

NV-Wrapper

Technical Reference

Wrapper for non-volatile memory access

Version 1.0

Authors	Christian Bäuerle, Achim Strobel
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

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

1	Introduction	6
2	Functional Description	7
2.1	Architecture.....	7
2.2	Data structures	7
2.2.1	Single.....	7
2.2.2	List	7
2.2.3	Structure	8
2.2.4	Table	8
2.3	NV-Wrapper in Application	8
3	Integration.....	9
3.1	Scope of delivery	9
3.2	Include structure	9
3.3	Usage hints	10
3.3.1	Initialization	10
3.3.2	Memory accesses	10
4	Configuration.....	11
4.1	Basic settings.....	12
4.2	Tree view usage.....	12
4.2.1	Data node context menu.....	12
4.2.2	Data block/element context menu.....	12
4.3	Data configuration.....	13
4.3.1	Data Blocks.....	13
4.3.2	Data Element Settings	14
5	API Description	15
5.1	Naming conventions	15
5.1.1	Read functions	15
5.1.2	Write functions	15
5.2	Interface definitions.....	15
6	Glossary and Abbreviations.....	18
6.1	Glossary.....	18
6.2	Abbreviations	18
7	Contact.....	19

Illustrations

Figure 2-1	NV-Wrapper in Flash Bootloader.....	7
Figure 3-1	Include hierarchy of NV-Wrapper (with EEPROM manager in FBL)	9
Figure 4-1	Activation of NV-Wrapper in GENy	11
Figure 4-2	NV-Wrapper GENy module	11
Figure 4-3	Data Block configuration	13
Figure 4-4	Data Element configuration	14

Tables

Table 5-1	Single	16
Table 5-2	List.....	16
Table 5-3	Structure.....	17
Table 5-4	Table.....	17

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 or the EEPROM emulation by the Vector EEPROM manager can be accessed.

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

2 Functional Description

The NV-Wrapper module mainly consists of a generated file which provides access macros. Additionally, a C header file provides some supporting defines.

2.1 Architecture

The NV-Wrapper is a layer between the application / FBL and the EEPROM manager or EEPROM driver.

