

# NV-Wrapper

## Technical Reference

Wrapper for non-volatile memory access

Version 2.02

Authors	Christian Bäuerle, Achim Strobel, Marco Riedl
Status	Released

## Document Information

### History

Author	Date	Version	Remarks
Christian Bäuerle	2009-09-19	0.90	Initial Draft
Achim Strobelt	2012-01-26	1.00	Configuration with GENy
Marco Riedl	2016-03-02	2.00	Added Fee/NvM
Marco Riedl	2016-03-20	2.01	Changed API
Marco Riedl	2017-06-29	2.02	Added Ea

### Reference Documents

No.	Source	Title	Version
[1]	Vector	Flash Bootloader User Manual	2.7
[2]	Vector	EEPROM Manager, Technical Reference	1.3
[3]	Vector	Flash Driver Wrapper, Technical Reference	1.1



#### Caution

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

## Contents

<b>1</b>	<b>Introduction.....</b>	<b>6</b>
<b>2</b>	<b>Functional Description.....</b>	<b>7</b>
2.1	Architecture.....	7
2.2	Data structures .....	8
2.2.1	Single.....	8
2.2.2	List.....	8
2.2.3	Structure .....	8
2.2.4	Table .....	8
2.3	NV-Wrapper in Application .....	8
<b>3</b>	<b>Integration.....</b>	<b>10</b>
3.1	Scope of delivery .....	10
3.1.1	Core files.....	10
3.1.2	Application files .....	10
3.1.3	Generated files.....	11
3.2	Include structure .....	11
3.2.1	Initialization .....	11
3.2.2	Memory accesses .....	12
<b>4</b>	<b>API Description.....</b>	<b>13</b>
4.1	Function API .....	13
4.1.1	Function Description .....	13
4.1.1.1	WrapNv_Init.....	13
4.1.1.2	Synchronous API .....	13
4.1.1.2.1	WrapNv_ReadSync .....	13
4.1.1.2.2	WrapNv_ReadPartialSync .....	14
4.1.1.2.3	WrapNv_WriteSync.....	14
4.1.1.2.4	WrapNv_DeleteSync.....	15
4.1.1.3	Asynchronous API .....	15
4.1.1.3.1	WrapNv_ReadAsync.....	15
4.1.1.3.2	WrapNv_ReadPartialAsync.....	16
4.1.1.3.3	WrapNv_WriteAsync.....	17
4.1.1.3.4	WrapNv_DeleteAsync.....	17
4.2	Naming conventions .....	18
4.3	Interface definitions.....	19
<b>5</b>	<b>Configuration.....</b>	<b>22</b>
5.1	GENy configuration.....	22

5.1.1	Basic settings.....	22
5.1.2	Tree view usage.....	23
5.1.2.1	Data node context menu.....	23
5.1.2.2	Data block/element context menu.....	23
5.1.3	Data configuration.....	23
5.1.3.1	Data Blocks .....	24
5.1.3.2	Data Element Settings .....	25
<b>6</b>	<b>Terminology .....</b>	<b>26</b>
6.1	Abbreviations .....	26
<b>7</b>	<b>Contact.....</b>	<b>27</b>

## Illustrations

Figure 2-1	NV-Wrapper in Flash Bootloader .....	7
Figure 5-1	Activation of NV-Wrapper in GENy .....	22
Figure 5-2	NV-Wrapper GENy module.....	22
Figure 5-3	Data Block configuration .....	24
Figure 5-4	Data Element configuration.....	25

## Tables

Table 3-1	Core files .....	10
Table 3-2	Application files.....	11
Table 3-3	Generted files .....	11
Table 4-1	WrapNv_Init.....	13
Table 4-2	WrapNv_ReadSync .....	14
Table 4-3	WrapNv_ReadPartialSync .....	14
Table 4-4	WrapNv_WriteSync .....	15
Table 4-5	WrapNv_DeleteSync .....	15
Table 4-6	WrapNv_ReadAsync .....	16
Table 4-7	WrapNv_ReadPartialAsync .....	16
Table 4-8	WrapNv_WriteAsync.....	17
Table 4-9	WrapNv_DeleteAsync.....	18
Table 4-10	Single .....	19
Table 4-11	List.....	20
Table 4-12	Structure.....	20
Table 4-13	Table.....	21
Table 5-1	GENy configuration options for Nv-Wrapper .....	23
Table 5-2	Data Block configuration .....	24
Table 5-3	Data Block configuration .....	25

## 1 Introduction

The NV-Wrapper component provides an ID-based interface to access data in non-volatile memory. Based on this interface, data located in EEPROM devices with a HIS device driver interface (DrvEep), the EEPROM emulation by the Vector EEPROM manager (EepM), Fee, Ea and the NvM can be accessed.

Each data item is identified by a unique ID. When an address-based DrvEep is used, the NV-Wrapper performs a conversion from ID to address.

