AN1135: Using Third Generation Non-Volatile Memory (NVM3) Data Storage

The NVM3 driver provides a means to write and read data objects (key/value pairs) stored in flash. Wear-leveling is applied to reduce erase and write cycles and maximize flash lifetime. The driver is resilient to power loss and reset events, ensuring that objects retrieved from the driver are always in a valid state. A single NVM3 instance can be shared among several wireless stacks and application code, making it well-suited for multiprotocol applications. This application note explains how NVM3 can be used as non-volatile data storage in Zigbee (EmberZNet), Open Thread, Z-Wave, Bluetooth, and Connect applications.

Version 3.x of the Gecko SDK Suite, used with Simplicity Studio 5, introduced a new component-based project architecture that replaced AppBuilder. Eventually all wireless protocols will move to the component-based architecture. This document addresses both this new approach to NVM3 configuration as well as the AppBuilder configuration still in use.

KEY POINTS

- Key/value pair data object storage in flash
- Wear-leveling to maximize flash lifetime
- Resilient to power and reset events
- Shared by Zigbee, Connect, OpenThread, Z-Wave, and Bluetooth stacks
- Compatible with PS Store and Token APIs through wrappers
- Data upgradable from Simulated EEPROM version 2 to NVM3

1. Introduction

The third generation Non-Volatile Memory (NVM3) data storage driver is for storing persistent data in flash.

NVM3 is designed to work on all Silicon Labs wireless stacks running on EFR32 as well as MCU applications running on EFM32.

Some of the main features of NVM3 are:

- Key/value pair data storage in flash
- Runtime object creation and deletion
- Persistence across power loss and reset events
- Wear-leveling to maximize flash lifetime
- Object sizes configurable up to 4096 bytes
- Configurable flash storage size (minimum 3 flash pages)
- Cache with configurable size for fast object access
- Data and counter object types
- Compatibility layers included for several Silicon Labs persistent storage APIs
- Single shared storage instance in multiprotocol applications
- Repack API to allow application to run clean-up page erases during periods with low CPU load

NVM3 is described in detail in the NVM3 Documentation on https://docs.silabs.com/. The rest of this document assumes you are familiar with that content.

While the NVM3 API can be used directly, NVM3 is also used as the underlying storage mechanism for several other persistent storage APIs provided by Silicon Labs:

- Token API used in EmberZnet and Connect applications
- Persistent Storage API used Silicon Labs Bluetooth applications

2. NVM3 Default Instance

Several NVM3 instances can be created on a device and live independently of each other, but to save memory it is usually desirable to use only one NVM3 instance as each instance adds some overhead. For applications based on Silicon Labs wireless stacks, a common default instance is used. This allows Dynamic Multiprotocol (DMP) applications that combine several wireless stacks to share the same NVM3 instance.

The number of flash pages used for the NVM3 default instance is configurable, but this setting must match if an application includes several stacks that all use the default instance.

IMPORTANT: When creating an application that includes an NVM3 instance for a device that already contains an NVM3 instance in flash, the number of flash pages configured for the NVM3 instance must match the number of flash pages for the NVM3 instance already found on the device. Therefore it is not possible to change the size of an NVM3 instance once it has been installed on a device, without first erasing the flash pages holding the NVM3 instance and the NVM3 objects stored there.

NVM3 has a cache to speed up access to NVM3 objects. The cache size must be set to a value greater than or equal to the number of objects found in NVM3. This includes the number of NVM3 objects created through the native NVM3 API and any objects created through higher level APIs such as the Token API. The cache must also be large enough to hold any deleted NVM3 objects. The `nvm3_countObjects()` and functions can be used to find the number of live and deleted objects in NVM3 at any given point. Silicon Labs recommends checking these functions after initialization of all NVM3 objects, both through the native NVM3 API and higher level APIs such as the token API, to figure out the correct size of the NVM3 default cache size.

