RTI Bypass Blockset

MATLAB API Reference

For RTI Bypass Blockset 3.16

Release 2021-A – May 2021

dSPACE

How to Contact dSPACE

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

How to Contact dSPACE Support

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

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

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

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

Software Updates and Patches

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

Important Notice

This publication and the contents hereof are subject to change without notice.

Contents

Parameters 73

About this Reference

Contents

This reference gives you detailed information on the automation API of the RTI Bypass Blockset.

Required knowledge

Knowledge in handling the host PC and the Microsoft Windows operating system is presupposed. You should also be familiar with programming MATLAB.

Symbols

dSPACE user documentation uses the following symbols:

Symbol	Description
⚠ DANGER	Indicates a hazardous situation that, if not avoided, will result in death or serious injury.
⚠ WARNING	Indicates a hazardous situation that, if not avoided, could result in death or serious injury.
⚠ CAUTION	Indicates a hazardous situation that, if not avoided, could result in minor or moderate injury.
NOTICE	Indicates a hazard that, if not avoided, could result in property damage.
Note	Indicates important information that you should take into account to avoid malfunctions.
Tip	Indicates tips that can make your work easier.
⍰	Indicates a link that refers to a definition in the glossary, which you can find at the end of the document unless stated otherwise.
📖	Precedes the document title in a link that refers to another document.

Naming conventions

dSPACE user documentation uses the following naming conventions:

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

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

Examples:

- Where you find terms such as `rti<XXXX>` replace them by the RTI platform support you are using, for example, `rti1007`.
- Where you find terms such as `<model>` or `<submodel>` in this document, replace them by the actual name of your model or submodel. For example, if the name of your Simulink model is `smd_1007_sl.slx` and you are asked to edit the `<model>_usr.c` file, you actually have to edit the `smd_1007_sl_usr.c` file.

RTI block name conventions All I/O blocks have default names based on dSPACE's board naming conventions:

- Most RTI block names start with the board name.
- A short description of functionality is added.
- Most RTI block names also have a suffix.

Suffix	Meaning
B	Board number (for PHS-bus-based systems)
M	Module number (for MicroAutoBox II)
C	Channel number
G	Group number
CON	Converter number
BL	Block number
P	Port number
I	Interrupt number

A suffix is followed by the appropriate number. For example, DS2201IN_B2_C14 represents a digital input block located on a DS2201 board. The suffix indicates board number 2 and channel number 14 of the block. For more general block naming, the numbers are replaced by variables (for example, DS2201IN_Bx_Cy).

Special folders

Some software products use the following special folders:

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

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

or

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

Documents folder A standard folder for user-specific documents.

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

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

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

Accessing dSPACE Help and PDF Files

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

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

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

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

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

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

Safety Precautions

Introduction

To avoid risk of injury and/or damage, read and ensure compliance with the safety precautions given.

General Warning When Using the Internal Bypass Plug-In for the RTI Bypass Blockset

Introduction

Note the following warning when using the Internal Bypass Plug-In for the RTI Bypass Blockset.

Danger potential

Using this product can be dangerous. You must observe the following safety instructions and the relevant instructions in the user documentation.

⚠ **WARNING**

Improper or negligent use can result in serious personal injury and/or property damage.

The Internal Bypass Plug-In for the RTI Bypass Blockset allows the integration of function code and associated data in ECU image and ECU variable description files. Programming ECUs with these image files and accessing ECUs via calibration tools with these ECU description files can have a direct effect on networked electronic systems and may lead to unforeseeable system behavior with an increased risk of property damage or personal injury.

Only persons who are qualified to use the Internal Bypass Plug-In for the RTI Bypass Blockset, who have been informed about the above dangers, and who are able to assess the possible consequences to take appropriate precautions, are permitted to use this product.

All applications where malfunctions or misoperation involve danger of property damage, injury or death must be examined for potential hazards by the user, who must if necessary take additional measures for protection, for example, by implementing an emergency off switch, and/or by clearly labeling files to prevent original ECU image and ECU description files being confused with those modified by the Internal Bypass Plug-In for the RTI Bypass Blockset.

Liability

It is your responsibility to adhere to instructions and warnings. Any unskilled operation or other improper use of this product in violation of the respective safety instructions, warnings, or other instructions contained in the user documentation constitutes contributory negligence, which may lead to a limitation of liability by dSPACE GmbH, its representatives, agents and regional dSPACE companies, to the point of total exclusion, as the case may be. Any exclusion or limitation of liability according to other applicable regulations, individual agreements, and applicable general terms and conditions remain unaffected.

Introduction to the RTI Bypass API

Where to go from here

Information in this section

Basics on the RTI Bypass API

Introduction

The RTI Bypass API provides RTI block-specific use cases with methods and block-specific parameters for automating the creation and configuration of RTI MATLAB models.

Use cases and block parameters

You can work with these and other use cases of the RTI Bypass API:

- Accessing block parameters
- Handling database (A2L) files
- Configuring ECU events

- Selecting ECU interfaces
- Importing and selecting compiler configurations
- Summarizing models with RTI Bypass blocks
- Handling variables
- Configuring and starting the build process for the internal (and external) bypass parts in the model
- Adding custom C code
- Generating Simulink parameters and signals

You can get and set block parameters such as the following:

- BypassInterfaceName
- InitData
- InterfaceIP
- UseRelativePath

RTI Bypass API

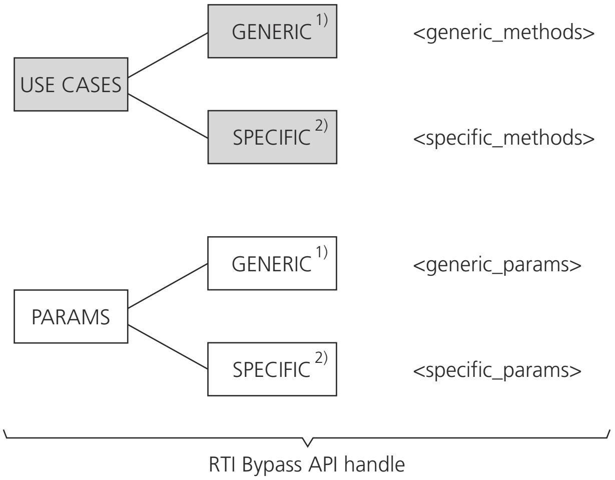
The RTI Bypass API consists of:

- A generic use case that lets you configure both the generic and the specific block parameters.
- Generic use cases and parameters that let you configure RTI Bypass blocks independently from the current configuration such as the active bypass interface type.
- Specific use cases and parameters that depend on the selected bypass interface, bypass method, prototyping hardware, or I/O board.

RTI Bypass API handle

You have to generate an RTI Bypass API handle for each block in a MATLAB model to work with the RTI Bypass API. An API handle provides access to a block's use case methods and block parameters. You have to generate a new RTI Bypass API handle after selecting or changing a bypass interface.

The following illustration shows the structure of an RTI Bypass API handle.



1) Common block parameters and use case methods available for all the bypass interfaces.
2) Specific block parameters and use case methods available only for specific bypass interfaces.

Creating and Configuring RTI MATLAB Models

Introduction

Together with MATLAB's M-Script API, the RTI Bypass API lets you write batch scripts for automating the creation and configuration of RTI MATLAB models.

Creating MATLAB elements

The following listings show how you can create models, add blocks such as blocks from the RTI Bypass Block Library to a model, and get and set the MATLAB-specific block parameters using MATLAB-specific functions.

Tip

For information on using MATLAB-specific functions to work with models, refer to MATLAB's documentation. For examples for using MATLAB-specific functions to work with a model, refer to the demo scripts in the `%ProgramData%\dSPACE\<InstallationGUID>\Demos\RTIBYPASS\Automation` folder.

You can access the `%ProgramData%\dSPACE\<InstallationGUID>` folder via a shortcut in the Windows Start menu below dSPACE RCP and HIL <version>.

Creating models

```
new_model = new_system ('HelloWorld');
open_system(new_model);
rtilib = load_system('rtibypasslib');
```

MATLAB creates the `HelloWorld` model and opens it. The `rtibypasslib` blockset library is loaded to access its blocks.

Adding blocks

```
setup_block = add_block('rtibypasslib/RTIBYPASS_SETUP_BL1',...
'Hello World/RTIBYPASS_SETUP_BL1');
```

A Setup block from the RTI Bypass Blockset is added to the model. The block's unique ID is assigned to the `setup_block` variable.

Accessing MATLAB parameters

```
params = get_param(setup_block,'objectparameters');
set_param(setup_block,'Position',[65 70 185 120]);
```

The position of the block is changed to the given position parameters. These parameters are MATLAB-specific block parameters.

Tip

Type *help <command>* for reference information on MATLAB API commands. For example, type `help add_block` to get reference information on the `add_block` command.

Generating an RTI Bypass API handle

Generating an API handle for valid RTI Bypass blocks The listing below shows how you can create block-specific RTI Bypass API handles to work with RTI Bypass-specific methods and block parameters for all the RTI Bypass blocks. Each block must be valid, i.e., reside in a Simulink model and have a configured Setup block or be a Setup block itself.

```
block_api = rtibypassexecute(block);
```

This creates the RTI Bypass API handle of a block, which allows you to work with the block's use cases and parameters. The unique block ID or full path string of the block is provided to the `rtibypassexecute` command via the `block` variable. The API handle is assigned to the `block_api` variable.

Assigning a Setup block and generating an API handle The listing below shows how you can assign a Setup block to a block and create an RTI Bypass API handle for all the RTI Bypass blocks except the Setup block.

```
block_api = rtibypassexecute(block, setup_block);
```

The RTI Bypass API handle of the block is created. The unique block ID or full path string of the Setup block is provided to the `rtibypassexecute` command via the `setup_block` variable.

Tip

Type `rtibypassexecute` in MATLAB's Command Window for reference information.

Generating new RTI Bypass API handles

If you change the active RTI bypass interface type or name, you have to generate new RTI Bypass API handles to configure the interface's specific parameters. You have to generate new RTI Bypass API handles for all the blocks that depend on the changed interface.

Activating the Debug mode for working with the RTI Bypass API

The following listing shows how you can activate the Debug mode for getting additional run-time information when working with the RTI Bypass MATLAB API if activated warning and info messages are printed in addition to error messages during run time in MATLAB's Command Window.

```
DSPACE_RTIBYPASS_Config.RTI_AUTOMATION_DEBUG = true;
```

This creates the `DSPACE_RTIBYPASS_Config.RTI_AUTOMATION_DEBUG` MATLAB workspace variable. If the variable is set to `true` the Debug mode is activated, otherwise the Debug mode is deactivated.

Working with RTI Bypass API Handles

Introduction

You have to work with RTI Bypass API handles to access an RTI Bypass block's use case methods and block parameters.

Configuring RTI Bypass blocks

To configure RTI Bypass blocks in MATLAB models, you have to provide an RTI Bypass API handle, the desired method or parameter name, and, if required, parameter values to the `rtibypassexecute` command. A method or parameter is a part of each block-specific API handle. Methods and parameters are categorized as either generic or specific. You have to specify methods or parameters relative to the API handle.

> **Tip**
>
> Use MATLAB to write batch scripts for configuring RTI Bypass blocks in MATLAB models. MATLAB lets you select from the available methods and parameters of an API handle during script writing.

Calling methods of a use case

The following listing shows how you can call use case methods.

```
success = rtibypassexecute(setup_api,...
setup_api.METHODS.GENERIC.ADD_DBFILES,...
{[dspaceroot '\Demos\RTIBYPASS\test_byp_xcp_on_can.a2l']})
```

The test_byp_xcp_on_can.a2l file is added to the RTI Bypass Setup block. The ADD_DBFILES method is called for this purpose. The method returns the logical status of the operation that is assigned to the success variable.

If you want to call a method of a specific use case, write <api_handle>.METHODS.SPECIFIC.<method> instead of <api_handle>.METHODS.GENERIC.<method>.

Tip

Use case methods provide useful comments. Type <api_handle>.METHODS.GENERIC/SPECIFIC.<method>.COMMENT() to display the comments. For example, type <api_handle>.METHODS.GENERIC.ADD_DBFILES.COMMENT() to display the comment of the ADD_DBFILES method.

Accessing parameters

The following listings show how you can get and set parameters.

Getting parameters

```
isEnabled = rtibypassexecute(read_api,...
 read_api.METHODS.GENERIC.GET_PARAM,...
 read_api.PARAMS.SPECIFIC.WaitEnable);
```

The value of the specific WaitEnable parameter is assigned to the isEnabled variable. Each API handle provides the generic GET_PARAM and SET_PARAM use case methods for you to get and set both the generic and the specific parameters. If you want to get or set a generic parameter, write <api_handle>.PARAMS.GENERIC.<parameter> instead of <api_handle>.PARAMS.SPECIFIC.<parameter>.

Setting parameters

```
success = rtibypassexecute(read_api,...
 read_api.METHODS.GENERIC.SET_PARAM,...
 read_api.PARAMS.SPECIFIC.WaitEnable,true);
```

The specific WaitEnable parameter is set to 'true'. The SET_PARAM method returns a Boolean with the status of the set operation that is assigned to the success variable.

Using the RTI Bypass MATLAB API Without Database Files

Introduction

The RTI Bypass MATLAB API allows you to work with the RTI Bypass Blockset without database files.

Configuring the active bypass interface

The *ECU Interface Selection* use case lets you configure the active bypass interface. The following listing shows how you can configure the active bypass interface without adding database files.

```
success = rtibypassexecute(setup_api,
setup_api.METHODS.GENERIC.SET_INTERFACE, 'xcp_on_can');
```

The active bypass interface of the Setup block that corresponds to the `setup_api` is changed to the `xcp_on_can` ECU interface. The required database file parameters are set to defaults. You can change the database file parameters subsequently.

Supported ECU interfaces

Using the RTI Bypass MATLAB API without database files is supported for the following ECU interfaces:

- `'ccp'`
- `'dspace_on_dpmem'`
- `'internal'`
- `'xcp_on_can'`
- `'xcp_on_flexray'`
- `'xcp_on_udp_ip'`

Configuring database file parameters

The *Block Parameter Access* use case lets you get and set database file parameters just like all the other parameters of the RTI Bypass MATLAB API.

Tip

Refer to the index for a list of all the parameters of the RTI Bypass MATLAB API. The keyword *database file parameter* lists all the parameters that can be specified in database files. The keyword *parameter* lists all the parameters that cannot be specified via database files.

Use Cases

Where to go from here

Information in this section

Generic Use Cases

Where to go from here

Information in this section

Block Parameter Access

Purpose

To get and set block parameter values.

Methods

You can use the following methods to get and set block parameters:

- <api_handle>.METHODS.GENERIC.GET_PARAM
- <api_handle>.METHODS.GENERIC.SET_PARAM

The methods are available for the following blocks:

- Build block
- Calpageswitch block
- Download block
- Function block
- Info block
- Interrupt block
- Read block
- Setup block
- Upload block
- Write block

GET_PARAM

Description Returns the value of a block parameter.

Examples

```
param = rtibypassexecute(setup_api, setup_api.METHODS.GENERIC.GET_PARAM, setup_api.PARAMS.GENERIC.UseDispId);
```

Parameters

Parameter	Type	Description
Output	block parameter type	The `rtibypassexecute` command returns the value of the block parameter.

SET_PARAM

Description Sets the value of a block parameter.

Examples

- Activating display identifiers in the variable list:

```
val = true;
success = rtibypassexecute(setup_api, setup_api.METHODS.GENERIC.SET_PARAM, setup_api.PARAMS.GENERIC.UseDispId, val);
```

- Adding a bypass interface specified in an RTIBYPASS_SETUP_BLx block to a Function block:

```
success = rtibypassexecute(function_api, function_api.METHODS.GENERIC.SET_PARAM,
function_api.PARAMS.GENERIC.BypassInterfaceName, 'EcuIf');
```

Parameters

Parameter	Type	Description
Input	block parameter type	Value of the block parameter to be set.
Output	logical	The `rtibypassexecute` command returns the status of the operation.

Related topics

Basics

Working with RTI Bypass API Handles..21

Build Process Configuration and Start

Purpose

To configure and start the build process for the internal and external bypass parts in the model.

Note

This use case is available for the following ECU interfaces:

- `'internal'`

Methods

You can use the following methods to handle free memory segments:

- <api_handle>.METHODS.GENERIC.SET_RCP_AUTO_LOAD
- <api_handle>.METHODS.GENERIC.GET_RCP_AUTO_LOAD
- <api_handle>.METHODS.GENERIC.SET_SELECTED_INTBYP_SETUP_BLOCKS
- <api_handle>.METHODS.GENERIC.GET_SELECTED_INTBYP_SETUP_BLOCKS
- <api_handle>.METHODS.GENERIC.GET_ALL_INTBYP_SETUP_BLOCKS
- <api_handle>.METHODS.GENERIC.HAS_EXTERNAL_BYPASS
- <api_handle>.METHODS.GENERIC.RUN
- <api_handle>.METHODS.GENERIC.LOADX86APP
- <api_handle>.METHODS.GENERIC.SET_INTBYP_AUTO_FLASH
- <api_handle>.METHODS.GENERIC.GET_INTBYP_AUTO_FLASH

The methods are available for the following blocks:

- Build block

SET_RCP_AUTO_LOAD

Description Sets whether to automatically download the generated external bypass application to the RCP system after the build process has finished.

Examples

```
auto_load = true;
success = rtibypassexecute(build_api, build_api.METHODS.GENERIC.SET_RCP_AUTO_LOAD, auto_load);
```

Parameters

Parameter	Type	Description
Input	logical	You have to specify whether the generated external bypass ECU application is to be automatically downloaded to the RCP system after the build process has finished.
Output	logical	The `rtibypassexecute` command returns the status of the operation.

GET_RCP_AUTO_LOAD

Description Returns whether automatic download of the generated ECU application to the RCP system on completion of the build process is enabled for the external bypass parts in the model.

Examples

```
auto_load = rtibypassexecute(build_api, build_api.METHODS.GENERIC.GET_RCP_AUTO_LOAD);
```

Parameters

Parameter	Type	Description
Output	logical	The `rtibypassexecute` command returns the information whether automatic download of the generated external bypass ECU application to the RCP system on completion of the build process is enabled for the external bypass parts in the model.

SET_SELECTED_INTBYP_SETUP_BLOCKS

Description Sets the bypass interfaces of the internal bypass parts that are to be included in the build process.

Examples

```
intbyp_setup_blocks = rtibypassexecute(build_api, build_api.METHODS.GENERIC.GET_ALL_INTBYP_SETUP_BLOCKS);
if_1 = intbyp_setup_blocks{1};
if_2 = intbyp_setup_blocks{2};
success = rtibypassexecute(build_api, build_api.METHODS.GENERIC.SET_SELECTED_INTBYP_SETUP_BLOCKS, {if_1, if_2});
```

Parameters

Parameter	Type	Description
Input	cell({char})	List of bypass interface names to be set.
Output	logical	The `rtibypassexecute` command returns the status of the operation.

GET_SELECTED_INTBYP_SETUP_BLOCKS

Description Returns the Setup blocks of the internal bypass parts that are selected to be included in the build process.

Examples

```
[interface_names, block_handles] = rtibypassexecute(build_api,
build_api.METHODS.GENERIC.GET_SELECTED_INTBYP_SETUP_BLOCKS);
```

Parameters

Parameter	Type	Description
Output	interface_names cell({char}), block_handles cell({double})	The `rtibypassexecute` command returns the Setup blocks currently selected for the internal bypass build process. The first list contains the bypass interface names, the second list contains the related handles.

GET_ALL_INTBYP_SETUP_BLOCKS

Description Returns all the Setup blocks of the model configured for internal bypassing.

Examples

```
[interface_names, block_handles] = rtibypassexecute(build_api, build_api.METHODS.GENERIC.GET_ALL_INTBYP_SETUP_BLOCKS);
```

Parameters

Parameter	Type	Description
Output	interface_names cell({char}), block_handles cell({double})	The `rtibypassexecute` command returns all Setup blocks configured in the model for internal bypassing. The first list contains the bypass interface names, the second list contains the related handles.

HAS_EXTERNAL_BYPASS

Description Returns whether the Simulink model has external bypass parts.

Examples

```
has_extbyp = rtibypassexecute(build_api, build_api.METHODS.GENERIC.HAS_EXTERNAL_BYPASS);
```

Parameters

Parameter	Type	Description
Output	logical	The `rtibypassexecute` command returns whether the Simulink model has external bypass parts.

RUN

Description Starts the build process for the selected Setup blocks.

Examples

```
success = rtibypassexecute(build_api, build_api.METHODS.GENERIC.RUN);
```

Parameters

Parameter	Type	Description
Output	logical	The `rtibypassexecute` command returns the status of the operation.

LOADX86APP

Description Starts the download of the generated ELF file to the VEOS VPU.

Examples

```
success = rtibypassexecute(build_api, build_api.METHODS.GENERIC.LOADX86APP);
```

Parameters

Parameter	Type	Description
Output	logical	The `rtibypassexecute` command returns the status of the operation.

SET_INTBYP_AUTO_FLASH

Description Specifies whether to automatically flash the internal bypass application generated for the selected Setup blocks to the ECU after the build process has finished.

Examples

```
auto_flash = true;
success = rtibypassexecute(build_api, build_api.METHODS.GENERIC.SET_INTBYP_AUTO_FLASH, auto_flash);
```

Parameters

Parameter	Type	Description
Input	logical	Specifies whether to automatically flash the internal bypass application generated for the selected Setup blocks to the ECU after the build process has finished.
Output	logical	The `rtibypassexecute` command returns the status of the operation.

GET_INTBYP_AUTO_FLASH

Description Returns whether automatic flashing on completion of the build process is enabled for the internal bypass parts in the model.

Examples

```
auto_flash = rtibypassexecute(build_api, build_api.METHODS.GENERIC.GET_INTBYP_AUTO_FLASH);
```

Parameters

Parameter	Type	Description
Output	logical	The `rtibypassexecute` command returns whether automatic flashing of the generated internal bypass ECU applications on completion of the build process is enabled for the internal bypass parts in the model.

Related topics

Basics

Working with RTI Bypass API Handles..21

Bypass Interface Update

Purpose

To update the bypass interface names for all blocks connected to an Interrupt block.