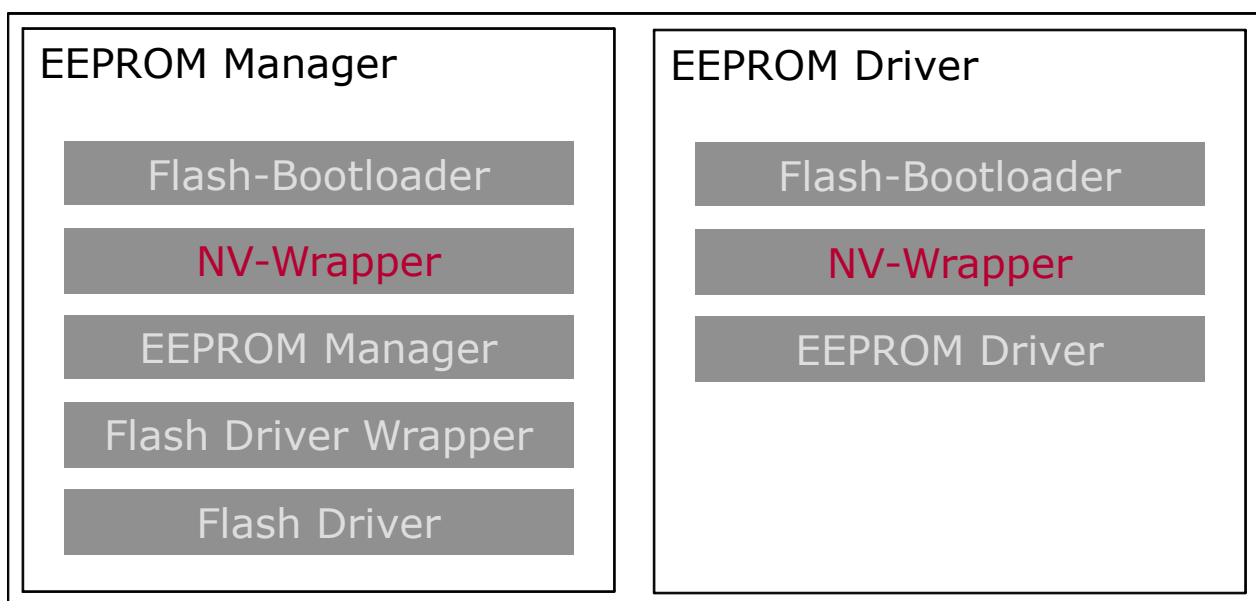


Figure 2-1 NV-Wrapper in Flash Bootloader

The left part of Figure 2-1 shows the usage of the NV-Wrapper with the EEPROM Manager and the right part the usage in connection with an EEPROM driver.

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.

2.2.2 List

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

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.

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.

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.

3.1 Scope of delivery

The delivery of the NV-Wrapper module consists of the following files:

File name	Description	Status
WrapNv.h	Main module header file. It contains some preprocessor macros needed by the generated files.	
_WrapNv_Inc.h	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. Copy this file into your working folder, remove the leading underscore and make your modifications as needed by e.g. adding or modifying include statements.	

3.2 Include structure

The NV-Wrapper centralizes the import of all required header files in WrapNv_inc.h. The exported API prototypes and macros are specified in WrapNv.h. The configuration settings are located in WrapNv_cfg.h.

If the EEPROM Manager is used, the include hierarchy looks as depicted in Figure 3 1. Otherwise, the file WrapNv_Inc.h would include the header files of the EEPROM driver used by the NV-Wrapper module.

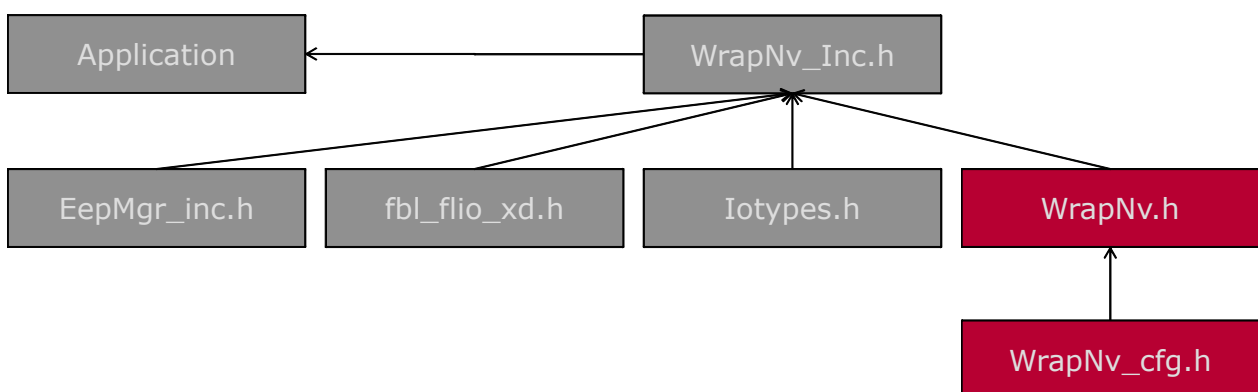


Figure 3-1 Include hierarchy of NV-Wrapper (with EEPROM manager in FBL)

3.3 Usage hints

3.3.1 Initialization

The NV-Wrapper itself doesn't require an 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.