2.1 NVM3 Default Instance Key Space

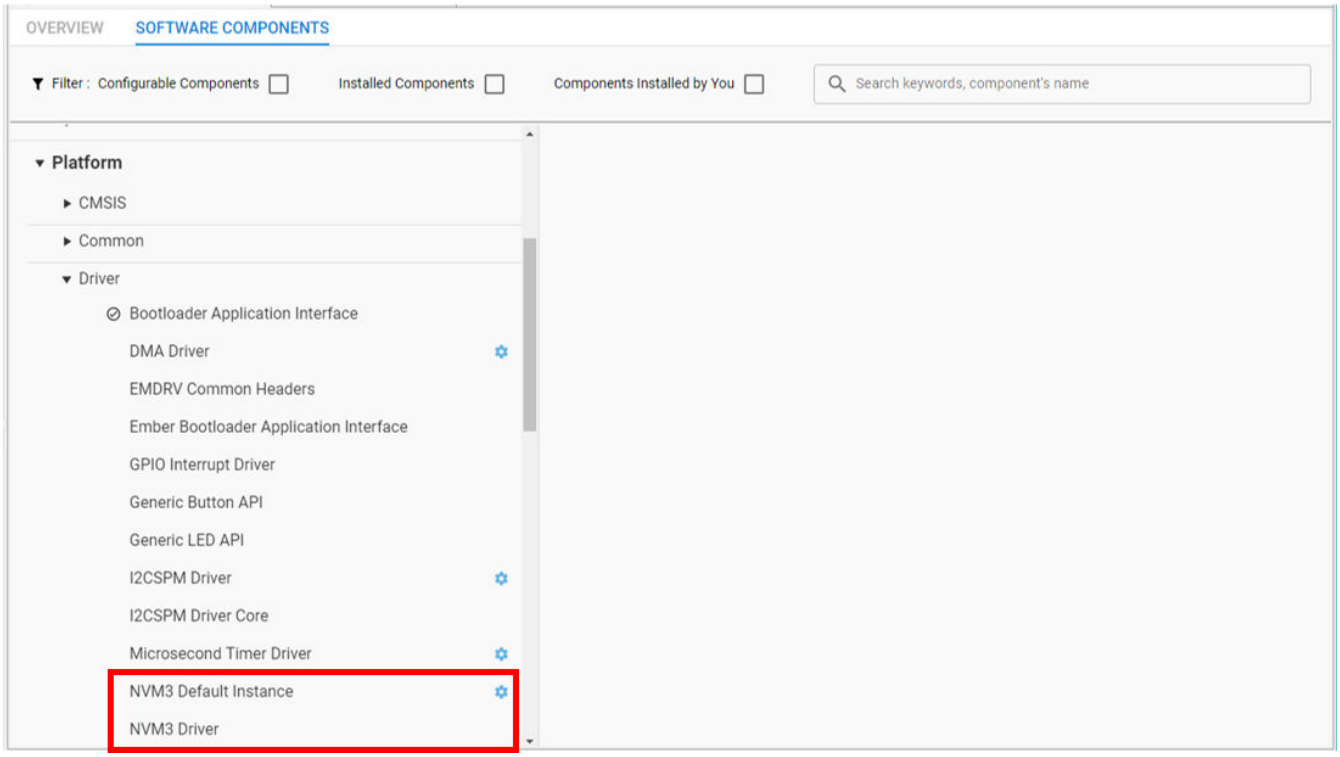
NVM3 uses a 20-bit key to identify each object. To avoid using the same key for more than one object, the NVM3 key space for the default NVM3 instance has been divided into several domains as outlined in the following table. For example, NVM3 objects defined in the EmberZNet stack should use NVM3 keys in the range 0x10000 to 0x1FFFF, while user application tokens should use keys below 0x10000. Note that any user defined NVM3 objects should be placed below 0x10000.

Table 2.1. NVM3 Default Instance Key Space

Domain	NVM3 Key
User	0x00000 - 0x0FFFF
EmberZNet stack	0x10000 - 0x1FFFF
OpenThread stack	0x20000 - 0x2FFFF
Connect stack	0x30000 - 0x3FFFF
Bluetooth stack	0x40000 - 0x4FFFF
Z-Wave stack	0x50000 - 0x5FFFF
Reserved	0x60000 - 0xFFFFF

3. NVM3 in the Simplicity Studio 5 Project Configurator

The Simplicity Studio 5 Project Configurator, used with Silicon Labs Bluetooth v3.x, Connect v3.x and OpenThread SDK applications, includes an **NVM3 Driver** component, found under Platform->Driver in the Software Components view. A separate component is also provided for the NVM3 default instance, which will initialize this instance.



The **NVM3 Default Instance** component provides the following configurations:

- **Size**: Number of bytes pages to use for NVM3 data storage. This must be set to match an integer number of flash pages, 3 pages at minimum.
- **Cache Size:** Number of objects to cache. To reduce access times, this number should be equal to or higher than the number of live and deleted objects stored in NVM3 at any time.
- **Max Object Size:** Size of largest allowed NVM3 object in bytes. Must be between 208 and 4096 bytes.
- **User Repack Headroom:** Headroom determining how many bytes below the forced repack limit the user repack limit is placed. The default value is 0, which means that the forced and the user repack limits are the same.

If the **NVM3 Default Instance** component is included in a project, the default instance will be initialized automatically during *sl_system_init()* if the **System Init** component found under Platform->Service->System is included in the project.

4. Using NVM3 with Silicon Labs Connect

This chapter applies to Connect in the Proprietary Flex SDK v3.x used with the Project Configurator in Simplicity Studio 5. Connect in Proprietary Flex SDK v2.x uses AppBuilder in Simplicity Studio 4. AppBuilder use is described in 7. Using NVM3 in AppBuilder-Based Applications.

For Silicon Labs Connect v3.x applications, a **Token Manager** component is available under Platform->Driver in the Simplicity Studio Project Configurator Software Component view. While the token manager provides the 9.1 Token API, an additional component must be selected for the token storage backend, either **Token Manager using Sim EEPROM 1, Token Manager using Sim EEPROM 2**, or **Token Manager using NVM3.** If NVM3 is chosen as the storage backend, **NVM3** and the **NVM3 Default Instance** components are included in the project.

5. Using NVM3 with Silicon Labs OpenThread Applications

This section applies to the Silicon Labs OpenThread SDK v1.x used with the Simplicity Project Configurator in Simplicity Studio 5. All OpenThread sample applications in the GSDK by default are configured to use NVM3 to store data in non-volatile memory. When doing so, these applications:

- Utilize the common default NVM3 instance.
- Include the NVM3 Driver and the NVM3 Default Instance components in the project.
- Use the native NVM3 API to access the NVM3 object

The NVM3 key space used by the OpenThread stack is 0x20000 to 0x2FFFF.

