Navigation & Multimedia System & Architecture
Specification of NVM component

1 Introduction

This document describes the application specific interface of the Non Volatile Memory functions. The APIs Specification reports all the structures and function call available at higher level related to the NVM component.

2 Contents

2.1 Index

2.2 List of Tables

2.3 List of Figures

Document Management

2.4 Revision History

Rev	Date	Author	Notes
1.0		P. Bagnall	Release for Vespucci
2.0	2006/05/22	O. Ballan	Reviewed
3.0	2007/12/09	F. Boggia	Totally rewritten
3.1	2007/12/13	F. Boggia	New chapter describing component; New values for return status.
3.2	2008/01/29	F. Boggia	Corrected `nvm_create` API.
3.3	2009/05/14	F. Boggia	Updated document layout
3.4	2011/04/20	F. Boggia	Updated document layout
3.5	2012/04/30	F. Boggia	Updated document layout Added application note

Table 1: Revision history

2.5 Acronyms

Keyword	Definition

Table 2. Acronyms

2.6 Reference Documents

None

3 Interface description

The NVM component is studied to support a unified access method to store information that should be recallable. The specific implementation changes based on the hardware and software components, and is realized in terms of BSPs specific for the application. It could provide a direct access to NAND, NOR or to a volatile buffer in main memory which the customer application must take care of storing it.

Figure 1 shows a simple block diagram that summarizes the operations implemented in the NVM component.

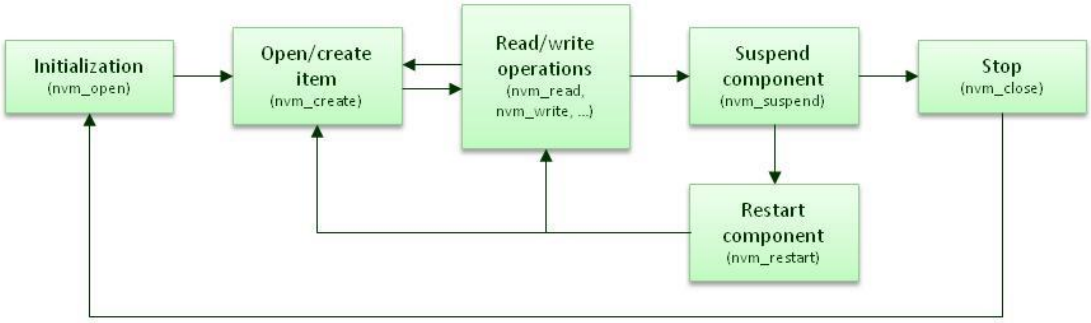

Figure 1: NVM operations diagram

After the initialization of the component, each item stored in the NVM component must be created. This item could already exist or it's created from scratch. Once the item is created, read/write operations could be done on it. For the read operation, two methods are allowed: direct access to the item in the NVM component (`nvm_read`) or a copy of the component to a local item (`nvm_copy`). Whenever the application wants to perform specific tasks (write back of the buffer, shut_down, …) a suspend command must be called to stop NVM working. When done, a restart command will re enable the access to the component to the other components operating on the system. NVM component could be stopped only after a suspend command.

For some implementations (example direct access to a NOR), a double banks implementation could be realized. In these cases, a method to force a swap between the two banks is provided.

4 Interface specification

This section describes type and function components of the NVM library.

4.1 Type Interface Definitions

This section describes the data type components of the NVM component, visible to the Application.

4.1.1 Constants

None.

4.1.2 Structures

The following structures are all contained in `nvm.h`:

```
typedef enum nvm_status_e
{
  NVM_NO_ERROR       = 0,
  NVM_ERROR          = 1,
  NVM_ITEM_VALID     = 2,
  NVM_ITEM_NOTVALID  = 3
} nvm_status_t;
```

This is the type returned by all NVM access methods. It indicates the success (`NVM_NO_ERROR`) or failure (`NVM_ERROR`) of the method called. Some methods could return a specific status (see methods descriptions).

4.2 Function call interfaces

This section describes in a one-function-per-page fashion the prototypes and the input/output parameters of each NVM function visible to the user.

4.2.1 nvm_open

Initializes the NVM component.

Synopsis:

```
#include "nvm.h"

nvm_status_t nvm_open(
  const unsigned int version_id,
  void *primary_bank_ptr,
  void *secondary_bank_ptr,
  const unsigned int size
);
```

Arguments:

`int version_id`	Version of the information stored in NVM.
`void *primary_bank_ptr`	Pointer to the primary bank location.
`void *secondary_bank_ptr`	Pointer to the secondary bank location.
`const unsigned int size`	Size of each bank.

Results:

Returns `NVM_NO_ERROR` if initialization is successful, NVM_ERROR otherwise.

Errors:

Returns `NVM_ERROR` if an error occurs.

Description:

Initializes the NVM as specified by parameters. Banks that must be used by NVM must be preliminarly allocated. Use of secondary bank depends on implementation. If implementation does not need it, a `NULL` pointer must be passed.

