:AppProtocol and urn:din:70121:2012:MsgDef as specified in DIN 70121. EXI provides a structure based data representation of the XSD data types. For a detailed description how XSD data types are represented in the Exi_SchemaTypes.h see chapter 5.1.

3.8.1 Initialize Decoding Workspace

The `Exi_DecodeWorkspaceType` is used for EXI decoding. This type consists of two elements, the basic stream decoding workspace (`Exi_BSDecodeWorkspaceType`) for input data and the decoder output data (`Exi_DecoderOutputDataType`).

Caution

`Exi_InitDecodeWorkspace` shall be used to set up the workspace before decoding the data. This is required before every new decoding process.

For initialization of the decoding workspace a pointer to the received EXI stream as input data that should be decoded and a pointer to the storage section where the C-structure representation of the EXI data should be stored are required. The offset parameter can be used to define an offset in the output buffer where the EXI stream starts. This could be helpful when working with `IpBase_PbufType` to avoid an additional copy step.

Without any knowledge about the namespace the EXI stream is based on a decoding is not possible. This requires setting a so called schema set ID (`Exi_DecoderOutputDataType`, Element `SchemaSetId`) after initialization. Now all preconditions are fulfilled to be able to encode the data.

Example: Create and initialize decoding workspace

```
/* variable declaration */
Exi_DecodeWorkspaceType Appl_DecWs;
IpBase_PbufType ExiPbuf;
/* this byte array represents an EXI encoded supportedAppProtocolReq message */
const unsigned char supportedAppProtocolReq[34] = {
  0x80, 0x00, 0xDB, 0xAB, 0x93, 0x71, 0xD3, 0x23, 0x4B, 0x71, 0xD1, 0xB9,
  0x81, 0x89, 0x91, 0x89, 0xD1, 0x91, 0x81, 0x89, 0x91, 0xD2, 0x6B, 0x9B,
  0x3A, 0x23, 0x2B, 0x30, 0x02, 0x00, 0x00, 0x04, 0x00, 0x40
};
uint8 StructBuffer[APPL_STRUCT_BUFFER_SIZE];
Exi_ReturnType RetValue = EXI_E_NOT_OK;
uint8 index;
/* setup PBuf (a normal application will get this buffer from lower layer TcpIp
component */
ExiPbuf.payload = (uint8*)&supportedAppProtocolReq[0];
ExiPbuf.totLen = sizeof(supportedAppProtocolReq);
ExiPbuf.len = sizeof(supportedAppProtocolReq);

/* initialize workspace */
Exi_InitDecodeWorkspace(&Appl_DecWs, &ExiPbuf, &StructBuffer[0],
sizeof(StructBuffer), 0);
/* we want to decode a received message from the urn:iso:15118:2:2010:AppProtocol
namespace */
Appl_DecWs.OutputData.SchemaSetId = EXI_SCHEMA_SET_SAP_TYPE;
```

3.8.2 Basic Decoding

Decoding process is started with `Exi_Decode` call. EXI decoder requires all mandatory elements to be present in the stream and all values must be valid according to the schema defined type and value range. If this is not the case return value will be `EXI_E_INV_EVENT_CODE`, `EXI_E_ARR_OVERFLOW` or `EXI_E_INT_OVERFLOW`.

Decoder will place all decoded C-structures inside the given storage section and link them together using pointers. The root structure (entry point) is placed at memory position zero or at the via the offset parameter specified location. The root element ID will be set to the type ID of the entry point to enable the application to cast and read the correct data structure.

Example: Decode supportedAppProtocolReq

```
/* start decoding */
RetValue = Exi_Decode(&ExiClient_DecWs);
if(E_OK == RetValue)
{
  /* check if decoded message was a supportedAppProtocolReq */
  if(EXI_SAP_SUPPORTED_APP_PROTOCOL_REQ_TYPE ==
ExiClient_DecWs.OutputData.RootElementId)
  {
    /* cast storage pointer to supportedAppProtocolReq */
    Exi_SAP_supportedAppProtocolReqType* msg =
(Exi_SAP_supportedAppProtocolReqType*)&ExiClient_DecWs.OutputData.StoragePtr[0];
    if(TRUE == Appl_CompareStrings(&DinNamespace[0], sizeof(DinNamespace) - 1,
               &msg->AppProtocol->ProtocolNamespace->Buffer[0],
               msg->AppProtocol->ProtocolNamespace->Length))
    {
      /* namespace match */
      if((2 == msg->AppProtocol->VersionNumberMajor) &&
         (0 == msg->AppProtocol->VersionNumberMinor))
      {
        /* version match, the supported version is DIN 70121 */
      }
    }
  }
}
```

Optional elements are signaled using the flag element. The value should only be evaluated in case the flag is set to 1. Else runtime exceptions may occur because the structure value is initialized by zero.

3.9 Substitution Groups and Choice Elements

ISO 15118 and DIN 70121 makes use of XML substitution groups e.g. to substitute the BodyElement inside a V2G_Message with the correct message type, e.g. SessionSetupReq. To handle this with C-structures EXI implements an additional parameter that identifies the substituted element with the corresponding `Exi_RootElementIdType`.

Example: Using Substitution Groups

```
Exi_DIN_BodyType V2G_MsgBody;
Exi_DIN_SessionSetupReqType SessionSetupReq;
SessionSetupReq.EVCCID = &EVCCID;
V2G_MsgBody.BodyElementElementId = EXI_DIN_SESSION_SETUP_REQ_TYPE;
V2G_MsgBody.BodyElement = (Exi_DIN_BodyBaseType*)&SessionSetupReq;
V2G_MsgBody.BodyElementFlag = 1;
```

Choice Elements are used in the XML signature namespace as well as in the ServiceDetailsRes message. Choices are implemented using a structure that contains a union including the choice values and flags for all possible choice elements.

Example: Using Choice Elements

```
Exi_DIN_ParameterChoiceType ParameterChoice;
ParameterChoice.boolValueFlag = 1;
ParameterChoice.byteValueFlag = 0;
ParameterChoice.intValueFlag = 0;
ParameterChoice.physicalValueFlag = 0;
ParameterChoice.shortValueFlag = 0;
ParameterChoice.stringValueFlag = 0;
ParameterChoice.ChoiceValue.boolValue = true;
```

4 Integration

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

4.1 Scope of Delivery

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

4.1.1 Static Files

File Name	Description
Exi.c	Static source for EXI core.
Exi.h	Static header file for EXI API.
Exi_BSDecoder.c	Static source for basic stream decoding.
Exi_BSDecoder.h	Static header file for basic stream decoding.
Exi_BSEncoder.c	Static source for basic stream encoding.
Exi_BSEncoder.h	Static header file for basic stream encoding.
Exi_Cbk.h	Static header file for EXI callback API
Exi_Lcfg.h	Static header file for link time configuration data.
Exi_PBcfg.h	Static header file for post-build time configuration data.
Exi_Priv.h	Static header file for internal macro and variable declaration.
Exi_SchemaDecoder.c	Static source for schema dependent stream decoding.
Exi_SchemaDecoder.h	Static header file for schema dependent stream decoding.
Exi_SchemaEncoder.c	Static source for schema dependent stream encoding.
Exi_SchemaEncoder.h	Static header file for schema dependent stream encoding.
Exi_SchemaTypes.h	Static header file for schema dependent type definitions.
Exi_Types.h	Static header file for Exi type definitions.

