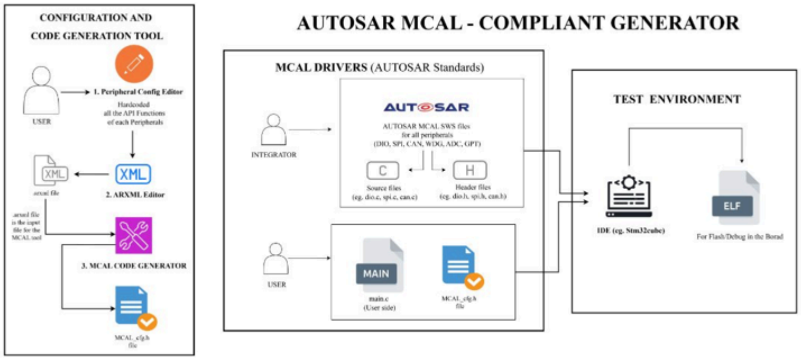
AUTOSAR MCAL - Overall User Manual

ARCHITECTURE


CONFIGURATION AND
CODE GENERATION TOOL
USER
1. Peripheral Config Editor
Hardcoded all the API Functions of each Peripherals
XML
arxml file
2. ARXML Editor
.arxml file is the input file for the MCAL tool
3. MCAL CODE GENERATOR
MCAL_cfg.h file
AUTOSAR MCAL - COMPLIANT GENERATOR
MCAL DRIVERS (AUTOSAR Standards)
INTEGRATOR
AUTOSAR
AUTOSAR MCAL SWS files for all peripherals (DIO, SPI, CAN, WDG, ADC, GPT)
C
Source files (eg. dio.c, spi.c, can.c)
H
Header files (eg. dio.h, spi.h, can.h)
USER
MAIN
main.c (User side)
MCAL_cfg.h file
TEST ENVIRONMENT
IDE (eg. Stm32cube)
ELF
For Flash/Debug in the Borad

TOOL USAGE

Installation

A desktop application for generating AUTOSAR MCAL (Microcontroller Abstraction Layer) configurations and code from ARXML files. This tool simplifies the process of configuring and generating drivers for the DIO, ADC, CAN module.

Git installation (For Integrators)

If you want to modify the code or build the application from source, follow these instructions.

Prerequisites

- Python 3.x
- pip (Python package installer)
- Git

Clone the repository

```
git clone https://github.com/Creamcollar/autosar-arxml-codegen.git
```

Navigate to the project directory

```
cd autosar-arxml-codegen
```

Create an env

```
python -m venv venv
# On Unix/Linux/macOS:
source venv/bin/activate

# On Windows Command Prompt:
venv\Scripts\activate
```

Install the required dependencies

```
pip install -r requirements.txt
```

Running the Application from Source

```
python main.py
```

Technologies Used

- **Language:** Python
- **GUI Framework:** Tkinter
- **Template Engine:** Parse library
- **Data Format:** ARXML (AUTOSAR XML)

Feature usage

Peripheral Config editor

Select your peripheral using the dropdown feature, once selected what are the possible configuration parameter is available in the selected peripheral in the **AUTOSAR standards R24-11**

Options

- Select peripheral from the module dropdown
- Enter the location to save in the file path or browse the path.
- Generate ARXML

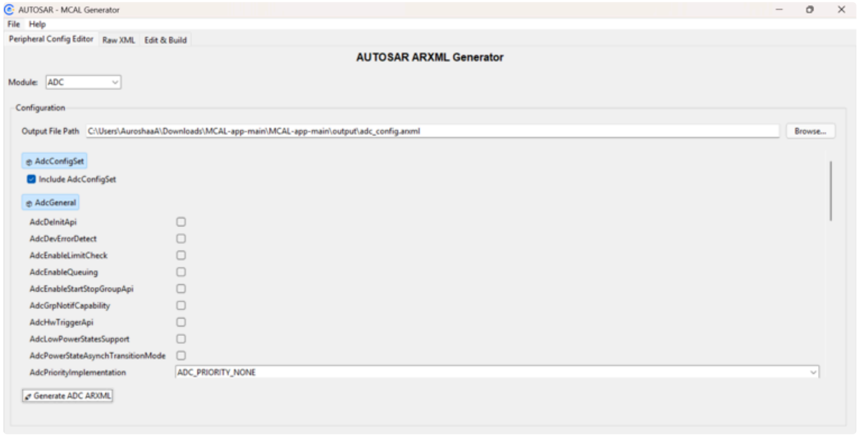
Raw XML

- Upload .arxml file
- It will populate the Raw XML.
- Save your modified arxml file (if done any modifications)

Edit and build

- Select the driver from dropdown.
- Browser the arxml which u generated and click “Parse ARXML” and view it in extracted configuration tab
- Need to adjust or modify anything can be done in the Generated tab
- Finally click on “Save driver files”
- Along with Driver files (.c and .h) the generated configuration file will also be downloaded.

Screenshots



AUTOSAR ARXML Generator

Module: ADC

Configuration

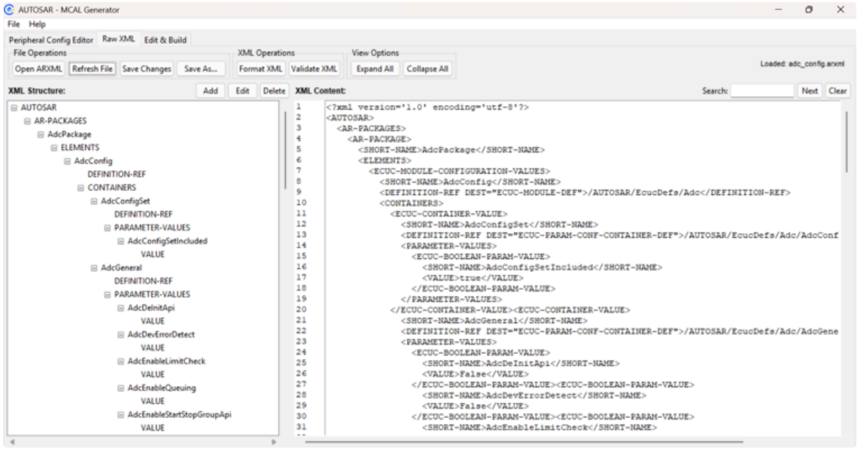
Output File Path: C:\Users\AuroshaaA\Downloads\MCAL-app-main\MCAL-app-main\output\adc_config.arxml

AdcConfigSet

☑ Include AdcConfigSet

AdcGeneral

- AdcDeInitApi ☐
- AdcDevErrorDetect ☐
- AdcEnableLimitCheck ☐
- AdcEnableQueuing ☐
- AdcEnableStartStopGroupApi ☐
- AdcGrpNotifCapability ☐
- AdcHwTriggerApi ☐
- AdcLowPowerStatesSupport ☐
- AdcPowerStateAsynchTransitionMode ☐
- AdcPriorityImplementation: ADC_PRIORITY_NONE