6. Using NVM3 with Silicon Labs Bluetooth Applications

Traditionally the Bluetooth stack uses its own proprietary solution to store data in non-volatile memory, called **Persistent Store (PS Store)**. PS Store stores both data handled by the stack (such as temporary Bluetooth address, bonding keys, and so on) and user data (such as the device state) that has to be preserved on resetting the device. To learn more about PS Store, read the related section of the Bluetooth API Reference Guide.

Some of the sample applications in the Bluetooth SDK are still configured to use PS Store, while others are already configured to use NVM3. This means that:

- On Series 1 devices sample apps are configured to use PS Store, except Bluetooth Mesh NCP sample projects, where NVM3 is used by default.
- On Series 2 devices all sample apps are configured to use NVM3.

While Series 2 devices support NVM3 only, on Series 1 devices both PS Store and NVM3 can be used. This section describes how to configure NVM3 in the Bluetooth SDK, and how to switch between PS Store and NVM3 if needed.

6.1 Configuring NVM3 in the Bluetooth SDK

With Simplicity Studio 5 and Bluetooth SDK v3.x

NVM3 can be configured with the Project Configurator as described in section 3. NVM3 in the Simplicity Studio 5 Project Configurator.

With Simplicity Studio 4 and Bluetooth SDK v2.x

The Project Configurator is not available to configure NVM3 parameters. Therefore, the parameters have to be defined manually. To overwrite the default parameters:

1. Open the project settings.
2. Find the defined symbols:
   a. If you use GCC as your compiler, go to C/C++ Build>Settings>GNU ARM C Compiler>Symbols>Defined Symbols
   b. If you use IAR as your compiler, go to C/C++ Build>Settings>IAR C/C++ compiler for ARM>Preprocessor>Defined Symbols
3. Add any of the following defines to overwrite the default parameters:

   NVM3_DEFAULT_NVM_SIZE

   NVM3_DEFAULT_CACHE_SIZE

   NVM3_DEFAULT_MAX_OBJECT_SIZE

   NVM3_DEFAULT_REPACK_HEADROOM

You can find the description of each parameter in section 3. NVM3 in the Simplicity Studio 5 Project Configurator Be careful when providing the size of NVM3, as it must be a multiple of the flash page size. Note: On Series 1 devices the flash page size is typically 2 kB, while on Series 2 devices the flash page size is typically 8 kB. Check the data sheet for your device. The NVM size must be at least 3 flash pages.

6.2 Switching from PS Store to NVM3

Beginning with Bluetooth SDK v2.13.0, both PS Store and NVM3 are supported as non-volatile memory solutions on Series 1 devices. Most sample applications are configured to use PS Store by default, but for some applications (where larger non-volatile memory is needed) NVM3 may be a better solution.

Note: PS Store and NVM3 are not compatible with each other, therefore upgrading an already existing application from PS Store to NVM3 will result in losing all data stored on the device. If you have an application running in the field, it may be wiser to stay with PS Store. If you still want to upgrade, see *AN1086: Using the Gecko Bootloader with the Silicon Labs Bluetooth® Application* for details.

Note: PS Store uses only 2 flash pages (=4 kB on an EFR32BG1/12/13 device). Therefore, changing to NVM3 will affect the available space in flash. You must be particularly careful when you upgrade the firmware not to overwrite the NVM3 area with the application.

With Simplicity Studio 5 and Bluetooth SDK v3.x

To change your project configuration from PS Store to NVM3, simply install the **NVM3 Default Instance** component in the Project Configurator as discussed in section 3. NVM3 in the Simplicity Studio 5 Project Configurator. This will automatically uninstall the (otherwise hidden) PS Store component.

With Simplicity Studio 4 and Bluetooth SDK v2.x

To change your project configuration from PS Store to NVM3, use the following procedure.

1. Copy the following folder with all of its content:

   `C:\SiliconLabs\SimplicityStudio\v4\developer\sdks\gecko_sdk_suite\<version>\platform\emdrv\nvm3`

   under the `/platform/emdrv` folder of your project.
2. Remove the following files from the project:
   - `/platform/emdrv/nvm3/src/nvm3_hal_extflash.c`
   - `/platform/emdrv/nvm3/src/nvm3_default_extflash.c` (NVM3 use with external flash is deprecated)
3. If you use Apploader in your project, also copy the NVM3 version of Apploader from

   `C:\SiliconLabs\SimplicityStudio\v4\developer\sdks\gecko_sdk_suite\<version>\protocol\bluetooth\lib\<device>\<compiler>\binapploader_nvm3.o`

   into the `/protocol/bluetooth/lib/<device>/<compiler>` folder of your project.
4. **If you use GCC as a compiler:**
   a. Go to `Project > Properties > C/C++ Build > Settings > GNU ARM C Compiler > Includes`.
   b. Add `${workspace_loc:/${ProjName}/platform/emdrv/nvm3/inc}` to the include directory.
   c. Go to `Project > Properties > C/C++ Build > Settings > GNU ARM C Linker > Miscellaneous`.
   d. Remove `${workspace_loc:/${ProjName}/protocol/bluetooth/lib/<device>/<compiler>/libpsstore.a}`.
   e. add `${workspace_loc:/${ProjName}/platform/emdrv/nvm3/lib/libnvm3_CM4_gcc.a}`