Methods

You can use the following methods to update the bypass interface names:

- <api_handle>.METHODS.GENERIC.UPDATE_SETUP_REF_OF_SUBSYS_BLOCKS

The methods are available for the following blocks:

- Interrupt block

UPDATE_SETUP_REF_OF_SUBSYS_BLOCKS

Description Updates the bypass interface names of all blocks connected to an Interrupt block.

Examples

```
old_name = rtibypassexecute(setup_api1, setup_api1.METHODS.GENERIC.GET_PARAM,
setup_api1.PARAMS.GENERIC.BypassInterfaceName);
new_name = rtibypassexecute(setup_api2, setup_api2.METHODS.GENERIC.GET_PARAM,
setup_api2.PARAMS.GENERIC.BypassInterfaceName);
success = rtibypassexecute(interrupt_api, interrupt_api.METHODS.GENERIC.UPDATE_SETUP_REF_OF_SUBSYS_BLOCKS, old_name,
new_name);
```

Parameters

Parameter	Type	Description
Input	char	Bypass interface name of the old Setup block.
Input	char	Bypass interface name of the new Setup block. All connected blocks with the old bypass interface name are updated to the new bypass interface name. The interface-specific parameters are also updated.
Output	logical	The `rtibypassexecute` command returns the status of the operation.

Related topics

Basics

Working with RTI Bypass API Handles........21

Database File Handling

Purpose

To handle A2L files of a Setup block.

Methods

You can use the following methods to handle A2L files:

- <api_handle>.METHODS.GENERIC.ADD_DBFILES
- <api_handle>.METHODS.GENERIC.GET_DBFILES
- <api_handle>.METHODS.GENERIC.REMOVE_DBFILES
- <api_handle>.METHODS.GENERIC.UPDATE_DBFILES

The methods are available for the following blocks:

- Setup block

ADD_DBFILES

Description Adds A2L files to the TargetLink Data Dictionary.

Examples

```
dbfiles = {'c:\databases\test_byp_xcp_on_can.a2l','c:\databases\ecu_test_xcp_on_can.a2l'};
success = rtibypassexecute(setup_api, setup_api.METHODS.GENERIC.ADD_DBFILES, dbfiles);
```

Parameters

Parameter	Type	Description
Input	cell({char})	List of A2L files to be added.
Output	logical	The `rtibypassexecute` command returns the status of the operation.

GET_DBFILES

Description Gets the list of available A2L files.

Examples

```
dbfiles = rtibypassexecute(setup_api, setup_api.METHODS.GENERIC.GET_DBFILES);
file_1 = dbfiles{1};
file_2 = dbfiles{2};
```

Parameters

Parameter	Type	Description
Output	cell({char})	The `rtibypassexecute` command returns the list of database file names.

REMOVE_DBFILES

Description Removes A2L files from the TargetLink Data Dictionary.

Examples

```
dbfiles = {'c:\databases\test_byp_xcp_on_can.a2l','c:\databases\ecu_test_xcp_on_can.a2l'};
success = rtibypassexecute(setup_api, setup_api.METHODS.GENERIC.REMOVE_DBFILES,{dbfiles{1}});
```

Parameters

Parameter	Type	Description
Input	cell({char})	List of A2L files to be removed.
Output	logical	The `rtibypassexecute` command returns the status of the operation.

UPDATE_DBFILES

Description Updates A2L files in the TargetLink Data Dictionary.

Examples

```
dbfiles = {'c:\databases\test_byp_xcp_on_can.a2l','c:\databases\ecu_test_xcp_on_can.a2l'};
success = rtibypassexecute(setup_api, setup_api.METHODS.GENERIC.UPDATE_DBFILES,dbfiles);
```

```
old_dbfiles = {'c:\databases\test_byp_xcp_on_can.a2l'};
new_dbfiles = {'c:\databases\test.a2l'};
success = rtibypassexecute(setup_api, setup_api.METHODS.GENERIC.UPDATE_DBFILES,old_dbfiles,new_dbfiles);
```

Parameters

Parameter	Type	Description
Input	cell({char})	List of A2L files to be updated.
(Optional) Input	cell({char})	You can provide a second A2L file list. If the second list is available, database files from the first list are replaced with files from the second list. Files with equal indices are replaced.
Output	logical	The `rtibypassexecute` command returns the status of the operation.

Related topics

Basics

Working with RTI Bypass API Handles.......21

Define Handling

Purpose

To handle defines for a Function block. The defines can be used for preprocessing the source files before parsing and compiling.

Methods

You can use the following methods to handle defines:

- <api_handle>.METHODS.GENERIC.ADD_DEFINES
- <api_handle>.METHODS.GENERIC.GET_DEFINES
- <api_handle>.METHODS.GENERIC.REMOVE_DEFINES

The methods are available for the following blocks:

- Function block

ADD_DEFINES

Description Adds preprocessor defines to a Function block.

Examples

```
define_names = {'INTBYP', 'UINT'};
define_values = {'1', 'unsigned int'};
success = rtibypassexecute(function_api, function_api.METHODS.GENERIC.ADD_DEFINES, {define_names, define_values});
```

Parameters

Parameter	Type	Description
Input	cell({key:cell({char}), value:cell({char})})	You have to provide two lists: The first list must contain the define names to be added, and the second list must contain the corresponding define values.

Parameter	Type	Description
		The two lists must have the same number of elements. The define names must be unique.
Output	logical	The rtibypassexecute command returns the status of the operation.

GET_DEFINES

Description Returns the defines of a Function block.

Examples

```
defines = rtibypassexecute(function_api, function_api.METHODS.GENERIC.GET_DEFINES);
```

Parameters

Parameter	Type	Description
Output	cell({key:cell({char}), value:cell({char})})	The rtibypassexecute command returns a cell with the available defines. The first list contains the define names, the second list contains the values.

REMOVE_DEFINES

Description Removes defines from a Function block.

Examples

```
success = rtibypassexecute(function_api, function_api.METHODS.GENERIC.REMOVE_DEFINES, {'INTBYP', 'UINT'});
```

Parameters

Parameter	Type	Description
Input	cell({char})	List of define names. The name value pairs belonging to the define names from the list are removed.
Output	logical	The rtibypassexecute command returns the status of the operation.

Related topics

Basics

Working with RTI Bypass API Handles..21

ECU Interface Selection

Purpose

To configure the active bypass interface.

Methods

You can use the following methods for configuring the active bypass interface:

- <api_handle>.METHODS.GENERIC.GET_INTERFACES
- <api_handle>.METHODS.GENERIC.SET_INTERFACE
- <api_handle>.METHODS.GENERIC.UNSET_INTERFACE

The methods are available for the following blocks:

- Setup block

GET_INTERFACES

Description Returns a struct with interfaces, sorted by the database files in which the interfaces are defined.

Examples

```
ifaces = rtibypassexecute(setup_api, setup_api.METHODS.GENERIC.GET_INTERFACES);
file_1 = ifaces{1}.dbfile;
interface_1 = ifaces{1}.interfaceList{1};
```

Parameters

Parameter	Type	Description
Output	cell({dbfile char, interfaceList cell{char}})	The `rtibypassexecute` command returns a struct with database files and interface lists.

SET_INTERFACE

Description Sets the active bypass interface.

Examples The following listing shows how you can set a bypass interface described by a database file:

```
ifaces = rtibypassexecute(setup_api, setup_api.METHODS.GENERIC.GET_INTERFACES);
file_1 = ifaces{1}.dbfile;
interface_1 = ifaces{1}.interfaceList{1};
success = rtibypassexecute(setup_api, setup_api.METHODS.GENERIC.SET_INTERFACE, interface_1, file_1);
```

Parameters

Parameter	Type	Description
Input	char	Name of the interface to be set as the active interface.
Input	char	Database file
Output	logical	The `rtibypassexecute` command returns the status of the operation.

Note

If you change the active RTI bypass interface, you have to generate new RTI Bypass API handles to configure the interface's specific parameters. You have to generate new RTI Bypass API handles for all the blocks that depend on the reconfigured Setup block.

UNSET_INTERFACE

Description Unsets the active interface.

Examples

```
success = rtibypassexecute(setup_api, setup_api.METHODS.GENERIC.UNSET_INTERFACE);
```

Parameters

Parameter	Type	Description
Output	logical	The `rtibypassexecute` command returns the status of the operation.

Related topics

Basics

Working with RTI Bypass API Handles..21

Function Handling

Purpose

To specify a C function for a Function block.

Tip

Working with functions consists of several steps, to be performed in the following order:

1. Add source files (refer to Source File Handling on page 46)
2. Add defines (refer to Define Handling on page 34)
3. Add includes (refer to Include Folder Handling on page 41)
4. Add preprocessor options (refer to Preprocessor Configuration on page 43)
5. Get all functions (see below)
6. Set function (see below)
7. Get all global variables (see below)
8. Add global variables (see below)

Methods

You can use the following methods to specify the function to be called:

- <api_handle>.METHODS.GENERIC.GET_ALL_FUNCTIONS
- <api_handle>.METHODS.GENERIC.SET_FUNCTION
- <api_handle>.METHODS.GENERIC.GET_FUNCTION
- <api_handle>.METHODS.GENERIC.ANALYZE_FUNCTION
- <api_handle>.METHODS.GENERIC.GET_ALL_GLOBAL_VARIABLES
- <api_handle>.METHODS.GENERIC.ADD_GLOBAL_VARIABLES
- <api_handle>.METHODS.GENERIC.REMOVE_GLOBAL_VARIABLES
- <api_handle>.METHODS.GENERIC.GET_ALL_FUNCTIONS_AND_ALL_GLOBAL_VARIABLES

The methods are available for the following blocks:

- Function block

GET_ALL_FUNCTIONS

Description Returns all the functions contained in the source files added to the model.

Examples

```
sources = rtibypassexecute(function_api, function_api.METHODS.GENERIC.GET_ALL_FUNCTIONS);
fnc_1 = sources(1).functions(1);
fnc_1_name = sources(1).functions(1).name;
fnc_2 = sources(1).functions(2);
fnc_2_name = sources(1).functions(2).name;
```

Parameters

Parameter	Type	Description
Output	cell(struct)[1]	The `rtibypassexecute` command returns a cell of structs. Each struct consists of the following fields: ▪ source_file (char) This field represents the path to the current C module. ▪ functions (cell of structs) This field lists all the functions that are contained in the C module.

[1] Detailed knowledge of the structure of the struct is not essential for using this method.

SET_FUNCTION

Description Sets the current function of a Function block.

Examples

```
sources = rtibypassexecute(function_api, function_api.METHODS.GENERIC.GET_ALL_FUNCTIONS);
function_1 = sources(1).functions(1);
success = rtibypassexecute(function_api, function_api.METHODS.GENERIC.SET_FUNCTION, function_1);
```

Parameters

Parameter	Type	Description
Input	function struct[1]	You have to provide a complete function (as a function struct). The `rtibypassexecute` command sets the function to the Function block and analyzes it to determine its parameters and locally referenced global variables. It then configures the inports and outports of the Function block according to the variable access types.
Output	logical	The `rtibypassexecute` command returns the status of the operation.

[1] Detailed knowledge of the structure of the function struct is not essential for using this method.

GET_FUNCTION

Description Returns the current function of a Function block.

Examples

```
function = rtibypassexecute(function_api, function_api.METHODS.GENERIC.GET_FUNCTION);
```

Parameters

Parameter	Type	Description
Output	function struct[1]	The `rtibypassexecute` command returns the current function and all the currently selected variables.

[1] Detailed knowledge of the structure of the function struct is not essential for using this method.

ANALYZE_FUNCTION

Description Analyzes a function to determine its parameters and locally referenced global variables.

Examples

```
sources = rtibypassexecute(function_api, function_api.METHODS.GENERIC.GET_ALL_FUNCTIONS);
analyzed_fun = rtibypassexecute(function_api, function_api.METHODS.GENERIC.ANALYZE_FUNCTION, sources(1).functions(1));
var_1 = analyzed_fun.variables(1);
```

Parameters

Parameter	Type	Description
Output	function struct[1]	The `rtibypassexecute` command returns the analyzed function and its variables.

[1] Detailed knowledge of the structure of the function struct is not essential for using this method.

GET_ALL_GLOBAL_VARIABLES

Description Returns all the global variables that are defined in the source files added to the model.

Examples

```
sources = rtibypassexecute(function_api, function_api.METHODS.GENERIC.GET_ALL_GLOBAL_VARIABLES);
global_var_1 = sources(1).global_variables.variables(1);
global_var_1_name = sources(1).global_variables.variables(1).name;
```

Parameters

Parameter	Type	Description
Output	cell(struct)[1]	The `rtibypassexecute` command returns a cell of structs. Each struct consists of the following fields: ▪ source_file (char) This field represents the path to the current C module. ▪ global_variables (cell of structs) This field lists all the global variables that are contained in the C module.

[1] Detailed knowledge of the structure of the struct is not essential for using this method.

ADD_GLOBAL_VARIABLES

Description Adds global variables to be used as additional input and/or output values to the Function block.

Examples

```
sources = rtibypassexecute(function_api, function_api.METHODS.GENERIC.GET_ALL_GLOBAL_VARIABLES);
global_var_1 = sources(1).global_variables.variables(1);
global_var_1.ReadAccess = 1;        %Results in an input port
global_var_1.WriteAccess = 1;       %Results in an output port
success = rtibypassexecute(function_api, function_api.METHODS.GENERIC.ADD_GLOBAL_VARIABLES, {global_var_1});
```

Parameters

Parameter	Type	Description
Output	logical	The `rtibypassexecute` command returns the status of the operation.

REMOVE_GLOBAL_ VARIABLES

Description Removes global variables from the Function block.

Examples

```
function = rtibypassexecute(function_api, function_api.METHODS.GENERIC.GET_FUNCTION);
referenced_var = function.variables(1);
success = rtibypassexecute(function_api,function_api.METHODS.GENERIC.REMOVE_GLOBAL_VARIABLES,{referenced_var});
```

Parameters

Parameter	Type	Description
Output	logical	The `rtibypassexecute` command returns the status of the operation.

GET_ALL_FUNCTIONS_AND_ ALL_GLOBAL_VARIABLES

Description Returns all the functions and global variables contained in the source files that are added to the model.

Examples

```
sources = rtibypassexecute(function_api, function_api.METHODS.GENERIC.GET_ALL_FUNCTIONS_AND_ALL_GLOBAL_VARIABLES);
fnc_1 = sources(1).functions(1);
fnc_1_name = sources(1).functions(1).name;
glob_var_1 = sources(1).global_variables.variables(1);
global_var_1_name = sources(1).global_variables.variables(1).name;
```

Parameters

Parameter	Type	Description
Output	cell(struct)[1]	The `rtibypassexecute` command returns a cell of structs. Each struct consists of the following fields: ▪ source_file (char) This field represents the path to the current C module. ▪ functions (cell of structs) This field lists all the functions that are contained in the C module. ▪ global_variables (cell of structs) This field lists all the global variables that are contained in the C module.

[1] Detailed knowledge of the structure of the struct is not essential for using this method.

Related topics

Basics

Working with RTI Bypass API Handles..21

Include Folder Handling

Purpose

To handle folders for inclusion in source file preprocessing during parsing and compiling.

Methods

You can use the following methods to handle include folders:

- <api_handle>.METHODS.GENERIC.ADD_INCLUDE_DIRS
- <api_handle>.METHODS.GENERIC.GET_INCLUDE_DIRS
- <api_handle>.METHODS.GENERIC.REMOVE_INCLUDE_DIRS

The methods are available for the following blocks:

- Function block

ADD_INCLUDE_DIRS

Description Adds additional preprocessor include folders to the Function block.

Examples

```
includes = {'c:\include'};
success = rtibypassexecute(function_api, function_api.METHODS.GENERIC.ADD_INCLUDE_DIRS, includes);
```

Parameters

Parameter	Type	Description
Input	cell({char})	List of include folders to be added.
Output	logical	The `rtibypassexecute` command returns the status of the operation.

GET_INCLUDE_DIRS

Description Returns the current include folders of the Function block.

Examples

```
includes = rtibypassexecute(function_api, function_api.METHODS.GENERIC.GET_INCLUDE_DIRS);
folder_1 = includes{1};
folder_2 = includes{2};
```

Parameters

Parameter	Type	Description
Output	cell({char})	The `rtibypassexecute` command returns the list of database file names.

REMOVE_INCLUDE_DIRS

Description Removes include folders from the Function block.

Examples

```
includes = rtibypassexecute(function_api, function_api.METHODS.GENERIC.GET_INCLUDE_DIRS);
include_1 = includes{1};
success = rtibypassexecute(function_api, function_api.METHODS.GENERIC.REMOVE_INCLUDE_DIRS, {include_1});
```

Parameters

Parameter	Type	Description
Input	cell({char})	List of include folders to be removed.
Output	logical	The `rtibypassexecute` command returns the status of the operation.

Related topics

Basics

Working with RTI Bypass API Handles..21

Model Summary

Purpose

To get a summary of the model.

Methods

You can use the following methods to get a summary of the model:

- <api_handle>.METHODS.GENERIC.GET_MODELINFO

The methods are available for the following blocks:

- Info block

GET_MODELINFO

Description Returns a summary of the model.

Examples

```
summary = rtibypassexecute(info_api, info_api.METHODS.GENERIC.GET_MODELINFO);
```

Parameters

Parameter	Type	Description
Output	cell(char)	The `rtibypassexecute` command returns a summary of the model.

Related topics

Basics

Working with RTI Bypass API Handles..21

Preprocessor Configuration

Purpose

To configure preprocessor options for a Function block. In some cases, the preprocessor requires additional options for parsing the function prototypes from the header file.

Note

The additional cpp options are not used for compiling the source files. If you need to provide additional flags for compiling, you have to use the Code Generation build options.

Methods

You can use the following methods to configure preprocessor options:

- <api_handle>.METHODS.GENERIC.SET_CPP_OPTIONS
- <api_handle>.METHODS.GENERIC.GET_CPP_OPTIONS

The methods are available for the following blocks:

- Function block

SET_CPP_OPTIONS

Description Sets additional preprocessor options of a Function block.

Examples

```
success = rtibypassexecute(function_api, function_api.METHODS.GENERIC.SET_CPP_OPTIONS, '-C');
```

Tip

For a list of the available options, type `<api_handle>.METHODS.GENERIC.SET_CPP_OPTIONS,'-help'`.

Parameters

Parameter	Type	Description
Input	char	Preprocessor options to be used.
Output	logical	The `rtibypassexecute` command returns the status of the operation.

GET_CPP_OPTIONS

Description Returns the current preprocessor options of a Function block.

Examples

```
options = rtibypassexecute(function_api, function_api.METHODS.GENERIC.GET_CPP_OPTIONS);
```

Parameters

Parameter	Type	Description
Output	char	The `rtibypassexecute` command returns the current preprocessor options.

Related topics

Basics

Simulink Parameter and Signal Generation

Purpose

To automatically create workspace Simulink parameters and Simulink signals for a model.

Methods

You can use the following methods to create Simulink parameters and signals:

- <api_handle>.METHODS.GENERIC.CREATE_SIMULINK_PARAMETERS
- <api_handle>.METHODS.GENERIC.CREATE_SIMULINK_SIGNALS

The methods are available for the following blocks:

- Setup block

CREATE_SIMULINK_PARAMETERS

Description Creates Simulink parameters.

Simulink parameters (Characteristics) are generated ECU variables that can be used to calibrate the internal bypass application. Simulink parameters can be generated into an A2L file during code generation.

Examples

```
system_handle = gcs;
block_type = 'Constant';
block_access = 'Value';
create_unique_names = true;
success = ribypassexecute(setup_api, setup_api.METHODS.GENERIC.CREATE_SIMULINK_PARAMETERS, system_handle, block_type,
block_access, create_unique_names);
```

Parameters

Parameter	Type	Description
Input	char/numeric	The system containing blocks of the specified block type for which you want to create Simulink parameters.
Input	char	The block type you want to create Simulink parameters for. To get the block type of a block, use `get_param(blkh,'BlockType');`
Input	char	How to access the blocks' parameters, i.e., how to set/get the parameter values of the blocks. Specify the following values: ▪ For the Constant block type, use 'Value'. ▪ For the Gain block type, use 'Gain'.
Input	logical	Specifies whether unique parameter names are to be created. The variable names will be prefixed with the full path of the block (except for the root/model name). An existing workspace variable is reused if it is not yet referenced by another block in the model. Its value is updated with the new value of the block. If the existing workspace variable is already referenced by another block, the new name is enumerated. If the block already contains a string name, that string name is reused as the parameter name.
Output	logical	The `rtibypassexecute` command returns the status of the operation.

CREATE_SIMULINK_SIGNALS

Description Creates Simulink signals. Simulink signals (Measurements) are the variables that are intended for measuring. Simulink signals can be generated into an A2L file during code generation.

Examples

```
system_handle = gcs;
block_type = 'SubSystem';
create_unique_names = true;
success = ribypassexecute(setup_api, setup_api.METHODS.GENERIC.CREATE_SIMULINK_SIGNALS, system_handle, block_type,
create_unique_names);
```

Parameters

Parameter	Type	Description
Input	char/numeric	The system containing blocks of the specified block type and their outports for which you want to create Simulink signals.
Input	char	The block type for which Simulink signals are generated on each output port.
Input	logical	Specifies whether unique signal names are to be created. The variable names will be prefixed with the full path of the block (except for the root/model name). An existing workspace variable is reused if it is not yet referenced by another signal in the model. If it is already referenced, the new name is enumerated. If the signal already contains a string name, that string name is reused as the signal name.
Output	logical	The `rtibypassexecute` command returns the status of the operation.

Related topics

Basics

Source File Handling

Purpose

To handle the header and source files of a model. The files are available to each Function block in the model.

Methods

You can use the following methods to handle the header and source files of a model:

- <api_handle>.METHODS.GENERIC.ADD_SOURCE_FILES
- <api_handle>.METHODS.GENERIC.GET_SOURCE_FILES
- <api_handle>.METHODS.GENERIC.REMOVE_SOURCE_FILES

The methods are available for the following blocks:

- Function block

ADD_SOURCE_FILES

Description Adds header and source files to a model.

Examples

```
source_files = {'c:\ecu.c','c:\ecu.h'};
success = rtibypassexecute(function_api, function_api.METHODS.GENERIC.ADD_SOURCE_FILES, source_files);
```

Parameters

Parameter	Type	Description
Input	cell({char})	List of header and source files to be added.
Output	logical	The `rtibypassexecute` command returns the status of the operation.