The `version_id` is used to validate data that NVM module will find in banks specified. It is used as a tag, and if the tag is not matched in all specified banks, the banks will be treated as uninitialized and will be completely erased and initialized to be usable by NVM module.

Note: All information stored in memory banks will be lost if the `version_id` tag is not matched.

4.2.2 nvm_create

Open/create a set of copies of an item stored/to be stored in NVM component.

Synopsis:

```
#include "nvm.h"

nvm_status_t nvm_create(
   const unsigned int id,
   const int max_items,
   const int max_copies,
   const int item_size
);
```

Arguments:

`const unsigned int id`	ID of the item set to be opened/created;
`const int max_items`	Max number of items of this type to be allocated;
`const int max_copies`	Number of copies to be stored for each item;
`const int item_size`	Size of each item.

Results:

It returns:

- `NVM_NO_ERROR` if the set of items is successfully opened/created;
- `NVM_ERROR` otherwise.

Errors:

Returns `NVM_ERROR` if an error occurs.

Description:

This method must be used every time the user wants to add a new item or open an existing one in NVM component. If the item already exists, the method will exit and returns no error. If the item does not exist, the method tries to allocate a new one in NVM component (for both primary and secondary banks if present). If there is enough space for all copies of all items, the needed space is allocated and no error is returned. If there is not enough space, an error is returned.

Note: if the NVM is allocated in RAM, the `max_copies` parameter is not effective.

4.2.3 nvm_write

Writes an item to NVM component.

Synopsis:

```
#include "nvm.h"

nvm_status_t nvm_write(
   const unsigned int id,
   const int item_number,
   const void * data_ptr
);
```

Arguments:

`const unsigned int id`	ID of the item to be written;
`const int item_number`	Item to be written for provided ID;
`const void * data_ptr`	Pointer to buffer containing item.

Results:

Returns:

- `NVM_NO_ERROR` if the item is written successfully on NVM component;
- `NVM_ERROR` otherwise.

Errors:

Returns `NVM_ERROR` if an error occurs.

Description:

This method writes an item in a set identified by id previously allocated with `nvm_create`.

4.2.4 nvm_read

Reads an item from NVM component.

Synopsis:

```
#include "nvm.h"

nvm_status_t nvm_read(
   const unsigned int id,
   const int item_number,
   void ** data_ptr
);
```

Arguments:

`const unsigned int id`	ID of the item set to be read;
`const int item_number`	Item to be read for provided ID;
`void ** data_ptr`	Pointer where to copy the pointer to the item in NVM component.

Results:

Returns:

- `NVM_ITEM_VALID` if the item is read successfully from NVM component;
- `NVM_ITEM_NOTVALID` if the item is allocated but not valid;
- `NVM_ERROR` if an error occurred while reading or the item does not exist.

Errors:

Returns `NVM_ERROR` if an error occurs.

Description:

This method reads an item from a set identified by id previously allocated with nvm_create. If there are no copy already written for the item, `NVM_ERROR` is returned.

Note that the method returns the pointer to the item, but does not copy the item in NVM to a local item. This could be done with `nvm_copy`.

4.2.5 nvm_copy

Copies an item from NVM component.

Synopsis:

```
#include "nvm.h"

nvm_status_t nvm_copy(
   const unsigned int id,
   const int item_number,
   void * data_ptr
);
```

Arguments:

`const unsigned int id`	ID of the item set to be read;
`const int item_number`	Item to be read for provided ID;
`void * data_ptr`	Pointer where to copy the item from NVM component.

Results:

Returns:

- `NVM_ITEM_VALID` if the item is copied successfully from NVM component;
- `NVM_ITEM_NOTVALID` if the item does not exist;
- `NVM_ERROR` if an error occurred while reading.

Errors:

Returns `NVM_ERROR` if an error occurs.

Description:

This method copies an item from a set identified by id previously allocated with `nvm_create`. If there are no copy already written for the item, `NVM_ERROR` is returned.

Note that the method copies an entire item where requested. Instead `nvm_copy` can be used to point directly to the item in NVM component.

4.2.6 nvm_set_item_invalid

Invalidates an item from NVM component.

Synopsis:

```
#include "nvm.h"

nvm_status_t nvm_set_item_invalid(
   const unsigned int id,
   const int item_number
);
```

Arguments:

`const unsigned int id` ID of the item set to be invalidated;

`const int item_number` Item to be invalidated for provided ID.

Results:

Returns:

- `NVM_NO_ERROR` if the item is successfully invalidated in NVM component;
- `NVM_ERROR` if the item does not exist or an error occurred while trying to invalidate it.

Errors:

Returns `NVM_ERROR` if an error occurs.

Description:

This method tries to invalidate an item from a set identified by id previously allocated with `nvm_create`.

4.2.7 nvm_swap

Swaps primary and secondary banks.

Synopsis:

```
#include "nvm.h"

nvm_status_t nvm_swap( void);
```

Arguments:

None.

Results:

Returns:

- `NVM_NO_ERROR` if the swap is successfully;
- `NVM_ERROR` otherwise.