## 2 Functional Description

The NV-Wrapper module consists of a generated file which provides access macros. Those macros will then call the respective interface. In case the NV-Wrapper is used to access the Fee or NvM, a c-file which implements a synchronous and an asynchronous interface is used. For the DrvEep or the EepM the access macros are directly mapped to the respective module.

### 2.1 Architecture

The NV-Wrapper is a layer between the application / FBL and the NV-memory (DrvEep, EepM, Fee, Ea or NvM).

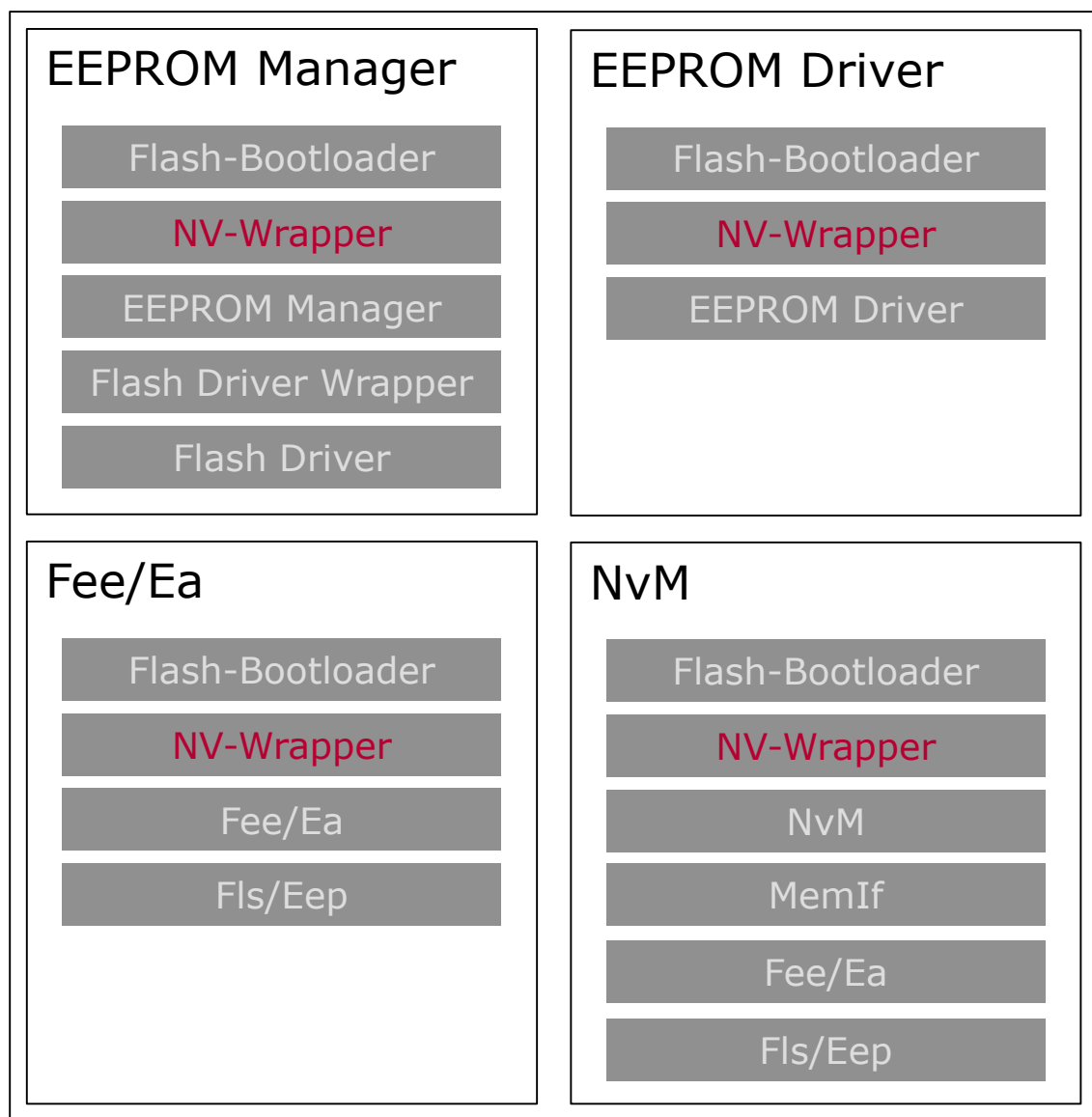


Figure 2-1 NV-Wrapper in Flash Bootloader

Figure 2-1 shows the usage of the NV-Wrapper with the EepM, DrvEep, Fee/Ea and NvM.

## 2.2 Data structures

This section gives a short overview about the supported data types. Each data element stored by the NV-Wrapper is either a single byte or a stream of bytes. The NV-Wrapper module generates access macros for 4 types of data structures:

### 2.2.1 Single

A single type data structure is the simplest data structure supported by NV-Wrapper. It can be compared to a single variable like an integer variable, e.g. the Security Access Delay flag used by the Bootloader.

- ▶ Data block multiplicity equals one
- ▶ One data element configured

### 2.2.2 List

A list can be used to store an array of one variable, e.g. a fingerprint history list.