GET_SOURCE_FILES

Description Returns the header and source files that are currently available in the model.

Examples

```
source_files = rtibypassexecute(function_api, function_api.METHODS.GENERIC.GET_SOURCE_FILES);
file_1 = source_files{1};
file_2 = source_files{2};
```

Parameters

Parameter	Type	Description
Output	cell({char})	The `rtibypassexecute` command returns the list of header and source files currently added to the model.

REMOVE_SOURCE_FILES

Description Removes header and source files from the model.

Examples

```
source_files = rtibypassexecute(function_api, function_api.METHODS.GENERIC.GET_SOURCE_FILES);
file_1 = source_files{1};
success = rtibypassexecute(function_api, function_api.METHODS.GENERIC.REMOVE_SOURCE_FILES, {file_1});
```

Parameters

Parameter	Type	Description
Input	cell({char})	List of header and source files to be removed.
Output	logical	The `rtibypassexecute` command returns the status of the operation.

Related topics

Basics

Working with RTI Bypass API Handles...21

Variable Handling

Purpose

To specify variables.

Methods

You can use the following methods to specify variables:

- <api_handle>.METHODS.GENERIC.ADD_VARIABLES
- <api_handle>.METHODS.GENERIC.GET_VARIABLE_TEMPLATE
- <api_handle>.METHODS.GENERIC.GET_VARIABLES
- <api_handle>.METHODS.GENERIC.GET_VARIABLES_CURRENT
- <api_handle>.METHODS.GENERIC.REMOVE_VARIABLES
- <api_handle>.METHODS.GENERIC.REMOVE_VARIABLES_ALL
- <api_handle>.METHODS.GENERIC.UPDATE_VARIABLES

The methods are available for the following blocks:

- Download block
- Read block
- Upload block
- Write block

Variable template

A variable template supports you when you specify variables. The following listing shows the structure of the variable template.

```
var_temp struct
{
   Address vector {double, double} : [0 0]
   BitMask double : 0
   CompuMethod struct : [1x1 struct]
   RecordLayout struct : [1x1 struct]
   DisplayIdentifier char : ""
   FailSafeValue vector {double, double} : [0 0]
   Index double : -1
   LongIdentifier char : ""
   MinMax struct : [1x1 struct]
   Name char : ""
   Signed double : 0
   Database char : ""
   DatabaseDate char : ""
   ByteOrder char : ""
   DisplayDataType char : "Unsigned bit 8"
   Type double : 8
   EnableRemapping double : 0
   RemappingSupported double : 0
   AliasName char : ""
   AliasAddress vector {double, double} : [0 0]
}
```

ADD_VARIABLES

Description Adds variables to a block.

Examples

```
var_temp = rtibypassexecute(read_api, read_api.METHODS.GENERIC.GET_VARIABLE_TEMPLATE);
var_temp.Name = "myVariable";
var_temp.Address = [hex2dec('D000') hex2dec('D02C')];
var_temp.Type = 32;
var_temp.Signed = 1;
var_temp.DisplayDataType = "Signed 32 bit";
success = rtibypassexecute(read_api, read_api.METHODS.GENERIC.ADD_VARIABLES, {var_temp});
```

Parameters

Parameter	Type	Description
Input	{var_temp struct}[1]	List of variables to be added.
Output	logical	The `rtibypassexecute` command returns the status of the operation.

[1] For information on the structure of the var_temp struct, refer to Variable template on page 48.

GET_VARIABLE_TEMPLATE

Description Returns the variable template that you can use for creating custom variables.

Examples

```
template = rtibypassexecute(read_api, read_api.METHODS.GENERIC.GET_VARIABLE_TEMPLATE);
```

Parameters

Parameter	Type	Description
Output	var_temp struct[1]	The `rtibypassexecute` command returns a struct with the available variable properties.

[1] For information on the structure of the var_temp struct, refer to Variable template on page 48.

GET_VARIABLES

Description Returns a cell of structs, that contain the database file name and its variables.

Examples

```
var = rtibypassexecute(read_api, read_api.METHODS.GENERIC.GET_VARIABLES);
file_1 = var{1}.dbfile;
var_1 = var{1}.varList{1};
var_1_name = var_1.Name;
```

Parameters

Parameter	Type	Description
Output	cell({dbfile char, varList cell{var_temp struct}})[1]	The `rtibypassexecute` command returns a cell of structs that contain a database file name and variable list.

[1] For information on the structure of the var_temp struct, refer to Variable template on page 48.

GET_VARIABLES_CURRENT

Description Returns the currently added variables of a block.

Examples

```
current_vars = rtibypassexecute(read_api, read_api.METHODS.GENERIC.GET_VARIABLES_CURRENT);
var_1 = current_vars{1};
var_1_name = var_1.Name;
```

Parameters

Parameter	Type	Description
Output	cell{var_temp struct}[1]	The `rtibypassexecute` command returns a list of the currently added variables.

[1] For information on the structure of the var_temp struct, refer to Variable template on page 48.

REMOVE_VARIABLES

Description Removes variables from a block.

Examples

```
var = rtibypassexecute(read_api, read_api.METHODS.GENERIC.GET_VARIABLES);
var_1 = var{1}.varList{1};
success = rtibypassexecute(read_api, read_api.METHODS.GENERIC.REMOVE_VARIABLES, {var_1});
```

Parameters

Parameter	Type	Description
Input	cell{var_temp struct}[1]	List of variables to be removed.
Output	logical	The `rtibypassexecute` command returns the status of the operation.

[1] For information on the structure of the var_temp struct, refer to Variable template on page 48.

REMOVE_VARIABLES_ALL

Description

Removes all variables of a block.

Examples

```
success = rtibypassexecute(read_api, read_api.METHODS.GENERIC.REMOVE_VARIABLES_ALL);
```

Parameters

Parameter	Type	Description
Output	logical	The `rtibypassexecute` command returns the status of the operation.

UPDATE_VARIABLES

Description

Updates a list of variables.

Examples

```
var = rtibypassexecute(read_api, read_api.METHODS.GENERIC.GET_VARIABLES);
var_1 = var{1}.varList{1};
var_2 = var{1}.varList{2};
success = rtibypassexecute(read_api, read_api.METHODS.GENERIC.UPDATE_VARIABLES, {var_1}, {var_2});
```

Parameters

Parameter	Type	Description
Input	cell{var_temp struct}[1]	List of variables. The variables from the list are updated.
Input	cell{var_temp struct}[1]	Second list of variables. Variables from the first list are replaced with variables from the second list. Variables with equal indices are replaced.
Output	logical	The `rtibypassexecute` command returns the status of the operation.

[1] For information on the structure of the var_temp struct, refer to Variable template on page 48.

Related topics

Basics

Working with RTI Bypass API Handles..21

Specific Use Cases

Where to go from here

Information in this section

Compiler Configuration Handling

Purpose

To handle compiler configurations of a Setup block.

Note

This use case is available for the following ECU interfaces:

- `'internal'`

Methods

You can use the following methods to handle compiler configurations:

- <api_handle>.METHODS.SPECIFIC.IMPORT_COMPILERS
- <api_handle>.METHODS.SPECIFIC.GET_COMPILERS
- <api_handle>.METHODS.SPECIFIC.SELECT_COMPILER

The methods are available for the following blocks:

- Setup block

IMPORT_COMPILERS

Description Imports compiler configurations from a compiler configuration file.

Examples

```
compiler_config_file_path = 'C:\compilerConfig.mat';
success = rtibypassexecute(setup_api, setup_api.METHODS.SPECIFIC.IMPORT_COMPILERS, compiler_config_file_path);
```

Parameters

Parameter	Type	Description
Input	char	Compiler configuration file.
Output	logical	The `rtibypassexecute` command returns the status of the operation.

GET_COMPILERS

Description Returns the available compiler configurations of the selected ECU.

Examples

```
compilers = rtibypassexecute(setup_api, setup_api.METHODS.SPECIFIC.GET_COMPILERS);
```

Parameters

Parameter	Type	Description
Output	cell({char})	The `rtibypassexecute` command returns a list of the compiler configurations.

SELECT_COMPILER

Description Selects the compiler configuration to be used for the build process.

Examples

```
to_be_used_compiler = compilers{1};
success = rtibypassexecute(setup_api, setup_api.METHODS.SPECIFIC.SELECT_COMPILER, to_be_used_compiler);
```

Parameters

Parameter	Type	Description
Input	cell(char)	Compiler configuration to be used.
Output	logical	The `rtibypassexecute` command returns the status of the operation.

Related topics

Basics

DAQ List Handling

Purpose

To handle DAQ list definitions of a Setup block.

Note

This use case is available only for CCP-based ECU interfaces and XCP-based ECU interfaces with static DAQ lists. Select a matching interface via the Setup block's ECU Interface Selection generic use case.

Methods

You can use the following methods to handle DAQ lists:

- <api_handle>.METHODS.SPECIFIC.GET_DAQ_LISTS
- <api_handle>.METHODS.SPECIFIC.GET_DAQ_LIST_TEMPLATE
- <api_handle>.METHODS.SPECIFIC.ADD_DAQ_LISTS
- <api_handle>.METHODS.SPECIFIC.REMOVE_DAQ_LISTS

The methods are available for the following blocks:

- Setup block

Tip

Use the above methods for handling DAQ lists if no database file is used in the related Setup block.

DAQ list template

A DAQ list template supports you when you handle DAQ lists. The following listings show the structure of the DAQ list templates for XCP-based and CCP-based ECU interfaces.

```
xcp_temp struct
{
   DAQ_LIST_NUMBER double
   DAQ_LIST_TYPE double
   MAX_ODT double
   MAX_ODT_ENTRIES double
   FIRST_PID double
   CAN_ID_FIXED double
   CAN_ID_VARIABLE double
   EVENT_FIXED double
}
```

```
ccp_temp struct
{
   DAQ_LIST_NUMBER double
   MAX_ODT double
   RASTER_IDs cell
   CAN_ID_FIXED double
   CAN_ID_VARIABLE double
   FIRST_PID double
}
```

GET_DAQ_LISTS

Description Returns the available DAQ list definitions of the Setup block.

Examples

```
lists = rtibypassexecute(setup_api, setup_api.METHODS.SPECIFIC.GET_DAQ_LISTS);
list_1_tmp = lists{1};
```

Parameters

Parameter	Type	Description
Output	cell({lists cell{list_tmp struct}})[1]	The `rtibypassexecute` command returns a list of DAQ lists.

[1] For information on the structure of the list_tmp struct, refer to DAQ list template on page 53.

GET_DAQ_LIST_TEMPLATE

Description Returns the DAQ list template that you can use for creating custom DAQ lists. Different DAQ list templates are returned for XCP-based and CCP-based ECU interfaces.

Examples

```
list_temp = rtibypassexecute(setup_api, setup_api.METHODS.SPECIFIC.GET_DAQ_LIST_TEMPLATE);
```

Parameters

Parameter	Type	Description
Output	list_temp struct[1]	The `rtibypassexecute` command returns a struct that depends on the selected ECU interface with the available DAQ list properties.

[1] For information on the structure of the list_temp struct, refer to DAQ list template on page 53.

ADD_DAQ_LISTS

Description Adds DAQ lists to the Setup block.

Examples

```
list_temp = rtibypassexecute(setup_api, setup_api.METHODS.SPECIFIC.GET_DAQ_LIST_TEMPLATE);
list_temp.DAQ_LIST_NUMBER = 1;
list_temp.MAX_ODT = 5;
success = rtibypassexecute(setup_api, setup_api.METHODS.SPECIFIC.ADD_DAQ_LISTS, {list_temp});
```

Parameters

Parameter	Type	Description
Input	{list_temp struct}[1]	List of DAQ lists to be added.
Output	logical	The `rtibypassexecute` command returns the status of the operation.

[1] For information on the structure of the list_temp struct, refer to DAQ list template on page 53.

REMOVE_DAQ_LISTS

Description Removes DAQ lists from a Setup block.

Examples

```
lists = rtibypassexecute(setup_api, setup_api.METHODS.SPECIFIC.GET_DAQ_LISTS);
list_1 = lists{1};
success = rtibypassexecute(setup_api, setup_api.METHODS.SPECIFIC.REMOVE_DAQ_LISTS, {list_1});
```

Parameters

Parameter	Type	Description
Input	cell{list_temp struct}[1]	List of DAQ lists to be removed.
Output	logical	The `rtibypassexecute` command returns the status of the operation.

[1] For information on the structure of the list_temp struct, refer to DAQ list template on page 53.

Related topics

Basics

Working with RTI Bypass API Handles...21

ECU Application Binary Files Selection

Purpose

To return the paths to the ECU application binary files (HEX, SREC) as configured in the imported A2L file and manually added.

Note

This use case is available for the following ECU interfaces:

- `'internal'`

Methods

You can use the following methods to search for the ECU applications for internal bypass code integration:

- <api_handle>.METHODS.SPECIFIC.GET_ALL_SRC_ECU_APPLICATIONS

The method is available for the following blocks:

- Setup block

GET_ALL_SRC_ECU_APPLICATIONS

Description Returns the paths of the available ECU applications as configured in the imported A2L files or manually added. The returned ECU applications are possible source ECU applications for inserting the internal bypass code. You can later select one of the ECU applications as the original input ECU application you want to merge the internal bypass code with.

Examples

```
ecu_apps = rtibypassexecute(setup_api, setup_api.METHODS.SPECIFIC.GET_ALL_SRC_ECU_APPLICATIONS);
success = rtibypassexecute(setup_api, setup_api.METHODS.GENERIC.SET_PARAM, setup_api.PARAMS.SPECIFIC.SrcEcuApplication,
ecu_apps{1});
```

Parameters

Parameter	Type	Description
Output	cell({char})	The `rtibypassexecute` command returns a list of the paths to the available ECU application binary files (HEX, SREC).

Related topics

Basics

Working with RTI Bypass API Handles..21

ECU Event Configuration

Purpose

To configure ECU events.

Note

This use case is available for the following ECU interfaces:

- `'ccp'`
- `'internal'`
- `'vecu'`
- `'xcp_on_can'`
- `'xcp_on_udp_ip'`

Methods

You can use the following methods to configure ECU events:

- <api_handle>.METHODS.SPECIFIC.GET_EVENTS
- <api_handle>.METHODS.SPECIFIC.SET_EVENT

The methods are available for the following blocks:

- Interrupt block
- Read block
- Write block

GET_EVENTS

Description Returns the available ECU events.

Examples

```
events = rtibypassexecute(read_api, read_api.METHODS.SPECIFIC.GET_EVENTS);
event_1 = events{1};
event_1_name = event_1.name;
event_1_id = event_1.id;
```

Parameters

Parameter	Type	Description
Output	cell({name char, id double})	The `rtibypassexecute` command returns a struct with event names and event ids.

SET_EVENT

Description Sets the active ECU event.

Examples

```
event_1 = {'10ms DAQ','0'};
success = rtibypassexecute(read_api, read_api.METHODS.SPECIFIC.SET_EVENT, event_1);
```

Parameters

Parameter	Type	Description
Input	cell(name char, id double)	Event to be set.
Output	logical	The `rtibypassexecute` command returns the status of the operation.

Related topics

Basics

Working with RTI Bypass API Handles..21

ECU Event Handling

Purpose

To handle ECU event definitions of a Setup block.

> **Note**
>
> This use case is available for the following ECU interfaces:
> - `'ccp'`
> - `'vecu'`
> - `'xcp_on_can'`
> - `'xcp_on_udp_ip'`

Methods

You can use the following methods to handle ECU events:

- <api_handle>.METHODS.SPECIFIC.GET_EVENTS
- <api_handle>.METHODS.SPECIFIC.GET_EVENT_TEMPLATE
- <api_handle>.METHODS.SPECIFIC.ADD_EVENTS
- <api_handle>.METHODS.SPECIFIC.REMOVE_EVENTS

The methods are available for the following blocks:

- Setup block

> **Tip**
>
> Use the above methods for handling ECU events if no database file is used in the related Setup block.

ECU event template

An ECU event template supports you when you handle ECU events. The following listings show the structure of the ECU event templates for XCP-based, CCP-based, and dSPACE Calibration and Bypassing Service-based ECU interfaces.

```
xcp_temp struct
{
   EVENT_CHANNEL_NAME char
   EVENT_CHANNEL_NUMBER double
   MAX_DAQ_LIST double
   TIME_CYCLE double
   TIME_UNIT double
   PRIORITY double
}
```

```
ccp_temp struct
{
   EVENT_CHANNEL_NAME char
   EVENT_CHANNEL_NUMBER double
   PERIOD_DEFINITION double
   SAMPLE_STATE double
}
```

```
dspace_temp struct
{
   EVENT_CHANNEL_LONG_NAME char
   SERVICE_ID double
   EVENT_PERIOD double
   ADDRESS_EXTENSION double
}
```

GET_EVENTS

Description Returns the available ECU events of the Setup block.

Examples

```
events = rtibypassexecute(setup_api, setup_api.METHODS.SPECIFIC.GET_EVENTS);
event_1_tmp = events{1};
```

Parameters

Parameter	Type	Description
Output	cell({events cell{event_tmp struct}})[1]	The `rtibypassexecute` command returns a list of ECU events.

[1] For information on the structure of the event_tmp struct, refer to ECU event template on page 58.

GET_EVENT_TEMPLATE

Description Returns the ECU event template that you can use for creating custom ECU events. Different ECU event templates are returned for XCP-based, CCP-based, and dSPACE Calibration and Bypassing Service-based ECU interfaces.

Examples

```
event_temp = rtibypassexecute(setup_api, setup_api.METHODS.SPECIFIC.GET_EVENT_TEMPLATE);
```

Parameters

Parameter	Type	Description
Output	event_temp struct[1]	The `rtibypassexecute` command returns a struct that depends on the selected ECU interface with the available ECU event properties.

[1] For information on the structure of the event_temp struct, refer to ECU event template on page 58.

ADD_EVENTS

Description Adds ECU events to the Setup block.

Examples

```
event_temp = rtibypassexecute(setup_api, setup_api.METHODS.SPECIFIC.GET_EVENT_TEMPLATE);
event_temp.EVENT_CHANNEL_NAME = 'myEvent';
event_temp.EVENT_CHANNEL_NUMBER = 1;
success = rtibypassexecute(setup_api, setup_api.METHODS.SPECIFIC.ADD_EVENTS, {event_temp});
```

Parameters

Parameter	Type	Description
Input	{event_temp struct}[1]	List of ECU events to be added.
Output	logical	The `rtibypassexecute` command returns the status of the operation.

[1] For information on the structure of the event_temp struct, refer to ECU event template on page 58.

REMOVE_EVENTS

Description Removes ECU events from a Setup block.

Examples

```
events = rtibypassexecute(setup_api, setup_api.METHODS.SPECIFIC.GET_EVENTS);
event_1 = events{1};
success = rtibypassexecute(setup_api, setup_api.METHODS.SPECIFIC.REMOVE_EVENTS, {event_1});
```

Parameters

Parameter	Type	Description
Input	cell{event_temp struct}[1]	List of ECU events to be removed.
Output	logical	The `rtibypassexecute` command returns the status of the operation.

[1] For information on the structure of the event_temp struct, refer to ECU event template on page 58.

Related topics

Basics

Working with RTI Bypass API Handles..21

Flash Project Root Selection

Purpose

To configure the working folder and the flash projects to be used for flashing the ECU application.

Note

This use case is available for the following ECU interfaces:

- `'internal'`

Methods

You can use the following methods to configure the working folder and the flash projects to be used:

- <api_handle>.METHODS.SPECIFIC.SET_DSPACE_FLASH_PROJECT_ROOT
- <api_handle>.METHODS.SPECIFIC.GET_ALL_DSPACE_FLASH_PROJECTS
- <api_handle>.METHODS.SPECIFIC.SET_DSPACE_FLASH_PROJECT

The methods are available for the following blocks:

- Setup block

SET_DSPACE_FLASH_PROJECT_ROOT

Description Sets the working folder which contains the flash project you want to use for flashing the ECU application with the dSPACE ECU Flash Programming Tool.

Examples

```
projcet_root = {'e:\work\Flashing'};
success = rtibypassexecute(setup_api, setup_api.METHODS.SPECIFIC.SET_DSPACE_FLASH_PROJECT_ROOT, project_root);
```

Parameters

Parameter	Type	Description
Input	char	Working folder for the flash projects to be used.
Output	logical	The `rtibypassexecute` command returns the status of the operation.

GET_ALL_DSPACE_FLASH_PROJECTS

Description Returns all the flash projects that are located in a flash project root folder.

Examples

```
projects = rtibypassexecute(setup_api, setup_api.METHODS.SPECIFIC.GET_ALL_DSPACE_FLASH_PROJECTS);
project_1 = projects{1};
```

Parameters

Parameter	Type	Description
Output	cell({char})	The `rtibypassexecute` command returns a list of all the flash projects located in the working folder.

SET_DSPACE_FLASH_PROJECT

Description Sets the flash project to be used.

Examples

```
projects = rtibypassexecute(setup_api, setup_api.METHODS.SPECIFIC.GET_ALL_DSPACE_FLASH_PROJECTS);
project_1 = projects{1};
success = rtibypassexecute(setup_api, setup_api.METHODS.SPECIFIC.SET_DSPACE_FLASH_PROJECT, project_1);
```

Parameters

Parameter	Type	Description
Input	char	Flash project name
Output	logical	The `rtibypassexecute` command returns the status of the operation.

Related topics

Basics

Working with RTI Bypass API Handles..21

Free Memory Segment Handling

Purpose

To handle the free memory segment definitions of a Setup block.

> **Note**
>
> This use case is available for the following ECU interfaces:
> - `'internal'`

Methods

You can use the following methods to handle the free memory segments:

- <api_handle>.METHODS.SPECIFIC.GET_FREE_MEMORY_SEGMENTS
- <api_handle>.METHODS.SPECIFIC.GET_FREE_MEMORY_SEGMENT_TEMPLATE
- <api_handle>.METHODS.SPECIFIC.ADD_FREE_MEMORY_SEGMENTS
- <api_handle>.METHODS.SPECIFIC.REMOVE_FREE_MEMORY_SEGMENTS
- <api_handle>.METHODS.SPECIFIC.UPDATE_FREE_MEMORY_SEGMENTS

The methods are available for the following blocks:

- Setup block

Free memory segment template

A free memory segment template helps you handle free memory segments. The following listing shows the structure of the free memory segment template.

```
free_segment_temp struct
{
   START_ADDRESS char
   LENGTH char
   TYPE char
   CODE double
   PARAMETER double
   VARIABLE double
   SELECTED double
}
```

GET_FREE_MEMORY_SEGMENTS

Description Returns the available free memory segments of the Setup block.