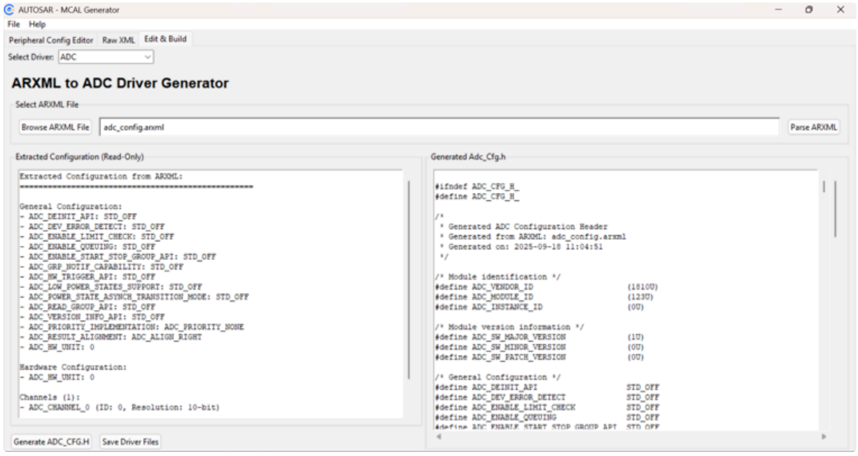
Loaded: adc_config.arxml

XML Structure:

```
⊟ AUTOSAR
  ⊟ AR-PACKAGES
    ⊟ AdcPackage
      ⊟ ELEMENTS
        ⊟ AdcConfig
            DEFINITION-REF
          ⊟ CONTAINERS
            ⊟ AdcConfigSet
                DEFINITION-REF
              ⊟ PARAMETER-VALUES
                ⊟ AdcConfigSetIncluded
                    VALUE
            ⊟ AdcGeneral
                DEFINITION-REF
              ⊟ PARAMETER-VALUES
                ⊟ AdcDeInitApi
                    VALUE
                ⊟ AdcDevErrorDetect
                    VALUE
                ⊟ AdcEnableLimitCheck
                    VALUE
                ⊟ AdcEnableQueuing
                    VALUE
                ⊟ AdcEnableStartStopGroupApi
                    VALUE
```

XML Content:

```xml
<?xml version='1.0' encoding='utf-8'?>
<AUTOSAR>
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>AdcPackage</SHORT-NAME>
      <ELEMENTS>
        <ECUC-MODULE-CONFIGURATION-VALUES>
          <SHORT-NAME>AdcConfig</SHORT-NAME>
          <DEFINITION-REF DEST="ECUC-MODULE-DEF">/AUTOSAR/EcucDefs/Adc</DEFINITION-REF>
          <CONTAINERS>
            <ECUC-CONTAINER-VALUE>
              <SHORT-NAME>AdcConfigSet</SHORT-NAME>
              <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/EcucDefs/Adc/AdcConf
              <PARAMETER-VALUES>
                <ECUC-BOOLEAN-PARAM-VALUE>
                  <SHORT-NAME>AdcConfigSetIncluded</SHORT-NAME>
                  <VALUE>true</VALUE>
                </ECUC-BOOLEAN-PARAM-VALUE>
              </PARAMETER-VALUES>
            </ECUC-CONTAINER-VALUE><ECUC-CONTAINER-VALUE>
              <SHORT-NAME>AdcGeneral</SHORT-NAME>
              <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR/EcucDefs/Adc/AdcGene
              <PARAMETER-VALUES>
                <ECUC-BOOLEAN-PARAM-VALUE>
                  <SHORT-NAME>AdcDeInitApi</SHORT-NAME>
                  <VALUE>False</VALUE>
                </ECUC-BOOLEAN-PARAM-VALUE><ECUC-BOOLEAN-PARAM-VALUE>
                  <SHORT-NAME>AdcDevErrorDetect</SHORT-NAME>
                  <VALUE>False</VALUE>
                </ECUC-BOOLEAN-PARAM-VALUE><ECUC-BOOLEAN-PARAM-VALUE>
                  <SHORT-NAME>AdcEnableLimitCheck</SHORT-NAME>
```



Select Driver: ADC

adc_config.arxml

```
Extracted Configuration from ARXML:
==================================================

General Configuration:
- ADC_DEINIT_API: STD_OFF
- ADC_DEV_ERROR_DETECT: STD_OFF
- ADC_ENABLE_LIMIT_CHECK: STD_OFF
- ADC_ENABLE_QUEUING: STD_OFF
- ADC_ENABLE_START_STOP_GROUP_API: STD_OFF
- ADC_GRP_NOTIF_CAPABILITY: STD_OFF
- ADC_HW_TRIGGER_API: STD_OFF
- ADC_LOW_POWER_STATES_SUPPORT: STD_OFF
- ADC_POWER_STATE_ASYNCH_TRANSITION_MODE: STD_OFF
- ADC_READ_GROUP_API: STD_OFF
- ADC_VERSION_INFO_API: STD_OFF
- ADC_PRIORITY_IMPLEMENTATION: ADC_PRIORITY_NONE
- ADC_RESULT_ALIGNMENT: ADC_ALIGN_RIGHT
- ADC_HW_UNIT: 0

Hardware Configuration:
- ADC_HW_UNIT: 0

Channels (1):
- ADC_CHANNEL_0 (ID: 0, Resolution: 10-bit)
```

```
#ifndef ADC_CFG_H_
#define ADC_CFG_H_

/*
 * Generated ADC Configuration Header
 * Generated from ARXML: adc_config.arxml
 * Generated on: 2025-09-18 11:04:51
 */

/* Module identification */
#define ADC_VENDOR_ID                     (1810U)
#define ADC_MODULE_ID                     (123U)
#define ADC_INSTANCE_ID                   (0U)

/* Module version information */
#define ADC_SW_MAJOR_VERSION              (1U)
#define ADC_SW_MINOR_VERSION              (0U)
#define ADC_SW_PATCH_VERSION              (0U)

/* General Configuration */
#define ADC_DEINIT_API                    STD_OFF
#define ADC_DEV_ERROR_DETECT              STD_OFF
#define ADC_ENABLE_LIMIT_CHECK            STD_OFF
#define ADC_ENABLE_QUEUING                STD_OFF
#define ADC_ENABLE_START_STOP_GROUP_API   STD_OFF
```

USER DOCUMENT

ARXML structure

- **ARXML** = AUTOSAR XML file, used to describe ECU configuration, system descriptions, software components, etc.
- In your case, it's specifically for **ECU Configuration (ECUC)** → DIO (Digital Input/Output) module.

Your exporter generates a **simplified AUTOSAR-compliant ECUC configuration** for the DIO module.

---

1. AUTOSAR Root

```
<AUTOSAR>
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>DioPackage</SHORT-NAME>
      <ELEMENTS>
        <ECUC-MODULE-CONFIGURATION-VALUES> ... </ECUC-MODULE-CONFIGURATION-VALUES>
      </ELEMENTS>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AUTOSAR>

```

- `AUTOSAR` : Root element for the file.
- `AR-PACKAGES` : Container for multiple packages.
- `AR-PACKAGE` : Holds related configuration data.
- `SHORT-NAME` : Unique name identifier (here `DioPackage`).
- `ELEMENTS` : Holds ECUC configuration elements.