   If you use IAR as a compiler:
   a. Go to `Project > Properties > C/C++ Build > Settings > IAR C/C++ Compiler for ARM > Preprocessor`.
   b. Add `${workspace_loc:/${ProjName}/platform/emdrv/nvm3/inc}` to the include directory.
   c. Go to `Project > Properties > C/C++ Build > Settings > IAR Linker for ARM > Library`.
   d. Remove `${workspace_loc:/${ProjName}/protocol/bluetooth/lib/<device>/<compiler>/libpsstore.a}`.
   e. Add `${workspace_loc:/${ProjName}/platform/emdrv/nvm3/lib/libnvm3_CM4_iar.a}`.
5. If you use Apploader, also modify

   `${workspace_loc:/${ProjName}/protocol/bluetooth/lib/<device>/<compiler>/binapploader.o}`

   to `${workspace_loc:/${ProjName}/protocol/bluetooth/lib/<device>/<compiler>/binapploader_nvm3.o}`.
6. Configure NVM3 as described in section 6.1 Configuring NVM3 in the Bluetooth SDK.

6.3 Switching from NVM3 to PS Store

It may happen that, for a reason such as backward compatibility, you have to change the configuration from NVM3 to PS Store.

With Simplicity Studio 5 and Bluetooth SDK v3.x

To change your project configuration from NVM3 to PS Store, simply uninstall the NVM3 Default Instance component. This will automatically install the (otherwise hidden) PS Store component. Note, that this can only be done on series 1 devices, as series 2 devices do not support PS Store

With Simplicity Studio 4 and Bluetooth SDK v2.x

To change your project configuration from NVM3 to PS Store, use the following procedure:

1. Remove the `/platform/emdrv/nvm3` folder from your project.
2. Copy the PS Store library from

   `C:\SiliconLabs\SimplicityStudio\v4\developer\sdks\gecko_sdk_suite\<version>\protocol\bluetooth\lib\<device>\<compiler>\libpsstore.a`

   into the `/protocol/bluetooth/lib/<device>/<compiler>` folder of your project.
3. If you use Apploader in your project, also copy the PS Store version of Apploader from

   `C:\SiliconLabs\SimplicityStudio\v4\developer\sdks\gecko_sdk_suite\<version>\protocol\bluetooth\lib\<device>\<compiler>\binapploader.o`

   into the `/protocol/bluetooth/lib/<device>/<compiler>` folder of your project.

   Note: PS Store is not supported on Series 2 devices (EFR32xG2x), therefore there is only an NVM3 version of the Apploader for these devices.

4. **If you use GCC as your compiler**
   a. Go to `Project > Properties > C/C++ Build > Settings > GNU ARM C Compiler > Includes`.
   b. Remove `${workspace_loc:/${ProjName}/platform/emdrv/nvm3/inc}` from the include directory.
   c. Go to `Project > Properties > C/C++ Build > Settings > GNU ARM C Linker > Miscellaneous`.
   d. Add `${workspace_loc:/${ProjName}/protocol/bluetooth/lib/<device>/<compiler>/libpsstore.a}`.
   e. Remove `${workspace_loc:/${ProjName}/platform/emdrv/nvm3/lib/libnvm3_CM4_gcc.a}`.

   If you use IAR as your compiler
   a. Go to `Project > Properties > C/C++ Build > Settings > IAR C/C++ Compiler for ARM > Preprocessor`.
   b. Remove `${workspace_loc:/${ProjName}/platform/emdrv/nvm3/inc}` from the include directories.
   c. Go to `Project > Properties > C/C++ Build > Settings > IAR Linker for ARM > Library`.
   d. Add `${workspace_loc:/${ProjName}/protocol/bluetooth/lib/<device>/<compiler>/libpsstore.a}`.
   e. Remove `${workspace_loc:/${ProjName}/platform/emdrv/nvm3/lib/libnvm3_CM4_iar.a}`.
5. If you use Apploader, also change

   `${workspace_loc:/${ProjName}/protocol/bluetooth/lib/<device>/<compiler>/binapploader_nvm3.o}`

   to `${workspace_loc:/${ProjName}/protocol/bluetooth/lib/<device>/<compiler>/binapploader.o}`.

7. Using NVM3 in AppBuilder-Based Applications

This chapter explains how NVM3 can be used for non-volatile storage in Appbuilder-based applications like EmberZNet 6.7.n and 6.8.n, EmberZNet-based DMP applications, and Connect applications in Gecko SDK Suite 2.7 and earlier.

7.1 NVM3 Library Plugin

To use NVM3 with an AppBuilder-based example application, the **NVM3 Library** plugin should be included in the project. All PS Store and SimEE plugins should be deselected.