Table 4-1 Static files

4.1.2 Dynamic Files

The dynamic files are generated by the configuration tool GENy.

File Name	Description
Exi_Cfg.h	Generated header file for pre-compile time configuration data.
Exi_Lcfg.c	Generated source for link time configuration data.
Exi_PBcfg.c	Generated source for post-build time configuration data.

Table 4-2 Generated files

4.2 Compiler Abstraction and Memory Mapping

The objects (e.g. variables, functions, constants) are declared by compiler independent definitions – the compiler abstraction definitions. Each compiler abstraction definition is assigned to a memory section.

The following table contains the memory section names and the compiler abstraction definitions of the EXI and illustrates their assignment among each other.

Compiler Abstraction Definitions / Memory Mapping Sections	EXI_CONST	EXI_CODE	EXI_PBCFG	EXI_PBCFG_ROOT	EXI_VAR_NOINIT	EXI_VAR_ZERO_INIT	EXI_APPL_DATA	EXI_APPL_VAR
EXI_START_SEC_CODE EXI_STOP_SEC_CODE		■					■	■
EXI_START_SEC_CONST_8BIT EXI_STOP_SEC_CONST_8BIT	■							
EXI_START_SEC_CONST_32BIT EXI_STOP_SEC_CONST_32BIT	■							
EXI_START_SEC_CONST_UNSPECIFIED EXI_STOP_SEC_CONST_UNSPECIFIED	■	■	■					
EXI_START_SEC_VAR_ZERO_INIT_8BIT EXI_STOP_SEC_VAR_ZERO_INIT_8BIT						■		
EXI_START_SEC_VAR_NOINIT_UNSPECIFIED EXI_STOP_SEC_VAR_NOINIT_UNSPECIFIED				■	■			

Table 4-3 Compiler abstraction and memory mapping

5 API Description

For an interfaces overview please see Figure 2-3.

5.1 Types derived from the XSD Schema Definitions

The types defined by the EXI are described in this chapter. Each XSD complex type is represented by a structure in the Exi_SchemaTypes.h. Simple types by the corresponding C data type representation, enumerations by enums. Complex types that are contained in the sequence or choice of another complex type are included using pointers. Multiple occurrences are handled using pointers (for complex types, strings and byte arrays) or arrays (simple types).

Because of the number of derived structures from the XSD schema this chapter does not list the provided structures and types.

Structures based on XSD schema are derived as follows: There exists one structure per complex type. The naming is Exi_<SchemaSet>_<SchemaType>. If the structure contains a sequence, all elements inside the sequence are represented as structure elements. Element names are derived directly from the schema element name. The element type is given by the schema element type. Simple types are represented as the corresponding C data type, complex types, strings and binary data are included using pointers. For optional elements an additional flag element with the name <ParameterName>Flag is added to the structure. Elements with multiple occurrences are derived as arrays for simple types or for complex types the type is extended by a next pointer (Next<ElementName>Ptr).

To be able to handle substitution groups an additional element <ElementName>ElementId is added to the base structure. This is set to the corresponding element ID (`Exi_RootElementIdType`). The element pointer must be casted into the correct type to read the substitution value correctly.

Choices are represented by a new structure named Exi_<SchemaSet>_<ElementName>ChoiceType. The new choice structure is included in the complex type structure via the element named ChoiceElement using pointers. The choice structure itself contains a union for the choice value and flags for all possible choices. For a better understanding the following examples may help you.

Example: XSD ParameterType with Choice Elements

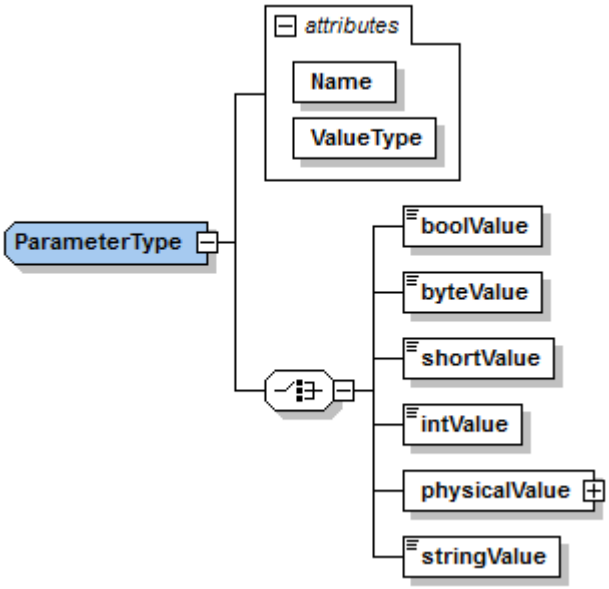

Figure 5-1 ParameterType XSD example

Example: ParameterType structures in Exi

```
typedef struct Exi_DIN_ParameterChoiceType Exi_DIN_ParameterChoiceType;

struct Exi_DIN_ParameterChoiceType
{
  union
  {
    Exi_DIN_PhysicalValueType* physicalValue;
    Exi_DIN_stringType* stringValue;
    sint32 intValue;
    sint16 shortValue;
    boolean boolValue;
    sint8 byteValue;
  } ChoiceValue;
  Exi_BitType physicalValueFlag : 1;
  Exi_BitType stringValueFlag : 1;
  Exi_BitType intValueFlag : 1;
  Exi_BitType shortValueFlag : 1;
  Exi_BitType boolValueFlag : 1;
  Exi_BitType byteValueFlag : 1;
};

typedef struct Exi_DIN_ParameterType Exi_DIN_ParameterType;

struct Exi_DIN_ParameterType
{
  Exi_DIN_AttributeNameType* Name;
  Exi_DIN_ParameterChoiceType* ChoiceElement;
  struct Exi_DIN_ParameterType* NextParameterPtr;
```

```
  Exi_DIN_AttributeValueType ValueType;
};
```

5.2 Services provided by EXI

5.2.1 Exi_InitMemory

Prototype	
`void Exi_InitMemory (void)`	
Parameter	
void	
Return code	
void	
Functional Description	
This function is used to initialize the global variables of the Exi at startup.	
Particularities and Limitations	
AUTOSAR extension Service ID: see table 'Service IDs'	
	Caution This function shall be called before Exi_Init.
Call Context	
This function can be called in any context.	