---

2. ECUC Module Configuration

```
<ECUC-MODULE-CONFIGURATION-VALUES>
  <SHORT-NAME>DioConfigName</SHORT-NAME>
  <DEFINITION-REF DEST="ECUC-MODULE-DEF">/AUTOSAR/EcucDefs/Dio</DEFINITION-REF>
  <CONTAINERS> ... </CONTAINERS>
</ECUC-MODULE-CONFIGURATION-VALUES>

```

- `ECUC-MODULE-CONFIGURATION-VALUES` : Top-level ECU configuration for one module.
- `SHORT-NAME` : User-defined config name (from `model.DioConfig.DioConfigShortName`).

- `DEFINITION-REF` : References AUTOSAR DIO definition (`/AUTOSAR/EcucDefs/Dio`).
- `CONTAINERS` : Holds parameter containers.

---

3. ECUC Containers (Logical Groupings)

Example: DioConfigSet

```
<ECUC-CONTAINER-VALUE>
  <SHORT-NAME>DioConfigSet</SHORT-NAME>
  <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">
    /AUTOSAR/EcucDefs/Dio/DioConfigSet
  </DEFINITION-REF>
  <PARAMETER-VALUES>
    <ECUC-BOOLEAN-PARAM-VALUE>
      <SHORT-NAME>DioConfigSetIncluded</SHORT-NAME>
      <VALUE>true</VALUE>
    </ECUC-BOOLEAN-PARAM-VALUE>
  </PARAMETER-VALUES>
</ECUC-CONTAINER-VALUE>

```

- `ECUC-CONTAINER-VALUE` : Represents one configuration container.
- `SHORT-NAME` : Name of this container (e.g., `DioConfigSet`).
- `DEFINITION-REF` : Reference to AUTOSAR definition path.
- `PARAMETER-VALUES` : Holds values for parameters.

---

4. Parameter Types inside Containers

Your exporter maps Python object fields (`bool` , `int` , `string`) to AUTOSAR param types:

Boolean Example

```
<ECUC-BOOLEAN-PARAM-VALUE>
  <SHORT-NAME>DioSafety</SHORT-NAME>
  <VALUE>true</VALUE>
</ECUC-BOOLEAN-PARAM-VALUE>

```

Numerical Example

```
<ECUC-NUMERICAL-PARAM-VALUE>
  <SHORT-NAME>DioMaxPortNum</SHORT-NAME>
  <VALUE>32</VALUE>
</ECUC-NUMERICAL-PARAM-VALUE>

```

Textual Example

```
<ECUC-TEXTUAL-PARAM-VALUE>
  <SHORT-NAME>DioDriverVendor</SHORT-NAME>
  <VALUE>ABC_Semiconductors</VALUE>
</ECUC-TEXTUAL-PARAM-VALUE>

```

Here’s a list of **all ARXML tags** your code generates:

- **Top-Level**
  - `<AUTOSAR>`
  - `<AR-PACKAGES>`
  - `<AR-PACKAGE>`

    - `<SHORT-NAME>`
    - `<ELEMENTS>`
- **Module Config**
    - `<ECUC-MODULE-CONFIGURATION-VALUES>`
    - `<DEFINITION-REF DEST="ECUC-MODULE-DEF">`
    - `<CONTAINERS>`
- **Containers**
    - `<ECUC-CONTAINER-VALUE>`
    - `<SHORT-NAME>`
    - `<DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">`
    - `<PARAMETER-VALUES>`
- **Parameter Types**
    - `<ECUC-BOOLEAN-PARAM-VALUE>`
    - `<ECUC-NUMERICAL-PARAM-VALUE>`
    - `<ECUC-TEXTUAL-PARAM-VALUE>`
    - Inside each:
        - `<SHORT-NAME>`
        - `<VALUE>`

Definition

DIO

Type Definitions

Type Name	Description
`Dio_ChannelType`	Numeric identifier of a single DIO channel (pin). Used to specify an individual pin in a port.
`Dio_PortType`	Numeric identifier of a DIO port (group of pins). Each port contains multiple channels.
`Dio_LevelType`	Represents the logical level of a channel: `STD_LOW (0)` or `STD_HIGH (1)`.
`Dio_PortLevelType`	Represents the combined levels (bit values) of all channels in a port. Each bit corresponds to a channel’s logic level.

`Dio_ChannelGroupType`	Structure defining a group of adjacent channels within a port. It contains: • `mask` – bitmask for selecting pins • `offset` – start position • `port` – port identifier.

Function Prototypes

Service Name	Service ID (hex)	Prototype	Parameters (in)	Description
`Dio_ReadChannel`	`0x00`	`Dio_LevelType Dio_ReadChannel(Dio_ChannelType ChannelId)`	`ChannelId` → ID of the channel to read	Reads the level (HIGH/LOW) of a single DIO channel (pin).
`Dio_WriteChannel`	`0x01`	`void Dio_WriteChannel(Dio_ChannelType ChannelId, Dio_LevelType Level)`	`ChannelId` → ID of the channel `Level` → Value to write (HIGH/LOW)	Writes a logical value (HIGH/LOW) to a single DIO channel.
`Dio_ReadPort`	`0x02`	`Dio_PortLevelType Dio_ReadPort(Dio_PortType PortId)`	`PortId` → ID of the port	Reads the level of all channels (pins) in a port and returns the combined value.
`Dio_WritePort`	`0x03`	`void Dio_WritePort(Dio_PortType PortId, Dio_PortLev`	`PortId` → ID of the port `Level` → Bit pattern to write to the port	Writes a bit pattern to all pins in a port simultaneously.

		`elType Level)`		
`Dio_ReadChannelGroup`	`0x04`	`Dio_PortLevelType Dio_ReadChannelGroup(const Dio_ChannelGroupType* ChannelGroupIdPtr)`	`ChannelGroupIdPtr` → Defines port, mask, offset	Reads the level of a group of adjacent pins in a port.
`Dio_WriteChannelGroup`	`0x05`	`void Dio_WriteChannelGroup(const Dio_ChannelGroupType* ChannelGroupIdPtr, Dio_PortLevelType Level)`	`ChannelGroupIdPtr` → Defines port, mask, offset `Level` → Value to write	Writes a value to a group of adjacent pins in a port without affecting others.
`Dio_GetVersionInfo`	`0x12`	`void Dio_GetVersionInfo(Std_VersionInfoType* versioninfo)`	`versioninfo` → Pointer to structure for version information	Returns version information of the DIO module (vendor ID, module ID, SW version).
`Dio_FlipChannel`	`0x11`	`Dio_LevelType Dio_FlipChannel(Dio_ChannelType ChannelId)`	`ChannelId` → ID of the channel to toggle	Toggles the level of a channel and returns the new value.