- ▶ Data block multiplicity is more than one
- ▶ One data element configured

### 2.2.3 Structure

Structures are used to group several different members of a data structure. This data type can be compared to a C struct. One access macro is generated for every member of the structure.

- ▶ Data block count equals one
- ▶ At least two data elements configured

### 2.2.4 Table

A table is a list of structures. It represents an array of structures and can be used to access data elements like the meta data table stored for every logical block.

- ▶ Data block multiplicity is more than one
- ▶ At least two data elements configured

## 2.3 NV-Wrapper in Application

The NV-Wrapper can be used in the application as well. The architecture used there is basically the same compared to the architecture used in the Bootloader. However, special care has to be taken to ensure that the used memory access driver, e.g. EEPROM Manager, is integrated properly into the application environment.



**Caution**

If the NV-Wrapper is used within the application in connection with the EEPROM Manager, an additional wrapper layer has to be used between EEPROM Manager and Flash Driver Wrapper. Please see [3] for further hints.

## 3 Integration

This chapter provides information how to integrate and configure the NV-Wrapper. The configuration is mainly done within the GENy module supplied with the NV-Wrapper module.

**Note**

Configurator5 does not support the configuration of the NV-Wrapper module.

**Note**

GENy does not support the configuration of the Fee/Ea and NvM.

### 3.1 Scope of delivery

All files of the NV-Wrapper are located in the BSW\WrapNv folder of your delivery and consist of core files, adaptable application files as well as generated files. The application files are indicated by a leading underscore.

Some files are only present or generated if the NV-Wrapper is used in a special use case, e.g. Fee or NvM.

#### 3.1.1 Core files

The first group contains core files. Do not modify the contents of the files in this group without prior written permission from Vector.

File	Use Case	Description
WrapNv.h	DrvEep, EepM, Fee, Ea, NvM	Main module header file. It contains some preprocessor macros needed by the generated files.
WrapNv.c	Fee, Ea, NvM	Main module source file. It contains the abstraction of the Fee/Ea and NvM access to a synchronous/asynchronous API. The generated access macros are using these functions.

Table 3-1 Core files

#### 3.1.2 Application files

The second group consists of components that must be customized for your configuration. You should copy these files to your project folder and rename them, removing the leading underscore from the filename.

File	Use Case	Description
_WrapNv_inc.h	DrvEep, EepM, Fee, Ea, NvM	This is a template for the include file. The default constant of this file is a collection of file includes that are necessary for the NV-Wrapper in the used scope.

Table 3-2 Application files

### 3.1.3 Generated files

File	Use Case	Description
WrapNv_Cfg.c	Fee, Ea, NvM	This file consists of a look-up table, which the NV-Wrapper uses.
WrapNv_Cfg.h	DrvEep, EepM, Fee, Ea, NvM	Consists of all necessary access macros to read from/write to the NV-memory and the size.

Table 3-3 Generted files

## 3.2 Include structure

The NV-Wrapper centralizes the includes of all required header files in WrapNv\_inc.h.

### 3.2.1 Initialization

In case the NV-Wrapper is used with the DrvEep or the EepM, then the NV-Wrapper itself doesn't require an explicit initialization, but the memory drivers or managers used by the NV-Wrapper normally do. Please make sure all modules needed to access the memory are initialized before the first memory access macro is used.

If the NV-Wrapper is used to access the Fee, Ea or the NvM, the NV-Wrapper will initialize the respective memory drivers.

**Example**

Initialization if EepM is used:

```
/* Initialize EEPROM manager and the used flash driver */
Flash_XDDriver_InitSync(NULL);
(void)EepMgrInitPowerOn(kEepMgrInitFull);
```

Initialization if a EEPROM driver is used

```
/* Initialize EEPROM-Driver */
(void)EepromDriver_InitSync(NULL);
```

Initialization if Fee, Ea or NvM is used:

```
/* Initialize NV-Wrapper */
WrapNv_Init();
```

After these initializations, the NV-Wrapper can be used.

### 3.2.2 Memory accesses

The memory access macros normally use a pointer to a byte-array to hand over read or write buffers.

**Example**

Read access to a single structure, 1 Byte:

```
vuint8 buffer;

/* Read security access delay flag */
/* Parameter is not used on a struct and set to 0 */
if (ApplFblNvReadSecAccessDelay(0u, &buffer) != WRAPNV_E_OK)
{
    /* Read failed! Return kEepSecAccessDelayActive */
    return kEepSecAccessDelayActive;
}
else
{
    return buffer;
}
```

Write access to a member of a table, 10 Bytes:

```
/* Write fingerprint (from RAM buffer) into meta data table */
returnCode = ApplFblNvWriteFingerprint(blockNr, blockFingerprint);
```

## 4 API Description

In case the NV-Wrapper is used with a Fee, Ea or NvM the following API description (chapter 4.1) applies. The naming conventions described in chapter 4.2 apply to all use cases.

### 4.1 Function API

Only the API of the global functions is listed here. The NV-Wrapper internal functions are not included.