Example

Initialization if EepMgr is used:

```
/* Initialize EEPROM-Driver */
Flash_XDDriver_InitSync(NULL);
(void)EepMgrInitPowerOn(kEepMgrInitFull);
```

Initialization if a EEPROM driver is used

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

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

3.3.2 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(0, &buffer) != kFb1Ok)
{
    /* 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 Configuration

The NV-Wrapper is configured in GENy. To use the module, it has to be activated in GENy's Component Selection:

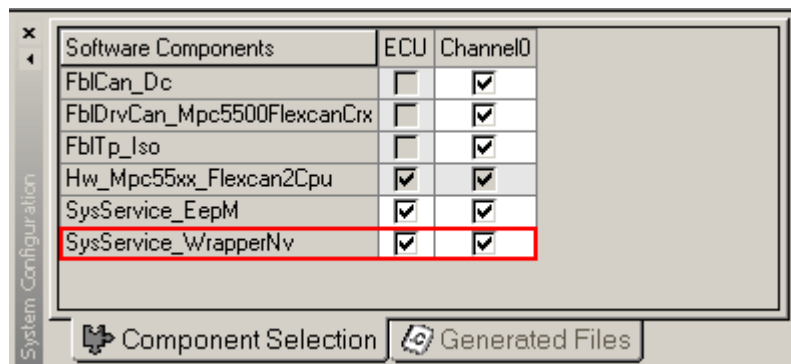


Figure 4-1 Activation of NV-Wrapper in GENy

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:

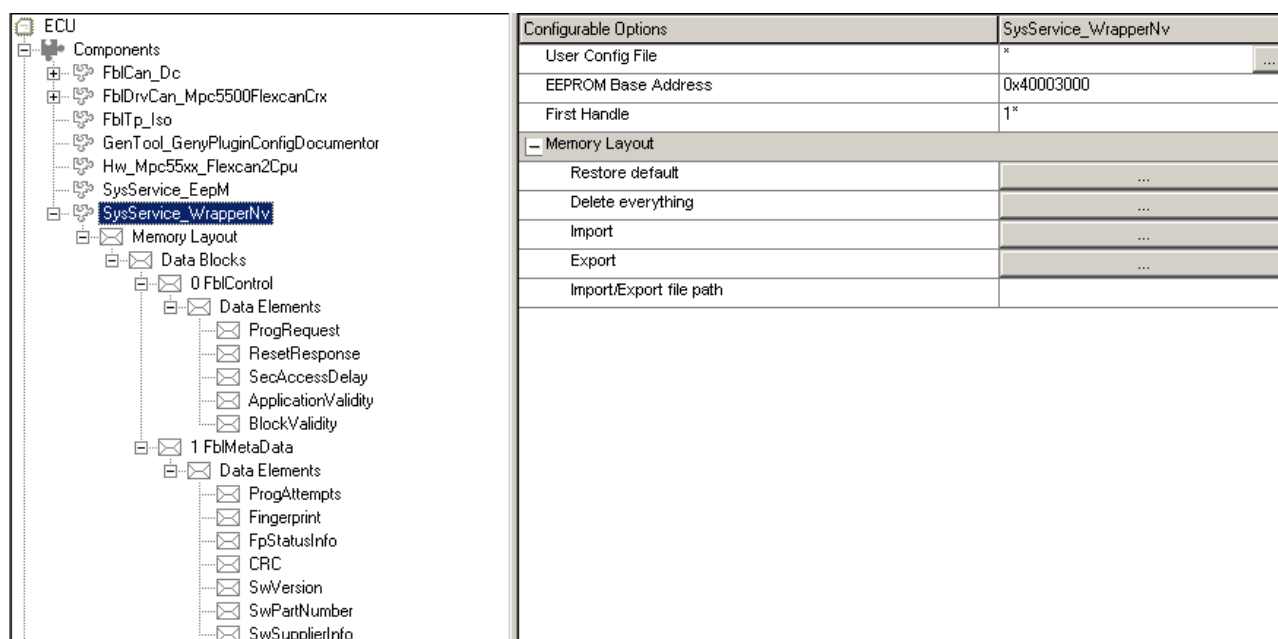


Figure 4-2 NV-Wrapper GENy module

4.1 Basic settings

The basic settings (see the detail view on the right side of Figure 4-2 NV-Wrapper GENy module) configure some global options used for all data structures:

- ▶ 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/Export: Read configuration from .xml file and 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.