Examples

```
free_memory_segments = rtibypassexecute(setup_api, setup_api.METHODS.SPECIFIC.GET_FREE_MEMORY_SEGMENTS);
segment_1 = free_memory_segments{1};
```

Parameters

Parameter	Type	Description
Output	cell{free_segment_temp struct}[1]	The `rtibypassexecute` command returns a list of free memory segments.

[1] For information on the structure of the free_segment_temp struct, refer to Free memory segment template on page 62.

GET_FREE_MEMORY_SEGMENT_TEMPLATE

Description Returns the free memory segment template that you can use for creating custom free memory segments.

Examples

```
segment_1 = rtibypassexecute(setup_api, setup_api.METHODS.SPECIFIC.GET_FREE_MEMORY_SEGMENT_TEMPLATE);
segment_1.START_ADDRESS = '0xA10115CC';
segment_1.LENGTH = '0x00013A34';
segment_1.TYPE = 'FLASH';
segment_1.CODE = 1;
segment_1.PARAMETER = 0;
segment_1.VARIABLE = 0;
segment_1.SELECTED = 1;
```

Parameters

Parameter	Type	Description
Output	free_segment_temp struct[1]	The `rtibypassexecute` command returns a struct with the memory segment properties.

[1] For information on the structure of the free_segment_temp struct, refer to Free memory segment template on page 62.

ADD_FREE_MEMORY_SEGMENTS

Description Adds free memory segments to the Setup block.

Examples

```
segment_1 = rtibypassexecute(setup_api, setup_api.METHODS.SPECIFIC.GET_FREE_MEMORY_SEGMENT_TEMPLATE);
segment_1.START_ADDRESS = '0xA10115CC';
segment_1.LENGTH = '0x00013A34';
segment_1.TYPE = 'FLASH';
segment_1.CODE = 1;
segment_1.PARAMETER = 0;
segment_1.VARIABLE = 0;
segment_1.SELECTED = 1;
success = rtibypassexecute(setup_api, setup_api.METHODS.SPECIFIC.ADD_FREE_MEMORY_SEGMENTS, {segment_1});
```

Parameters

Parameter	Type	Description
Input	{free_segment_temp struct}[1]	List of free memory segments to be added.
Output	logical	The `rtibypassexecute` command returns the status of the operation.

[1] For information on the structure of the free_segment_temp struct, refer to Free memory segment template on page 62.

REMOVE_FREE_MEMORY_SEGMENTS

Description Removes free memory segments from the Setup block.

Examples

```
free_memory_segments = rtibypassexecute(setup_api, setup_api.METHODS.SPECIFIC.GET_FREE_MEMORY_SEGMENTS);
segment_1 = free_memory_segments{1};
success = rtibypassexecute(setup_api, setup_api.METHODS.SPECIFIC.REMOVE_FREE_MEMORY_SEGMENTS, {segment_1});
```

Parameters

Parameter	Type	Description
Input	cell{free_segment_temp struct}[1]	List of free memory segments to be removed.
Output	logical	The `rtibypassexecute` command returns the status of the operation.

[1] For information on the structure of the free_segment_temp struct, refer to Free memory segment template on page 62.

UPDATE_FREE_MEMORY_SEGMENTS

Description Updates the free memory segments of the Setup block. You can also use this function to select/deselect memory segments.

Examples

```
old_memory_segments = rtibypassexecute(setup_api, setup_api.METHODS.SPECIFIC.GET_FREE_MEMORY_SEGMENTS);
update_segments = old_memory_segments;
segment_1 = old_memory_segments{1};
segment_2 = update_segments{1};
segment_2.SELECTED = false;
success = rtibypassexecute(setup_api, setup_api.METHODS.SPECIFIC.UPDATE_FREE_MEMORY_SEGMENTS, {segment_1}, {segment_2});
```

Parameters

Parameter	Type	Description
Input	cell{free_segment_temp struct}[1]	List of free memory segments. The free memory segments from the list are updated.
Input	cell{free_segment_temp struct}[1]	Second list of free memory segments. Memory segments from the first list are replaced with memory segments from the second list. Memory segments with equal indices are replaced.
Output	logical	The `rtibypassexecute` command returns the status of the operation.

[1] For information on the structure of the free_segment_temp struct, refer to Free memory segment template on page 62.

Related topics

Basics

Working with RTI Bypass API Handles..21

Memory Segment Handling

Purpose

To handle memory segment definitions of a Setup block.

Note

This use case is available for XCP-based ECU interfaces only. Select a matching interface via the Setup block's ECU Interface Selection generic use case.

Methods

You can use the following methods to handle memory segments:

- <api_handle>.METHODS.SPECIFIC.GET_MEMORY_SEGMENTS
- <api_handle>.METHODS.SPECIFIC.GET_MEMORY_SEGMENT_TEMPLATE
- <api_handle>.METHODS.SPECIFIC.ADD_MEMORY_SEGMENTS
- <api_handle>.METHODS.SPECIFIC.REMOVE_MEMORY_SEGMENTS

The methods are available for the following blocks:

- Setup block

Tip

Use the above methods for handling memory segments if no database file is used in the related Setup block.

Memory segment template

A memory segment template supports you when you handle memory segments. The following listing shows the structure of the memory segment template for XCP-based ECU interfaces.

```
xcp_temp struct
{
   SEGMENT_NUMBER double
   NUMBER_OF_PAGES double
   ADDRESS_EXTENSION double
   COMPRESSION_METHOD double
   ENCRYPTION_METHOD double
   CHECKSUM double
}
```

GET_MEMORY_SEGMENTS

Description Returns the available memory segments of the Setup block.

Examples

```
segments = rtibypassexecute(setup_api, setup_api.METHODS.SPECIFIC.GET_MEMORY_SEGMENTS);
segment_1 = segments{1};
```

Parameters

Parameter	Type	Description
Output	cell({segments cell{segment_tmp struct}})[1]	The `rtibypassexecute` command returns a list of memory segments.

[1] For information on the structure of the segment_tmp struct, refer to Memory segment template on page 65.

GET_MEMORY_SEGMENT_TEMPLATE

Description Returns the memory segment template that you can use for creating custom memory segments.

Examples

```
segment_1 = rtibypassexecute(setup_api, setup_api.METHODS.SPECIFIC.GET_MEMORY_SEGMENT_TEMPLATE);
segment_1.SEGMENT_NUMBER = 1;
segment_1.NUMBER_OF_PAGES = 5;
```

Parameters

Parameter	Type	Description
Output	segment_temp struct[1]	The `rtibypassexecute` command returns a struct that depends on the selected ECU interface with the available memory segment properties.

[1] For information on the structure of the segment_temp struct, refer to Memory segment template on page 65.

ADD_MEMORY_SEGMENTS

Description Adds memory segments to the Setup block.

Examples

```
segment_1 = rtibypassexecute(setup_api, setup_api.METHODS.SPECIFIC.GET_MEMORY_SEGMENT_TEMPLATE);
segment_1.SEGMENT_NUMBER = 1;
segment_1.NUMBER_OF_PAGES = 5;
success = rtibypassexecute(setup_api, setup_api.METHODS.SPECIFIC.ADD_MEMORY_SEGMENTS, {segment_1});
```

Parameters

Parameter	Type	Description
Input	{segment_temp struct}[1]	List of memory segments to be added.
Output	logical	The `rtibypassexecute` command returns the status of the operation.

[1] For information on the structure of the segment_temp struct, refer to Memory segment template on page 65.

REMOVE_MEMORY_SEGMENTS

Description Removes memory segments from a Setup block.

Examples

```
segments = rtibypassexecute(setup_api, setup_api.METHODS.SPECIFIC.GET_MEMORY_SEGMENTS);
segment_1 = segments{1};
success = rtibypassexecute(setup_api, setup_api.METHODS.SPECIFIC.REMOVE_MEMORY_SEGMENTS, {segment_1});
```

Parameters

Parameter	Type	Description
Input	cell{segment_temp struct}[1]	List of memory segments to be removed.
Output	logical	The `rtibypassexecute` command returns the status of the operation.

[1] For information on the structure of the segment_temp struct, refer to Memory segment template on page 65.

Related topics

Basics

Working with RTI Bypass API Handles..21

Memory Segment Page Handling

Purpose

To handle memory segment page definitions of a Setup block.

> **Note**
>
> This use case is available for XCP-based ECU interfaces only. Select a matching interface via the Setup block's ECU Interface Selection generic use case.

Methods

You can use the following methods to handle memory segment pages:

- <api_handle>.METHODS.SPECIFIC.GET_PAGES
- <api_handle>.METHODS.SPECIFIC.GET_PAGE_TEMPLATE
- <api_handle>.METHODS.SPECIFIC.ADD_PAGES
- <api_handle>.METHODS.SPECIFIC.REMOVE_PAGES

The methods are available for the following blocks:

- Setup block

> **Tip**
>
> Use the above methods for handling memory segment pages if no database file is used in the related Setup block.

Memory segment page template

A memory segment page template supports you when you handle memory segment pages. The following listing shows the structure of the memory segment page template for XCP-based ECU interfaces.

```
xcp_temp struct
{
   PAGE_NUMBER double
   ECU_ACCESS_TYPE double
   XCP_READ_ACCESS_TYPE double
   XCP_WRITE_ACCESS_TYPE double
}
```

You can specify the following values:

Parameter	Value	Description
ECU_ACCESS_TYPE	0	ECU_ACCESS_NOT_ALLOWED
	1	ECU_ACCESS_WITHOUT_XCP_ONLY
	2	ECU_ACCESS_WITH_XCP_ONLY
	3	ECU_ACCESS_DONT_CARE
XCP_READ_ACCESS_TYPE	0	XCP_READ_ACCESS_NOT_ALLOWED
	1	XCP_READ_ACCESS_WITHOUT_ECU_ONLY
	2	XCP_READ_ACCESS_WITH_ECU_ONLY
	3	XCP_READ_ACCESS_DONT_CARE
XCP_WRITE_ACCESS_TYPE	0	XCP_WRITE_ACCESS_NOT_ALLOWED
	1	XCP_WRITE_ACCESS_WITHOUT_ECU_ONLY
	2	XCP_WRITE_ACCESS_WITH_ECU_ONLY
	3	XCP_WRITE_ACCESS_DONT_CARE

GET_PAGES

Description Returns the available memory segment pages of the Setup block.

Examples

```
segments = rtibypassexecute(setup_api, setup_api.METHODS.SPECIFIC.GET_MEMORY_SEGMENTS);
segment_nr = segments{1}.SEGMENT_NUMBER;
pages = rtibypassexecute(setup_api, setup_api.METHODS.SPECIFIC.GET_PAGES, segment_nr);
page_1_tmp = pages{1};
```

Parameters

Parameter	Type	Description
Input	segment_nr double	The number of the memory segment you want to get the pages from.
Output	cell({pages cell{page_tmp struct}})[1]	The **rtibypassexecute** command returns a list of memory segment pages.

[1] For information on the structure of the page_tmp struct, refer to Memory segment page template on page 67.

GET_PAGE_TEMPLATE

Description Returns the memory segment page template that you can use for creating custom memory segment pages.

Examples

```
page_temp = rtibypassexecute(setup_api, setup_api.METHODS.SPECIFIC.GET_PAGE_TEMPLATE);
page_temp.PAGE_NUMBER = 1;
page_temp.ECU_ACCESS_TYPE = 1;
```

Parameters

Parameter	Type	Description
Output	page_temp struct[1]	The `rtibypassexecute` command returns a struct that depends on the selected ECU interface with the available memory segment page properties.

[1] For information on the structure of the page_temp struct, refer to Memory segment page template on page 67.

ADD_PAGES

Description Adds memory segment pages to the Setup block.

Examples

```
page_temp = rtibypassexecute(setup_api, setup_api.METHODS.SPECIFIC.GET_PAGE_TEMPLATE);
page_temp.PAGE_NUMBER = 1;
page_temp.ECU_ACCESS_TYPE = 1;
segments = rtibypassexecute(setup_api, setup_api.METHODS.SPECIFIC.GET_MEMORY_SEGMENTS);
segment_nr = segments{1}.SEGMENT_NUMBER;
success = rtibypassexecute(setup_api, setup_api.METHODS.SPECIFIC.ADD_MEMORY_SEGMENTS, segment_nr, {page_temp});
```

Parameters

Parameter	Type	Description
Input	segment_nr double	The number of the memory segment you want to add pages to.
Input	{segment_temp struct}[1]	List of memory segment pages to be added.
Output	logical	The `rtibypassexecute` command returns the status of the operation.

[1] For information on the structure of the page_temp struct, refer to Memory segment page template on page 67.

REMOVE_PAGES

Description Removes memory segment pages from a Setup block.

Examples

```
segments = rtibypassexecute(setup_api, setup_api.METHODS.SPECIFIC.GET_MEMORY_SEGMENTS);
segment_nr = segments{1}.SEGMENT_NUMBER;
pages = rtibypassexecute(setup_api, setup_api.METHODS.SPECIFIC.GET_PAGES, segment_nr);
page_1 = pages{1};
success = rtibypassexecute(setup_api, setup_api.METHODS.SPECIFIC.REMOVE_PAGES, segment_nr, {page_1});
```

Parameters

Parameter	Type	Description
Input	segment_nr double	The number of the memory segment you want to remove pages from.
Input	cell{page_temp struct}[1]	List of memory segment pages to be deleted.
Output	logical	The `rtibypassexecute` command returns the status of the operation.

[1] For information on the structure of the page_temp struct, refer to Memory segment page template on page 67.

Related topics

Basics

Working with RTI Bypass API Handles..21

Memory Tag Handling

Purpose

To handle memory tag definitions used to label the ECU application's binary content.

> **Note**
>
> This use case is available for the following ECU interfaces:
> - `'internal'`

Methods

You can use the following methods to handle memory tags:

- <api_handle>.METHODS.SPECIFIC.GET_MEMORY_TAGS
- <api_handle>.METHODS.SPECIFIC.GET_MEMORY_TAG_TEMPLATE
- <api_handle>.METHODS.SPECIFIC.ADD_MEMORY_TAGS
- <api_handle>.METHODS.SPECIFIC.REMOVE_MEMORY_TAGS

The methods are available for the following blocks:

- Setup block

Memory tag template

A memory tag template supports you when you handle memory tags. The following listing shows the structure of the memory tag template.

```
memory_tag_temp struct
{
   START_ADDRESS char
   TAG_DATA char
   LENGTH char
}
```

For further information on the elements of the structure, refer to MEMORY_TAG (Interface Description Data Reference 📖).

GET_MEMORY_TAGS

Description Returns the available memory tags of the Setup block.

Examples

```
memory_tags = rtibypassexecute(setup_api, setup_api.METHODS.SPECIFIC.GET_MEMORY_TAGS);
mem_tag_1 = memory_tags{1};
```

Parameters

Parameter	Type	Description
Output	cell({memory_tags cell{memory_tag_temp struct}})[1]	The `rtibypassexecute` command returns a list of memory tags.

[1] For information on the structure of the memory_tag_temp struct, refer to Memory tag template on page 70.

GET_MEMORY_TAG_TEMPLATE

Description Returns the memory tag template that you can use for creating a custom memory tag.

Examples

```
mem_tag_1 = rtibypassexecute(setup_api, setup_api.METHODS.SPECIFIC.GET_MEMORY_TAG_TEMPLATE);
mem_tag_1.START_ADDRESS = '0xA20200004';
mem_tag_1.TAG_DATA = '$xaa$x55$x66';
```

Parameters

Parameter	Type	Description
Output	memory_tag_temp struct[1]	The `rtibypassexecute` command returns a struct with the available memory tag properties.

[1] For information on the structure of the memory_tag_temp struct, refer to Memory tag template on page 70.

ADD_MEMORY_TAGS

Description Adds memory tags to the Setup block.

Examples

```
mem_tag_1 = rtibypassexecute(setup_api, setup_api.METHODS.SPECIFIC.GET_MEMORY_TAG_TEMPLATE);
mem_tag_1.START_ADDRESS = '0xA20200004';
mem_tag_1.TAG_DATA = '$xaa$x55$x66';
success = rtibypassexecute(setup_api, setup_api.METHODS.SPECIFIC.ADD_MEMORY_TAGS, {mem_tag_1});
```

Parameters

Parameter	Type	Description
Input	{memory_tag_temp struct}[1]	List of memory tags to be added.
Output	logical	The `rtibypassexecute` command returns the status of the operation.

[1] For information on the structure of the memory_tag_temp struct, refer to Memory tag template on page 70.

REMOVE_MEMORY_TAGS

Description Removes memory tags.

Examples

```
memory_tags = rtibypassexecute(setup_api, setup_api.METHODS.SPECIFIC.GET_MEMORY_TAGS);
mem_tag_1 = memory_tags{1};
success = rtibypassexecute(setup_api, setup_api.METHODS.SPECIFIC.REMOVE_MEMORY_TAGS, {mem_tag_1});
```

Parameters

Parameter	Type	Description
Input	cell{memory_tag_temp struct}[1)]	List of memory tags to be removed.
Output	logical	The `rtibypassexecute` command returns the status of the operation.

[1)] For information on the structure of the memory_tag_temp struct, refer to Memory tag template on page 70.

Related topics

Basics

Parameters

Where to go from here

Information in this section

Generic Parameters

Where to go from here

Information in this section

BuildExternalBypass

Access	You can access this parameter via the `<api_handle>.PARAMS.GENERIC.BuildExternalBypass` api handle path. The parameter is available for the following blocks: ▪ Build block
Description	Whether to auto build the external bypass model.
Parameter type	logical
Possible values	You can specify the following values: ▪ false ▪ true
User interface	For information on specifying the BuildExternalBypass block parameter interactively, refer to: ▪ RTI BYPASS BUILD Block (RTI Bypass Blockset Reference 🕮)

BuildInternalBypass

Access	You can access this parameter via the `<api_handle>.PARAMS.GENERIC.BuildInternalBypass` api handle path. The parameter is available for the following blocks: ▪ Build block
Description	Whether to auto build the selected internal bypass models.
Parameter type	logical
Possible values	You can specify the following values: ▪ false ▪ true

User interface

For information on specifying the BuildInternalBypass block parameter interactively, refer to:

- RTI BYPASS BUILD Block (RTI Bypass Blockset Reference 🕮)

BypassInterfaceName

Access

You can access this parameter via the `<api_handle>.PARAMS.GENERIC.BypassInterfaceName` api handle path.

The parameter is available for the following blocks:

- Calpageswitch block
- Download block
- Function block
- Interrupt block
- Read block
- Setup block
- Upload block
- Write block

Description

The bypass interface name of the corresponding setup block. Should be changed only via the setup block.

Parameter type

char

User interface

For information on specifying the BypassInterfaceName block parameter interactively, refer to:

- Unit Page (RTIBYPASS_CAL_PAGE_SWITCH_BLx) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_DOWNLOAD_BLx) (RTI Bypass Blockset Reference 🕮)
- Unit Page (RTIBYPASS_INTERRUPT_BLx) (RTI Bypass Blockset Reference 🕮)
- Unit Page (RTIBYPASS_DOWNLOAD_BLx) (RTI Bypass Blockset Reference 🕮)
- Unit Page (RTIBYPASS_SETUP_BLx) (RTI Bypass Blockset Reference 🕮)
- Unit Page (RTIBYPASS_UPLOAD_BLx) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_WRITE_BLx) (RTI Bypass Blockset Reference 🕮)
- Variables Page (RTIBYPASS_FUNCTION_BLx) (RTI Bypass Blockset Reference 🕮)

CleanExternalBypass

Access

You can access this parameter via the `<api_handle>.PARAMS.GENERIC.CleanExternalBypass` api handle path.

The parameter is available for the following blocks:

- Build block

Description

Whether to auto clean generated external bypass files.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the CleanExternalBypass block parameter interactively, refer to:

- RTI BYPASS BUILD Block (RTI Bypass Blockset Reference 📖)

CleanInternalBypass

Access

You can access this parameter via the `<api_handle>.PARAMS.GENERIC.CleanInternalBypass` api handle path.

The parameter is available for the following blocks:

- Build block

Description

Whether to auto clean generated internal bypass files.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the CleanInternalBypass block parameter interactively, refer to:

- RTI BYPASS BUILD Block (RTI Bypass Blockset Reference 📖)

ClearSearchBuffer

Access

You can access this parameter via the `<api_handle>.PARAMS.GENERIC.ClearSearchBuffer` api handle path.

The parameter is available for the following blocks:

- Download block
- Read block
- Upload block
- Write block

Description

With this option enabled, the search buffer is cleared automatically after you select variables from the variable list.

This parameter is used only within the UI.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the ClearSearchBuffer block parameter interactively, refer to:

- Options Page (RTIBYPASS_DOWNLOAD_BLx) (RTI Bypass Blockset Reference 📖)
- Options Page (RTIBYPASS_READ_BLx) (RTI Bypass Blockset Reference 📖)
- Options Page (RTIBYPASS_UPLOAD_BLx) (RTI Bypass Blockset Reference 📖)
- Options Page (RTIBYPASS_WRITE_BLx) (RTI Bypass Blockset Reference 📖)

CreateValidPortNames

Access

You can access this parameter via the `<api_handle>.PARAMS.GENERIC.CreateValidPortNames` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Whether to create unique and valid signal names for all blocks that have the ''Set output ports signal label'' option enabled.

Unique and valid signal names are required for generating Simulink objects.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the CreateValidPortNames block parameter interactively, refer to:

- Variables Options Page (RTIBYPASS_SETUP_BLx) (RTI Bypass Blockset Reference 📖)

DeleteUnusedDDs

Access

You can access this parameter via the `<api_handle>.PARAMS.GENERIC.DeleteUnusedDDs` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Whether to delete unused data dictionary files.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the DeleteUnusedDDs block parameter interactively, refer to:

- DD Options Page (RTIBYPASS_SETUP_BLx) (RTI Bypass Blockset Reference 🕮)

EnableBlockMode

Access

You can access this parameter via the `<api_handle>.PARAMS.GENERIC.EnableBlockMode` api handle path.