Table 5-1 Exi_InitMemory

5.2.2 Exi_Init

Prototype	
`void Exi_Init (const Exi_ConfigType *Exi_ConfigPtr)`	
Parameter	
Exi_ConfigPtr	Pointer to the post-build configuration data structure of the Exi. If the configuration variant pre-compile is used, the pointer given is ignored.
Return code	
void	
Functional Description	
This function is used to initialize the Exi component. The configuration data that shall be used by the Exi is passed as parameter.	
Particularities and Limitations	
Service ID: see table 'Service IDs'	

	Caution This function has to be called before usage of the module
Call Context	
This function shall be called for initialization.	

Table 5-2 Exi_Init

5.2.3 Exi_GetVersionInfo

Prototype	
`void Exi_GetVersionInfo (Std_VersionInfoType *VersionInfoPtr)`	
Parameter	
VersionInfoPtr	Pointer to a memory location where the EXI version information shall be stored.
Return code	
void	
Functional Description	
Returns the decimal encoded version information, vendor ID and AUTOSAR module ID of the EXI component. Note: additionally the version information can be read from the following pre-processor defines. BCD encoded: - SYSSERVICE_EXI_VERSION type uint16 - SYSSERVICE_EXI_RELEASE_VERSION type uint8 Decimal encoded: - EXI_SW_MAJOR_VERSION type uint8 - EXI_SW_MINOR_VERSION type uint8 - EXI_SW_PATCH_VERSION type uint8	
Particularities and Limitations	
Service ID: see table 'Service IDs' This function is only available if ExiVersionInfoApi is enabled.	
Call Context	
This function can be called in any context.	

Table 5-3 Exi_GetVersionInfo

5.2.4 Exi_MainFunction

Prototype
`void Exi_MainFunction (void)`

Parameter	
void	
Return code	
void	
Functional Description	
This function is used for basic administration.	
Particularities and Limitations	
Service ID: see table 'Service IDs'	
Call Context	
This function is called cyclically by the BSW Scheduler.	

Table 5-4 Exi_MainFunction

5.2.5 Exi_InitEncodeWorkspace using PBuf support

Prototype	
`Std_ReturnType Exi_InitEncodeWorkspace (Exi_EncodeWorkspaceType *EncWsPtr, const uint8 *InBufPtr, IpBase_PbufType *OutPBufPtr, uint16 OutBufOfs)`	
Parameter	
EncWsPtr	Pointer to EXI workspace containing the input and output data buffer
InBufPtr	Pointer to EXI input data buffer (EXI struct)
OutBufPtr	Pointer to EXI output data buffer (EXI stream)
OutBufOfs	Byte offset in output buffer at which encoding starts
Return code	
Std_ReturnType	E_OK: Finished successfully E_PENDING: More buffer requiered E_NOT_OK: Error
Functional Description	
This function is called to initialize an EXI encoding workspace. The encoding workspace is used to handle all data that is required to transform a schema conform data structures into a valid EXI stream.	
Particularities and Limitations	
Service ID: see table 'Service IDs'	
Call Context	
This function can be called in any context.	

Table 5-5 Exi_InitEncodeWorkspace using PBuf support

5.2.6 Exi_InitEncodeWorkspace using linear buffers

Prototype	
`Std_ReturnType Exi_InitEncodeWorkspace (Exi_EncodeWorkspaceType *EncWsPtr, const uint8 *InBufPtr, uint8 *OutBufPtr, uint16 OutBufLen, uint16 OutBufOfs)`	
Parameter	
EncWsPtr	Pointer to EXI workspace containing the input and output data buffer
InBufPtr	Pointer to EXI input data buffer (EXI struct)
OutBufPtr	Pointer to EXI output data buffer (EXI stream)
OutBufLen	Maximum EXI output data buffer length
OutBufOfs	Byte offset in output buffer at which encoding starts
Return code	
Std_ReturnType	E_OK: Finished successfully E_PENDING: More buffer requiered E_NOT_OK: Error
Functional Description	
This function is called to initialize an EXI encoding workspace. The encoding workspace is used to handle all data that is required to transform a schema conform data structures into a valid EXI stream.	
Particularities and Limitations	
Service ID: see table 'Service IDs'	
Call Context	
This function can be called in any context.	

Table 5-6 Exi_InitEncodeWorkspace using linear buffers

5.2.7 Exi_InitDecodeWorkspace using PBuf support

Prototype	
`Std_ReturnType Exi_InitDecodeWorkspace (Exi_DecodeWorkspaceType *DecWsPtr, const IpBase_PbufType *InPBufPtr, uint8 *OutBufPtr, uint16 InBufLen, uint16 OutBufLen, uint16 InBufOfs)`	
Parameter	
DecWsPtr	Pointer to EXI workspace containing the input and output data buffer
InBufPtr	Pointer to EXI input data buffer (EXI stream)
OutBufPtr	Pointer to EXI output data buffer (EXI struct)
InBufLen	EXI input data length
OutBufLen	Maximum EXI output data buffer length
InBufOfs	byte offset in input buffer at which decoding begings

Return code	
Std_ReturnType	E_OK: Finished successfully E_PENDING: More buffer requiered E_NOT_OK: Error
Functional Description	
This function is called to initialize an EXI decoding workspace.The decoding workspace is used to handle all data that is required to transform an EXI stream into a schema conform data structures.	
Particularities and Limitations	
Service ID: see table 'Service IDs'	
Call Context	
This function can be called in any context.	

Table 5-7 Exi_InitDecodeWorkspace using PBuf support

5.2.8 Exi_InitDecodeWorkspace using linear buffers

Prototype	
`Std_ReturnType` **`Exi_InitDecodeWorkspace`** `(Exi_DecodeWorkspaceType *DecWsPtr, const uint8 *InBufPtr, uint8 *OutBufPtr, uint16 OutBufLen, uint16 InBufOfs)`	
Parameter	
DecWsPtr	Pointer to EXI workspace containing the input and output data buffer
InBufPtr	Pointer to EXI input data buffer (EXI stream)
OutBufPtr	Pointer to EXI output data buffer (EXI struct)
OutBufLen	Maximum EXI output data buffer length
InBufOfs	byte offset in input buffer at which decoding begings
Return code	
Std_ReturnType	E_OK: Finished successfully E_PENDING: More buffer requiered E_NOT_OK: Error
Functional Description	
This function is called to initialize an EXI decoding workspace.The decoding workspace is used to handle all data that is required to transform an EXI stream into a schema conform data structures.	
Particularities and Limitations	
Service ID: see table 'Service IDs'	
Call Context	
This function can be called in any context.	

Table 5-8 Exi_InitDecodeWorkspace using linear buffers

5.2.9 Exi_Encode

Prototype	
`Exi_ReturnType Exi_Encode (Exi_EncodeWorkspaceType *EncWsPtr)`	
Parameter	
EncWsPtr	Pointer to EXI workspace containing the input and output data buffer
Return code	
Exi_ReturnType	EXI_E_OK: Finished successfully EXI_E_BUFFER_SIZE: Target buffer to small E_NOT_OK: Error
Functional Description	
This function is used to generate a schema conform EXI stream hat represents the data structure included in the encoding workspace.	
Particularities and Limitations	
Service ID: see table 'Service IDs'	
	Caution Make sure Exi_InitEncodeWorkspace is called before this function gets called.
Call Context	
This function can be called in any context.	