Note

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

4.2 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 2 types of context menus, the context menus of the Data¹ nodes and the context menus of the actual blocks and elements:

4.2.1 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.

4.2.2 Data block/element context menu

The data block or data element context menus provide 2 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.

¹ Data Blocks and Data Elements

4.3 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 1 of the 4 possible types depending on the count of data blocks respective child elements.

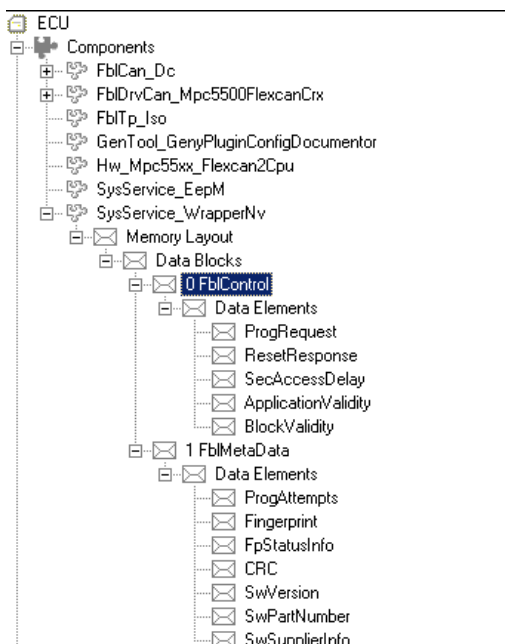
At least 1 data element is assigned to every data block. If 1 data element is assigned, a single or list data structure is generated, if more than 1 data element is assigned, a struct or table element is generated.

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.

4.3.1 Data Blocks

Data blocks are used to group data. There are some settings valid for all members of this data block:

- ▶ **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**
- ▶ **Count:** Specifies the number of instances of this data block. If Count is set to 1, the Data Block will be type “Single” or “Struct”, if Count is bigger than 1, it will be “List” or “Table”
- ▶ **Description**

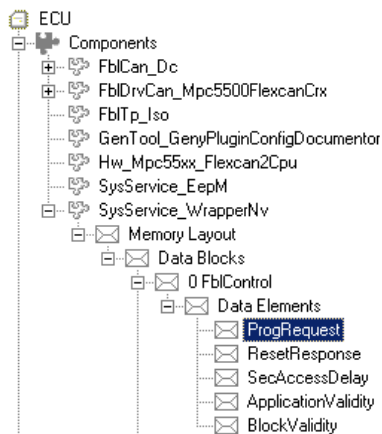


The screenshot shows the GENy interface with a tree view on the left and a configuration table on the right. The tree view shows the hierarchy: ECU > Components > FblCan_Dc > FblDrvCan_Mpc5500FlexcanCrx > FblTp_Iso > GenTool_GenyPluginConfigDocumentor > Hw_Mpc55xx_Flexcan2Cpu > SysService_EepM > SysService_WrapperNv > Memory Layout > Data Blocks > 0 FblControl. The configuration table on the right shows the following options:

Configurable Options		0 FblControl
Type		Struct
Generate		<input checked="" type="checkbox"/> *
Name		FblControl
Count		1*
Description		Data used to control FBL functions

Figure 4-3 Data Block configuration

4.3.2 Data Element Settings



Configurable Options	ProgRequest
Generate	<input checked="" type="checkbox"/> *
Name	ProgRequest
Length	1*
Default	*
Description	Programming request flag

Figure 4-4 Data Element configuration

Data Elements contain the settings used by 1 member of a Data Block. The following settings can be configured:

- ▶ **Generate:** Check if the Data Element should be generated
- ▶ **Name**
- ▶ **Length:** Length of variable in Bytes
- ▶ **Default:** Not implemented yet
- ▶ **Description**

5 API Description

The NV-Wrapper module doesn't include hardcoded functions, but the macros generated by the module use specified names and interfaces. These names and interfaces are described here.