The parameter is available for the following blocks:

- Download block
- Read block
- Upload block
- Write block

Description

Whether to enable the block transfer mode for dynamic variables.

Instead of defining each dynamic variable individually, you can use the block mode to read/write the Count number of variables from the specified start address.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the EnableBlockMode block parameter interactively, refer to:

- Options Page (RTIBYPASS_DOWNLOAD_BLx) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_READ_BLx) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_UPLOAD_BLx) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_WRITE_BLx) (RTI Bypass Blockset Reference 🕮)

EnableCalibrationPagePort

Access

You can access this parameter via the `<api_handle>.PARAMS.GENERIC.EnableCalibrationPagePort` api handle path.

The parameter is available for the following blocks:

- Calpageswitch block

Description

Whether to enable the Calibration page port.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the EnableCalibrationPagePort block parameter interactively, refer to:

- Unit Page (RTIBYPASS_CAL_PAGE_SWITCH_BLx) (RTI Bypass Blockset Reference 🕮)

EnableConversionPort

Access

You can access this parameter via the `<api_handle>.PARAMS.GENERIC.EnableConversionPort` api handle path.

The parameter is available for the following blocks:

- Download block
- Read block
- Upload block
- Write block

Description

Whether to enable the ExtVarConversion port.

This port allows you to define individual conversion methods for each dynamic variable.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the EnableConversionPort block parameter interactively, refer to:

- Options Page (RTIBYPASS_DOWNLOAD_BLx) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_READ_BLx) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_UPLOAD_BLx) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_WRITE_BLx) (RTI Bypass Blockset Reference 🕮)

EnableExtVars

Access

You can access this parameter via the `<api_handle>.PARAMS.GENERIC.EnableExtVars` api handle path.

The parameter is available for the following blocks:

- Download block
- Read block
- Upload block
- Write block

Description

Whether to enable the ExtVarCount, ExtVarAddress and ExtVarType ports to use dynamic variables.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the EnableExtVars block parameter interactively, refer to:

- Options Page (RTIBYPASS_DOWNLOAD_BLx) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_READ_BLx) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_UPLOAD_BLx) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_WRITE_BLx) (RTI Bypass Blockset Reference 🕮)

EnablePageStatusPort

Access

You can access this parameter via the `<api_handle>.PARAMS.GENERIC.EnablePageStatusPort` api handle path.

The parameter is available for the following blocks:

- Calpageswitch block

Description

Whether to enable the page status.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the EnablePageStatusPort block parameter interactively, refer to:

- Unit Page (RTIBYPASS_CAL_PAGE_SWITCH_BLx) (RTI Bypass Blockset Reference 🕮)

EnableServicePort

Access

You can access this parameter via the `<api_handle>.PARAMS.GENERIC.EnableServicePort` api handle path.

The parameter is available for the following blocks:

- Download block
- Function block
- Read block
- Upload block
- Write block

Description

Whether to enable the Enable Service port.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the EnableServicePort block parameter interactively, refer to:

- Options Page (RTIBYPASS_DOWNLOAD_BLx) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_READ_BLx) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_UPLOAD_BLx) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_WRITE_BLx) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_FUNCTION_BLx) (RTI Bypass Blockset Reference 🕮)

EnableStatusOutputPort

Access

You can access this parameter via the `<api_handle>.PARAMS.GENERIC.EnableStatusOutputPort` api handle path.

The parameter is available for the following blocks:

- Download block
- Read block
- Upload block
- Write block

Description

Whether to enable the Status port.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the EnableStatusOutputPort block parameter interactively, refer to:

- Options Page (RTIBYPASS_DOWNLOAD_BLx) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_READ_BLx) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_UPLOAD_BLx) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_WRITE_BLx) (RTI Bypass Blockset Reference 🕮)

EnableSwitchingPort

Access

You can access this parameter via the `<api_handle>.PARAMS.GENERIC.EnableSwitchingPort` api handle path.

The parameter is available for the following blocks:

- Calpageswitch block

Description

Whether to enable the Enable Switching port.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the EnableSwitchingPort block parameter interactively, refer to:

- Unit Page (RTIBYPASS_CAL_PAGE_SWITCH_BLx) (RTI Bypass Blockset Reference 🕮)

FillVarSel

Access

You can access this parameter via the `<api_handle>.PARAMS.GENERIC.FillVarSel` api handle path.

The parameter is available for the following blocks:

- Download block
- Read block
- Upload block
- Write block

Description

Fills the Variable Selector with the variables defined in the database files you imported for the selected bypass interface.

This parameter is used only within the UI.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the FillVarSel block parameter interactively, refer to:

- Options Page (RTIBYPASS_DOWNLOAD_BLx) (RTI Bypass Blockset Reference 📖)
- Options Page (RTIBYPASS_READ_BLx) (RTI Bypass Blockset Reference 📖)
- Options Page (RTIBYPASS_UPLOAD_BLx) (RTI Bypass Blockset Reference 📖)
- Options Page (RTIBYPASS_WRITE_BLx) (RTI Bypass Blockset Reference 📖)

MaxNrOfExtVars

Access

You can access this parameter via the `<api_handle>.PARAMS.GENERIC.MaxNrOfExtVars` api handle path.

The parameter is available for the following blocks:

- Download block
- Read block

- Upload block
- Write block

Description

Defines the maximum number of dynamic variables you can read/write.

Parameter type

double

User interface

For information on specifying the MaxNrOfExtVars block parameter interactively, refer to:

- Options Page (RTIBYPASS_DOWNLOAD_BLx) (RTI Bypass Blockset Reference 📖)
- Options Page (RTIBYPASS_READ_BLx) (RTI Bypass Blockset Reference 📖)
- Options Page (RTIBYPASS_UPLOAD_BLx) (RTI Bypass Blockset Reference 📖)
- Options Page (RTIBYPASS_WRITE_BLx) (RTI Bypass Blockset Reference 📖)

MeasurementsOnly

Access

You can access this parameter via the `<api_handle>.PARAMS.GENERIC.MeasurementsOnly` api handle path.

The parameter is available for the following blocks:

- Download block
- Read block
- Upload block
- Write block

Description

Whether to filter only the measurements.

This parameter is used only within the UI.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the MeasurementsOnly block parameter interactively, refer to:

- Options Page (RTIBYPASS_DOWNLOAD_BLx) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_READ_BLx) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_UPLOAD_BLx) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_WRITE_BLx) (RTI Bypass Blockset Reference 🕮)

PerformBlockDisableInForeground

Access

You can access this parameter via the `<api_handle>.PARAMS.GENERIC.PerformBlockDisableInForeground` api handle path.

The parameter is available for the following blocks:

- Download block
- Read block
- Upload block
- Write block

Description

Whether to place block disable code not only in the background, but additionally also in the foreground task.

Enable this flag if you notice a value is being written to the ECU and if the block was already disabled.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the PerformBlockDisableInForeground block parameter interactively, refer to:

- Options Page (RTIBYPASS_READ_BLx) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_UPLOAD_BLx) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_WRITE_BLx) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_DOWNLOAD_BLx) (RTI Bypass Blockset Reference 🕮)

PerformBlockEnableInForeground

Access

You can access this parameter via the `<api_handle>.PARAMS.GENERIC.PerformBlockEnableInForeground` api handle path.

The parameter is available for the following blocks:

- Download block
- Read block
- Upload block
- Write block

Description

Whether to place block enable code not only in the background, but additionally also in the foreground task.

Enable this flag if you notice that early values are not being written to the ECU and if the block was already enabled.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the PerformBlockEnableInForeground block parameter interactively, refer to:

- Options Page (RTIBYPASS_READ_BLx) (RTI Bypass Blockset Reference 📖)
- Options Page (RTIBYPASS_UPLOAD_BLx) (RTI Bypass Blockset Reference 📖)
- Options Page (RTIBYPASS_WRITE_BLx) (RTI Bypass Blockset Reference 📖)
- Options Page (RTIBYPASS_DOWNLOAD_BLx) (RTI Bypass Blockset Reference 📖)

PerformVarsCfgInForeground

Access

You can access this parameter via the `<api_handle>.PARAMS.GENERIC.PerformVarsCfgInForeground` api handle path.

The parameter is available for the following blocks:

- Download block
- Read block
- Upload block
- Write block

Description

Whether to perform the variable configuration in the foreground or background state of the block.

Enable this flag if you notice task overrun errors, when the attempt is made to read/write too many dynamic variables.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the PerformVarsCfgInForeground block parameter interactively, refer to:

- Options Page (RTIBYPASS_DOWNLOAD_BLx) (RTI Bypass Blockset Reference 📖)
- Options Page (RTIBYPASS_READ_BLx) (RTI Bypass Blockset Reference 📖)
- Options Page (RTIBYPASS_UPLOAD_BLx) (RTI Bypass Blockset Reference 📖)
- Options Page (RTIBYPASS_WRITE_BLx) (RTI Bypass Blockset Reference 📖)

SelectWithEnter

Access

You can access this parameter via the `<api_handle>.PARAMS.GENERIC.SelectWithEnter` api handle path.

The parameter is available for the following blocks:

- Download block
- Read block
- Upload block
- Write block

Description

Whether to select a variable with Enter.

This parameter is used only within the UI.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the SelectWithEnter block parameter interactively, refer to:

- Options Page (RTIBYPASS_DOWNLOAD_BLx) (RTI Bypass Blockset Reference 📖)
- Options Page (RTIBYPASS_READ_BLx) (RTI Bypass Blockset Reference 📖)
- Options Page (RTIBYPASS_UPLOAD_BLx) (RTI Bypass Blockset Reference 📖)
- Options Page (RTIBYPASS_WRITE_BLx) (RTI Bypass Blockset Reference 📖)

ServiceType

Access

You can access this parameter via the `<api_handle>.PARAMS.GENERIC.ServiceType` api handle path.

The parameter is available for the following blocks:

- Setup block

Parameter type

char

User interface

For information on specifying the ServiceType block parameter interactively, refer to:

- Unit Page (RTIBYPASS_SETUP_BLx) (RTI Bypass Blockset Reference 📖)

ShowHierarchical

Access

You can access this parameter via the `<api_handle>.PARAMS.GENERIC.ShowHierarchical` api handle path.

The parameter is available for the following blocks:

- Function block

Description

Whether to enable the hierarchical view on functions.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the ShowHierarchical block parameter interactively, refer to:

- Variables Page (RTIBYPASS_FUNCTION_BLx) (RTI Bypass Blockset Reference 🕮)

StartCleanDDs

Access

You can access this parameter via the `<api_handle>.PARAMS.GENERIC.StartCleanDDs` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

This parameter is only for the UI. It enables automatic searches for unused DDs every time the model is saved under a different name.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the StartCleanDDs block parameter interactively, refer to:

- DD Options Page (RTIBYPASS_SETUP_BLx) (RTI Bypass Blockset Reference 🕮)

SubIntLoc

Access

You can access this parameter via the `<api_handle>.PARAMS.GENERIC.SubIntLoc` api handle path.

The parameter is available for the following blocks:

- Interrupt block

Parameter type

char

Possible values

You can specify the following values:

- DSBYPASS_INTERRUPT_AFTER_READ
- DSBYPASS_INTERRUPT_AFTER_WRITE

User interface

For information on specifying the SubIntLoc block parameter interactively, refer to:

- Options Page (RTIBYPASS_INTERRUPT_BLx) (RTI Bypass Blockset Reference 📖)

SubIntNo

Access

You can access this parameter via the `<api_handle>.PARAMS.GENERIC.SubIntNo` api handle path.

The parameter is available for the following blocks:

- Interrupt block

Description

Specifies the type of the interrupt trigger (interrupt trigger location). It defines whether the related subsystem should be executed after execution of all the reads or writes of the selected service instance.

Parameter type

double

User interface

For information on specifying the SubIntNo block parameter interactively, refer to:

- Options Page (RTIBYPASS_INTERRUPT_BLx) (RTI Bypass Blockset Reference 📖)

UseDefaultConvert

Access

You can access this parameter via the `<api_handle>.PARAMS.GENERIC.UseDefaultConvert` api handle path.

The parameter is available for the following blocks:

- Download block
- Read block
- Upload block
- Write block

Description

With this option enabled, the ASAM MCD-2-defined conversion is enabled automatically after you select variables from the variable list.

This parameter is used only within the UI.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the UseDefaultConvert block parameter interactively, refer to:

- Options Page (RTIBYPASS_DOWNLOAD_BLx) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_READ_BLx) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_UPLOAD_BLx) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_WRITE_BLx) (RTI Bypass Blockset Reference 🕮)

UseDefaultPortDataType

Access

You can access this parameter via the `<api_handle>.PARAMS.GENERIC.UseDefaultPortDataType` api handle path.

The parameter is available for the following blocks:

- Download block
- Read block

- Upload block
- Write block

Description

With this option enabled, the port data type is automatically set as double after you select variables from the variable list.

This parameter is used only within the UI.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the UseDefaultPortDataType block parameter interactively, refer to:

- Options Page (RTIBYPASS_DOWNLOAD_BLx) (RTI Bypass Blockset Reference 📖)
- Options Page (RTIBYPASS_READ_BLx) (RTI Bypass Blockset Reference 📖)
- Options Page (RTIBYPASS_UPLOAD_BLx) (RTI Bypass Blockset Reference 📖)
- Options Page (RTIBYPASS_WRITE_BLx) (RTI Bypass Blockset Reference 📖)

UseDefaultRemapping

Access

You can access this parameter via the `<api_handle>.PARAMS.GENERIC.UseDefaultRemapping` api handle path.

The parameter is available for the following blocks:

- Read block
- Write block

Description

With this option enabled, the variable is remapped automatically after you select variables from the variable list.

This parameter is used only within the UI.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the UseDefaultRemapping block parameter interactively, refer to:

- Options Page (RTIBYPASS_READ_BLx) (RTI Bypass Blockset Reference 📖)
- Options Page (RTIBYPASS_WRITE_BLx) (RTI Bypass Blockset Reference 📖)

UseDispId

Access

You can access this parameter via the `<api_handle>.PARAMS.GENERIC.UseDispId` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Whether to use display identifiers.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the UseDispId block parameter interactively, refer to:

- Variables Options Page (RTIBYPASS_SETUP_BLx) (RTI Bypass Blockset Reference 📖)

UseRelativePath

Access

You can access this parameter via the `<api_handle>.PARAMS.GENERIC.UseRelativePath` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Whether the source file paths are stored relatively or absolutely.

Parameter type

logical

User interface

For information on specifying the UseRelativePath block parameter interactively, refer to:

- Unit Page (RTIBYPASS_SETUP_BLx) (RTI Bypass Blockset Reference 🕮)

VarNameAsPortLabel

Access

You can access this parameter via the `<api_handle>.PARAMS.GENERIC.VarNameAsPortLabel` api handle path.

The parameter is available for the following blocks:

- Download block
- Function block
- Read block
- Upload block
- Write block

Description

Whether to use the variable names as port names.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the VarNameAsPortLabel block parameter interactively, refer to:

- Options Page (RTIBYPASS_DOWNLOAD_BLx) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_READ_BLx) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_UPLOAD_BLx) (RTI Bypass Blockset Reference 🕮)

- Options Page (RTIBYPASS_WRITE_BLx) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_FUNCTION_BLx) (RTI Bypass Blockset Reference 🕮)

VarNameAsSignalLabel

Access

You can access this parameter via the `<api_handle>.PARAMS.GENERIC.VarNameAsSignalLabel` api handle path.

The parameter is available for the following blocks:

- Download block
- Function block
- Read block
- Upload block
- Write block

Description

Whether to use the variable names as signal labels.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the VarNameAsSignalLabel block parameter interactively, refer to:

- Unit Page (RTIBYPASS_DOWNLOAD_BLx) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_READ_BLx) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_UPLOAD_BLx) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_WRITE_BLx) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_FUNCTION_BLx) (RTI Bypass Blockset Reference 🕮)

Parameters for XCP-Based Interfaces

Where to go from here

Information in this section

ADDRESS

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.ADDRESS` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Value must be provided as an address string, e.g. 127.0.0.1

You can define the target (ECU) IP via either the HOST_NAME or the ADDRESS.

Note

The ADDRESS parameter can be specified via a database file.

Parameter type

char

ADDRESS_EXTENSION

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.ADDRESS_EXTENSION` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Address extension type.

0 (FREE): Address extension can be different within one and the same ODT.

1 (ODT): Address extension to be the same for all entries within one ODT.

3 (DAQ): Address extension to be the same for all entries within one DAQ.

Note

The ADDRESS_EXTENSION parameter can be specified via a database file.

Parameter type

double

Possible values

You can specify the following values:

- 0
- 1
- 3

ADDRESS_GRANULARITY

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.ADDRESS_GRANULARITY` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Address granularity

Values: 1=BYTE, 2=WORD, 4=DWORD

1: Addresses have BYTE (8-bit) granularity.

2: Addresses have WORD (16-bit) granularity.

4: Addresses have DWORD (32-bit) granularity.

> **Note**
>
> The ADDRESS_GRANULARITY parameter can be specified via a database file.

Parameter type

double

Possible values

You can specify the following values:

- 1
- 2
- 4

AssignmentType

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.AssignmentType` api handle path.

The parameter is available for the following blocks:

- Read block
- Write block

Description

Values: 1=Default, 2=ServiceInstance, 3=Manual

Parameter type

double

Possible values

You can specify the following values:

- 1
- 2
- 3

User interface

For information on specifying the AssignmentType block parameter interactively, refer to:

- Options Page (RTIBYPASS_READ_BLx for XCP on CAN) (RTI Bypass Blockset Reference 📖)
- Options Page (RTIBYPASS_WRITE_BLx for XCP on CAN) (RTI Bypass Blockset Reference 📖)
- Options Page (RTIBYPASS_READ_BLx for XCP on UDP/IP) (RTI Bypass Blockset Reference 📖)
- Options Page (RTIBYPASS_WRITE_BLx for XCP on UDP/IP) (RTI Bypass Blockset Reference 📖)

Autodecrement

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.Autodecrement` api handle path.

The parameter is available for the following blocks:

- Read block
- Write block

Description

Disables DAQ list priority auto decrement.

Can be set only if the assignment type is set to ServiceInstance or Manual.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the Autodecrement block parameter interactively, refer to:

- Options Page (RTIBYPASS_READ_BLx for XCP on CAN) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_WRITE_BLx for XCP on CAN) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_READ_BLx for XCP on UDP/IP) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_WRITE_BLx for XCP on UDP/IP) (RTI Bypass Blockset Reference 🕮)

BAUDRATE

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.BAUDRATE` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

CAN baud rate (in Hz).

This is an optional parameter.

> **Note**
>
> The BAUDRATE parameter can be specified via a database file.

Parameter type

double

BIT_STIM_SUPPORTED

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.BIT_STIM_SUPPORTED` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Support for bit stimulation.

Note

The BIT_STIM_SUPPORTED parameter can be specified via a database file.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

BTL_CYCLES

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.BTL_CYCLES` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

BTL cycles (in slots per bit time).

This is an optional parameter.

Note

The BTL_CYCLES parameter can be specified via a database file.

Parameter type

double

BYTE_ORDER

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.BYTE_ORDER` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Byte order of the XCP interface.

Values: 0=Last, 1=First

0: The XCP interface is little endian (MSB last).

1: The XCP interface is big endian (MSB first).

So far, the XCP interface endian must be identical to the ECU endian,

specified in the parameter dsbypass_service_init_struct_t::ecu_endian.

Note

The BYTE_ORDER parameter can be specified via a database file.

Parameter type

double

Possible values

You can specify the following values:

- 0
- 1

ByteOrder

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.ByteOrder` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Endian of the ECU (big or little)

BIG_ENDIAN : The ECU is big endian (MSB first).

LITTLE_ENDIAN : The ECU is little endian (MSB last).

> **Note**
>
> The ByteOrder parameter can be specified via a database file.

Parameter type

char

Possible values

You can specify the following values:

- LITTLE_ENDIAN
- BIG_ENDIAN

CAN_FD_MAX_DLC_REQUIRED

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.CAN_FD_MAX_DLC_REQUIRED` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Maximum DLC required on CAN FD frame data phase.

Master to slave frames.

Always use maximum data length (CAN FD frames -> MAX_DLC) to send the messages.

This is an optional parameter.

If CAN FD is disabled, this parameter will be ignored. Instead, use MAX_DLC_REQUIRED to configure the message size.

> **Note**
>
> The CAN_FD_MAX_DLC_REQUIRED parameter can be specified via a database file.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

can_id

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.can_id` api handle path.

The parameter is available for the following blocks:

- Read block
- Write block

Description

Defines the CAN ID used for communication. This parameter is evaluated if the related interface description (A2L) provides a static CAN ID configuration.

Parameter type

char

CAN_ID_BROADCAST

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.CAN_ID_BROADCAST` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

CAN ID for XCP broadcast messages.

Value must be provided as a HEX string.

> **Note**
>
> The CAN_ID_BROADCAST parameter can be specified via a database file.

Parameter type char

CAN_ID_MASTER

Access You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.CAN_ID_MASTER` api handle path.

The parameter is available for the following blocks:

- Setup block

Description CAN ID of the XCP master (here: the RCP system).

Value must be provided as a HEX string.

> **Note**
>
> The CAN_ID_MASTER parameter can be specified via a database file.

Parameter type char

can_id_pid_off

Access You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.can_id_pid_off` api handle path.

The parameter is available for the following blocks:

- Read block
- Write block

Description Can be set only if PIDTransmissionEnable is enabled.

Parameter type char

CAN_ID_SLAVE

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.CAN_ID_SLAVE` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

CAN ID of the XCP slave (here: the ECU).

Value must be provided as a HEX string.

Note

The CAN_ID_SLAVE parameter can be specified via a database file.

Parameter type

char

DAQ_ALTERNATING_SUPPORTED

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.DAQ_ALTERNATING_SUPPORTED` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Display_Event_Channel_Number.

This is an optional parameter.

Note

The DAQ_ALTERNATING_SUPPORTED parameter can be specified via a database file.

Parameter type

double

DAQ_CONFIG_TYPE

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.DAQ_CONFIG_TYPE` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

DAQ configuration.

0: Static

1: Dynamic

> **Note**
>
> The DAQ_CONFIG_TYPE parameter can be specified via a database file.

Parameter type

double

Possible values

You can specify the following values:

- 0
- 1

DAQListNo

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.DAQListNo` api handle path.

The parameter is available for the following blocks:

- Read block
- Write block

Description

Can be set only if the static DAQs are used.