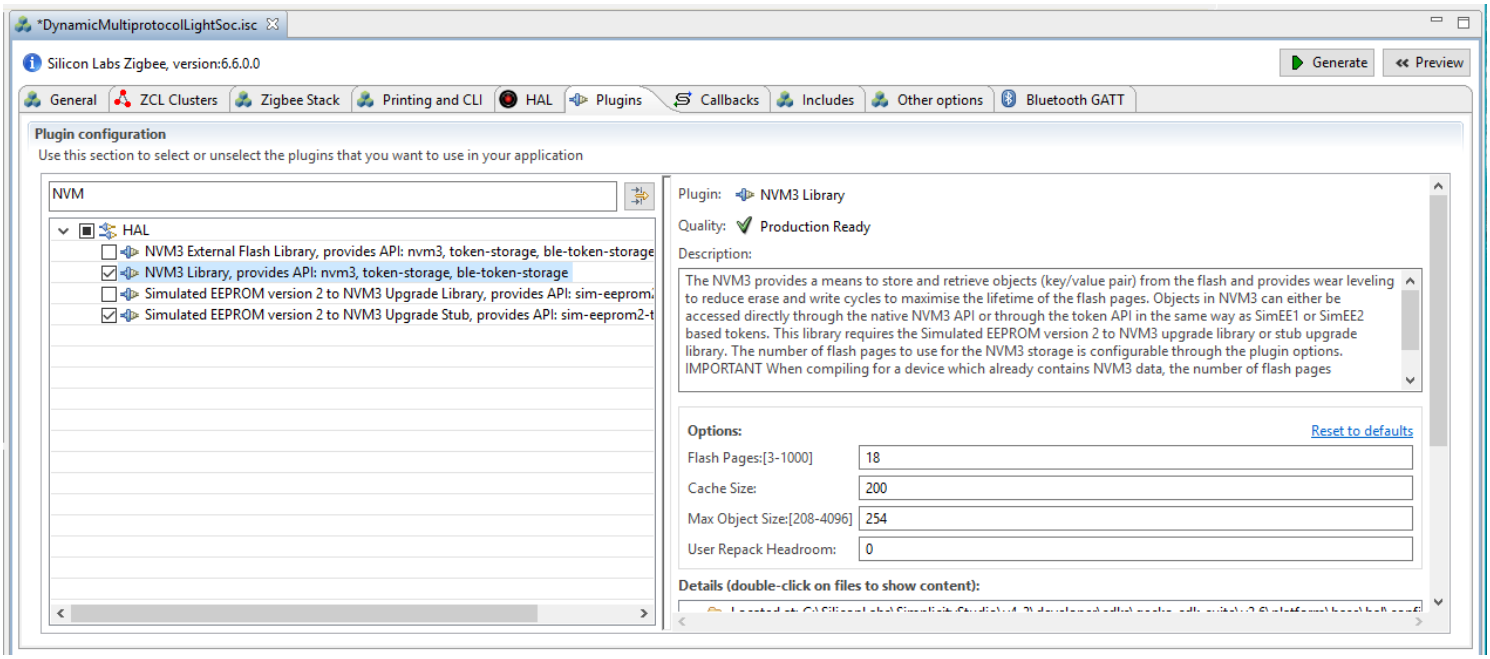


Figure 7.1. NVM3 Library Plugin in AppBuilder

The **NVM3 Library** plugin offers four plugin options:

- **Flash Pages**: Number of flash pages to use for NVM3 data storage. Must be 3 or higher. Default is 18 (36KB) for EFR32 Series-1 devices and 4 (32KB) for EFR32 Series-2 devices
- **Cache Size:** Number of objects to cache. To reduce access times, this number should be equal to or higher than the number of objects, including tokens and deleted objects, stored in NVM3 at any time.
- **Max Object Size:** Size of largest allowed NVM3 object in bytes. Must be between 208 and 4096 bytes. Note that the token API can only be used to access objects of 254 bytes or smaller. When accessing larger objects, the native NVM3 API must be used.
- **User Repack Headroom:** Headroom determining how many bytes below the forced repack limit the user repack limit is placed. The default value is 0, which means that the forced and the user repack limits are the same.

When the NVM3 Library plugin is used the **Simulated EEPROM version 2 to NVM3 Upgrade Library** or **Simulated EEPROM version 2 to NVM3 Upgrade Stub Library** must be included, as described in section 7.2 SimEEv2 to NVM3 Upgrade Plugin.

7.2 SimEEv2 to NVM3 Upgrade Plugin

An AppBuilder plugin (**Simulated EEPROM version 2 to NVM3 Upgrade Library**) is provided for EmberZNet applications that upgrade tokens stored in SimEEv2 to NVM3. For tokens to be successfully upgraded to NVM3, CREATOR_* and NVM3KEY_* defines must be added for all tokens as described in section 9.1 Token API. The upgrade plugin will replace the SimEEv2 storage in-place with an NVM3 storage instance. The plugin does this by compacting the SimEEv2 storage down to 12 kB, and then creates an NVM3 instance in the remaining 24 kB of the original 36 kB SimEEv2 storage space. After the token data has been copied over from SimEEv2 to NVM3, the SimEEv2 storage is erased and the NVM3 instance is resized to use the entire 36 kB storage space. Apart from the code space needed for the upgrade library code, the upgrade does not require any additional flash space to the 36 kB storage area. The upgrade plugin requires that the existing SimEEv2 storage space and new NVM3 storage space are located at the same address and have the same size.