Table 5-9 Exi_Encode

5.2.10 Exi_FinalizeExiStream

Prototype	
`Exi_ReturnType Exi_FinalizeExiStream (Exi_EncodeWorkspaceType *EncWsPtr)`	
Parameter	
EncWsPtr	Pointer to EXI workspace containing the input and output data buffer
Return code	
Exi_ReturnType	EXI_E_OK: Finished successfully EXI_E_BUFFER_SIZE: Target buffer to small E_NOT_OK: Error
Functional Description	
Finalize an EXI stream. Padding will be added and EncWsPtr->EncWs.BytePos will indicate EXI stream length.	
Particularities and Limitations	
Service ID: see table 'Service IDs'	
	Caution Make sure Exi_InitEncodeWorkspace is called before this function gets called.

Call Context
This function can be called in any context.

Table 5-10 Exi_FinalizeExiStream

5.2.11 Exi_Decode

Prototype	
`Exi_ReturnType Exi_Decode (Exi_DecodeWorkspaceType *DecWsPtr)`	
Parameter	
DecWsPtr	Pointer to EXI workspace containing the input and output data buffer
Return code	
Exi_ReturnType	EXI_E_OK: Finished successfully EXI_E_INV_EVENT_CODE: Unsupported event code EXI_E_BUFFER_SIZE: Target buffer to small EXI_E_NOT_OK: Error
Functional Description	
This function is used to decode an EXI stream and store the data structures in the decoding workspace output storage.	
Particularities and Limitations	
Service ID: see table 'Service IDs'	
Caution Make sure Exi_InitEncodeWorkspace is called before this function gets called.	
Call Context	
This function can be called in any context.	

Table 5-11 Exi_Decode

5.3 Services used by EXI

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

Component	API
DET	Det_ReportError
DEM	Dem_SetEventStatus
ECUM	EcuM_GeneratorCompatibilityError

Table 5-12 Services used by the EXI

6 Configuration

In the EXI the attributes can be configured according to/ with the following methods/ tools:

> Configuration in DaVinci Configurator Pro, see 6.2.

> Configuration in GENy, for a detailed description see 6.3.

> Configuration manually in header files, for a detailed description see 6.4.

6.1 Configuration Variants

The EXI supports the configuration variants

> `VARIANT-PRE-COMPILE`

The configuration classes of the EXI parameters depend on the supported configuration variants. For their definitions please see the Exi_bswmd.arxml file.

6.2 Configuration with DaVinci Configurator Pro

For a detailed description of the parameters and possible values see Help in DaVinci Configurator Pro.

6.3 Configuration with GENy

The EXI is configured with the help of the configuration tool GENy. Select the Exi component in the “Component Selection” section in GENy.

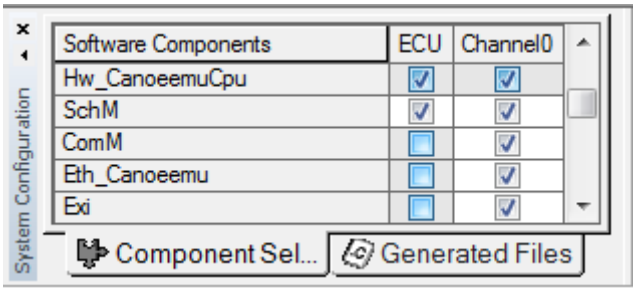

Figure 6-1 Exi has to be selected in the component selection tab.

6.3.1 Component configuration

Configurable Options	Exi
Common	
Configuration Variant	Variant 1 (Pre-compile Configuration)
Version Info Api	☐ *
Dev Error Detect	☐ *
User Config File	* ...
ISO 15118 / DIN 70121	
SupportedAppProtocol Support	☑ *
DIN 70121 Message Support	☑ *
AC EIM Profile	☐ *
AC PnC Profile	☐ *
DC EIM Profile	☑ *
DC PnC Profile	☐ *
Miscellaneous	
Allow Schema Deviation	☐ *
Decode String Truncation	☐ *
Max Occurrence Value Certificate Element	8*
Max Occurrence Value Consumption Cost Element	3*
Max Occurrence Value Cost Element	3*
Max Occurrence Value Generic Elements	16*
Max Occurrence Value Object Element	16*
Max Occurrence Value Parameter Element	2*
Max Occurrence Value Parameter Set Element	4*
Max Occurrence Value Payment Option Element	2*
Max Occurrence Value PMax Schedule Entry Element	12*
Max Occurrence Value Profile Entry Element	24*
Max Occurrence Value Reference Element	16*
Max Occurrence Value Root Certificate ID Element	20*
Max Occurrence Value Sales Tariff Entry Element	12*
Max Occurrence Value SA Schedule Tuple Element	3*
Max Occurrence Value Selected Service Element	3*
Max Occurrence Value Service Element	3*
Max Occurrence Value Signature Property Element	16*
Max Occurrence Value Transform Element	16*
Max Occurrence Value Unbounded Elements	32*
Internal Dev Error Detect	☐ *

Figure 6-2 The Exi component configuration

The following attributes can be configured for the Exi, if it is delivered as source code.

Attribute Name	Value Type	Values Default value is typed bold	Description
Configuration Variant	Enum	**Variant 1 (Pre-**	Select which configuration variant shall be

Attribute Name	Value Type	Values Default value is typed bold	Description
		Compile Configuration), Variant 2 (Link-time Configuration), Variant 3 (Post-build Configuration)	applied to your BSW module.
Version Info Api	Boolean	true, **false**	If 'Version Info Api' is enabled, the function Exi_GetVersionInfo() is available to get major, minor and patch version information.
Dev Error Detect	Boolean	true, **false**	Switches the Development Error Detection and Notification on or off. If “Dev Error Detection” is enabled, all development errors are reported to the Development Error Tracer (DET). The errors are described in 3.5.1. true: Development Error Detection and Notification on false: Development Error Detection and Notification off NOTE: In general, the development error detection is recommended during pre-test phase. It is not recommended to enable the development error detection in production code due to increased runtime and ROM needs.
User Config File	String		A configuration file is generated by GENy. If you want to overwrite settings in the generated configuration file, you can specify a path to a user defined configuration file. NOTE: The user defined configuration file will be included at the end of the generated file. Therefore definitions in the user defined configuration file can overwrite definitions in the generated configuration file. The contents of the user defined configuration file is copied to the end of Exi_Cfg.h.
SupportedAppProtocol Support	Boolean	**true**, fasle	If 'Supported App Protocol Support' is enabled, EXI Encoder/Decoder is able to handle Messages described in the ISO15118/DIN70121 supportedAppProtocol Schema.
DIN 70121 Message Support	Boolean	**true**, fasle	If ‘DIN 70121 Support' is enabled, EXI Encoder/Decoder is able to handle Messages described in the ISO15118/DIN70121 MsgDef