`Dio_MaskedWritePort`	`0x0A`	`void Dio_MaskedWritePort(Dio_PortType PortId, Dio_PortLevelType Level, Dio_PortLevelType Mask);`	• **PortId**: Identifier of the DIO port where bits will be modified. • **Level**: Value to be written to the selected masked bits. • **Mask**: Bitmask that specifies which pins in the port should be updated. Other pins remain unchanged.	This service writes a given logical value (`Level`) to specific pins of a port, as selected by a bitmask (`Mask`). Only the pins indicated in the mask are updated, while all other pins in the port retain their current state. This allows partial and controlled modification of a port without affecting unrelated channels.

Configuration Parameters

Field	Details
DioDevErrorDetect	Enables/Disables Development Error Detection (`boolean`). If enabled, runtime parameter checks are performed and errors are reported to DET.
DioFlipChannelApi	Enables/Disables availability of the `Dio_FlipChannel` API (`boolean`).
DioMaskedWritePortApi	Enables/Disables availability of the `Dio_MaskedWritePort` API

	(`boolean`).
DioVersionInfoApi	Enables/Disables availability of the `Dio_GetVersionInfo` API (`boolean`).
DioPortId	Identifier type for DIO Ports (e.g., `GPIOA` , `GPIOB`).
DioChannelId	Identifier type for DIO Channels (individual pins within a port).
DioChannelGroupIdentification	Identifies a group of adjacent pins within a port, usually defined by a mask, offset, and port reference.
DioPortMask	Bitmask used to define which pins belong to a `ChannelGroup` . Each set bit indicates membership in the group.
DioPortOffset	Starting bit position of the channel group within a port. Determines how the group aligns within the port's bitfield.

ADC

Type Definitions

Field	Details
Adc_ConfigType	Structure containing all configuration parameters for the ADC driver (channels, groups, resolution, timings, etc.).
Adc_ChannelType	Numeric identifier of a single ADC channel (e.g., mapped to a physical analog input pin).
Adc_GroupType	Identifier type for a logical group of ADC channels that are converted together based on configuration.
Adc_ValueGroupType	Data type to hold the digital conversion result of an ADC channel group (typically `uint16`).

Adc_PrescaleType	Defines the ADC clock prescaler (division factor applied to the system clock to derive ADC clock).
Adc_ConversionTimeType	Represents the time required to complete a single conversion, typically dependent on clock and resolution settings.
Adc_SamplingTimeType	Defines how long the ADC samples the input voltage before conversion begins.
Adc_ResolutionType	Defines resolution of conversion (e.g., 8-bit, 10-bit, 12-bit), i.e., how many discrete digital values are possible.
Adc_StatusType	Indicates current status of the ADC driver or a conversion group (e.g., idle, busy, completed).
Adc_TriggerSourceType	Specifies the trigger source for conversions: software trigger or hardware trigger (like timer or external event).
Adc_GroupConvModeType	Defines conversion mode of a group: one-shot (single conversion) or continuous.
Adc_GroupPriorityType	Defines priority level of an ADC group, used when multiple groups are scheduled.
Adc_GroupDefType	Defines how channels are assigned to an ADC group (e.g., ordered list of channels).
Adc_StreamNumSampleType	Defines the number of samples to be taken in streaming mode (e.g., 1, N, or infinite).
Adc_StreamBufferModeType	Defines how ADC results are stored in streaming: linear buffer (stop at end) or circular buffer (overwrite oldest).
Adc_GroupAccessModeType	Defines access type for group conversion results: single access or streaming access.

Adc_HwTriggerSignalType	Defines hardware trigger signal polarity/edge (e.g., rising edge, falling edge, both edges).
Adc_HwTriggerTimerType	Defines the timer resource used as trigger source for hardware-triggered conversions.
Adc_PriorityImplementationType	Defines how group priorities are handled by the ADC driver (preemptive or non-preemptive scheduling).
Adc_GroupReplacementType	Defines whether an ongoing group conversion can be replaced by another (e.g., cancel current and start new, or wait until finished).
Adc_ChannelRangeSelectType	Defines the selectable input range for channels (e.g., 0–5V, 0–3.3V), depending on ADC hardware support.
Adc_ResultAlignmentType	Defines alignment of result data in registers: left-aligned (MSB justified) or right-aligned (LSB justified).
Adc_PowerStateType	Defines possible power states of the ADC (e.g., normal, sleep, standby).
Adc_PowerStateRequestResultType	Defines the result of a power state change request (e.g., success, not supported, busy).

Function Prototypes

Service Name	Service ID (hex)	Prototype	Parameters (in)	Description
Adc_Init	0x00	`void Adc_Init(const Adc_ConfigType* ConfigPtr)`	`ConfigPtr` – Pointer to ADC configuration structure	Initializes the ADC driver, sets up channels, groups, resolution, timings, and hardware resources.

Adc_SetupResultBuffer	0x01	`Std_ReturnType Adc_SetupResultBuffer(Adc_GroupType Group, Adc_ValueGroupType* DataBufferPtr)`	`Group` – Group ID, `DataBufferPtr` – Pointer to buffer	Assigns a result buffer where conversion results for the given group will be stored.
Adc_DeInit	0x02	`void Adc_DeInit(void)`	None	Deinitializes ADC hardware and resets driver state.
Adc_StartGroupConversion	0x03	`void Adc_StartGroupConversion(Adc_GroupType Group)`	`Group` – Group identifier	Starts conversion of all channels assigned to the specified group.
Adc_StopGroupConversion	0x04	`void Adc_StopGroupConversion(Adc_GroupType Group)`	`Group` – Group identifier	Stops ongoing conversion of the specified group.
Adc_ReadGroup	0x05	`Std_ReturnType Adc_ReadGroup(Adc_GroupT`	`Group` – Group identifier, `DataBufferPtr` –	Reads the most recent conversion results of a group into

		`ype Group, Adc_ValueGroupType* DataBufferPtr)`	Pointer to store results	the provided buffer.
Adc_EnableHardwareTrigger	0x06	`void Adc_EnableHardwareTrigger(Adc_GroupType Group)`	`Group` – Group identifier	Enables hardware triggering for the specified group.
Adc_DisableHardwareTrigger	0x07	`void Adc_DisableHardwareTrigger(Adc_GroupType Group)`	`Group` – Group identifier	Disables hardware triggering for the specified group.
Adc_EnableGroupNotification	0x08	`void Adc_EnableGroupNotification(Adc_GroupType Group)`	`Group` – Group identifier	Enables end-of-conversion notifications for the specified group.
Adc_DisableGroupNotification	0x09	`void Adc_DisableGroupNotification(Adc_GroupTy`	`Group` – Group identifier	Disables notifications for the specified group.