Errors:

Returns `NVM_ERROR` if an error occurs.

Description:

This method swaps primary and secondary banks in NVM component. If no secondary bank was specified by `nvm_init`, `NVM_ERR`OR is returned.

4.2.8 nvm_suspend

Suspends NVM component operations.

Synopsis:

```
#include "nvm.h"

void nvm_suspend( void);
```

Arguments:

None.

Results:

None

Errors:

None.

Description:

This method waits the NVM component to be free and then locks the resource. It could be used to freeze the component status before a powerdown of the system.

4.2.9 nvm_restart

Restarts NVM component operations.

Synopsis:

```
#include "nvm.h"

void nvm_restart( void);
```

Arguments:

None.

Results:

None

Errors:

None.

Description:

This method re-enables the NVM standard operations. It should be used after a nvm_suspend to restart te component.

5 Application notes

As already anticipated, NVM module is a common interface to access and retrieve information from a non-volatile memory. The implementation of this module could vary a lot. Basically there are two main implementations:

- NVM module in a volatile RAM
- NVM module in a non-volatile memory (flash)

They mainly differ in how write operation is performed. This application note will clarify some aspects about these two implementations.

5.1 NVM module in a volatile RAM

The NVM module in RAM is useful when

- There is planty of RAM
- Running OS can store information on different places (file systems over SD or NOR or whatelse)

In this case, before starting using NVM APIs, a buffer in RAM must be prepared. The buffer must be allocated in RAM before `nvm_open` is called and must be filled with a copy from non-volatile support (a file for example) if it was saved before.

When NVM is allocated in a volatile RAM, there is no specific procedure to write it. So, a write always overwrites previous value in NVM. This means that the number of copies specified when invoking `nvm_create` is ineffective, as the copy written is always the same.

Before power down, the NVM area must be saved, otherwise all operations done on it would be lost forever. So, the NVM area must be copied back to a non-volatile support (a file for example).

The saving operation could also be done periodically if needed.

5.2 NVM module in a non-volatile memory (flash)

The NVM module in a non-volatile memory is mainly used in OS20+ based applications. Its main purpose is to deal with physical supports (NOR memories, SQI memories, …) that needs specific algorithms for writing operations.

Mainly, these memories allow to write a bit from 1 to 0, but when a bit must be set high, the sector containing the bit must be erased to get back the 1 state. The erasing operation leads to aging of memory itself. To avoid too many erase operations, the NVM module provide a way to have more copies of the same item. Of these copies, only one is valid. When there is no more space for new copies, the whole NVM area is copied in a secondary bank and the erase operation of the full sector is performed.

This means that the number of copies must be well chosen to minimize the number of erasing events. The best performances can be obtained if the `max_copies` parameter used when `nvm_create` is called obey to the following simple rule:

$$max\_copies > \frac{3h}{num\_of\_writes}$$

where `max_copies` is the parameter used when calling `nvm_create` and `num_of_writes` is the number of writes of the wanted item that should happen in two hours. If the number of copies is too small, the erasing could happen too often and performances of physical support could worsen.

5.3 NVM item IDs

Each NVM item has its own ID. A set of NVM IDs are used for GNSS library specific items and cannot be used by custom application. Here are a table of sets allocated by GNSS library and a set of IDs usable by custom application.

The maximum number of different IDs is 32. The maximum value allowed for NVM IDs is 255.

Here is a table describing currently allocated ranges.

NVM IDs Range	Usage
0-63	GNSS library
64-79	AGPS library
80-126	GNSS Application libraries
127	SW config module

Table 3: NVM IDs ranges

So, the NVM IDs available for custom data to be stored in NVM must be in range of 128-255.

6 Disclaimer

Please Read Carefully:

Information in this document is provided solely in connection with ST products. STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at any time, without notice.

All ST products are sold pursuant to ST’s terms and conditions of sale.

Purchasers are solely responsible for the choice, selection and use of the ST products and services described herein, and ST assumes no liability whatsoever relating to the choice, selection or use of the ST products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by ST for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

UNLESS OTHERWISE SET FORTH IN ST’S TERMS AND CONDITIONS OF SALE ST DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ST PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS EXPRESSLY APPROVED IN WRITING BY AN AUTHORIZED ST REPRESENTATIVE, ST PRODUCTS ARE NOT RECOMMENDED, AUTHORIZED OR WARRANTED FOR USE IN MILITARY, AIR CRAFT, SPACE, LIFE SAVING, OR LIFE SUSTAINING APPLICATIONS, NOR IN PRODUCTS OR SYSTEMS WHERE FAILURE OR MALFUNCTION MAY RESULT IN PERSONAL INJURY, DEATH, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE. ST PRODUCTS WHICH ARE NOT SPECIFIED AS "AUTOMOTIVE GRADE" MAY ONLY BE USED IN AUTOMOTIVE APPLICATIONS AT USER’S OWN RISK.

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries.

Information in this document supersedes and replaces all information previously supplied.

STMicroelectronics group of companies

Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America

http://www.st.com