Attribute Name	Value Type	Values Default value is typed bold	Description
			Schema Set urn:din:70121:2012:MsgDef, version 2.0.
AC EIM Profile	Boolean	true, **false**	If 'AC External Identification Means Profile Support' is enabled, EXI Encoder/Decoder is able to handle Messages described in the ISO15118/DIN70121 MsgDef Schema Set, that are included in the External Identification Means (EIM) Profile for AC charging.
AC PnC Profile	Boolean	true, **false**	If 'AC Plug and Charge Profile Support' is enabled, EXI Encoder/Decoder is able to handle Messages described in the ISO15118/DIN70121 MsgDef Schema Set, version 1.0 that are included in the Plug and Charge (PnC) Profile for AC charging.
DC EIM Profile	Boolean	true, **false**	If 'DC External Identification Means Profile Support' is enabled, EXI Encoder/Decoder is able to handle Messages described in the ISO15118/DIN70121 MsgDef Schema Set, that are included in the External Identification Means (EIM) Profile for DC charging.
DC PnC Profile	Boolean	true, **false**	If 'DC Plug and Charge Profile Support' is enabled, EXI Encoder/Decoder is able to handle Messages described in the ISO15118/DIN70121 MsgDef Schema Set, version 1.0 that are included in the Plug and Charge (PnC) Profile for DC charging.
Allow Schema Deviation	Boolean	**true**, false	If 'Allow Schema Deviation' is enabled, schema deviations in the decoding process will be skipped silently if possible. Setting the value to false will lead to a decoding error with return value EXI_INV_EVENT_CODE for all schema deviations. This option can be helpful to accept some schema deviations and avoid errors receiving strings that do not match the schema definition.
Decode String Truncation	Boolean	true, **false**	If 'Decode String Truncation' is enabled, string characters that do not fit into the schema dependent target buffer will be skipped silently. Setting the value to false will lead to a decoding error. This option can be helpful to accept some schema deviations and avoid errors receiving strings that do not match the schema definition.
Max Occurrence Value Certificate Element	Integer	**8**	Support for unbounded maximum occurrence values is not possible. Therefore this value is used as max value for Certificate schema

Attribute Name	Value Type	Values Default value is typed bold	Description
			element used in SubCertifcates complex type included in the CertificateChain used in Certificate Installation and Update messages as well as in the Payment Details Request. The XSD schema does not limit the maximum number of Certificates because the value is set to unbounded. RESTRICTION: Only required if ISO PnC identification mode should be supported.
Max Occurrence Value Consumption Cost Element	Integer	**3**	Support for unbounded maximum occurrence values is not possible. Therefore this value is used as max value for ConsumptionCost schema element used in SalesTariffEntry complex type included in the SalesTariff used in the Charge Parameter Discovery Response. The XSD schema does not limit the maximum number of ConsumptionCost entries because the value is set to unbounded. HINT: ISO Requirement [V2G2-326] limits the value to 3. RESTRICTION: Only required if ISO PnC identification mode should be supported.
Max Occurrence Value Cost Element	Integer	**3**	Support for unbounded maximum occurrence values is not possible. Therefore this value is used as max value for Cost schema element used in ConsumptionCost complex type included in the SalesTariff used in the Charge Parameter Discovery Response. The XSD schema does not limit the maximum number of Cost entries because the value is set to unbounded. HINT: ISO Requirement [V2G2-334] limits the value to 3. RESTRICTION: Only required if ISO PnC identification mode should be supported.
Max Occurrence Value Generic Elements	Integer	**16**	Support for unbounded maximum occurrence values is not possible. Therefore this value is used as max value for generic #any and #other schema elements used in CanonicalizationMethod, DigestMethod, PGPData and SignatureMethod complex type included in the Signature used in the Message Header. The XSD schema does not limit the maximum number of these entries because

Attribute Name	Value Type	Values Default value is typed bold	Description
			the value is set to unbounded. RESTRICTION: Only required if ISO PnC identification mode with XML security should be supported.
Max Occurrence Value Generic Elements	Integer	**16**	Support for unbounded maximum occurrence values is not possible. Therefore this value is used as max value for generic #any and #other schema elements used in CanonicalizationMethod, DigestMethod, PGPData and SignatureMethod complex type included in the Signature used in the Message Header. The XSD schema does not limit the maximum number of these entries because the value is set to unbounded. RESTRICTION: Only required if ISO PnC identification mode with XML security should be supported.
Max Occurrence Value Parameter Element	Integer	**2**	Support for unbounded maximum occurrence values is not possible. Therefore this value is used as max value for Parameter schema elements used in ParameterSet complex type included in the ParameterList used in the Service Details Response. The XSD schema does not limit the maximum number of these entries because the value is set to unbounded. HINT: ISO specified services internet access requieres 2 and certificate requires 1 Parameter. RESTRICTION: Only required if Services with Service Details (e.g. internet access, certificate service) should be supported.
Max Occurrence Value Parameter Set Element	Integer	**4**	Support for unbounded maximum occurrence values is not possible. Therefore this value is used as max value for ParameterSet schema elements used in ParameterList complex type included in the ParameterList used in the Service Details Response. The XSD schema does not limit the maximum number of these entries because the value is set to unbounded. HINT: ISO specified services internet access defines 4 and certificate defines 2 ParameterLists.

Attribute Name	Value Type	Values Default value is typed bold	Description
			RESTRICTION: Only required if Services with Service Details (e.g. internet access, certificate service) should be supported.
Max Occurrence Value Payment Option Element	Integer	**2**	Support for unbounded maximum occurrence values is not possible. Therefore this value is used as max value for PaymentOption schema elements used in PaymentOptions complex type included in the Service Discovery Response. The XSD schema does not limit the maximum number of these entries because the value is set to unbounded. HINT: ISO specified only 2 valid PaymentOptions: Contract and ExternalPayment.
Max Occurrence Value PMax Schedule Entry Element	Integer	**12**	Support for unbounded maximum occurrence values is not possible. Therefore this value is used as max value for PMaxScheduleEntry schema elements used in PMaxSchedule complex type included in the SAScheduleList used in the Charge Parameter Response. The XSD schema does not limit the maximum number of these entries because the value is set to unbounded. HINT: ISO Requirement [V2G2-312] limits the value to 12.
Max Occurrence Value Profile Entry Element	Integer	**24**	Support for unbounded maximum occurrence values is not possible. Therefore this value is used as max value for ProfileEntry schema elements used in ChargingProfile complex type included in the Power Delivery Request. The XSD schema does not limit the maximum number of these entries because the value is set to unbounded. HINT: ISO Requirement [V2G2-287] limits the value to 24.
Max Occurrence Value Reference Element	Integer	**16**	Support for unbounded maximum occurrence values is not possible. Therefore this value is used as max value for Reference schema elements used in Manifest or SignedInfo complex type. Manifest is a XML Security root element and may appear as a #any element below the Signature in the MessageHeader. SignedInfo complex type is included in the Signature in the MessageHeader. The XSD schema does not limit the maximum number of these entries because the value is set to unbounded.