These functions are used by the generated access macros.

#### 4.1.1 Function Description

##### 4.1.1.1 WrapNv\_Init

Prototype
<code>void WrapNv_Init( void )</code>
Functional Description
This function is called to initialize the NV-Wrapper module.
Particularities and Limitations
> This function should be called when the Application/FBL is started after reset.

Table 4-1      WrapNv\_Init

##### 4.1.1.2 Synchronous API

##### 4.1.1.2.1 WrapNv\_ReadSync

Prototype	
<code>WrapNv_ReturnType WrapNv_ReadSync(uint16 id, uint16 idx, tWrapNvRamData buffer)</code>	
Parameter	
<code>id</code>	ID of record
<code>idx</code>	Index of the dataset
<code>buffer</code>	Pointer to the target buffer
Return code	
<code>WRAPNV_E_OK</code>	Read operation successful
<code>WRAPNV_E_NOT_OK</code>	Read operation failed
Functional Description	
This function reads the data from Fee/Nvm in the respective buffer.	

### Particularities and Limitations

> NV-Wrapper has to be initialized.

Table 4-2 WrapNv\_ReadSync

#### 4.1.1.2.2 WrapNv\_ReadPartialSync

##### Prototype

```
WrapNv_ReturnType WrapNv_ReadPartialSync(uint16 id, uint16 idx,
tWrapNvRamData buffer, uint16 offset, uint16 length )
```

##### Parameter

id	ID of record
idx	Index of the dataset
buffer	Pointer to the target buffer
offset	Offset in element from where to start to read
length	Length of the partial read operation

##### Return code

WRAPNV_E_OK	Read operation successful
WRAPNV_E_NOT_OK	Read operation failed

##### Functional Description

This function reads the data from Fee/Nvm in the respective buffer.

##### Particularities and Limitations

> NV-Wrapper has to be initialized.

Table 4-3 WrapNv\_ReadPartialSync

#### 4.1.1.2.3 WrapNv\_WriteSync

##### Prototype

```
WrapNv_ReturnType WrapNv_WriteSync( uint16 id, uint16 idx, tWrapNvRamData
buffer)
```

##### Parameter

id	ID of record
idx	Index of the dataset
buffer	Pointer to the source buffer

Return code	
WRAPNV_E_OK	Read operation successful
WRAPNV_E_NOT_OK	Read operation failed
Functional Description	
This function reads the data from Fee/Nvm in the respective buffer.	
Particularities and Limitations	
> NV-Wrapper has to be initialized.	

Table 4-4 WrapNv\_WriteSync

#### 4.1.1.2.4 WrapNv\_DeleteSync

Prototype	
WrapNv_ReturnType WrapNv_DeleteSync( uint16 id, uint16 idx )	
Parameter	
id	ID of record
idx	Index of the dataset
Return code	
WRAPNV_E_OK	Delete (invalidate) operation successful
WRAPNV_E_NOT_OK	Delete (invalidate) operation failed
Functional Description	
This function reads the data from Fee/Nvm in the respective buffer.	
Particularities and Limitations	
> NV-Wrapper has to be initialized.	

Table 4-5 WrapNv\_DeleteSync

#### 4.1.1.3 Asynchronous API

##### 4.1.1.3.1 WrapNv\_ReadAsync

Prototype	
WrapNv_ReturnType WrapNv_ReadAsync(uint16 id, uint16 idx, tWrapNvRamData buffer, tWrapNvOpStatus opStatus )	
Parameter	
id	ID of record
idx	Index of the dataset

buffer	Pointer to the target buffer
opStatus	Operation status
<b>Return code</b>	
WRAPNV_E_OK	Read operation successful
WRAPNV_E_NOT_OK	Read operation failed
WRAPNV_E_PENDING	Operation is still in progress
<b>Functional Description</b>	
This function reads the data from Fee/Nvm in the respective buffer.	
<b>Particularities and Limitations</b>	
> NV-Wrapper has to be initialized.	

Table 4-6 WrapNv\_ReadAsync

#### 4.1.1.3.2 WrapNv\_ReadPartialAsync

<b>Prototype</b>	
WrapNv_ReturnType WrapNv_ReadPartialAsync(uint16 id, uint16 idx, tWrapNvRamData buffer, uint16 offset, uint16 length, tWrapNvOpStatus opStatus)	
<b>Parameter</b>	
id	ID of record
idx	Index of the dataset
buffer	Pointer to the target buffer
offset	Offset in element from where to start to read
length	Length of the partial read operation
opStatus	Operation status
<b>Return code</b>	
WRAPNV_E_OK	Read operation successful
WRAPNV_E_NOT_OK	Read operation failed
WRAPNV_E_PENDING	Operation is still in progress
<b>Functional Description</b>	
This function reads the data from Fee/Nvm in the respective buffer.	
<b>Particularities and Limitations</b>	
> NV-Wrapper has to be initialized.	