The **Simulated EEPROM version 2 to NVM3 Upgrade Library** plugin should be included to enable the upgrade as shown in the figure below. If no SimEEv2 token data is found, the upgrade plugin will look for NVM3 data, and if neither is found it will create a new NVM3 instance with tokens set to their default values. For applications that do not need to upgrade any SimEEv2 tokens, the **Simulated EEPROM version 2 to NVM3 Upgrade Stub** plugin should be included instead.

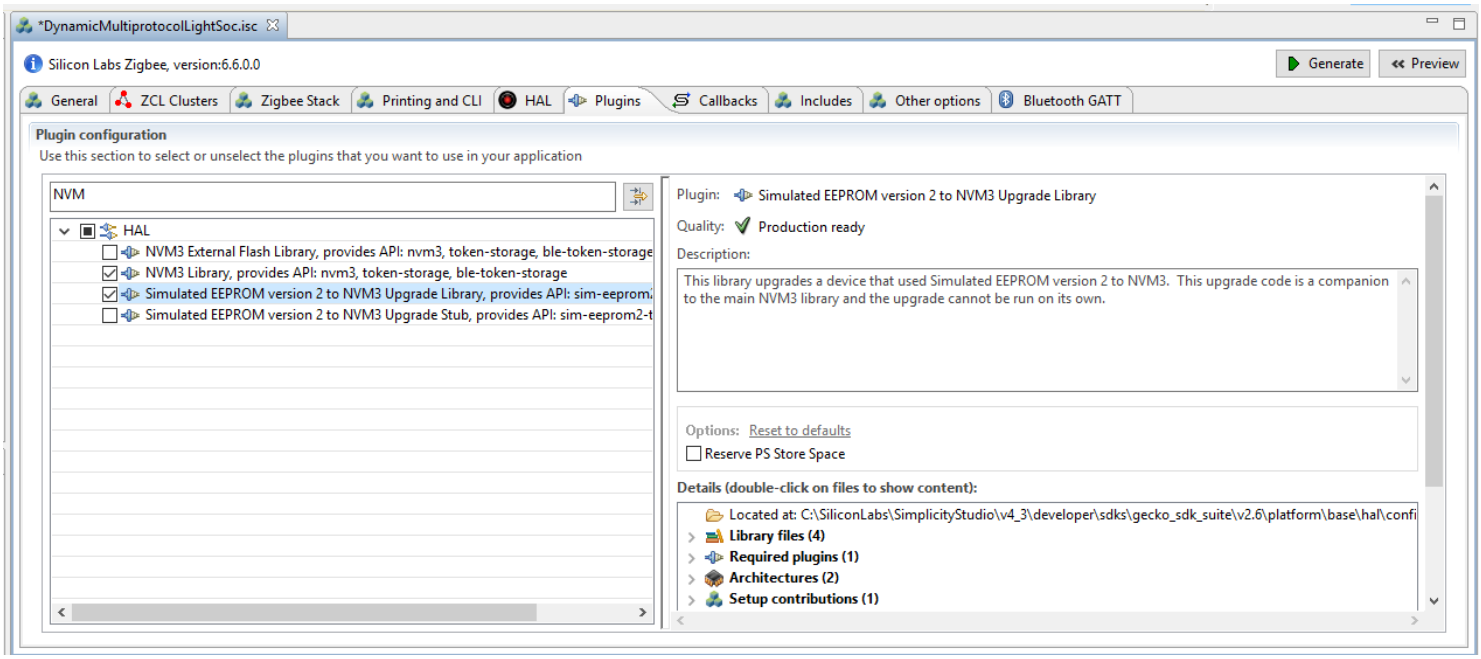


Figure 7.2. SimEEv2 to NVM3 Upgrade Library and Stub Plugins in AppBuilder

8. Using NVM3 with Z-Wave Applications

For details on using NVM3 with Z-Wave applications, see INS14250-10: Z-Wave Plus V2 Application Framework SDK7, section 6.4.

9. NVM3 API Options

This chapter describes the three different APIs available to access NVM3 objects.

- Token API
- Persistent Store API
- Native NVM3 API

9.1 Token API

The token API is used to access data stored in SimEEv1 and SimEEv2 with the EmberZNet and Connect stacks, as well as multiprotocol applications. Information on how to define and access tokens can be found in *AN1154: Using Tokens for Non-Volatile Data Storage*, and users should read this document before using the token API. When selecting the NVM3 Library plugin instead of one the SimEE plugins, the NVM3 default instance is used to store the token data instead of SimEE. The same token API can still be used to access tokens stored in NVM3, but the token definition needs some modifications to work with NVM3, as described below.

When defining a token to be used with SimEE, a creator code must be defined as an identifier for the token. Similarly, when defining a token to be used with NVM3, an NVM3 key must be defined for the token. A token definition that is compatible with both NVM3 and SimEE would include both an NVM3 key and a creator code and look like this:

```
#define CREATOR_name 16bit_value
#define NVM3KEY_name 20bit_value
#ifdef DEFINETYPES
  typedef data_type type
#endif
#ifdef DEFINETOKENS
  DEFINE_*_TOKEN(name, type, ... ,defaults)
#endif
```

Select a 20-bit NVM3 key for the token, according to the domains in Table 2.1 NVM3 Default Instance Key Space on page 3. Each token must have a unique NVM3 key, except for indexed tokens, where more NVM3 keys must be reserved as outlined in section 9.1.2 Special Considerations for Indexed Tokens.