Attribute Name	Value Type	Values Default value is typed bold	Description
			RESTRICTION: Only required if ISO PnC identification mode with XML security should be supported.
Max Occurrence Value Root Certificate ID Element	Integer	**20**	Support for unbounded maximum occurrence values is not possible. Therefore this value is used as max value for RootCertificateID schema element used in ListOfRootCertificateIDs complex type included in Certificate Installation and Update Request messages. The XSD schema does not limit the maximum number of RootCertificateID because the value is set to unbounded. HINT: ISO Requirement [V2G2-357] limits the value to 20. RESTRICTION: Only required if ISO PnC identification mode should be supported.
Max Occurrence Value Sales Tariff Entry Element	Integer	**12**	Support for unbounded maximum occurrence values is not possible. Therefore this value is used as max value for SalesTariffEntry schema element used in SalesTariff complex type included in the Charge Parameter Discovery Response message. The XSD schema does not limit the maximum number of SalesTariffEntry because the value is set to unbounded. HINT: ISO Requirement [V2G2-320] limits the value to 12. RESTRICTION: Only required if ISO PnC identification mode should be supported.
Max Occurrence Value SA Schedule Tuple Element	Integer	**3**	Support for unbounded maximum occurrence values is not possible. Therefore this value is used as max value for SAScheduleTuple schema element used in SAScheduleList complex type included in the Charge Parameter Discovery Response message. The XSD schema does not limit the maximum number of SAScheduleTuple because the value is set to unbounded. HINT: ISO Requirement [V2G2-295] limits the value to 3. RESTRICTION: Only required if ISO PnC

Attribute Name	Value Type	Values Default value is typed bold	Description
			identification mode should be supported.
Max Occurrence Value Selected Service Element	Integer	**3**	Support for unbounded maximum occurrence values is not possible. Therefore this value is used as max value for SelectedService schema element used in SelectedServiceList complex type included in the Service Payment Selection Request message. The XSD schema does not limit the maximum number of SelectedService because the value is set to unbounded. HINT: ISO Requirement [V2G2-417] only defines 3 known services (AC_DC_Charging, Certificate and InternetAccess).
Max Occurrence Value Service Element	Integer	**3**	Support for unbounded maximum occurrence values is not possible. Therefore this value is used as max value for Service schema element used in ServiceTagList complex type included in the Service Discovery Response message. The XSD schema does not limit the maximum number of Service because the value is set to unbounded. HINT: ISO Requirement [V2G2-417] only defines 3 known services (AC_DC_Charging, Certificate and InternetAccess).
Max Occurrence Value Signature Property Element	Integer	**16**	Support for unbounded maximum occurrence values is not possible. Therefore this value is used as max value for SignatureProperty schema element included in SignatureProperties complex type. SignatureProperty is a XML Security root element and may appear as a #any element below the Signature in the MessageHeader. The XSD schema does not limit the maximum number of SignatureProperty because the value is set to unbounded. RESTRICTION: Only required if ISO PnC identification mode with XML security should be supported.
Max Occurrence Value Transform Element	Integer	**16**	Support for unbounded maximum occurrence values is not possible. Therefore this value is used as max value for Transform schema element included in Transforms complex type is used in the Signature included in the MessageHeader. The XSD schema does not limit the maximum number of Transform because the value is set to unbounded.

Attribute Name	Value Type	Values Default value is typed bold	Description
			RESTRICTION: Only required if ISO PnC identification mode with XML security should be supported.
Max Occurrence Value Unbounded Elements	Integer	**32**	Support for unbounded maximum occurrence values is not possible. Therefore this value is used as max value for all elements in the schema where MaxOccurs is set to "unbounded" and no special optimization attribute is available.
Internal Dev Error Detect	Boolean	true, **false**	If 'Internal Dev Error Detection' is enabled, all development errors for internal EXI Encoder/Decoder functions are reported to the Development Error Tracer (DET). The errors are identified via module Instance ID 108 for Encoder and 109 for Decoder errors. Note: In general, the development error detection is recommended during pre-test phase. It is not recommended to enable the development error detection in production code due to increased runtime and ROM needs.

Table 6-1 GENy configuration parameter descriptions

6.4 Configuration manually in Header and Source Files

Manual configuration requires to create the normally tool-based created files Exi_cfg.h, Exi_Lcfg.c and Exi_PBcfg.c by hand.

6.4.1 Exi_cfg.h

For manual configuration in Exi_cfg.h make the configuration based on the following example as a template.