Table 4-7 WrapNv\_ReadPartialAsync



#### 4.1.1.3.3 WrapNv\_WriteAsync

Prototype	
<code>WrapNv_ReturnType WrapNv_WriteAsync( uint16 id, uint16 idx, tWrapNvRamData buffer, tWrapNvOpStatus opStatus )</code>	
Parameter	
<code>id</code>	ID of record
<code>idx</code>	Index of the dataset
<code>buffer</code>	Pointer to the source buffer
<code>opStatus</code>	Operation status
Return code	
<code>WRAPNV_E_OK</code>	Read operation successful
<code>WRAPNV_E_NOT_OK</code>	Read operation failed
<code>WRAPNV_E_PENDING</code>	Operation is still in progress
Functional Description	
This function reads the data from Fee/Nvm in the respective buffer.	
Particularities and Limitations	
> NV-Wrapper has to be initialized.	

Table 4-8      WrapNv\_WriteAsync

#### 4.1.1.3.4 WrapNv\_DeleteAsync

Prototype	
<code>WrapNv_ReturnType WrapNv_DeleteAsync( uint16 id, uint16 idx, tWrapNvOpStatus opStatus )</code>	
Parameter	
<code>id</code>	ID of record
<code>idx</code>	Index of the dataset
<code>opStatus</code>	Operation status
Return code	
<code>WRAPNV_E_OK</code>	Delete (invalidate) operation successful
<code>WRAPNV_E_NOT_OK</code>	Delete (invalidate) operation failed
<code>WRAPNV_E_PENDING</code>	Operation is still in progress

Functional Description
This function reads the data from Fee/Nvm in the respective buffer.
Particularities and Limitations
> NV-Wrapper has to be initialized.

Table 4-9 WrapNv\_DeleteAsync

## 4.2 Naming conventions

The access macros generated by the configuration tool have the following naming conventions:



### Example

`ApplWrapNv<operation><mode><element-name>()`

- ▶ The prefix of ApplWrapNv is for every access macro the same.
- ▶ Operation can either be Read, ReadPartial, Write or Delete.
  - ▶ Read: Reads a complete element from NV block
  - ▶ ReadPartial: Reads only a part of the element from the NV block
  - ▶ Write: Writes an element into the NV block
  - ▶ Delete: Deletes/invalidates a complete NV block
- ▶ Mode is used to distinguish between the synchronous and asynchronous API. If no mode is given, by default this represents the synchronous interface. For asynchronous interface the access macro has the mode "Async".
- ▶ The element name is generated from the data element specified in the configuration tool.



### Note

Not every NV module supports the Delete operation.

**Note**

In case the used NV driver can only be used synchronously, the asynchronous interface is mapped to the synchronous interface.

### 4.3 Interface definitions

Single	
synchronous	WrapNv_ReturnType <b>ApplWrapNv&lt;operation&gt;Single</b> (buf)
asynchronous	WrapNv_ReturnType <b>ApplWrapNv&lt;operation&gt;AsyncSingle</b> (buf, op)
Parameter	
uint8 * buf	Pointer to the target buffer (read) or the source buffer (write).
uint8 op	Operation status
Return code	
WrapNv_ReturnType	WRAPNV_E_OK if the memory access was successful, WRAPNV_E_NOT_OK if a failure occurred. WRAPNV_E_PENDING if operation is in progress (only in asynchronous mode)
Functional Description	
Reads or writes a single data structure from the configured memory device and address.	
Particularities and Limitations	
> There is no range check of the target buffer. Please make sure it is big enough, otherwise the buffer will overflow.	
Call context	
> Same call context as the used memory driver.	

Table 4-10 Single

List	
synchronous	WrapNv_ReturnType <b>ApplWrapNv&lt;operation&gt;List</b> (idx, buf)
asynchronous	WrapNv_ReturnType <b>ApplWrapNv&lt;operation&gt;AsyncList</b> (idx, buf, op)
Parameter	
uint8 idx	Index of the desired element within the list.
uint8* buf	Pointer to the target buffer (read) or the source buffer (write).
uint8 op	Operation status

Return code	
WrapNv_ReturnType	WRAPNV_E_OK if the memory access was successful, WRAPNV_E_NOT_OK if a failure occurred. WRAPNV_E_PENDING if operation is in progress (only in asynchronous mode)
Functional Description	
Reads or writes a list element from a list data structure.	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>&gt; The list index is not checked. Please make sure only valid list indexes are supplied.</li> <li>&gt; There is no range check of the target buffer. Please make sure it is big enough, otherwise the buffer will overflow.</li> </ul>	
Call context	
<ul style="list-style-type: none"> <li>&gt; Same call context as the used memory driver.</li> </ul>	