		`pe Group)`		
Adc_GetGroupStatus	0x0A	`Adc_StatusType Adc_GetGroupStatus(Adc_GroupType Group)`	`Group` – Group identifier	Returns the current status of the specified group (e.g., idle, busy, completed).
Adc_GetStreamLastPointer	0x0B	`Adc_ValueGroupType* Adc_GetStreamLastPointer(Adc_GroupType Group, Adc_StreamNumSampleType* PtrToSample)`	`Group` – Group identifier, `PtrToSample` – Pointer to last sample number	Returns a pointer to the buffer position of the last completed conversion in streaming mode.
Adc_GetVersionInfo	0x0C	`void Adc_GetVersionInfo(Std_VersionInfoType* versioninfo)`	`versioninfo` – Pointer to store version info	Returns version information of the ADC driver.
Adc_SetPowerState	0x0D	`Std_ReturnType Adc_SetPowerState(Adc_Po`	`Result` – Pointer to result of request	Requests ADC driver to switch to a target power state.

		`werStateRequestResultType* Result)`		
Adc_GetCurrentPowerState	0x0E	`Std_ReturnType Adc_GetCurrentPowerState(Adc_PowerStateType* CurrentState)`	`CurrentState` – Pointer to store current state	Returns the current power state of the ADC driver.
Adc_GetTargetPowerState	0x0F	`Std_ReturnType Adc_GetTargetPowerState(Adc_PowerStateType* TargetState)`	`TargetState` – Pointer to store target state	Returns the power state that ADC driver is requested to transition to.
Adc_PreparePowerState	0x10	`Std_ReturnType Adc_PreparePowerState(Adc_PowerStateType PowerState, Adc_PowerStateRe`	`PowerState` – Target state, `Result` – Pointer to result	Prepares ADC driver to transition into a requested power state.

		`questResultType* Result)`			
Adc_Main_PowerTransitionManager	0x11	`void Adc_Main_PowerTransitionManager(void)`	None	Manages ADC power state transitions in the main function loop.	

Configuration Parameters

General Container: `ADCGENERAL`

Parameter	Details
AdcDeInitApi	Enables/disables availability of `Adc_DeInit` API (boolean).
AdcDevErrorDetect	Enables/disables Development Error Detection (boolean). Reports to DET if enabled.
AdcEnableLimitCheck	Enables/disables runtime limit check functionality for channels (boolean).
AdcEnableQueuing	Enables/disables queuing of group conversion requests (boolean).
AdcEnableStartStopGroupApi	Enables/disables availability of Start/Stop group conversion APIs (boolean).
AdcGrpNotifCapability	Enables/disables group notification callbacks (boolean).
AdcHwTriggerApi	Enables/disables hardware trigger API support (boolean).
AdcLowPowerStatesSupport	Enables/disables low-power mode support (boolean).
AdcPowerStateAsynchTransitionMode	Defines whether power state transitions are handled asynchronously (boolean).
AdcReadGroupApi	Enables/disables `Adc_ReadGroup` API

	(boolean).
AdcVersionInfoApi	Enables/disables `Adc_GetVersionInfo` API (boolean).
AdcPriorityImplementation	Defines how group priorities are implemented (e.g., preemptive, non-preemptive).
AdcResultAlignment	Defines alignment of conversion result in register (left or right aligned).

Container: `AdcConfigSet`

Parameter	Details
AdcHwUnit	References ADC hardware units that belong to this configuration set.

Container: `AdcChannel`

Parameter	Details
AdcChannelConvTime	Conversion time for this channel (depends on resolution and clock).
AdcChannelHighLimit	Upper threshold value for limit check.
AdcChannelId	Unique identifier for the channel.
AdcChannelLimitCheck	Enables/disables limit checking for this channel (boolean).
AdcChannelLowLimit	Lower threshold value for limit check.
AdcChannelRangeSelect	Selectable input voltage range for the channel.
AdcChannelRefVoltsrcHigh	High reference voltage source for the channel.
AdcChannelRefVoltsrcLow	Low reference voltage source for the channel.
AdcChannelResolution	Resolution for this channel (e.g., 8, 10, 12 bits).

AdcChannelSampTime	Sampling time for the channel (time input is sampled before conversion).

Container: `AdcGroup`

Parameter	Details
AdcGroupAccessMode	Access type for results (single, streaming).
AdcGroupConversionMode	Conversion mode (one-shot or continuous).
AdcGroupId	Identifier for the group.
AdcGroupPriority	Priority assigned to this group.
AdcGroupReplacement	Whether ongoing conversion can be replaced by another request.
AdcGroupTriggSrc	Trigger source for the group (software, hardware).
AdcHwTrigSignal	Defines hardware trigger signal polarity/edge.
AdcHwTrigTimer	Timer resource used as trigger for this group.
AdcNotification	Callback function invoked after conversion completion.
AdcStreamingBufferMode	Streaming mode buffer type (linear/circular).
AdcStreamingNumSamples	Number of samples to be taken in streaming mode.
AdcGroupDefinition	List of channels that form this group.

Container: `AdcPublishedInformation`

Parameter	Details
AdcChannelValueSigned	Indicates if ADC channel results are signed/unsigned.
AdcGroupFirstChannelFixed	Defines whether group’s first channel is fixed in sequence.

AdcMaxChannelResolution	Maximum resolution supported by hardware.
AdcPowerStateConfig	Configuration for power states supported by ADC.
AdcPowerState	List of possible power states (e.g., normal, sleep).
AdcPowerStateReadyCbkRef	Reference to callback function executed when power state is ready.

Container: `AdcHwUnit`

Parameter	Details
AdcClockSource	Clock source used for ADC hardware unit.
AdcHwUnitId	Identifier for the ADC hardware unit.
AdcPrescale	Prescaler value used to derive ADC clock from system clock.

CAN

Type Definitions

Type Definition	Description	Possible Values (Dropdowns)
Can_ConfigType	Holds the overall configuration structure for CAN, including controller, baud rate, filters, and hardware mapping.	N/A
Can_PduType	Defines the structure of a Protocol Data Unit (PDU) for CAN transmission. It includes CAN ID, length, data pointer, and handle ID.	N/A
Can_IdType	Defines the type of CAN Identifier (11-bit	N/A

	standard or 29-bit extended).	
Can_HwHandleType	Defines a type for hardware object handles (e.g., Tx/Rx mailbox identifiers).	N/A
Can_HwType	Structure representing a specific CAN hardware object, including controller ID, handle type, and CAN ID.	N/A
Can_ErrorStateType	Represents the **error state** of the CAN controller (fault confinement states).	• `CAN_ERRORSTATE_ACTIVE` → Active state (normal operation) - `CAN_ERRORSTATE_PASSIVE` → Passive state (restricted transmission) - `CAN_ERRORSTATE_BUSOFF` → Bus-off state (controller disconnected from bus)
Can_ControllerStateType	Represents the **state** of the CAN controller (initialization and runtime states).	• `CAN_CS_UNINIT` → Controller not initialized - `CAN_CS_STARTED` → Controller actively transmitting/receiving - `CAN_CS_STOPPED` → Controller stopped - `CAN_CS_SLEEP` → Controller in low-power sleep mode