Example for Exi_cfg.h

```
/* ----------------------------------------------------------------------------
  Filename:    Exi_Cfg.h
 ---------------------------------------------------------------------------- */
#if !defined(EXI_CFG_H)
#define EXI_CFG_H
/* ----------------------------------------------------------------------------
    &&&~ Include
 ---------------------------------------------------------------------------- */

#include "Std_Types.h"

/* ----------------------------------------------------------------------------
    &&&~ Exi EcuC Global Configuration Container Name
 ---------------------------------------------------------------------------- */

/* use this define for Exi_Init()-call */
#define ExiConfigSet                        Exi_Config


/* ----------------------------------------------------------------------------
    &&&~ Defines
 ---------------------------------------------------------------------------- */
/* do not change the configuration variant! */
#define EXI_CONFIG_VARIANT                  1U

/* Support for unbounded maximum occurrence values is not possible. Therefore
this value is used as max value for Certificate schema element used in
SubCertifcates complex type included in the CertificateChain used in Certificate
Installation and Update messages as well as in the Payment Details Request. The
XSD schema does not limit the maximum number of Certificates because the value is
set to unbounded.

RESTRICTION: Only required if ISO PnC identification mode should be supported. */
#define EXI_MAXOCCURS_CERTIFICATE           8U
/* Support for unbounded maximum occurrence values is not possible. Therefore
this value is used as max value for ConsumptionCost schema element used in
SalesTariffEntry complex type included in the SalesTariff used in the Charge
Parameter Discovery Response. The XSD schema does not limit the maximum number of
ConsumptionCost entries because the value is set to unbounded.

HINT: ISO Requirement [V2G2-326] limits the value to 3.

RESTRICTION: Only required if ISO PnC identification mode should be supported. */
#define EXI_MAXOCCURS_CONSUMPTIONCOST       3U
/* Support for unbounded maximum occurrence values is not possible. Therefore
this value is used as max value for Cost schema element used in ConsumptionCost
complex type included in the SalesTariff used in the Charge Parameter Discovery
Response. The XSD schema does not limit the maximum number of Cost entries
because the value is set to unbounded.

HINT: ISO Requirement [V2G2-334] limits the value to 3.

RESTRICTION: Only required if ISO PnC identification mode should be supported. */
#define EXI_MAXOCCURS_COST                  3U
/* Support for unbounded maximum occurrence values is not possible. Therefore
this value is used as max value for generic #any and #other schema elements used
in CanonicalizationMethod, DigestMethod, PGPData and SignatureMethod complex type
```

```
included in the Signature used in the Message Header. The XSD schema does not
limit the maximum number of these entries because the value is set to unbounded.

RESTRICTION: Only required if ISO PnC identification mode with XML security
should be supported. */
#define EXI_MAXOCCURS_GENERICELEMENT         16U
/* Support for unbounded maximum occurrence values is not possible. Therefore
this value is used as max value for Object schema elements included in the
Signature complex type used in the Message Header. The XSD schema does not limit
the maximum number of these entries because the value is set to unbounded.

RESTRICTION: Only required if ISO PnC identification mode with XML security
should be supported. */
#define EXI_MAXOCCURS_OBJECT                 16U
/* Support for unbounded maximum occurrence values is not possible. Therefore
this value is used as max value for Parameter schema elements used in
ParameterSet complex type included in the ParameterList used in the Service
Details Response. The XSD schema does not limit the maximum number of these
entries because the value is set to unbounded.

HINT: ISO specified services internet access requieres 2 and certificate requires
1 Parameter.

RESTRICTION: Only required if Services with Service Details (e.g. internet
access, certificate service) should be supported. */
#define EXI_MAXOCCURS_PARAMETER              2U
/* Support for unbounded maximum occurrence values is not possible. Therefore
this value is used as max value for ParameterSet schema elements used in
ParameterList complex type included in the ParameterList used in the Service
Details Response. The XSD schema does not limit the maximum number of these
entries because the value is set to unbounded.

HINT: ISO specified services internet access defines 4 and certificate defines 2
ParameterLists.

RESTRICTION: Only required if Services with Service Details (e.g. internet
access, certificate service) should be supported. */
#define EXI_MAXOCCURS_PARAMETERSET           4U
/* Support for unbounded maximum occurrence values is not possible. Therefore
this value is used as max value for PaymentOption schema elements used in
PaymentOptions complex type included in the Service Discovery Response. The XSD
schema does not limit the maximum number of these entries because the value is
set to unbounded.

HINT: ISO specified only 2 valid PaymentOptions: Contract and ExternalPayment. */
#define EXI_MAXOCCURS_PAYMENTOPTION          2U
/* Support for unbounded maximum occurrence values is not possible. Therefore
this value is used as max value for PMaxScheduleEntry schema elements used in
PMaxSchedule complex type included in the SAScheduleList used in the Charge
Parameter Response. The XSD schema does not limit the maximum number of these
entries because the value is set to unbounded.

HINT: ISO Requirement [V2G2-312] limits the value to 12. */
#define EXI_MAXOCCURS_PMAXSCHEDULEENTRY      12U
/* Support for unbounded maximum occurrence values is not possible. Therefore
this value is used as max value for ProfileEntry schema elements used in
ChargingProfile complex type included in the Power Delivery Request. The XSD
schema does not limit the maximum number of these entries because the value is
set to unbounded.

HINT: ISO Requirement [V2G2-287] limits the value to 24. */
```

```
#define EXI_MAXOCCURS_PROFILEENTRY          24U
/* Support for unbounded maximum occurrence values is not possible. Therefore
this value is used as max value for Reference schema elements used in Manifest or
SignedInfo complex type. Manifest is a XML Security root element and may appear
as a #any element below the Signature in the MessageHeader. SignedInfo complex
type is included in the Signature in the MessageHeader. The XSD schema does not
limit the maximum number of these entries because the value is set to unbounded.

RESTRICTION: Only required if ISO PnC identification mode with XML security
should be supported. */
#define EXI_MAXOCCURS_REFERENCE             16U
/* Support for unbounded maximum occurrence values is not possible. Therefore
this value is used as max value for RootCertificateID schema element used in
ListOfRootCertificateIDs complex type included in Certificate Installation and
Update Request messages. The XSD schema does not limit the maximum number of
RootCertificateID because the value is set to unbounded.

HINT: ISO Requirement [V2G2-357] limits the value to 20.

RESTRICTION: Only required if ISO PnC identification mode should be supported. */
#define EXI_MAXOCCURS_ROOTCERTIFICATEID     20U
/* Support for unbounded maximum occurrence values is not possible. Therefore
this value is used as max value for SalesTariffEntry schema element used in
SalesTariff complex type included in the Charge Parameter Discovery Response
message. The XSD schema does not limit the maximum number of SalesTariffEntry
because the value is set to unbounded.

HINT: ISO Requirement [V2G2-320] limits the value to 12.

RESTRICTION: Only required if ISO PnC identification mode should be supported. */
#define EXI_MAXOCCURS_SALESTARIFFENTRY      12U
/* Support for unbounded maximum occurrence values is not possible. Therefore
this value is used as max value for SAScheduleTuple schema element used in
SAScheduleList complex type included in the Charge Parameter Discovery Response
message. The XSD schema does not limit the maximum number of SAScheduleTuple
because the value is set to unbounded.

HINT: ISO Requirement [V2G2-295] limits the value to 3.

RESTRICTION: Only required if ISO PnC identification mode should be supported. */
#define EXI_MAXOCCURS_SASCHEDULETUPLE       3U
/* Support for unbounded maximum occurrence values is not possible. Therefore
this value is used as max value for SelectedService schema element used in
SelectedServiceList complex type included in the Service Payment Selection
Request message. The XSD schema does not limit the maximum number of
SelectedService because the value is set to unbounded.

HINT: ISO Requirement [V2G2-417] only defines 3 known services (AC_DC_Charging,
Certificate and InternetAccess). */
#define EXI_MAXOCCURS_SELECTEDSERVICE       3U
/* Support for unbounded maximum occurrence values is not possible. Therefore
this value is used as max value for Service schema element used in ServiceTagList
complex type included in the Service Discovery Response message. The XSD schema
does not limit the maximum number of Service because the value is set to
unbounded.

HINT: ISO Requirement [V2G2-417] only defines 3 known services (AC_DC_Charging,
Certificate and InternetAccess). */
#define EXI_MAXOCCURS_SERVICE               3U
/* Support for unbounded maximum occurrence values is not possible. Therefore
this value is used as max value for SignatureProperty schema element included in
```

```
SignatureProperties complex type. SignatureProperty is a XML Security root
element and may appear as a #any element below the Signature in the
MessageHeader. The XSD schema does not limit the maximum number of
SignatureProperty because the value is set to unbounded.

RESTRICTION: Only required if ISO PnC identification mode with XML security
should be supported. */
#define EXI_MAXOCCURS_SIGNATUREPROPERTY      16U
/* Support for unbounded maximum occurrence values is not possible. Therefore
this value is used as max value for Transform schema element included in
Transforms complex type is used in the Signature included in the MessageHeader.
The XSD schema does not limit the maximum number of Transform because the value
is set to unbounded.

RESTRICTION: Only required if ISO PnC identification mode with XML security
should be supported. */
#define EXI_MAXOCCURS_TRANSFORM              16U
/* Support for unbounded maximum occurrence values is not possible. Therefore
this value is used as max value for all elements in the schema where MaxOccurs is
set to "unbounded" and no special optimization attribute is available. */
#define EXI_MAXOCCURS_UNBOUNDED              32U


/* ----------------------------------------------------------------------------
    &&&~ Precompile Config
 ---------------------------------------------------------------------------- */
/* Version Info Api support */
#define EXI_VERSION_INFO_API STD_OFF
/* Development error detection */
#define EXI_DEV_ERROR_DETECT STD_OFF
/* Internal development error detection */
#define EXI_INTERNAL_DEV_ERROR_DETECT STD_OFF
/* If 'Decode String Truncation' is enabled, string characters that do not fit
into the schema dependent target buffer will be skipped silently. Setting the
value to false will lead to a decoding error.

This option can be helpful to accept some schema deviations and avoid errors
receiving strings that do not match the schema definition.*/
#define EXI_ENABLE_DECODE_STRING_TRUNCATION STD_OFF
/* do not change this value because the feature is not supported at the moment */
#define EXI_ENABLE_STREAMING_SUPPORT STD_OFF
/* switch between IpBase_PbufType or linear buffer for Exi streams  */
#define EXI_ENABLE_PBUF_SUPPORT STD_ON
/* If 'EXI_ENABLE_SAP_MESSAGE_SET' is enabled, EXI Encoder/Decoder is able to
handle Messages described in the ISO15118/DIN70121 supportedAppProtocol Schema */
#define EXI_ENABLE_SAP_MESSAGE_SET STD_ON
/* If 'EXI_ENABLE_DIN_MESSAGE_SET' is enabled, EXI Encoder/Decoder is able to
handle Messages described in the ISO15118/DIN70121 MsgDef Schema Set
urn:din:70121:2012:MsgDef, version 2.0. */
#define EXI_ENABLE_DIN_MESSAGE_SET STD_ON
/* If 'EXI_ENABLE_EV_MESSAGE_SET' is enabled, EXI Encoder/Decoder is able to
handle the message set required for EV side */
#define EXI_ENABLE_EV_MESSAGE_SET STD_ON
/* If 'EXI_ENABLE_EVSE_MESSAGE_SET' is enabled, EXI Encoder/Decoder is able to
handle the message set required for EVSE side */
#define EXI_ENABLE_EVSE_MESSAGE_SET STD_OFF
/* If 'EXI_ENABLE_AC_BASIC_MESSAGE_SET' is enabled, EXI Encoder/Decoder is able
to handle Messages described in the ISO15118/DIN70121 MsgDef Schema Set that are
included in the External Identification Means (EIM) Profile for AC charging. */
#define EXI_ENABLE_AC_BASIC_MESSAGE_SET STD_OFF
/* If 'EXI_ENABLE_DC_BASIC_MESSAGE_SET' is enabled, EXI Encoder/Decoder is able
```

```
to handle Messages described in the ISO15118/DIN70121 MsgDef Schema Set that are
included in the External Identification Means (EIM) Profile for DC charging */
#define EXI_ENABLE_DC_BASIC_MESSAGE_SET STD_ON
/* If 'EXI_ENABLE_AC_EXTENDED_MESSAGE_SET ' is enabled, EXI Encoder/Decoder is
able to handle Messages described in the ISO15118/DIN70121 MsgDef Schema Set that
are included in the Plug and Charge (PnC) Profile for AC charging. */
#define EXI_ENABLE_AC_EXTENDED_MESSAGE_SET STD_OFF
/* If 'EXI_ENABLE_DC_EXTENDED_MESSAGE_SET ' is enabled, EXI Encoder/Decoder is
able to handle Messages described in the ISO15118/DIN70121 MsgDef Schema Set that
are included in the Plug and Charge (PnC) Profile for DC charging.  */
#define EXI_ENABLE_DC_EXTENDED_MESSAGE_SET STD_OFF
/* If 'EXI_ENABLE_XML_SEC_MESSAGE_SET ' is enabled, EXI Encoder/Decoder is able
to handle Messages described in the XML Security Schema Set that are required for
Signatures required in the Plug and Charge (PnC) Profile. */
#define EXI_ENABLE_XML_SEC_MESSAGE_SET STD_OFF
/* If 'EXI_ALLOW_SCHEMA_DEVIATION' is enabled, schema deviations in the decoding
process will be skipped silently if possible. Setting the value to false will
lead to a decoding error with return value EXI_INV_EVENT_CODE for all schema
deviations.

This option can be helpful to accept some schema deviations and avoid errors
receiving strings that do not match the schema definition. */
#define EXI_ALLOW_SCHEMA_DEVIATION STD_ON

#endif /* EXI_CFG_H */
```