Table 4-11 List

Structure	
synchronous	WrapNv_ReturnType <b>ApplWrapNv&lt;operation&gt;StructureMember</b> (idx, buf)
asynchronous	WrapNv_ReturnType <b>ApplWrapNv&lt;operation&gt;AsyncStructureMember</b> (idx, buf, op)
Parameter	
vuint8 idx	List index. This parameter is ignored for the actual memory driver call.
vuint8* buf	Pointer to the target buffer (read) or the source buffer (write).
uint8 op	Operation status
Return code	
WrapNv_ReturnType	WRAPNV_E_OK if the memory access was successful, WRAPNV_E_NOT_OK if a failure occurred. WRAPNV_E_PENDING if operation is in progress (only in asynchronous mode)
Functional Description	
Reads or writes a single member from a structure data element.	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>&gt; The list index is ignored.</li> <li>&gt; There is no range check of the target buffer. Please make sure it is big enough, otherwise the buffer will overflow.</li> </ul>	
Call context	
<ul style="list-style-type: none"> <li>&gt; Same call context as the used memory driver.</li> </ul>	

Table 4-12 Structure

Table	
synchronous	WrapNv_ReturnType <b>ApplWrapNv&lt;operation&gt;TableMember</b> (idx, buf)
asynchronous	WrapNv_ReturnType <b>ApplWrapNv&lt;operation&gt;AsyncTableMember</b> (idx, buf, op)
Parameter	
vuint8 idx	Index of the desired table row.
vuint8* buf	Pointer to the target buffer (read) or the source buffer (write).
uint8 op	Operation status
Return code	
WrapNv_ReturnType	WRAPNV_E_OK if the memory access was successful, WRAPNV_E_NOT_OK if a failure occurred. WRAPNV_E_PENDING if operation is in progress (only in asynchronous mode)
Functional Description	
Reads or writes a structure data element from the desired table row.	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>&gt; The list index is not checked. Please make sure only valid list indexes are supplied.</li> <li>&gt; There is no range check of the target buffer. Please make sure it is big enough, otherwise the buffer will overflow.</li> </ul>	
Call context	
<ul style="list-style-type: none"> <li>&gt; Same call context as the used memory driver.</li> </ul>	

Table 4-13      Table

## 5 Configuration

### 5.1 GENy configuration

To use the module in GENy, it has to be activated in GENy's Component Selection:

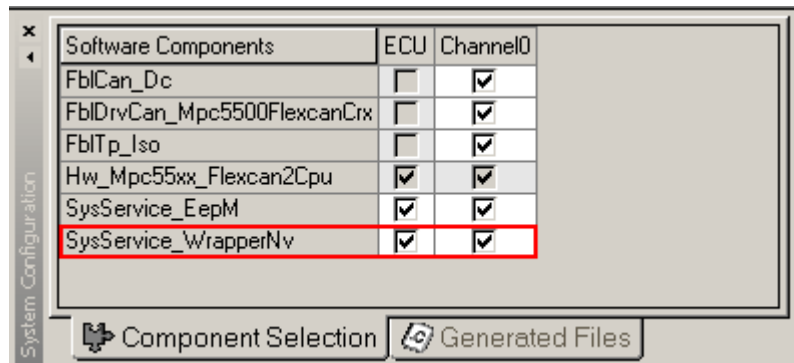


Figure 5-1 Activation of NV-Wrapper in GENy

The NV-Wrapper module can be divided into the tree view which represents the data structure on the left side of the GENy window and the detail view on the right side:

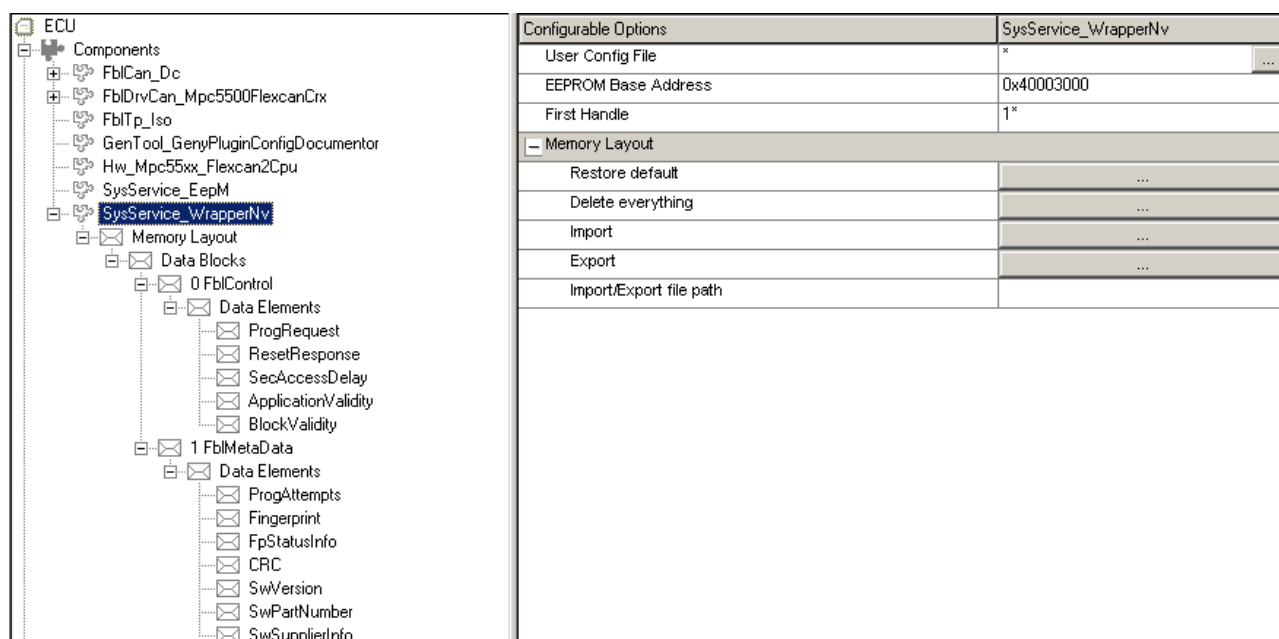


Figure 5-2 NV-Wrapper GENy module

#### 5.1.1 Basic settings

The basic settings (see the detail view on the right side of Figure 5-2) configure some global options used for all data structures.

Configuration Option	Description
User Config File	Text file with additional includes into WrapNv_Cfg.h
EEPROM Base Address	Address offset added to EEPROM driver calls
First Handle	Start values of the generated IDs
Restore default	Restores defaults from preconfiguration
Delete everything	Delete all configured structures
Import	Read configuration from .xml file
Export	Write configuration to .xml file
Import/Export file path	Path for import and export. If a relative path is entered, the GENy generation directory is used as base directory.

Table 5-1 GENy configuration options for Nv-Wrapper

**Note**

It is recommended to use an absolute path for import and export to avoid confusion regarding the used directory.

### 5.1.2 Tree view usage

The data structure used by NV-Wrapper is configured with the tree view on the left part of the NV-Wrapper GENy module. To modify the data structure, the context menu accessed with the right mouse button is used.

There are two types of context menus, the context menus of the Data nodes and the context menus of the actual blocks and elements:

#### 5.1.2.1 Data node context menu

The data node context menu includes an “Add” context menu. This context menu adds a new node as child element to the selected node.

#### 5.1.2.2 Data block/element context menu

The data block or data element context menus provide three possibilities:

1. Delete: Removes the selected element.
2. Move up: Changes the selected element's position with the element placed in front of the selected element.
3. Move down: Moves the selected element to a position behind the following element.

### 5.1.3 Data configuration

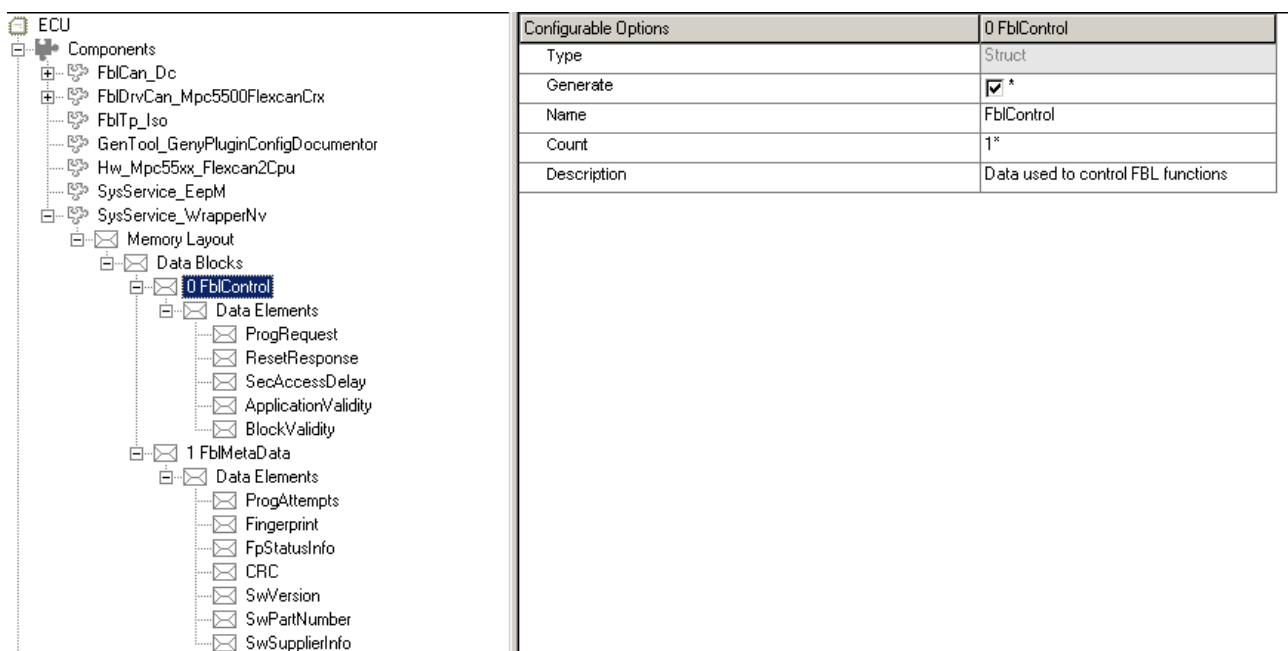
The data identifiers and structure is configured starting with the “Memory Layout\Data Blocks” node of the GENy component. A data block is the base node of every data

structure and is set to one of the four possible types depending on the count of data blocks respective child elements:

At least one data element is assigned to every data block. If one data element is assigned, a single or list data structure is generated, if more than one data element is assigned, a struct or table element is generated.