Can_ErrorType	Enumerates **error types** that may occur during CAN communication.	• `CAN_ERROR_BIT_MONITORING1` - `CAN_ERROR_BIT_MONITORING0` - `CAN_ERROR_BIT` - `CAN_ERROR_CHECK_ACK_FAILED` - `CAN_ERROR_ACK_DELIMITER` - `CAN_ERROR_ARBITRATION_LOST` - `CAN_ERROR_OVERLOAD` - `CAN_ERROR_CHECK_FORM_FAILED` - `CAN_ERROR_CHECK_STUFFING_FAILED` - `CAN_ERROR_CHECK_CRC_FAILED` - `CAN_ERROR_BUS_LOCK`
Can_TimeStampType	Defines a type for storing CAN message **timestamps**, usually in ticks or microseconds (depending on hardware).	N/A

Function Prototypes

Service Name	Service ID (Hex)	Prototype	Parameters (in)	Description
Can_Init	`0x00`	`void Can_Init(const Can_ConfigType* Config)`	`Config` → Pointer to CAN configuration set	Initializes the CAN driver and all controllers with the given configuration.
Can_GetVersionInfo	`0x07`	`void Can_GetVersionInfo(Std_VersionInfoType* versioninfo)`	`versioninfo` → Pointer to version info structure	Returns version information of the CAN driver.
Can_DeInit	`0x01`	`void Can_DeInit(void)`	None	Deinitializes the CAN driver (resets controllers and frees resources).
Can_SetBaudrate	`0x0F`	`Std_ReturnType Can_SetBaudrate(uint8 Controller, uint16 BaudRateConfigID)`	`Controller` → Controller ID `BaudRateConfigID` → Baudrate config reference	Sets the baudrate of the CAN controller during runtime.
Can_SetControllerMode	`0x03`	`Std_ReturnType Can_SetC`	`Controller` → Controller ID	Changes the operation mode of a CAN

		`ontrollerMode(uint8 Controller, Can_ControllerStateType Transition)`	`Transition` → Requested mode	controller (Start, Stop, Sleep, Wakeup).
Can_DisableControllerInterrupts	`0x04`	`void Can_DisableControllerInterrupts(uint8 Controller)`	`Controller` → Controller ID	Disables interrupts for the given controller.
Can_EnableControllerInterrupts	`0x05`	`void Can_EnableControllerInterrupts(uint8 Controller)`	`Controller` → Controller ID	Enables interrupts for the given controller.
Can_CheckWakeup	`0x0A`	`Std_ReturnType Can_CheckWakeup(uint8 Controller)`	`Controller` → Controller ID	Checks if a wakeup occurred for the CAN controller.
Can_GetControllerErrorState	`0x11`	`Std_ReturnType Can_GetC`	`Controller` → Controller ID	Gets the error state (Active, Passive,

		`ontrollerErrorState(uint8 Controller, Can_ErrorStateType* ErrorState)`	`ErrorState` → Pointer to error state variable	Bus-off) of the CAN controller.
Can_GetControllerMode	`0x12`	`Std_ReturnType Can_GetControllerMode(uint8 Controller, Can_ControllerStateType* ControllerModePtr)`	`Controller` → Controller ID `ControllerModePtr` → Pointer to mode variable	Returns the current mode (Uninit, Started, Stopped, Sleep) of the controller.
Can_GetControllerRxErrorCounter	`0x13`	`Std_ReturnType Can_GetControllerRxErrorCounter(uint8 Controller, uint8* RxErrorCounter)`	`Controller` → Controller ID `RxErrorCounter` → Pointer to Rx error counter	Returns the receive error counter of the CAN controller.

Can_GetControllerTxErrorCounter	`0x14`	`Std_ReturnType Can_GetControllerTxErrorCounter(uint8 Controller, uint8* TxErrorCounter)`	`Controller` → Controller ID `TxErrorCounter` → Pointer to Tx error counter	Returns the transmit error counter of the CAN controller.
Can_GetCurrentTime	`0x15`	`Std_ReturnType Can_GetCurrentTime(uint8 Controller, Can_TimeStampType* TimeStamp)`	`Controller` → Controller ID `TimeStamp` → Pointer to timestamp	Returns the current time for a CAN controller (used for time-triggered CAN).
Can_EnableEgressTimeStamp	`0x16`	`Std_ReturnType Can_EnableEgressTimeStamp(uint8 Controller, Can_HwHandleType Hoh)`	`Controller` → Controller ID `Hoh` → Hardware Object Handle	Enables timestamping for transmitted PDUs (egress).

Can_GetEgressTimeStamp	`0x17`	`Std_ReturnType Can_GetEgressTimeStamp(uint8 Controller, Can_HwHandleType Hoh, Can_TimeStampType* TimeStamp)`	`Controller` → Controller ID `Hoh` → Hardware Object Handle `TimeStamp` → Pointer to timestamp	Gets the egress timestamp for a transmitted CAN message.
Can_GetIngressTimeStamp	`0x18`	`Std_ReturnType Can_GetIngressTimeStamp(uint8 Controller, Can_HwHandleType Hoh, Can_TimeStampType* TimeStamp)`	`Controller` → Controller ID `Hoh` → Hardware Object Handle `TimeStamp` → Pointer to timestamp	Gets the ingress timestamp for a received CAN message.
Can_Write	`0x06`	`Std_ReturnType Can_Write(Can_HwHandleTy`	`Hth` → Hardware Transmit Handle `PduInfo`	Requests transmission of a CAN message.

		`pe Hth, const Can_PduType* PduInfo)`	→ Pointer to PDU structure	
Can_MainFunction_Write	`0x08`	`void Can_MainFunction_Write(void)`	None	Handles pending CAN transmit jobs (polling mode).
Can_MainFunction_Read	`0x09`	`void Can_MainFunction_Read(void)`	None	Handles pending CAN receive jobs (polling mode).
Can_MainFunction_BusOff	`0x0B`	`void Can_MainFunction_BusOff(void)`	None	Handles bus-off recovery tasks in polling mode.
Can_MainFunction_Wakeup	`0x0C`	`void Can_MainFunction_Wakeup(void)`	None	Handles wakeup events in polling mode.
Can_MainFunction_Mode	`0x0D`	`void Can_MainFunction_Mode(void)`	None	Handles mode transitions in polling mode.

Configuration Parameters

Container: Can

Field	Description

CanConfigSet	Enables or disables the CAN configuration set. Acts as the root container holding all other settings.
CanGeneral	Container for general CAN module settings such as error detection, baudrate API, global time, and main function periods.

Container: **CanGeneral**

Field	Description
CanDevErrorDetect	Enables development error detection for invalid API usage or incorrect parameters.
CanEnableSecurityEventReporting	Enables reporting of security-related events on the CAN network.
CanGlobalTimeSupport	Enables global time synchronization support for time-triggered communication.
CanIndex	Unique index identifying the CAN configuration.
CanLPduReceiveCalloutFunction	Callback function executed when a CAN LPDU is received.
CanMainFunctionBusoffPeriod	Periodic interval for executing bus-off recovery main function.
CanMainFunctionModePeriod	Periodic interval for executing mode handling main function.
CanMainFunctionWakeupPeriod	Periodic interval for executing wakeup main function.
CanMultiplexedTransmission	Configures support for multiplexed CAN message transmission.
CanSetBaudrateApi	Enables API to change controller baudrate at runtime.
CanTimeoutDuration	Timeout period for CAN operations like transmission or mode changes.
CanVersionInfoApi	Enables API to read the CAN driver version information.