5.1 Naming conventions

5.1.1 Read functions

The read functions are composed using the following name parts:



Example

```
ApplFblNvReadVariableName()
```

- ▶ Prefix ApplFblNvRead: Used as prefix by all NV-Wrapper read macros.
- ▶ VariableName: Name of the Data Element specified in GENy.

5.1.2 Write functions

The write functions are composed following the same rules than the read functions, but the prefix ApplFblNvWrite is used:



Example

```
ApplFblNvWriteVariableName()
```

5.2 Interface definitions

Single	
Read function	<code>vuint8 ApplFblNvReadSingle(buf)</code>
Write function	<code>vuint8 ApplFblNvWriteSingle(buf)</code>
Parameter	
<code>vuint8* buf</code>	Pointer to the target buffer (read) or the source buffer (write).
Return code	
<code>vuint8</code>	<code>kFbIOk</code> if the memory access was successful, <code>kFblFailed</code> if a failure occurred.

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 5-1 Single

List	
Read function	<code>vuint8 ApplFblNvReadList(idx, buf)</code>
Write function	<code>vuint8 ApplFblNvWriteList(idx, buf)</code>
Parameter	
<code>vuint8 idx</code>	Index of the desired element within the list.
<code>vuint8* buf</code>	Pointer to the target buffer (read) or the source buffer (write).
Return code	
<code>vuint8</code>	<code>kFblOk</code> if the memory access was successful, <code>kFblFailed</code> if a failure occurred.
Functional Description	
Reads or writes a list element from a list data structure.	
Particularities and Limitations	
<ul style="list-style-type: none">> The list index is not checked. Please make sure only valid list indexes are supplied.> There is no range check of the target buffer. Please make sure it is big enough, otherwise the buffer will overflow.	
Call context	
<ul style="list-style-type: none">> Same call context as the used memory driver.	

Table 5-2 List

Structure	
Read function	vuint8 ApplFblNvReadStructureMember (idx, buf)
Write function	vuint8 ApplFblNvWriteStructureMember (idx, buf)
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).

Return code	
vuint8	kFbIOk if the memory access was successful, kFbIFailed if a failure occurred.
Functional Description	
Reads or writes a single member from a structure data element.	
Particularities and Limitations	
<ul style="list-style-type: none"> > The list index is ignored. > There is no range check of the target buffer. Please make sure it is big enough, otherwise the buffer will overflow. 	
Call context	
<ul style="list-style-type: none"> > Same call context as the used memory driver. 	

Table 5-3 Structure

Table	
Read function	vuint8 App1Fb1NvReadTableMember (idx, buf)
Write function	vuint8 App1Fb1NvWriteTableMember (idx, buf)
Parameter	
vuint8 idx	Index of the desired table row.
vuint8* buf	Pointer to the target buffer (read) or the source buffer (write).
Return code	
vuint8	kFbIOk if the memory access was successful, kFbIFailed if a failure occurred.
Functional Description	
Reads or writes a structure data element from the desired table row.	
Particularities and Limitations	
<ul style="list-style-type: none"> > The list index is not checked. Please make sure only valid list indexes are supplied. > There is no range check of the target buffer. Please make sure it is big enough, otherwise the buffer will overflow. 	
Call context	
<ul style="list-style-type: none"> > Same call context as the used memory driver. 	

Table 5-4 Table

6 Glossary and Abbreviations

6.1 Glossary

none

6.2 Abbreviations

Abbreviation	Description
FLIO	Flash IO
Nv	Non-volatile

7 Contact

Visit our website for more information on

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

www.vector.com