Parameter type

char

Possible values

You can specify the following values:

- guibypass_xcp_on_can_readwrite(''GetPopupEntriesList'', dialogData, guibypassinterface(''PopupDAQListGet'', dialogData.simulink.hBlock,dialogData.param.blockparameters.BypassInterfaceName{1}, dialogData.param.blockparameters.ServiceInstanceStruct{1}.ServiceInstanceName))

User interface

For information on specifying the DAQListNo block parameter interactively, refer to:

- Options Page (RTIBYPASS_READ_BLx for XCP on CAN) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_WRITE_BLx for XCP on CAN) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_READ_BLx for XCP on UDP/IP) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_WRITE_BLx for XCP on UDP/IP) (RTI Bypass Blockset Reference 🕮)

DoubleBufferEnable

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.DoubleBufferEnable` api handle path.

The parameter is available for the following blocks:

- Read block
- Write block

Description

Can be set only if DoubleBuffer is supported by the A2L file.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the DoubleBufferEnable block parameter interactively, refer to:

- Options Page (RTIBYPASS_READ_BLx for XCP on CAN) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_WRITE_BLx for XCP on CAN) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_READ_BLx for XCP on UDP/IP) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_WRITE_BLx for XCP on UDP/IP) (RTI Bypass Blockset Reference 🕮)

DSpaceXcpVersion

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.DSpaceXcpVersion` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Version number of the IF_DATA dSPACE_XCP block.

Value must be provided as a HEX string.

> **Note**
>
> The DSpaceXcpVersion parameter can be specified via a database file.

Parameter type

char

ECUIP

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.ECUIP` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

This defines the target IP address.

Parameter type

char

User interface

For information on specifying the ECUIP block parameter interactively, refer to:

- Options Page (RTIBYPASS_SETUP_BLx for XCP on UDP/IP) (RTI Bypass Blockset Reference 🕮)

ECUSendsDAQPermanently

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.ECUSendsDAQPermanently` api handle path.

The parameter is available for the following blocks:

- Setup block

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the ECUSendsDAQPermanently block parameter interactively, refer to:

- Options Page (RTIBYPASS_SETUP_BLx for XCP on CAN) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_SETUP_BLx for XCP on UDP/IP) (RTI Bypass Blockset Reference 🕮)

EnablePageSwitching

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.EnablePageSwitching` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Can be set only if dynamic DAQs are used.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the EnablePageSwitching block parameter interactively, refer to:

- Options Page (RTIBYPASS_SETUP_BLx for XCP on CAN) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_SETUP_BLx for XCP on UDP/IP) (RTI Bypass Blockset Reference 🕮)

FailureLimit

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.FailureLimit` api handle path.

The parameter is available for the following blocks:

- Read block
- Write block

Description

Can be set only if DoubleBufferEnable is enabled.

Parameter type

char

User interface

For information on specifying the FailureLimit block parameter interactively, refer to:

- Options Page (RTIBYPASS_READ_BLx for XCP on CAN) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_WRITE_BLx for XCP on CAN) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_READ_BLx for XCP on UDP/IP) (RTI Bypass Blockset Reference 🕮)

- Options Page (RTIBYPASS_WRITE_BLx for XCP on UDP/IP) (RTI Bypass Blockset Reference 📖)

ForegroundDownload

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.ForegroundDownload` api handle path.

The parameter is available for the following blocks:

- Download block

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the ForegroundDownload block parameter interactively, refer to:

- Options Page (RTIBYPASS_DOWNLOAD_BLx for XCP on CAN) (RTI Bypass Blockset Reference 📖)
- Options Page (RTIBYPASS_DOWNLOAD_BLx for XCP on UDP/IP) (RTI Bypass Blockset Reference 📖)

GRANULARITY_ODT_ENTRY_SIZE_DAQ

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.GRANULARITY_ODT_ENTRY_SIZE_DAQ` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Granularity of ODT entries for data acquisition (DAQ).

1: ODT entries have BYTE (8-bit) granularity.

2: ODT entries have WORD (16-bit) granularity.

4: ODT entries have DWORD (32-bit) granularity.

8: ODT entries have DLONG (64-bit) granularity.

> **Note**
>
> The GRANULARITY_ODT_ENTRY_SIZE_DAQ parameter can be specified via a database file.

Parameter type

double

Possible values

You can specify the following values:

- 1
- 2
- 4
- 8

GRANULARITY_ODT_ENTRY_SIZE_STIM

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.GRANULARITY_ODT_ENTRY_SIZE_STIM` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Support for stimulation.

Granularity of ODT entries for data stimulation (STIM).

1: ODT entries have BYTE (8-bit) granularity

2: ODT entries have WORD (16-bit) granularity

4: ODT entries have DWORD (32-bit) granularity

8: ODT entries have DLONG (64-bit) granularity

> **Note**
>
> The GRANULARITY_ODT_ENTRY_SIZE_STIM parameter can be specified via a database file.

Parameter type

double

Possible values

You can specify the following values:

- 1
- 2
- 4
- 8

HOST_NAME

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.HOST_NAME` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Value must be provided as an address string, e.g. localhost

You can define the target (ECU) IP via either the HOST_NAME or the ADDRESS.

Note

The HOST_NAME parameter can be specified via a database file.

Parameter type

char

IDENTIFICATION_FIELD

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.IDENTIFICATION_FIELD` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Type of identification fields in DAQ packets.

0: DAQ packet identification by absolute ODT number.

1: DAQ packet identification by relative ODT number, absolute DAQ list number (BYTE).

2: DAQ packet identification by relative ODT number, absolute DAQ list number (WORD).

3: DAQ packet identification by relative ODT number, absolute DAQ list number (WORD, aligned).

> **Note**
>
> The IDENTIFICATION_FIELD parameter can be specified via a database file.

Parameter type

double

Possible values

You can specify the following values:

- 0
- 1
- 2
- 3

IFDataVersion

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.IFDataVersion` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Version of the XCP protocol layer.

Value must be provided as a double string.

> **Note**
>
> The IFDataVersion parameter can be specified via a database file.

Parameter type

char

IgnoreECUResourcesStatus

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.IgnoreECUResourcesStatus` api handle path.

The parameter is available for the following blocks:

- Setup block

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the IgnoreECUResourcesStatus block parameter interactively, refer to:

- Options Page (RTIBYPASS_SETUP_BLx for XCP on CAN) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_SETUP_BLx for XCP on UDP/IP) (RTI Bypass Blockset Reference 🕮)

InterfaceIP

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.InterfaceIP` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

This defines the source IP address.

Parameter type

char

User interface

For information on specifying the InterfaceIP block parameter interactively, refer to:

- Options Page (RTIBYPASS_SETUP_BLx for XCP on UDP/IP) (RTI Bypass Blockset Reference 🕮)

InterPacketGap

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.InterPacketGap` api handle path.

The parameter is available for the following blocks:

- Setup block

Parameter type

char

User interface

For information on specifying the InterPacketGap block parameter interactively, refer to:

- Options Page (RTIBYPASS_SETUP_BLx for XCP on CAN) (RTI Bypass Blockset Reference 📖)
- Options Page (RTIBYPASS_SETUP_BLx for XCP on UDP/IP) (RTI Bypass Blockset Reference 📖)

IntervalCheck

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.IntervalCheck` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Can be set only if EnablePageSwitching is enabled.

Parameter type

char

User interface

For information on specifying the IntervalCheck block parameter interactively, refer to:

- Options Page (RTIBYPASS_SETUP_BLx for XCP on CAN) (RTI Bypass Blockset Reference 📖)
- Options Page (RTIBYPASS_SETUP_BLx for XCP on UDP/IP) (RTI Bypass Blockset Reference 📖)

MAX_CTO

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.MAX_CTO` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Maximum length of an XCP command packet.

> **Note**
>
> The MAX_CTO parameter can be specified via a database file.

Parameter type

double

MAX_DAQ

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.MAX_DAQ` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Total number of available DAQ lists.

> **Note**
>
> The MAX_DAQ parameter can be specified via a database file.

Parameter type

double

MAX_DLC

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.MAX_DLC` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Maximum data length of CAN FD frames.

> **Note**
>
> The MAX_DLC parameter can be specified via a database file.

Parameter type

double

Possible values

You can specify the following values:

- 8
- 12
- 16
- 20
- 24
- 32
- 48
- 64

MAX_DLC_REQUIRED

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.MAX_DLC_REQUIRED` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Maximum DLC required.

Master to slave frames.

Use maximum size (8) for all CAN frames -> MAX_DLC_REQUIRED

This is an optional parameter.

If CAN FD is enabled, this parameter will be ignored. Instead, use CAN_FD_MAX_DLC_REQUIRED to configure the message size.

> **Note**
>
> The MAX_DLC_REQUIRED parameter can be specified via a database file.

Parameter type

logical

Possible values

You can specify the following values:
- false
- true

MAX_DTO

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.MAX_DTO` api handle path.

The parameter is available for the following blocks:
- Setup block

Description

Maximum length of an XCP data packet.

> **Note**
>
> The MAX_DTO parameter can be specified via a database file.

Parameter type

double

MAX_EVENT_CHANNEL

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.MAX_EVENT_CHANNEL` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Total number of available event channels.

Note

The MAX_EVENT_CHANNEL parameter can be specified via a database file.

Parameter type

double

MAX_ODT_ENTRY_SIZE_DAQ

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.MAX_ODT_ENTRY_SIZE_DAQ` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Maximum ODT entry size for data acquisition (DAQ).

Note

The MAX_ODT_ENTRY_SIZE_DAQ parameter can be specified via a database file.

Parameter type

double

MAX_ODT_ENTRY_SIZE_STIM

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.MAX_ODT_ENTRY_SIZE_STIM` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Maximum ODT entry size for data stimulation (STIM).

> **Note**
>
> The MAX_ODT_ENTRY_SIZE_STIM parameter can be specified via a database file.

Parameter type

double

MaximumBandwidth

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.MaximumBandwidth` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Indicates the interface maximum bandwidth in Mbps.

Parameter type

double

Possible values

You can specify the following values:

- 100
- 1000

User interface

For information on specifying the MaximumBandwidth block parameter interactively, refer to:

- Options Page (RTIBYPASS_SETUP_BLx for XCP on UDP/IP) (RTI Bypass Blockset Reference 🕮)

MIN_DAQ

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.MIN_DAQ` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Total number of predefined DAQ lists, unsigned 16-bit value.

> **Note**
>
> The MIN_DAQ parameter can be specified via a database file.

Parameter type

double

MIN_ST_STIM

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.MIN_ST_STIM` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Separation time between DTOs.

Time in units of 100 microseconds.

> **Note**
>
> The MIN_ST_STIM parameter can be specified via a database file.

Parameter type

double

OPTIMISATION_TYPE

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.OPTIMISATION_TYPE` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

ODT optimization type which the master should preferably use.

0: Default optimization type.

1: Optimization type 16-bit.

2: Optimization type 32-bit.

3: Optimization type 64-bit.

4: Optimization type alignment.

5: Optimization type max entry size.

> **Note**
>
> The OPTIMISATION_TYPE parameter can be specified via a database file.

Parameter type

double

Possible values

You can specify the following values:

- 0
- 1
- 2
- 3
- 4
- 5

OVERLOAD_INDICATION

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.OVERLOAD_INDICATION` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Overload indication type.

0: No overload indication

1: Overload indication in MSB of PID.

2: Overload indication by event packet

> **Note**
>
> The OVERLOAD_INDICATION parameter can be specified via a database file.

Parameter type

double

Possible values

You can specify the following values:

- 0
- 1
- 2

PageNumber

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.PageNumber` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Can be set only if EnablePageSwitching is enabled.

The maximum number of pages varies according to the selected segment number. You should therefore generate a new API if the segment changes.

Parameter type

double

User interface

For information on specifying the PageNumber block parameter interactively, refer to:

- Options Page (RTIBYPASS_SETUP_BLx for XCP on CAN) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_SETUP_BLx for XCP on UDP/IP) (RTI Bypass Blockset Reference 🕮)

PID_OFF_SUPPORTED

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.PID_OFF_SUPPORTED` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Support for PID off transmission.

This is an optional parameter.

> **Note**
>
> The PID_OFF_SUPPORTED parameter can be specified via a database file.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

PIDTransmissionEnable

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.PIDTransmissionEnable` api handle path.

The parameter is available for the following blocks:

- Read block
- Write block

Description

Can be set only if dynamic DAQs are used.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the PIDTransmissionEnable block parameter interactively, refer to:

- Options Page (RTIBYPASS_READ_BLx for XCP on CAN) (RTI Bypass Blockset Reference 📖)
- Options Page (RTIBYPASS_WRITE_BLx for XCP on CAN) (RTI Bypass Blockset Reference 📖)

PORT

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.PORT` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

> **Note**
>
> The PORT parameter can be specified via a database file.

Parameter type

double

PRESCALER_SUPPORTED

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.PRESCALER_SUPPORTED` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Support for a prescaler.

This is an optional parameter.

> **Note**
>
> The PRESCALER_SUPPORTED parameter can be specified via a database file.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

Priority

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.Priority` api handle path.

The parameter is available for the following blocks:

- Read block
- Write block

Description

Manually assigned priority.

Can be set only if the assignment type is set to Manual.

Parameter type

char

User interface

For information on specifying the Priority block parameter interactively, refer to:

- Options Page (RTIBYPASS_READ_BLx for XCP on CAN) (RTI Bypass Blockset Reference 📖)
- Options Page (RTIBYPASS_WRITE_BLx for XCP on CAN) (RTI Bypass Blockset Reference 📖)
- Options Page (RTIBYPASS_READ_BLx for XCP on UDP/IP) (RTI Bypass Blockset Reference 📖)
- Options Page (RTIBYPASS_WRITE_BLx for XCP on UDP/IP) (RTI Bypass Blockset Reference 📖)

ProtocolLayerVersion

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.ProtocolLayerVersion` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Version of the XCP protocol layer.

Value must be provided as a double string.

> **Note**
>
> The ProtocolLayerVersion parameter can be specified via a database file.

Parameter type

char

RESOLUTION

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.RESOLUTION` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Resolution of the time stamp.

0: The DAQ time stamp unit is 1 nanosecond

1: The DAQ time stamp unit is 10 nanoseconds

2: The DAQ time stamp unit is 100 nanoseconds

3: The DAQ time stamp unit is 1 microsecond

4: The DAQ time stamp unit is 10 microseconds

5: The DAQ time stamp unit is 100 microseconds

6: The DAQ time stamp unit is 1 millisecond

7: The DAQ time stamp unit is 10 milliseconds

8: The DAQ time stamp unit is 100 milliseconds

9: The DAQ time stamp unit is 1 second

Note

The RESOLUTION parameter can be specified via a database file.

Parameter type

double

Possible values

You can specify the following values:

- 0
- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9

RESUME_SUPPORTED

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.RESUME_SUPPORTED` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Support for resume mode.

This is an optional parameter.

Note

The RESUME_SUPPORTED parameter can be specified via a database file.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

SAMPLE_POINT

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.SAMPLE_POINT` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Sample point (as percentage of entire bit time).

This is an optional parameter.

Note

The SAMPLE_POINT parameter can be specified via a database file.

Parameter type

double

SAMPLE_RATE

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.SAMPLE_RATE` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Sample rate.

1 (SINGLE): 1 sample per bit

3 (TRIPLE): 3 samples per bit

This is an optional parameter.

> **Note**
>
> The SAMPLE_RATE parameter can be specified via a database file.

Parameter type

double

Possible values

You can specify the following values:

- 1
- 3

SeedAndKeyFile

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.SeedAndKeyFile` api handle path.

The parameter is available for the following blocks:

- Setup block

Parameter type

char

User interface

For information on specifying the SeedAndKeyFile block parameter interactively, refer to:

- Options Page (RTIBYPASS_SETUP_BLx for XCP on CAN) (RTI Bypass Blockset Reference 📖)
- Options Page (RTIBYPASS_SETUP_BLx for XCP on UDP/IP) (RTI Bypass Blockset Reference 📖)

SegmentNumber

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.SegmentNumber` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Can be set only if EnablePageSwitching is enabled.

Parameter type

double

User interface

For information on specifying the SegmentNumber block parameter interactively, refer to:

- Options Page (RTIBYPASS_SETUP_BLx for XCP on CAN) (RTI Bypass Blockset Reference 📖)
- Options Page (RTIBYPASS_SETUP_BLx for XCP on UDP/IP) (RTI Bypass Blockset Reference 📖)

SendDTOImmediately

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.SendDTOImmediately` api handle path.

The parameter is available for the following blocks:

- Write block

Description

Specifies to send the current data transmission object directly after the current block is processed. If disabled, the data transmission object is sent if the last write block of the configured service instance was processed.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the SendDTOImmediately block parameter interactively, refer to:

- Options Page (RTIBYPASS_WRITE_BLx for XCP on UDP/IP) (RTI Bypass Blockset Reference 📖)

SendExtendedIdentifierBit

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.SendExtendedIdentifierBit` api handle path.

The parameter is available for the following blocks:

- Setup block

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the SendExtendedIdentifierBit block parameter interactively, refer to:

- Options Page (RTIBYPASS_SETUP_BLx for XCP on CAN) (RTI Bypass Blockset Reference 📖)

SendOneODTperFrame

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.SendOneODTperFrame` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Specifies to send only one ODT per Ethernet frame. This parameter should be enabled if your ECU does not support decoding several ODT packets packed in one Ethernet frame.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the SendOneODTperFrame block parameter interactively, refer to:

- Options Page (RTIBYPASS_SETUP_BLx for XCP on UDP/IP) (RTI Bypass Blockset Reference 🕮)

ServiceInstanceID

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.ServiceInstanceID` api handle path.

The parameter is available for the following blocks:

- Read block
- Write block
- Interrupt block

Description

This parameter should be modified only via the SET_EVENT routine.

Parameter type

double

User interface

For information on specifying the ServiceInstanceID block parameter interactively, refer to:

- Options Page (RTIBYPASS_READ_BLx for XCP on CAN) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_WRITE_BLx for XCP on CAN) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_READ_BLx for XCP on UDP/IP) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_WRITE_BLx for XCP on UDP/IP) (RTI Bypass Blockset Reference 🕮)
- Unit Page (RTIBYPASS_INTERRUPT_BLx) (RTI Bypass Blockset Reference 🕮)

ServiceInstanceName

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.ServiceInstanceName` api handle path.

The parameter is available for the following blocks:

- Read block
- Write block
- Interrupt block

Description

This parameter should be modified only via the SET_EVENT routine.

Parameter type

char

User interface

For information on specifying the ServiceInstanceName block parameter interactively, refer to:

- Options Page (RTIBYPASS_READ_BLx for XCP on CAN) (RTI Bypass Blockset Reference 📖)
- Options Page (RTIBYPASS_WRITE_BLx for XCP on CAN) (RTI Bypass Blockset Reference 📖)
- Options Page (RTIBYPASS_READ_BLx for XCP on UDP/IP) (RTI Bypass Blockset Reference 📖)
- Options Page (RTIBYPASS_WRITE_BLx for XCP on UDP/IP) (RTI Bypass Blockset Reference 📖)
- Unit Page (RTIBYPASS_INTERRUPT_BLx) (RTI Bypass Blockset Reference 📖)

SJW

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.SJW` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Length of the synchronization segment (in BTL cycles).

This is an optional parameter.

Note

The SJW parameter can be specified via a database file.

Parameter type

double

STORE_DAQ_SUPPORTED

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.STORE_DAQ_SUPPORTED` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Support for resume mode.

This is an optional parameter.

> **Note**
>
> The STORE_DAQ_SUPPORTED parameter can be specified via a database file.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

SUPPORT_DOUBLE_BUFFER

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.SUPPORT_DOUBLE_BUFFER` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Flag for double buffer mode support.

This is an optional parameter.

> **Note**
>
> The SUPPORT_DOUBLE_BUFFER parameter can be specified via a database file.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

SUPPORT_FAILSAFE_PAGE

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.SUPPORT_FAILSAFE_PAGE` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Flag for fail-safe page support.

This is an optional parameter.

> **Note**
>
> The SUPPORT_FAILSAFE_PAGE parameter can be specified via a database file.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

SUPPORT_FAILURE_CHECKING

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.SUPPORT_FAILURE_CHECKING` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Flag for failure checking support.

This is an optional parameter.

Note

The SUPPORT_FAILURE_CHECKING parameter can be specified via a database file.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

SUPPORT_WAIT

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.SUPPORT_WAIT` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Flag for wait mechanism support.

This is an optional parameter.

Note

The SUPPORT_WAIT parameter can be specified via a database file.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

SYNC_EDGE

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.SYNC_EDGE` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Synchronization edge.

1 (SINGLE): On falling edge only

2 (DUAL): On falling and rising edge

This is an optional parameter.

> **Note**
>
> The SYNC_EDGE parameter can be specified via a database file.

Parameter type

double

Possible values

You can specify the following values:

- 1
- 2

T1

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.T1` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Timeout value T1 (in milliseconds).

> **Note**
>
> The T1 parameter can be specified via a database file.

Parameter type

double

T2

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.T2` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Timeout value T2 (in milliseconds).

> **Note**
>
> The T2 parameter can be specified via a database file.

Parameter type

double

T3

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.T3` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Timeout value T3 (in milliseconds).

> **Note**
>
> The T3 parameter can be specified via a database file.

Parameter type

double

T4

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.T4` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Timeout value T4 (in milliseconds).

> **Note**
>
> The T4 parameter can be specified via a database file.

Parameter type

double

T5

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.T5` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Timeout value T5 (in milliseconds).

> **Note**
>
> The T5 parameter can be specified via a database file.

Parameter type

double

T6

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.T6` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Timeout value T6 (in milliseconds).

> **Note**
>
> The T6 parameter can be specified via a database file.

Parameter type

double

T7

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.T7` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Timeout value T7 (in milliseconds).

> **Note**
>
> The T7 parameter can be specified via a database file.

Parameter type

double

Timeout

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.Timeout` api handle path.

The parameter is available for the following blocks:

- Read block
- Write block

Description

Can be set only if WaitEnable is enabled.

Parameter type

char

User interface

For information on specifying the Timeout block parameter interactively, refer to:

- Options Page (RTIBYPASS_READ_BLx for XCP on CAN) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_WRITE_BLx for XCP on CAN) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_READ_BLx for XCP on UDP/IP) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_WRITE_BLx for XCP on UDP/IP) (RTI Bypass Blockset Reference 🕮)

TIMESTAMP_FIXED

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.TIMESTAMP_FIXED` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Fixed time stamp.