Container: **CanController**

Field	**Description**
CanBusoffProcessing	Configures bus-off recovery handling: via interrupts or polling.
CanControllerActivation	Enables or disables the CAN controller.
CanControllerBaseAddress	Base memory address of the CAN controller registers.
CanControllerId	Unique identifier for the CAN controller.
CanHwPnSupport	Enables support for Partial Networking (PN) for selective ECU wakeup.
CanRxProcessing	Defines reception processing mode: interrupt, polling, or mixed.
CanTxProcessing	Defines transmission processing mode: interrupt, polling, or mixed.
CanWakeupProcessing	Wakeup event handling mode: interrupt or polling.
CanWakeupSupport	Enables detection and processing of wakeup events.

Container: CanControllerBaudrateConfig

Field	**Description**
CanControllerBaudRate	Nominal baudrate of the CAN controller.
CanControllerBaudRateConfigID	Identifier for specific baudrate configuration.
CanControllerPropSeg	Propagation segment time used for bit timing.
CanControllerSeg1	Phase segment 1 time.
CanControllerSeg2	Phase segment 2 time.
CanControllerSyncJumpWidth	Maximum allowed resynchronization adjustment.

Container: CanControllerFdBaudrateConfig

Field	Description
CanControllerFdBaudRate	Data phase baudrate for CAN FD controller.
CanControllerPropSeg	Propagation segment time for CAN FD.
CanControllerSeg1	Phase segment 1 for CAN FD.
CanControllerSeg2	Phase segment 2 for CAN FD.
CanControllerSyncJumpWidth	Synchronization jump width for CAN FD.
CanControllerSspOffset	Enables sample point offset for CAN FD.
CanControllerTxBitRateSwitch	Enables bit rate switching for CAN FD transmissions.

Container: **CanPartialNetwork**

Field	Description
CanPnEnabled	Enables Partial Networking for selective wakeup.
CanPnFrameCanId	CAN ID of the PN frame used for wakeup or communication.
CanPnFrameCanIdMask	Mask applied to filter relevant PN frames.
CanPnFrameDlc	Data length of the PN frame.

Container: CanPnFrameDataMaskSpec

Field	Description
CanPnFrameDataMask	Specifies relevant bits of the PN frame for wakeup detection.
CanPnFrameDataMaskIndex	Index of the PN frame data mask in the PN configuration list.

Container: CanHardwareObject

Field	Description
CanFdPaddingValue	Value used to pad CAN FD frames when data length is less than maximum.
CanHandleType	Defines whether the hardware object is basic or full CAN.
CanHardwareObjectUsesPolling	Indicates if this hardware object uses polling.
CanHwObjectCount	Number of hardware objects assigned to the controller.
CanIdType	Identifier type of CAN message: standard, extended, or mixed.
CanObjectId	Unique ID of the hardware object.
CanObjectPayloadLength	Length of the CAN object payload.
CanObjectType	Type of object: receive or transmit.
CanTriggerTransmitEnable	Enables trigger transmit for this hardware object.

Container: CanHwFilter

Field	Description
CanHwFilterCode	Filter code applied to received messages.
CanHwFilterMask	Filter mask to select relevant bits for acceptance filtering.

Container: **CanMainFunctionRWPeriods**

Field	Description
CanMainFunctionPeriod	Defines main function periodicity for read/write operations.

Container: CanTTController

Field	Description

CanTTControllerApplWatchdogLimit	Maximum allowed time for application watchdog.
CanTTControllerCycleCountMax	Maximum cycle count in time-triggered communication.
CanTTControllerExpectedTxTrigger	Expected transmit trigger time.
CanTTControllerExternalClockSynchronisation	Enables synchronization with external clock.
CanTTControllerGlobalTimeFiltering	Enables filtering of messages based on global time.
CanTTControllerInitialRefOffset	Initial reference offset in time-triggered cycle.
CanTTControllerInterruptEnable	Enables interrupts for TT controller events.
CanTTControllerLevel2	Enables Level 2 time-triggered features.
CanTTControllerNTUConfig	Configures network time unit parameters.
CanTTControllerOperationMode	Operation mode: event sync time-triggered, event triggered, time triggered.
CanTTControllerSyncDeviation	Allowed deviation in time synchronization.
CanTTControllerTimeMaster	Enables this controller as time master.
CanTTControllerTimeMasterPriority	Priority level of the time master controller.
CanTTControllerTURRestore	Enables restoration of TT schedules after communication disruptions.
CanTTControllerTxEnableWindowLength	Duration of allowed transmit window in TT cycle.
CanTTControllerWatchTriggerGapTimeMark	Watch trigger gap time marker.
CanTTControllerWatchTriggerTimeMark	Watch trigger time marker.
CanTTIRQProcessing	TT controller interrupt processing mode: interrupt or polling.

Container: CanTTHardwareObjectTrigger

Field	Description
CanTTHardwareObjectBaseCycle	Base cycle number for triggering this hardware object.
CanTTHardwareObjectCycleRepetition	Number of cycles after which trigger repeats.
CanTTHardwareObjectTimeMark	Time mark within cycle for triggering.
CanTTHardwareObjectTriggerId	Unique ID for the TT hardware object trigger.
CanTTHardwareObjectTriggerType	Type of TT trigger: RX, TX reference, TX gap, TX exclusive, merged, or single.

Container: CanXLGeneral

Field	Description
CanXLEthGlobalTimeSupport	Enables global time support for CAN XL Ethernet communication.

Container: **CanXLController**

Field	Description
CanXLCtrlEthDefaultPriority	Default priority for Ethernet CAN XL messages.
CanXLEthDefaultQueue	Default Ethernet queue used by CAN XL controller.
CanXLEthPhysAddress	Physical address of CAN XL Ethernet interface.

Container: **CanXLHardwareObject**

Field	Description
CanXLObjectId	Unique ID for the hardware object in CAN XL controller.

CanXLHwFilter	Enables or disables hardware filtering for CAN XL messages.

Container: **CanXLBaudrateConfig**

Field	Description
CanXLBaudRate	Nominal baud rate for CAN XL controller.
CanXLErrorSignaling	Enables error signaling.
CanXLPropSeg	Propagation segment time.
CanXLPwmL	PWM low time.
CanXLPwmO	PWM offset time.
CanXLPwmS	PWM sampling time.
CanXLSeg1	Phase segment 1 time.
CanXLSeg2	Phase segment 2 time.
CanXLSspOffset	Sample point offset.
CanXLSyncJumpWidth	Synchronization jump width.
CanXLTrcvPwmMode	Enables PWM mode for transceiver.