The options of present objects are configured on the right side of the GENy window. Objects can be added, moved and removed by the context menu of the tree structure.

### 5.1.3.1 Data Blocks



The screenshot shows the GENy window with a tree structure on the left and a configuration table on the right. The tree structure is as follows:

- ECU
  - Components
    - FblCan\_Dc
    - FblDrvCan\_Mpc5500FlexcanCrx
    - FblTp\_Iso
    - GenTool\_GenyPluginConfigDocumentor
    - Hw\_Mpc55xx\_Flexcan2Cpu
    - SysService\_EepM
    - SysService\_WrapperNv
      - Memory Layout
        - Data Blocks
          - 0 FblControl** (selected)
            - Data Elements
              - ProgRequest
              - ResetResponse
              - SecAccessDelay
              - ApplicationValidity
              - BlockValidity
            - 1 FblMetaData
              - Data Elements
                - ProgAttempts
                - Fingerprint
                - FpStatusInfo
                - CRC
                - SwVersion
                - SwPartNumber
                - SwSupplierInfo

The configuration table on the right is titled 'Configurable Options' and has the following data:

Configurable Options	
Type	Struct
Generate	<input checked="" type="checkbox"/> *
Name	FblControl
Count	1*
Description	Data used to control FBL functions

Figure 5-3 Data Block configuration

Data blocks are used to group data elements. There are some settings valid for all members of this data block. Each field is described below:

Configuration Option	Description
Type	Single, List, Struct or Table depending on the Count setting and the number of Data Elements
Generate	If checked, the data block will be included into the generated data structure. This includes reserved memory respective IDs and the access macros. Otherwise, the data blocks are not visible in the generated code
Name	Name of data block
Count	Specifies the number of instances of this data block. If Count is set to one, the Data Block will be type "Single" or "Struct", otherwise it will be "List" or "Table"
Description	Description of the data block

Table 5-2 Data Block configuration



### 5.1.3.2 Data Element Settings

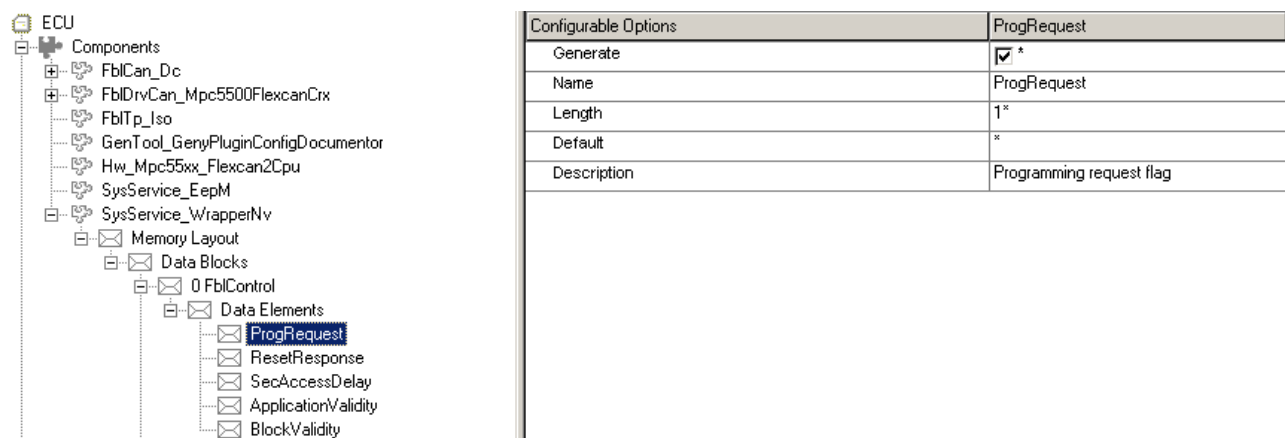


Figure 5-4 Data Element configuration

Data Elements contain the settings used by one member of a Data Block. The following settings can be configured:

Configuration Option	Description
Generate	Check if the Data Element should be generated
Name	Name of data block
Length	Length of data block in bytes
Default	Not implemented yet
Description	Description of the data block

Table 5-3 Data Block configuration

## 6 Terminology

### 6.1 Abbreviations

Abbreviation	Description
API	Application Programming Interface
DrvEep	Eeprom Driver
EepM	Eeprom Manager
Ea	EEPROM Abstraction (Autosar)
FBL	Flash Bootloader
Fee	Flash EEPROM Emulation Module (Autosar)
FLIO	Flash IO
Fls	Flash Driver (Autosar)
NV	Non-Volatile
NvM	Non Volatile RAM Manager (Autosar)

## 7 Contact

Visit our website for more information on

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

**[www.vector.com](http://www.vector.com)**