9.1.1 Deleting Tokens

As tokens are created at compile time, they cannot be created or deleted at run time. NVM3 objects are, however, created and deleted at run time, and the token initialization function creates NVM3 objects for each defined token if they do not already exist. The token initialization generally does not delete NVM3 objects found that do not have a corresponding token associated with them. Therefore, if a token is no longer included in an application, the application should manually delete the associated NVM3 object by using the NVM3 Native API described in section 9.3 Native NVM3 API. For indexed tokens, however, the token initialization checks if indexed tokens have more or less indexes than the number of NVM3 objects found in the indexed token's NVK3KEY range. If there are fewer indexes, the token initialization deletes the extra NVM3 objects. If the number of indexes has been increased, new NVM3 objects will be created to hold these indexes.

When NVM3 objects are deleted, the actual object data remains in NVM3 but is marked as deleted. The deleted object data remains in NVM3 and consumes cache space until NVM3 repacks have erased the page(s) holding all versions of these objects.

9.1.2 Special Considerations for Indexed Tokens

NVM3 does not have native support for indexed tokens. Therefore an extra requirement is imposed on the NVM3 key selection for indexed tokens. With NVM3, indexed tokens are implemented by storing each index in a separate object, starting with index 0 stored at the defined NVM3KEY_name key value and the last index (127) stored with key NVM3KEY_name + 127. Because of this implementation, 128 NVM3 keys must be reserved for each indexed token. The user still only defines one NVM3KEY_name key value, but no other tokens should be defined with key values in the 127 values following this defined key. Even if the token is defined with fewer than 128 indices, all 128 indices should be reserved as the token might be expanded with more indices later on.

The example below shows two indexed tokens defined in the user key domain:

```
// This key is used for an indexed token and the subsequent 0x7F keys are also reserved
#define NVM3KEY_MY_INDEXED_TOKEN_A 0x00000
// This key is used for an indexed token and the subsequent 0x7F keys are also reserved
#define NVM3KEY_MY_INDEXED_TOKEN_B 0x00080
```

Table 9.1. Indexed Token NVM3 Key Selection Example

NVM3KEY	NVM3 Objects Contents
0x00000	Reserved for TOKEN_MY_INDEXED_TOKEN_A index 0
0x00001	Reserved for TOKEN_MY_INDEXED_TOKEN_A index 1
0x00002	Reserved for TOKEN_MY_INDEXED_TOKEN_A index 2
…	
0x0007F	Reserved for TOKEN_MY_INDEXED_TOKEN_A index 127
0x00080	Reserved for TOKEN_MY_INDEXED_TOKEN_B index 0
0x00081	Reserved for TOKEN_MY_INDEXED_TOKEN_B index 1
…	
0x000FF	Reserved for TOKEN_MY_INDEXED_TOKEN_B index 127

9.2 Bluetooth NVM API

The Bluetooth NVM API that was originally designed for PS Store can be used in the same way when using NVM3 as when using PS Store. The Bluetooth stack will automatically translate its NVM API calls to PS Store API calls or to NVM3 API calls in the background, depending on what components the project uses. The same applies for Zigbee + Bluetooth DMP projects, where NVM3 is applied as the storage mechanism, but the Bluetooth NVM API can still be used. The Bluetooth API is documented in the Bluetooth API Reference Manual. In Bluetooth SDK v2.x it can be found under the *Flash* class (commands starting with gecko_cmd_flash_), while in Bluetooth SDK v3.x it can be found under the *NVM* class (commands starting with sl_bt_nvm).

16-bit keys are used with the Bluetooth NVM API, which are then mapped to a 20-bit NVM3 key when NVM3 is used as storage mechanism. The four most significant bits are set to 0x4 to place these objects in the Bluetooth domain of the NVM3 default instance key space. As the Bluetooth NVM API is fixed to use only the Bluetooth domain, any objects to be placed in other domains, for example the User domain, must be created and accessed using the native NVM3 API.

If you want to use the Bluetooth NVM API and the native NVM3 API in the same app, then:

1. Call `gecko_init(pconfig)` to initialize the Bluetooth stack and open its own NVM3 instance. This is done in all Bluetooth sample apps.
2. Open NVM3 by calling the `nvm3_open()` function with the default (!) parameters to open your NVM3 instance:

```
nvm3_open(nvm3_defaultHandle, nvm3_defaultInit);
```

User data can now be saved to:

- PS key range 0x4000 - 0x407F. All other PS keys (0x0000-0xFFFF) are reserved for the stack (for example for storing bonding data).
- NVM3 key range 0x00000-0x0FFFF (NVM3 user data area), and 0x44000-0x4407F (PS Store user data area).