Container: **CanXLEthEgressFifo**

Field	Description
CanXLEthIngressFifoCanXLQueue	Queue index for ingress FIFO.
CanXLEthIngressFifoIdx	FIFO index in Ethernet interface.
CanXLEthIngressFifoVcid	Virtual channel ID associated with ingress FIFO.

SPI

Configuration Parameters

Container: **SpiDemEventParameterRefs**

Field	Description
SPI_E_HARDWARE_ERROR	Enables reporting of hardware errors via the DEM (Diagnostic

	Event Manager).

Container: **SpiGeneral**

Field	Description
SpiCancelApi	Enables API to cancel ongoing SPI transmissions.
SpiChannelBuffersAllowed	Defines the maximum number of buffers allowed per SPI channel.
SpiDevErrorDetect	Enables development error detection for invalid API calls or parameters.
SpiHwStatusApi	Enables API to query the hardware status of SPI channels or jobs.
SpiInterruptibleSeqAllowed	Enables support for interruptible sequences, allowing preemption.
SpiLevelDelivered	Sets the default delivered level for SPI signals.
SpiMainFunctionPeriod	Periodicity for executing the SPI main function.
SpiSupportConcurrentSyncTransmit	Enables support for concurrent synchronous transmissions.
SpiVersionInfoApi	Enables the API to retrieve the SPI driver version information.

Container: **SpiSequence**

Field	Description
SpiInterruptibleSequence	Marks the sequence as interruptible.
SpiSeqEndNotification	Enables notification when the sequence ends.
SpiSequenceId	Unique identifier for the sequence.
SpiJobAssignment	Indicates assignment of jobs to this sequence.

Container: **SpiChannel**

Field	Description
SpiChannelId	Unique identifier for the SPI channel.
SpiChannelType	Indicates the type of SPI channel (e.g., master/slave).
SpiDataWidth	Width of data per SPI transfer.
SpiDefaultData	Default data transmitted when no actual data is available.
SpiEbMaxLength	Maximum length of the external buffer for the channel.
SpiIbNBuffers	Number of internal buffers available for the channel.
SpiTransferStart	Indicates whether the transfer should start automatically.

Container: **SpiChannelList**

Field	Description
SpiChannelIndex	Index of the channel in the list.
SpiChannelAssignment	Indicates whether the channel is assigned to a job or sequence.

Container: **SpiJob**

Field	Description
SpiJobEndNotification	Enables notification when a job ends.
SpiJobId	Unique identifier for the SPI job.
SpiJobPriority	Priority level of the SPI job.
SpiDeviceAssignment	Indicates which SPI device the job is assigned to.

Container: **SpiExternalDevice**

Field	Description

SpiBaudrate	Configures the communication baudrate for the external SPI device.
SpiCsIdentifier	User-defined identifier for the chip select (CS) signal.
SpiCsBehavior	Configures the CS behavior: keep asserted or toggle.
SpiCsPolarity	Polarity of CS signal: high or low active.
SpiCsSelection	Method for CS control: via GPIO or peripheral engine.
SpiDataShiftEdge	Defines whether data is sampled on leading or trailing edge of clock.
SpiEnableCs	Enables the chip select line.
SpiHwUnit	Hardware unit used for the SPI device (CSIB0–CSIB3).
SpiShiftClockIdleLevel	Idle level of the SPI clock: high or low.
SpiTimeClk2Cs	Delay from clock edge to CS activation.
SpiTimeCs2Clk	Delay from CS to clock edge.
SpiTimeCs2Cs	Minimum time between two consecutive CS activations.

Container: **SpiDriver**

Field	Description
SpiMaxChannel	Maximum number of SPI channels supported by the driver.
SpiMaxJob	Maximum number of jobs supported by the driver.
SpiMaxSequence	Maximum number of sequences supported by the driver.

GPT

Configuration Parameters

Container: **GptDriverConfiguration**

Field	Description
GptDevErrorDetect	Enables development error detection for invalid API calls or incorrect parameters.
GptPredefTimer100us32bitEnable	Enables predefined 100 µs 32-bit timer.
GptPredefTimer1usEnablingGrade	Configures enabling grade for predefined 1 µs timers: 16-bit, 16/24-bit, 16/24/32-bit, or disabled.
GptReportWakeupSource	Enables reporting of wakeup events detected by the GPT module.

Container: **GptClockReferencePoint**

Field	Description
GptClockReference	Reference source for GPT clock, used for synchronizing timers.

Container: **GptChannelConfigSet**

Field	Description
(Set value)	Numeric identifier for the channel configuration set.

Container: **GptChannelConfiguration**

Field	Description
GptChannelId	Unique identifier for the GPT channel.
GptChannelMode	Timer mode: Continuous (repeating) or One-shot.
GptChannelTickFrequency	Frequency of timer ticks for the channel.

GptChannelTickValueMax	Maximum tick value the timer can count.
GptEnableWakeup	Enables GPT channel to wake up the MCU from sleep mode.
GptNotification	User-defined callback function triggered on timer events.
GptChannelClkSrcRef	Reference source for the channel clock.

Container: **GptWakeupConfiguration**

Field	Description
GptWakeupSourceRef	Reference to a wakeup source associated with GPT channels.

Container: **GptConfigurationOfOptApiServices**

Field	Description
GptDeinitApi	Enables the API to deinitialize the GPT module.
GptEnableDisableNotificationApi	Enables the API to enable or disable notifications for timer events.
GptTimeElapsedApi	Enables the API to read elapsed time for a timer channel.
GptTimeRemainingApi	Enables the API to read remaining time for a timer channel.
GptVersionInfoApi	Enables the API to read GPT driver version information.
GptWakeupFunctionalityApi	Enables API support for GPT wakeup functionality.

WDG

Configuration Parameters

Container: **Wdg**

Field	Description

WdgSettingsConfig	Root container for Watchdog configuration settings. Holds all mode-specific settings.
WdgGeneral	General container for WDG module configuration and global settings.

Container: **WdgGeneral**

Field	Description
WdgDevErrorDetect	Enables development error detection for invalid API calls or incorrect parameters.
WdgDisableAllowed	Enables or disables the ability to stop or disable the watchdog at runtime.
WdgIndex	Unique index identifying the watchdog configuration instance.
WdgInitialTimeout	Initial timeout value for the watchdog timer.
WdgMaxTimeout	Maximum allowable timeout for the watchdog timer.
WdgRunArea	Defines the memory area where the WDG runs: RAM or ROM.
WdgVersionInfoApi	Enables the API to retrieve WDG driver version information.

Container: **WdgSettingsConfig**

Field	Description
WdgDefaultMode	Default operating mode of the watchdog: FAST, OFF, or SLOW.
WdgExternalConfiguration	Enables or disables external configuration of watchdog settings.
WdgSettingsFast	Enables configuration for FAST mode.
WdgSettingsOff	Enables configuration for OFF mode.

WdgSettingsSlow	Enables configuration for SLOW mode.

Container: **WdgPublishedInformation**

Field	Description
WdgTriggerMode	Defines how the watchdog is triggered: both edges (DG_BOTH), toggle, or window mode.