This is an optional parameter.

Note

The TIMESTAMP_FIXED parameter can be specified via a database file.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

TIMESTAMP_SIZE

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.TIMESTAMP_SIZE` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Support for timestamping.

Size of the time stamps in bytes.

0: No time stamp

1: Time stamp size BYTE (8-bit)

2: Time stamp size WORD (16-bit)

4: Time stamp size DWORD (32-bit)

Note

The TIMESTAMP_SIZE parameter can be specified via a database file.

Parameter type

double

Possible values

You can specify the following values:

- 0
- 1
- 2
- 4

TIMESTAMP_TICKS

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.TIMESTAMP_TICKS` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Time stamp ticks per unit.

> **Note**
>
> The TIMESTAMP_TICKS parameter can be specified via a database file.

Parameter type

double

TRANSPORT_LAYER_INSTANCE

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.TRANSPORT_LAYER_INSTANCE` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

The ID of the interface.

This is an optional parameter.

> **Note**
>
> The TRANSPORT_LAYER_INSTANCE parameter can be specified via a database file.

Parameter type

char

TransportLayerVersion

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.TransportLayerVersion` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Version of the XCP on CAN transport layer.

Value must be provided as double string.

> **Note**
>
> The TransportLayerVersion parameter can be specified via a database file.

Parameter type

char

TriggerIntAlways

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.TriggerIntAlways` api handle path.

The parameter is available for the following blocks:

- Setup block

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the TriggerIntAlways block parameter interactively, refer to:

- Options Page (RTIBYPASS_SETUP_BLx for XCP on CAN) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_SETUP_BLx for XCP on UDP/IP) (RTI Bypass Blockset Reference 🕮)

UnlockCAL

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.UnlockCAL` api handle path.

The parameter is available for the following blocks:

- Setup block

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the UnlockCAL block parameter interactively, refer to:

- Options Page (RTIBYPASS_SETUP_BLx for XCP on CAN) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_SETUP_BLx for XCP on UDP/IP) (RTI Bypass Blockset Reference 🕮)

UnlockDAQ

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.UnlockDAQ` api handle path.

The parameter is available for the following blocks:

- Setup block

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the UnlockDAQ block parameter interactively, refer to:

- Options Page (RTIBYPASS_SETUP_BLx for XCP on CAN) (RTI Bypass Blockset Reference 📖)
- Options Page (RTIBYPASS_SETUP_BLx for XCP on UDP/IP) (RTI Bypass Blockset Reference 📖)

UnlockSTIM

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.UnlockSTIM` api handle path.

The parameter is available for the following blocks:

- Setup block

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the UnlockSTIM block parameter interactively, refer to:

- Options Page (RTIBYPASS_SETUP_BLx for XCP on CAN) (RTI Bypass Blockset Reference 📖)
- Options Page (RTIBYPASS_SETUP_BLx for XCP on UDP/IP) (RTI Bypass Blockset Reference 📖)

WaitEnable

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.WaitEnable` api handle path.

The parameter is available for the following blocks:

- Read block
- Write block

Description

Can be set only if DoubleBufferEnable is enabled.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the WaitEnable block parameter interactively, refer to:

- Options Page (RTIBYPASS_READ_BLx for XCP on CAN) (RTI Bypass Blockset Reference 📖)
- Options Page (RTIBYPASS_WRITE_BLx for XCP on CAN) (RTI Bypass Blockset Reference 📖)
- Options Page (RTIBYPASS_READ_BLx for XCP on UDP/IP) (RTI Bypass Blockset Reference 📖)
- Options Page (RTIBYPASS_WRITE_BLx for XCP on UDP/IP) (RTI Bypass Blockset Reference 📖)

Parameters for CCP-Based Interfaces

Where to go from here

Information in this section

ADDRESS_EXTENSION

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.ADDRESS_EXTENSION` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Address extension type.

0: The ECU supports only one address extension within a DAQ.

1: The ECU supports only one address extension within an ODT.

> **Note**
>
> The ADDRESS_EXTENSION parameter can be specified via a database file.

Parameter type

double

Possible values

You can specify the following values:

- 0
- 1

BAUDRATE

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.BAUDRATE` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

CAN baud rate (in Hz).

This is an optional parameter.

> **Note**
>
> The BAUDRATE parameter can be specified via a database file.

Parameter type double

Blob_Version

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.Blob_Version` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Version of the AML file for which the A2L file was written

Value must be provided as a double string.

> **Note**
>
> The Blob_Version parameter can be specified via a database file.

Parameter type char

BTL_CYCLES

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.BTL_CYCLES` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

BTL cycles (in slots per bit time).

This is an optional parameter.

> **Note**
>
> The BTL_CYCLES parameter can be specified via a database file.

Parameter type

double

BTR0

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.BTR0` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Bit timing register 0.

You can either specify the baud rate of the CAN channel or

specify the bit timing parameters individually. If both are specified,

the setting of the baud rate parameter will be used.

> **Note**
>
> The BTR0 parameter can be specified via a database file.

Parameter type

double

BTR1

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.BTR1` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Bit timing register 1.

You can either specify the baud rate of the CAN channel or,

specify the bit timing parameters individually. If both are specified, the setting of the baud rate parameter will be used.

Note

The BTR1 parameter can be specified via a database file.

Parameter type

double

BYTE_ORDER

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.BYTE_ORDER` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Byte order of the CCP interface.

1: The CCP interface is little endian

2: The CCP interface is big endian

So far, the CCP interface endian must be identical to the ECU endian,

specified in the parameter dsbypass_service_init_struct_t::ecu_endian

Note

The BYTE_ORDER parameter can be specified via a database file.

Parameter type

double

Possible values

You can specify the following values:

- 1
- 2

ByteOrder

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.ByteOrder` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Endian of the ECU (big or little)

BIG_ENDIAN : The ECU is big endian (MSB first).

LITTLE_ENDIAN : The ECU is little endian (MSB last).

Note

The ByteOrder parameter can be specified via a database file.

Parameter type

char

Possible values

You can specify the following values:

- LITTLE_ENDIAN
- BIG_ENDIAN

BYTES_ONLY

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.BYTES_ONLY` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

BYTES only.

The ECU supports max. elements of one byte size

otherwise the ECU supports different data types.

This is an optional parameter.

Note

The BYTES_ONLY parameter can be specified via a database file.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

can_id

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.can_id` api handle path.

The parameter is available for the following blocks:

- Read block

Description

Defines the CAN ID used for communication. This parameter is evaluated if the related interface description (A2L) provides a static CAN ID configuration.

Parameter type

char

CAN_ID_MASTER

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.CAN_ID_MASTER` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

CAN ID of the CCP master (here: the RCP system).

Value must be provided as a HEX string.

> **Note**
>
> The CAN_ID_MASTER parameter can be specified via a database file.

Parameter type

char

CAN_ID_SLAVE

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.CAN_ID_SLAVE` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

CAN ID of the CCP slave (here: the ECU).

Value must be provided as a HEX string.

> **Note**
>
> The CAN_ID_SLAVE parameter can be specified via a database file.

Parameter type

char

CCP_Version

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.CCP_Version` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Version of the CCP protocol.

Value must be provided as a double string.

> **Note**
>
> The CCP_Version parameter can be specified via a database file.

Parameter type

char

CCPCommandTimeout

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.CCPCommandTimeout` api handle path.

The parameter is available for the following blocks:

- Setup block

Parameter type

char

User interface

For information on specifying the CCPCommandTimeout block parameter interactively, refer to:

- Options Page (RTIBYPASS_SETUP_BLx for CCP) (RTI Bypass Blockset Reference 📖)

CONSISTENCY

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.CONSISTENCY` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

CONSISTENCY mode.

0: Consistency of an entire DAQ is guaranteed.

1: Consistency of an entire ODT is guaranteed.

This is an optional parameter.

> **Note**
>
> The CONSISTENCY parameter can be specified via a database file.

Parameter type

double

Possible values

You can specify the following values:

- 0
- 1

DAQ_MODE

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.DAQ_MODE` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

DAQ mode.

0: ALTERNATING

1: BURST

This is an optional parameter.

> **Note**
>
> The DAQ_MODE parameter can be specified via a database file.

Parameter type

double

Possible values

You can specify the following values:

- 0
- 1

DAQListNo

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.DAQListNo` api handle path.

The parameter is available for the following blocks:

- Read block

Parameter type

char

Possible values

You can specify the following values:

- guibypass_ccp_readwrite(''GetPopupEntriesList'', guibypassinterface(''PopupDAQListGet'', dialogData.simulink.hBlock, dialogData.param.blockparameters.BypassInterfaceName{1}, dialogData.param.blockparameters.ServiceInstanceStruct{1}.ServiceInstanceName))

User interface

For information on specifying the DAQListNo block parameter interactively, refer to:

- Options Page (RTIBYPASS_READ_BLx for CCP) (RTI Bypass Blockset Reference 📖)

ECUSendsDAQPermanently

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.ECUSendsDAQPermanently` api handle path.

The parameter is available for the following blocks:

- Setup block

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the ECUSendsDAQPermanently block parameter interactively, refer to:

- Options Page (RTIBYPASS_SETUP_BLx for CCP) (RTI Bypass Blockset Reference 🕮)

ForegroundDownload

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.ForegroundDownload` api handle path.

The parameter is available for the following blocks:

- Download block

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the ForegroundDownload block parameter interactively, refer to:

- Options Page (RTIBYPASS_DOWNLOAD_BLx for CCP) (RTI Bypass Blockset Reference 🕮)

FREQUENCY

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.FREQUENCY` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Quartz frequency of the ECU (in Hz).

Note

The FREQUENCY parameter can be specified via a database file.

Parameter type

double

IgnoreECUResourcesStatus

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.IgnoreECUResourcesStatus` api handle path.

The parameter is available for the following blocks:

- Setup block

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the IgnoreECUResourcesStatus block parameter interactively, refer to:

- Options Page (RTIBYPASS_SETUP_BLx for CCP) (RTI Bypass Blockset Reference 🕮)

InterPacketGap

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.InterPacketGap` api handle path.

The parameter is available for the following blocks:

- Setup block

Parameter type

char

User interface

For information on specifying the InterPacketGap block parameter interactively, refer to:

- Options Page (RTIBYPASS_SETUP_BLx for CCP) (RTI Bypass Blockset Reference 🕮)

MAX_DLC_REQUIRED

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.MAX_DLC_REQUIRED` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Master to slave frames.

Use maximum size (8) for all CAN frames -> MAX_DLC_REQUIRED

This is an optional parameter.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

RESUME_SUPPORTED

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.RESUME_SUPPORTED` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

The ECU supports the Resume function.

This is an optional parameter.

> **Note**
>
> The RESUME_SUPPORTED parameter can be specified via a database file.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

SAMPLE_POINT

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.SAMPLE_POINT` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Sample point (as percentage of entire bit time).

This is an optional parameter.

> **Note**
>
> The SAMPLE_POINT parameter can be specified via a database file.

Parameter type

double

SAMPLE_RATE

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.SAMPLE_RATE` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Sample rate.

1: 1 sample per bit

3: 3 samples per bit

This is an optional parameter.

> **Note**
>
> The SAMPLE_RATE parameter can be specified via a database file.

Parameter type

double

Possible values

You can specify the following values:

- 1
- 3

SeedAndKeyFile

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.SeedAndKeyFile` api handle path.

The parameter is available for the following blocks:

- Setup block

Parameter type

char

User interface

For information on specifying the SeedAndKeyFile block parameter interactively, refer to:

- Options Page (RTIBYPASS_SETUP_BLx for CCP) (RTI Bypass Blockset Reference 🕮)

ServiceInstanceID

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.ServiceInstanceID` api handle path.

The parameter is available for the following blocks:

- Read block
- Interrupt block

Description

This parameter should be modified only via the SET_EVENT routine.

Parameter type

double

User interface

For information on specifying the ServiceInstanceID block parameter interactively, refer to:

- Options Page (RTIBYPASS_READ_BLx for CCP) (RTI Bypass Blockset Reference 🕮)
- Unit Page (RTIBYPASS_INTERRUPT_BLx) (RTI Bypass Blockset Reference 🕮)

ServiceInstanceName

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.ServiceInstanceName` api handle path.

The parameter is available for the following blocks:

- Read block
- Interrupt block

Description

This parameter should be modified only via the SET_EVENT routine.

Parameter type

char

User interface

For information on specifying the ServiceInstanceName block parameter interactively, refer to:

- Options Page (RTIBYPASS_READ_BLx for CCP) (RTI Bypass Blockset Reference 🕮)
- Unit Page (RTIBYPASS_INTERRUPT_BLx) (RTI Bypass Blockset Reference 🕮)

SJW

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.SJW` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Length of the synchronization segment (in BTL cycles).

This is an optional parameter.

> **Note**
>
> The SJW parameter can be specified via a database file.

Parameter type

double

StationAddress

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.StationAddress` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Station address.

Value must be provided as a HEX string.

> **Note**
>
> The StationAddress parameter can be specified via a database file.

Parameter type

char

STORE_SUPPORTED

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.STORE_SUPPORTED` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

The ECU supports the Store function.

This is an optional parameter.

> **Note**
>
> The STORE_SUPPORTED parameter can be specified via a database file.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

SYNC_EDGE

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.SYNC_EDGE` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Synchronization edge.

0 (SINGLE): On falling edge only

1 (DUAL): On falling and rising edge

This is an optional parameter.

> **Note**
>
> The SYNC_EDGE parameter can be specified via a database file.

Parameter type

double

Possible values

You can specify the following values:

- 0
- 1

UnlockCAL

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.UnlockCAL` api handle path.

The parameter is available for the following blocks:

- Setup block

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the UnlockCAL block parameter interactively, refer to:

- Options Page (RTIBYPASS_SETUP_BLx for CCP) (RTI Bypass Blockset Reference 📖)

UnlockDAQ

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.UnlockDAQ` api handle path.

The parameter is available for the following blocks:

- Setup block

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the UnlockDAQ block parameter interactively, refer to:

- Options Page (RTIBYPASS_SETUP_BLx for CCP) (RTI Bypass Blockset Reference 🕮)

Parameters for dSPACE Calibration and Bypassing Service-Based Interfaces

Where to go from here

Information in this section

ADDRESS_FACTOR

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.ADDRESS_FACTOR` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Factor for address computation.

Note

The ADDRESS_FACTOR parameter can be specified via a database file.

Parameter type

double

aml_version

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.aml_version` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

AML version of the dSPACE AML.

Value must be provided as a HEX string.

> **Note**
>
> The aml_version parameter can be specified via a database file.

Parameter type

char

ByteOrder

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.ByteOrder` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Endian of the ECU (big or little)

BIG_ENDIAN : The ECU is big endian (MSB first).

LITTLE_ENDIAN : The ECU is little endian (MSB last).

> **Note**
>
> The ByteOrder parameter can be specified via a database file.

Parameter type

char

Possible values

You can specify the following values:

- LITTLE_ENDIAN
- BIG_ENDIAN

DATA_FORMAT

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.DATA_FORMAT` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

0: PACKED

1: SCATTERED

> **Note**
>
> The DATA_FORMAT parameter can be specified via a database file.

Parameter type

double

DEVICE_DESCRIPTION

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.DEVICE_DESCRIPTION` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

> **Note**
>
> The DEVICE_DESCRIPTION parameter can be specified via a database file.

Parameter type

char

DEVICE_NAME

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.DEVICE_NAME` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Note

The DEVICE_NAME parameter can be specified via a database file.

Parameter type

char

DEVICE_TYPE

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.DEVICE_TYPE` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Note

The DEVICE_TYPE parameter can be specified via a database file.

Parameter type

char

DoubleBufferEnable

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.DoubleBufferEnable` api handle path.

The parameter is available for the following blocks:

- Read block
- Write block

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the DoubleBufferEnable block parameter interactively, refer to:

- Options Page (RTIBYPASS_READ_BLx for dSPACE on DPMEM) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_WRITE_BLx for dSPACE on DPMEM) (RTI Bypass Blockset Reference 🕮)

DPMEM_SIZE_IN_ECU

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.DPMEM_SIZE_IN_ECU` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Size in ECU.

Note

The DPMEM_SIZE_IN_ECU parameter can be specified via a database file.

Parameter type

double

DPMEM_SIZE_IN_RCP

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.DPMEM_SIZE_IN_RCP` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Size in interface.

Note

The DPMEM_SIZE_IN_RCP parameter can be specified via a database file.

Parameter type

double

DPMEM_START_IN_ECU

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.DPMEM_START_IN_ECU` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

DPMEM start address in ECU memory map.

Note

The DPMEM_START_IN_ECU parameter can be specified via a database file.

Parameter type

double

DPMEM_START_IN_RCP

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.DPMEM_START_IN_RCP` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

DPMEM start address in interface memory.

> **Note**
>
> The DPMEM_START_IN_RCP parameter can be specified via a database file.

Parameter type

double

DPMEM_WIDTH

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.DPMEM_WIDTH` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Width.

> **Note**
>
> The DPMEM_WIDTH parameter can be specified via a database file.

Parameter type

double

FailureLimit

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.FailureLimit` api handle path.

The parameter is available for the following blocks:

- Read block
- Write block

Description

Can be set only if DoubleBufferEnable is enabled.

Parameter type

char

User interface

For information on specifying the FailureLimit block parameter interactively, refer to:

- Options Page (RTIBYPASS_READ_BLx for dSPACE on DPMEM) (RTI Bypass Blockset Reference 📖)
- Options Page (RTIBYPASS_WRITE_BLx for dSPACE on DPMEM) (RTI Bypass Blockset Reference 📖)

HW_INTERRUPT_ADDRESS

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.HW_INTERRUPT_ADDRESS` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Offset of the subinterrupt handling memory in the tool DPMEM (in words).

> **Note**
>
> The HW_INTERRUPT_ADDRESS parameter can be specified via a database file.

Parameter type

double

LENGTH_SERVICES_MEM

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.LENGTH_SERVICES_MEM` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Length of the dSPACE service space in the ECU memory (in bytes).

> **Note**
>
> The LENGTH_SERVICES_MEM parameter can be specified via a database file.

Parameter type

double

MAX_NO_SERVICES

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.MAX_NO_SERVICES` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Maximum number of service instances which can be assigned to this service.

> **Note**
>
> The MAX_NO_SERVICES parameter can be specified via a database file.

Parameter type

double

MAX_NO_SUBINTERRUPTS

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.MAX_NO_SUBINTERRUPTS` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Subinterrupt handling mechanism.

Maximum number of subinterrupts.

The maximum number of possible subinterrupts depends on the

setting for the number of bits per subinterrupt.

Between 2 and 127 for bit-based

Between 4 and 128 and multiple of 4 for byte-based Ex: 4, 8, ..., 128

> **Note**
>
> The MAX_NO_SUBINTERRUPTS parameter can be specified via a database file.

Parameter type

double

NO_BITS_PER_SUBINTERRUPT

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.NO_BITS_PER_SUBINTERRUPT` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Number of bits per subinterrupt.

16: Bit-based (16 bits) subinterrupt mechanism.

> **Note**
>
> The NO_BITS_PER_SUBINTERRUPT parameter can be specified via a database file.

Parameter type

double

SCS_OFFSET_IN_SERVICES_MEM

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.SCS_OFFSET_IN_SERVICES_MEM` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Offset of the service configuration section (SCS) in the tool DPMEM (in words).

> **Note**
>
> The SCS_OFFSET_IN_SERVICES_MEM parameter can be specified via a database file.

Parameter type

double

ServiceInstanceID

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.ServiceInstanceID` api handle path.

The parameter is available for the following blocks:

- Read block
- Write block
- Interrupt block

Description

This parameter should be modified only via the SET_EVENT routine.

Parameter type

double

User interface

For information on specifying the ServiceInstanceID block parameter interactively, refer to:

- Unit Page (RTIBYPASS_READ_BLx) (RTI Bypass Blockset Reference 📖)
- Unit Page (RTIBYPASS_WRITE_BLx) (RTI Bypass Blockset Reference 📖)
- Unit Page (RTIBYPASS_INTERRUPT_BLx) (RTI Bypass Blockset Reference 📖)

ServiceInstanceName

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.ServiceInstanceName` api handle path.

The parameter is available for the following blocks:

- Read block
- Write block
- Interrupt block

Description

This parameter should be modified only via the SET_EVENT routine.

Parameter type

char

User interface

For information on specifying the ServiceInstanceName block parameter interactively, refer to:

- Unit Page (RTIBYPASS_READ_BLx) (RTI Bypass Blockset Reference 📖)
- Unit Page (RTIBYPASS_WRITE_BLx) (RTI Bypass Blockset Reference 📖)
- Unit Page (RTIBYPASS_INTERRUPT_BLx) (RTI Bypass Blockset Reference 📖)

SINT_BLOCK_START_ADDRESS

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.SINT_BLOCK_START_ADDRESS` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Offset of the subinterrupt handling memory in the tool DPMEM (in words).

> **Note**
>
> The SINT_BLOCK_START_ADDRESS parameter can be specified via a database file.

Parameter type

double

START_ADDRESS_SERVICES_MEM

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.START_ADDRESS_SERVICES_MEM` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Start address of the dSPACE service in the ECU memory map (in bytes).

> **Note**
>
> The START_ADDRESS_SERVICES_MEM parameter can be specified via a database file.

Parameter type

double

SUPPORT_08BIT_TRANSFERS

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.SUPPORT_08BIT_TRANSFERS` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

8-bit access to the ECU memory is supported.

> **Note**
>
> The SUPPORT_08BIT_TRANSFERS parameter can be specified via a database file.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

SUPPORT_16BIT_TRANSFERS

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.SUPPORT_16BIT_TRANSFERS` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

16-bit access to the ECU memory is supported.

> **Note**
>
> The SUPPORT_16BIT_TRANSFERS parameter can be specified via a database file.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

SUPPORT_32BIT_TRANSFERS

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.SUPPORT_32BIT_TRANSFERS` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

32-bit access to the ECU memory is supported.

> **Note**
>
> The SUPPORT_32BIT_TRANSFERS parameter can be specified via a database file.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

SUPPORT_64BIT_TRANSFERS

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.SUPPORT_64BIT_TRANSFERS` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

64-bit access to the ECU memory is supported.