For example, the following API calls will have the same effect (writing to the same area)::

- `nvm3_writeData(nvm3_defaultHandle,0x44000,(void*)data,len);`
- `gecko_cmd_flash_ps_save(0x4000,len,data); //in Bluetooth SDK v2.x`
- `sl_bt_nvm_save(0x4000,len,data); //in Bluetooth SDK v3.x`

Similarly, you can read the same data with the following API calls:

- `nvm3_readData(nvm3_defaultHandle,0x44000,(void*)read_buffer,maxlen);`
- `gecko_cmd_flash_ps_load(0x4000); //in Bluetooth SDK v2.x`
- `sl_bt_nvm_load(0x4000,maxlen,&read_len,(uint8_t*)read_buffer); //in Bluetooth SDK v3.x`

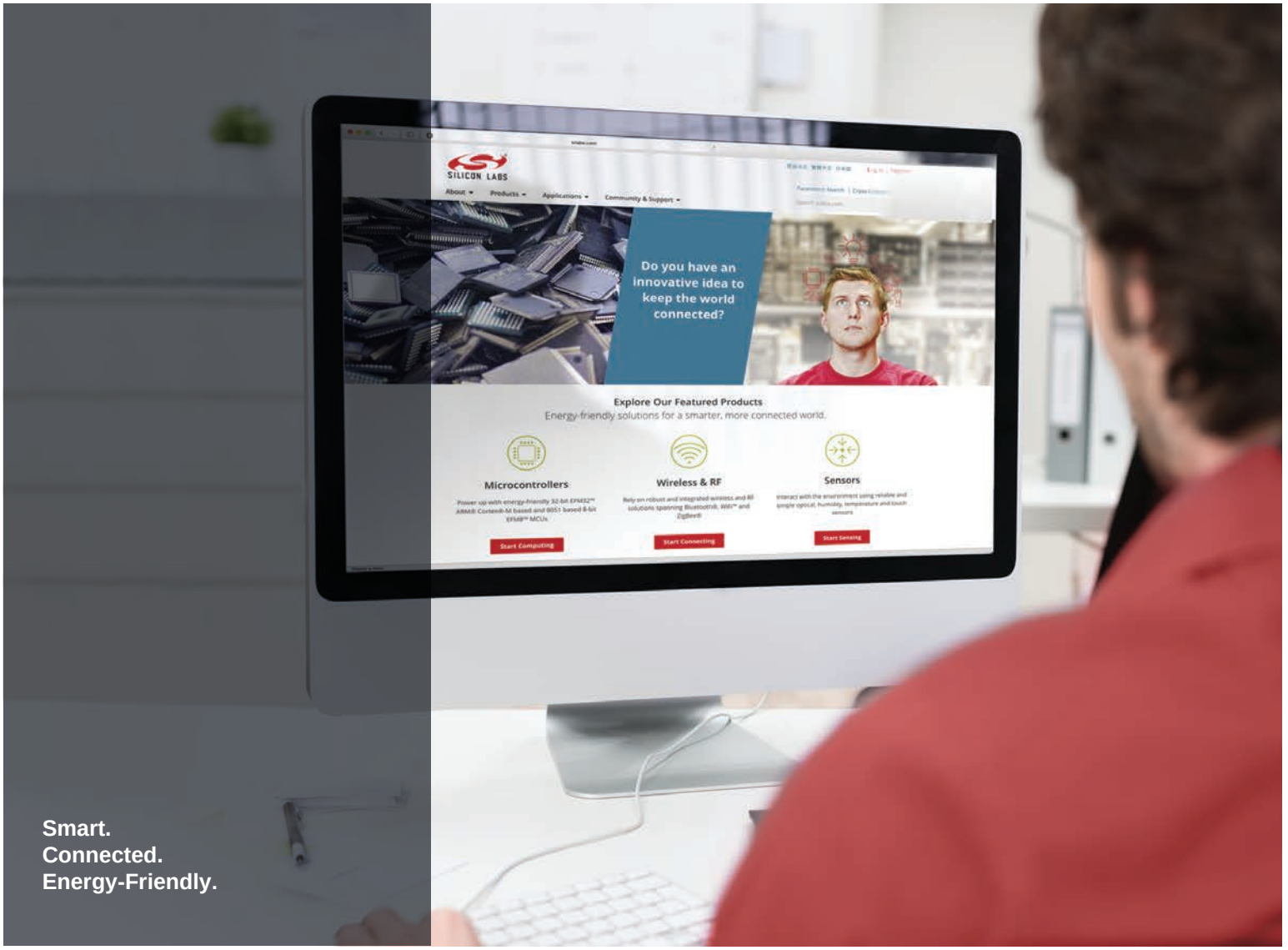
9.3 Native NVM3 API

For code accessing NVM3 objects that does not need to be compatible with the token or PS Store APIs, using the native NVM3 API to access NVM3 data is recommended to reduce code size and allow using the full feature set of NVM3. Any PS Store object or token can also be accessed through the native NVM3 API using the same NVM3 key. Complete documentation of this API is found in the EMDRV ->NVM3 section of Gecko HAL & Driver API Reference Guide.

10. Simplicity Commander and NVM3

Simplicity Commander is a single, all-purpose tool to be used in a production environment. It is invoked using a simple CLI (Command Line Interface) that is also scriptable. Simplicity Commander supports reading out the NVM3 data area from a device and parsing the NVM3 data to extract stored values. This can be useful in a debugging scenario where you may need to find out the stored state of an application that has been running for some time.

More information on how to use the Simplicity Commander with NVM3 can be found in *UG162: Simplicity Commander Reference Guide*.



Products
www.silabs.com/products

Quality
www.silabs.com/quality

Support and Community
community.silabs.com

SILICON LABS

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

http://www.silabs.com