6.4.2 Exi_Lcfg.c

For manual configuration in Exi_Lcfg.c make the configuration based on the following example as a template.

Example for Exi_Lcfg.c

```
/* ----------------------------------------------------------------------------
  Filename:    Exi_Lcfg.c
 ---------------------------------------------------------------------------- */
/* ----------------------------------------------------------------------------
    &&&~ Include
 ---------------------------------------------------------------------------- */

#include "Exi_Lcfg.h"
#include "Exi.h"


#define EXI_START_SEC_CONST_UNSPECIFIED
/* PRQA S 5087 1 */ /* MD_MSR_19.1 */
#include "MemMap.h"

/* could be used as config pointer in config variant 1 and 2 */
CONST(Exi_ConfigType, EXI_CONST) Exi_Config = 0U;

#define EXI_STOP_SEC_CONST_UNSPECIFIED
/* PRQA S 5087 1 */ /* MD_MSR_19.1 */
#include "MemMap.h"
```

6.4.3 Exi_PBcfg.c

For manual configuration the Exi_PBcfg.c could be left empty in this version.

7 Glossary and Abbreviations

7.1 Glossary

Term	Description
GENy	Generation tool for CANbedded and MICROSAR components
SysService_Exi	Vector Informatik component name of the MICROSAR Exi Parser and Generator module

Table 7-1 Glossary

7.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
DEM	Diagnostic Event Manager
DET	Development Error Tracer
EAD	Embedded Architecture Designer
ECU	Electronic Control Unit
EIM	External Identification Means
EV	Electric Vehicle
EVSE	Electric Vehicle Supply Equipment
EXI	Efficient XML Interchange
HIS	Hersteller Initiative Software
ISR	Interrupt Service Routine
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
PnC	Plug and Charge
PPort	Provide Port
RPort	Require Port
RTE	Runtime Environment
SRS	Software Requirement Specification
SWC	Software Component
SWS	Software Specification

Table 7-2 Abbreviations

8 Contact

Visit our website for more information on

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

www.vector.com