> **Note**
>
> The SUPPORT_64BIT_TRANSFERS parameter can be specified via a database file.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

SUPPORT_BITS_TRANSFERS

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.SUPPORT_BITS_TRANSFERS` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Bit access to the ECU memory is supported.

Note

The SUPPORT_BITS_TRANSFERS parameter can be specified via a database file.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

SUPPORT_BLOCK_COPY

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.SUPPORT_BLOCK_COPY` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

The block copy mode is supported.

> **Note**
>
> The SUPPORT_BLOCK_COPY parameter can be specified via a database file.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

SUPPORT_DOUBLE_BUFFER

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.SUPPORT_DOUBLE_BUFFER` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Support for the double buffer mechanism.

This is an optional parameter.

> **Note**
>
> The SUPPORT_DOUBLE_BUFFER parameter can be specified via a database file.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

SUPPORT_FAILSAFE_PAGE

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.SUPPORT_FAILSAFE_PAGE` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Support for the failsafe page mechanism.

This is an optional parameter.

Note

The SUPPORT_FAILSAFE_PAGE parameter can be specified via a database file.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

SUPPORT_FAILURE_CHECKING

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.SUPPORT_FAILURE_CHECKING` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Support for the failure checking mechanism.

This is an optional parameter.

> **Note**
>
> The SUPPORT_FAILURE_CHECKING parameter can be specified via a database file.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

SUPPORT_MORE_ATs_PER_SERVICE

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.SUPPORT_MORE_ATs_PER_SERVICE` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

The assignment of more than one address table to one service instance call is supported.

> **Note**
>
> The SUPPORT_MORE_ATs_PER_SERVICE parameter can be specified via a database file.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

SUPPORT_WAIT

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.SUPPORT_WAIT` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Support for the ECU wait mechanism.

This is an optional parameter.

> **Note**
>
> The SUPPORT_WAIT parameter can be specified via a database file.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

SyncServiceID

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.SyncServiceID` api handle path.

The parameter is available for the following blocks:

- Write block

Description

Can be set only if DoubleBufferEnable is enabled.

Parameter type

char

User interface

For information on specifying the SyncServiceID block parameter interactively, refer to:

- Options Page (RTIBYPASS_WRITE_BLx for dSPACE on DPMEM) (RTI Bypass Blockset Reference 🕮)

Timeout

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.Timeout` api handle path.

The parameter is available for the following blocks:

- Read block
- Write block

Description

Can be set only if DoubleBufferEnable is enabled.

Parameter type

char

User interface

For information on specifying the Timeout block parameter interactively, refer to:

- Options Page (RTIBYPASS_READ_BLx for dSPACE on DPMEM) (RTI Bypass Blockset Reference 📖)
- Options Page (RTIBYPASS_WRITE_BLx for dSPACE on DPMEM) (RTI Bypass Blockset Reference 📖)

WaitEnable

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.WaitEnable` api handle path.

The parameter is available for the following blocks:

- Read block
- Write block

Description

Can be set only if DoubleBufferEnable is enabled.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the WaitEnable block parameter interactively, refer to:

- Options Page (RTIBYPASS_READ_BLx for dSPACE on DPMEM) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_WRITE_BLx for dSPACE on DPMEM) (RTI Bypass Blockset Reference 🕮)

Parameters for Internal Bypassing

Where to go from here

Information in this section

AML_VERSION

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.AML_VERSION` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

AML version of the dSPACE AML.

Value must be provided as a HEX string.

> **Note**
>
> The AML_VERSION parameter can be specified via a database file.

Parameter type

char

CodeGenerator

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.CodeGenerator` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Select Target Code Generator.

Parameter type

char

Possible values

You can specify the following values:

- Simulink Coder
- dSPACE TargetLink Production Code Generator

User interface

For information on specifying the CodeGenerator block parameter interactively, refer to:

- Build Page (RTIBYPASS_SETUP_BLx for INTERNAL) (RTI Bypass Blockset Reference 🕮)

CustomFlashToolCommand

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.CustomFlashToolCommand` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

The custom flash command.

The following macros can be used.

$(ModelName), $(InputA2l), $(OutputA2l), $(InputEcu), $(OutputEcu)

Make sure to double-quote whitespaces in paths.

Parameter type

char

User interface

For information on specifying the CustomFlashToolCommand block parameter interactively, refer to:

- Build Page (RTIBYPASS_SETUP_BLx for INTERNAL) (RTI Bypass Blockset Reference 🕮)

CustomSrcEcuApplication

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.CustomSrcEcuApplication` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

The user defined source file that is merged with the generated internal bypass binary.

Parameter type

char

User interface

For information on specifying the CustomSrcEcuApplication block parameter interactively, refer to:

- Build Page (RTIBYPASS_SETUP_BLx for INTERNAL) (RTI Bypass Blockset Reference 🕮)

DstA2LFileName

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.DstA2LFileName` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

The destination file name used for extended A2L file generation.

Parameter type

char

User interface

For information on specifying the DstA2LFileName block parameter interactively, refer to:

- Build Page (RTIBYPASS_SETUP_BLx for INTERNAL) (RTI Bypass Blockset Reference 🕮)

DstEcuApplication

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.DstEcuApplication` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

The destination file name used for merging the ECU application.

Parameter type

char

User interface

For information on specifying the DstEcuApplication block parameter interactively, refer to:

- Build Page (RTIBYPASS_SETUP_BLx for INTERNAL) (RTI Bypass Blockset Reference 📖)

EnableDouble2SingleFloatConversion

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.EnableDouble2SingleFloatConversion` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Whether to automatically downgrade all occurrences of 64-bit double-precision floats to 32-bit single-precision floats (IEEE754).

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the EnableDouble2SingleFloatConversion block parameter interactively, refer to:

- Build Page (RTIBYPASS_SETUP_BLx for INTERNAL) (RTI Bypass Blockset Reference 📖)

EnableExtA2LGeneration

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.EnableExtA2LGeneration` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Whether to enable the extended A2L file generation.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the EnableExtA2LGeneration block parameter interactively, refer to:

- Build Page (RTIBYPASS_SETUP_BLx for INTERNAL) (RTI Bypass Blockset Reference 🕮)

EnableFlashing

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.EnableFlashing` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Whether to flash the generated ECU application after the internal bypass build process.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the EnableFlashing block parameter interactively, refer to:

- Build Page (RTIBYPASS_SETUP_BLx for INTERNAL) (RTI Bypass Blockset Reference 🕮)

EnableFunctionBlockVariable2EcuVariableMapping

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.EnableFunctionBlockVariable2EcuVariableMapping` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Whether to enable the mapping of internal function block variables to internal ECU variables.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the EnableFunctionBlockVariable2EcuVariableMapping block parameter interactively, refer to:

- Build Page (RTIBYPASS_SETUP_BLx for INTERNAL) (RTI Bypass Blockset Reference 📖)

EnableMerging

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.EnableMerging` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Whether to merge the generated internal bypass binary into an existing ECU binary.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the EnableMerging block parameter interactively, refer to:

- Build Page (RTIBYPASS_SETUP_BLx for INTERNAL) (RTI Bypass Blockset Reference 🕮)

EnablePostProcessing

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.EnablePostProcessing` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Whether to execute a post process command after the internal bypass build process.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the EnablePostProcessing block parameter interactively, refer to:

- Build Page (RTIBYPASS_SETUP_BLx for INTERNAL) (RTI Bypass Blockset Reference 🕮)

EnableSimulinkParameter2EcuVariableMapping

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.EnableSimulinkParameter2EcuVariableMapping` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Whether to enable the mapping of Simulink Parameter objects to internal ECU variables.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the EnableSimulinkParameter2EcuVariableMapping block parameter interactively, refer to:

- Build Page (RTIBYPASS_SETUP_BLx for INTERNAL) (RTI Bypass Blockset Reference 🕮)

EnableSimulinkParameterGeneration

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.EnableSimulinkParameterGeneration` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Whether to enable the automatic generation of Simulink parameters from referenced numeric workspace variables.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the EnableSimulinkParameterGeneration block parameter interactively, refer to:

- Build Page (RTIBYPASS_SETUP_BLx for INTERNAL) (RTI Bypass Blockset Reference 🕮)

EnableSimulinkSignal2EcuVariableMapping

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.EnableSimulinkSignal2EcuVariableMapping` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Whether to enable the mapping of Simulink Signal objects to internal ECU variables.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the EnableSimulinkSignal2EcuVariableMapping block parameter interactively, refer to:

- Build Page (RTIBYPASS_SETUP_BLx for INTERNAL) (RTI Bypass Blockset Reference 🕮)

EnableSimulinkSignalGeneration

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.EnableSimulinkSignalGeneration` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Whether to enable the automatic generation of Simulink Signal objects from signal names.

Parameter type

logical

Possible values

You can specify the following values:

- false
- true

User interface

For information on specifying the EnableSimulinkSignalGeneration block parameter interactively, refer to:

- Build Page (RTIBYPASS_SETUP_BLx for INTERNAL) (RTI Bypass Blockset Reference 🕮)

FlashTool

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.FlashTool` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

The tool that is used for flashing.

You can use the dSPACE flash tool or a custom command.

Parameter type

char

Possible values

You can specify the following values:

- ''dSPACE ECU Flash Programming Tool''
- ''Custom Flash Command''

User interface

For information on specifying the FlashTool block parameter interactively, refer to:

- Build Page (RTIBYPASS_SETUP_BLx for INTERNAL) (RTI Bypass Blockset Reference 📖)

HOST_NAME_IP_ADDRESS

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.HOST_NAME_IP_ADDRESS` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

The host name or IP address for accessing internal bypass service DLL (A2L).

> **Note**
>
> The HOST_NAME_IP_ADDRESS parameter can be specified via a database file.

Parameter type

char

INTBYP_CONFIG_START_ADDRESS

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.INTBYP_CONFIG_START_ADDRESS` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Start address of the internal bypass configuration structure.

Value must be provided as a HEX string.

> **Note**
>
> The INTBYP_CONFIG_START_ADDRESS parameter can be specified via a database file.

Parameter type

char

INTBYP_SERVICE_VERSION

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.INTBYP_SERVICE_VERSION` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

AML version of the dSPACE AML.

Value must be provided as a HEX string.

> **Note**
>
> The INTBYP_SERVICE_VERSION parameter can be specified via a database file.

Parameter type

char

PORT_NUMBER

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.PORT_NUMBER` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

The port number for accessing internal bypass service DLL (A2L).

> **Note**
>
> The PORT_NUMBER parameter can be specified via a database file.

Parameter type

double

PostProcessCommand

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.PostProcessCommand` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

The post process command that is executed after the internal bypass build process.

The following macros can be used.

$(ModelName), $(InputA2l), $(OutputA2l), $(InputEcu), $(OutputEcu)

Make sure to double-quote whitespaces in paths.

Parameter type

char

User interface

For information on specifying the PostProcessCommand block parameter interactively, refer to:

- Build Page (RTIBYPASS_SETUP_BLx for INTERNAL) (RTI Bypass Blockset Reference 🕮)

PROCESSOR_TYPE

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.PROCESSOR_TYPE` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Target processor type.

i.e. TRICORE

> **Note**
>
> The PROCESSOR_TYPE parameter can be specified via a database file.

Parameter type

char

Possible values

You can specify the following values:

- ''TRICORE''
- ''MPC5XXX''
- ''V850X''
- ''X86''
- ''ARM''

REGISTER_PROTECTION

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.REGISTER_PROTECTION` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Whether to save the registers before context switches.

> **Note**
>
> The REGISTER_PROTECTION parameter can be specified via a database file.

Parameter type

double

Possible values

You can specify the following values:

- 1
- 0

ROOT_TASK_MAPPING

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.ROOT_TASK_MAPPING` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

The service instance in which the timer task must be called (A2L).

> **Note**
>
> The ROOT_TASK_MAPPING parameter can be specified via a database file.

Parameter type

double

Possible values

You can specify the following values:

- -1
- rtibypass_internal(''GetServiceInstanceIDList'', dialogData.param.blockparameters.ServiceAndHardwareStruct{1}.EVENTs)

RootTaskMapping

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.RootTaskMapping` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

The service instance in which the timer task must be called.

Parameter type

double

Possible values

You can specify the following values:

- -1
- rtibypass_internal(''GetServiceInstanceIDList'', dialogData.param.blockparameters.ServiceAndHardwareStruct{1}.EVENTs)

User interface

For information on specifying the RootTaskMapping block parameter interactively, refer to:

- Options Page (RTIBYPASS_SETUP_BLx for INTERNAL) (RTI Bypass Blockset Reference 📖)

SERVICE_CONFIG_START_ADDRESS

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.SERVICE_CONFIG_START_ADDRESS` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Start address of the service configuration structure.

Value must be provided as a HEX string.

Note

The SERVICE_CONFIG_START_ADDRESS parameter can be specified via a database file.

Parameter type

char

SimulinkObjectA2LPostfix

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.SimulinkObjectA2LPostfix` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Suffix that is added to the name of each generated A2L variable.

Parameter type

char

User interface

For information on specifying the SimulinkObjectA2LPostfix block parameter interactively, refer to:

- Build Page (RTIBYPASS_SETUP_BLx for INTERNAL) (RTI Bypass Blockset Reference 🕮)

SimulinkObjectA2LPrefix

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.SimulinkObjectA2LPrefix` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Prefix that is added to the name of each generated A2L variable.

Parameter type

char

User interface

For information on specifying the SimulinkObjectA2LPrefix block parameter interactively, refer to:

- Build Page (RTIBYPASS_SETUP_BLx for INTERNAL) (RTI Bypass Blockset Reference 🕮)

SrcA2LFileName

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.SrcA2LFileName` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

The source file that is extended by the generated A2L.

Parameter type

char

User interface

For information on specifying the SrcA2LFileName block parameter interactively, refer to:

- Build Page (RTIBYPASS_SETUP_BLx for INTERNAL) (RTI Bypass Blockset Reference 🕮)

SrcEcuApplication

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.SrcEcuApplication` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

The A2L source file that is merged with the generated internal bypass binary.

Parameter type

char

User interface

For information on specifying the SrcEcuApplication block parameter interactively, refer to:

- Build Page (RTIBYPASS_SETUP_BLx for INTERNAL) (RTI Bypass Blockset Reference 🕮)

IO Board-Specific Parameters

Where to go from here

Information in this section

Common IO Parameters

Where to go from here

Information in this section

BoardNo

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.BoardNo` api handle path.

The parameter is available for the following blocks:

- Setup block

Parameter type

double

User interface

For information on specifying the BoardNo block parameter interactively, refer to:

- Options Page (RTIBYPASS_SETUP_BLx for CCP) (RTI Bypass Blockset Reference 📖)
- Options Page (RTIBYPASS_SETUP_BLx for XCP on CAN) (RTI Bypass Blockset Reference 📖)
- Options Page (RTIBYPASS_SETUP_BLx for XCP on UDP/IP) (RTI Bypass Blockset Reference 📖)
- Options Page (RTIBYPASS_SETUP_BLx for dSPACE on DPMEM) (RTI Bypass Blockset Reference 📖)

BoardType

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.BoardType` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

The possible values vary according to the configured interface. You should therefore generate a new API when the interface changes.

Parameter type

char

User interface

For information on specifying the BoardType block parameter interactively, refer to:

- Options Page (RTIBYPASS_SETUP_BLx for CCP) (RTI Bypass Blockset Reference 📖)

- Options Page (RTIBYPASS_SETUP_BLx for XCP on CAN) (RTI Bypass Blockset Reference 📖)
- Options Page (RTIBYPASS_SETUP_BLx for XCP on UDP/IP) (RTI Bypass Blockset Reference 📖)
- Options Page (RTIBYPASS_SETUP_BLx for dSPACE on DPMEM) (RTI Bypass Blockset Reference 📖)

ControllerNo

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.ControllerNo` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

The maximum number of controllers varies according to the selected board type. You should therefore generate a new API when the board type changes.

Parameter type

double

User interface

For information on specifying the ControllerNo block parameter interactively, refer to:

- Options Page (RTIBYPASS_SETUP_BLx for CCP) (RTI Bypass Blockset Reference 📖)
- Options Page (RTIBYPASS_SETUP_BLx for XCP on CAN) (RTI Bypass Blockset Reference 📖)
- Options Page (RTIBYPASS_SETUP_BLx for XCP on UDP/IP) (RTI Bypass Blockset Reference 📖)
- Options Page (RTIBYPASS_SETUP_BLx for dSPACE on DPMEM) (RTI Bypass Blockset Reference 📖)

ModuleNo

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.ModuleNo` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Number of the module on the IP carrier board.

The maximum number of modules varies according to the selected board type. You should therefore generate a new API when the board type changes.

Parameter type

double

User interface

For information on specifying the ModuleNo block parameter interactively, refer to:

- Options Page (RTIBYPASS_SETUP_BLx for XCP on CAN) (RTI Bypass Blockset Reference 📖)

ModuleType

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.ModuleType` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Type of the module on the IP carrier board.

The module type varies according to the selected board type. You should therefore generate a new API when the board type changes.

Parameter type

char

User interface

For information on specifying the ModuleType block parameter interactively, refer to:

- Options Page (RTIBYPASS_SETUP_BLx for XCP on CAN) (RTI Bypass Blockset Reference 📖)

DS2202 CAN IO Parameters

Where to go from here

Information in this section

termination

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.termination` api handle path.

The parameter is available for the following blocks:

- Setup block

Parameter type

char

Possible values

You can specify the following values:

- 0
- 120

User interface

For information on specifying the termination block parameter interactively, refer to:

- Options Page (RTIBYPASS_SETUP_BLx for XCP on CAN) (RTI Bypass Blockset Reference 📖)
- Options Page (RTIBYPASS_SETUP_BLx for CCP) (RTI Bypass Blockset Reference 📖)

transceiver

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.transceiver` api handle path.

The parameter is available for the following blocks:

- Setup block

Parameter type

char

Possible values

You can specify the following values:

- ISO11898

User interface

For information on specifying the transceiver block parameter interactively, refer to:

- Options Page (RTIBYPASS_SETUP_BLx for XCP on CAN) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_SETUP_BLx for CCP) (RTI Bypass Blockset Reference 🕮)

DS2210 CAN IO Parameters

Where to go from here

Information in this section

termination

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.termination` api handle path.

The parameter is available for the following blocks:

- Setup block

Parameter type

char

Possible values

You can specify the following values:

- 0
- 120

User interface

For information on specifying the termination block parameter interactively, refer to:

- Options Page (RTIBYPASS_SETUP_BLx for XCP on CAN) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_SETUP_BLx for CCP) (RTI Bypass Blockset Reference 🕮)

transceiver

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.transceiver` api handle path.

The parameter is available for the following blocks:

- Setup block

Parameter type

char

Possible values

You can specify the following values:

- ISO11898

User interface

For information on specifying the transceiver block parameter interactively, refer to:

- Options Page (RTIBYPASS_SETUP_BLx for XCP on CAN) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_SETUP_BLx for CCP) (RTI Bypass Blockset Reference 🕮)

DS2211 CAN IO Parameters

Where to go from here

Information in this section

termination

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.termination` api handle path.

The parameter is available for the following blocks:

- Setup block

Parameter type

char

Possible values

You can specify the following values:

- 0
- 120

User interface

For information on specifying the termination block parameter interactively, refer to:

- Options Page (RTIBYPASS_SETUP_BLx for XCP on CAN) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_SETUP_BLx for CCP) (RTI Bypass Blockset Reference 🕮)

transceiver

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.transceiver` api handle path.

The parameter is available for the following blocks:

- Setup block

Parameter type

char

Possible values

You can specify the following values:

- ISO11898

User interface

For information on specifying the transceiver block parameter interactively, refer to:

- Options Page (RTIBYPASS_SETUP_BLx for XCP on CAN) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_SETUP_BLx for CCP) (RTI Bypass Blockset Reference 🕮)

DS4302 CAN IO Parameters

Where to go from here

Information in this section

termination

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.termination` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

The possible values vary according to the selected transceiver. You should therefore generate a new API when the transceiver changes.

Parameter type

char

User interface

For information on specifying the termination block parameter interactively, refer to:

- Options Page (RTIBYPASS_SETUP_BLx for XCP on CAN) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_SETUP_BLx for CCP) (RTI Bypass Blockset Reference 🕮)

transceiver

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.transceiver` api handle path.

The parameter is available for the following blocks:

- Setup block

Parameter type

char

Possible values

You can specify the following values:

- ISO11898
- RS485
- C252

User interface

For information on specifying the transceiver block parameter interactively, refer to:

- Options Page (RTIBYPASS_SETUP_BLx for XCP on CAN) (RTI Bypass Blockset Reference 🕮)
- Options Page (RTIBYPASS_SETUP_BLx for CCP) (RTI Bypass Blockset Reference 🕮)

DS4342 CAN IO Parameters

Where to go from here

Information in this section

baudrate_dataphase

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.baudrate_dataphase` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

CAN baud rate in data phase (in Hz).

This is an optional parameter.

Note

The baudrate_dataphase parameter can be specified via a database file.

Parameter type

double

CAN_FD_Enable

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.CAN_FD_Enable` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

Configure CAN FD mode.

Parameter type

char

Possible values

You can specify the following values:

- OFF
- NONE_ISO_CAN_FD
- ISO_CAN_FD

User interface

For information on specifying the CAN_FD_Enable block parameter interactively, refer to:

- Options Page (RTIBYPASS_SETUP_BLx for XCP on CAN) (RTI Bypass Blockset Reference 📖)

termination

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.termination` api handle path.

The parameter is available for the following blocks:

- Setup block

Description

The possible values vary according to the selected transceiver. You should therefore generate a new API when the transceiver changes.

Parameter type

char

Possible values

You can specify the following values:

- 0
- 120

User interface

For information on specifying the termination block parameter interactively, refer to:

- Options Page (RTIBYPASS_SETUP_BLx for XCP on CAN) (RTI Bypass Blockset Reference 🕮)

transceiver

Access

You can access this parameter via the `<api_handle>.PARAMS.SPECIFIC.transceiver` api handle path.

The parameter is available for the following blocks:

- Setup block

Parameter type

char

Possible values

You can specify the following values:

- ISO11898_2
- ISO11898_6

User interface

For information on specifying the transceiver block parameter interactively, refer to:

- Options Page (RTIBYPASS_SETUP_BLx for XCP on CAN) (RTI Bypass Blockset Reference 🕮)