AN14478

Extending i.MX 9 System Manager Functionality with Board Controls

Rev. 1.0 — 25 November 2024

Application note

Document information

Information	Content
Keywords	AN14478, system manager, i.MX 9, i.MX 94, i.MX 95
Abstract	This application note covers system manager and client implementations provided in the NXP 6.6.36_2.1.0 BSP release.

1 Introduction

Some processors in the i.MX 9 family (for example, i.MX 95 and i.MX 94) feature an M33 core dedicated to running a system manager (SM).

SM is a firmware provided by NXP, intended to manage low-level resources for the other processors in the SoC. It provides a standard Arm SCMI interface to its clients, extended with some NXP proprietary extensions.

SM is delivered in source form, allowing customers the freedom to modify and recompile its code to suit their needs. However, NXP advises that any changes made to extend SM functionality must be limited to the customer board layer. It avoids unnecessary and possibly dangerous modifications to SM code while keeping modifications and extensions well segregated from the SM core code.

The SM provides a mechanism to overload and extend nearly every provided SM entry point and redirect them to the customer board layer. Though most of SM-exposed RPC can be overloaded this way, the best way to extend the SM function is to add dedicated board controls.

This application note covers SM and client implementations provided in the NXP 6.6.36_2.1.0 BSP release.

2 System manager controls

To expose device or board settings that the standard SCMI interfaces cannot manipulate, SM provides access to controls. Accessing a control can trigger registers or variables to be set or retrieved, or more complex actions to occur.

A control is identified by its 32-bits ID. SM clients can manipulate the controls using the NXP MISC extension to the SCMI protocol.

SM provides a mechanism for customers to add their own controls by only modifying their board layer. It is the preferred way to extend SM functionalities.

3 RPC for managing controls

Table 1 lists the SCMI API managing controls, their intended usage, and the required API permission needed for a client to call them.

Table 1. SCMI API managing controls

SCMI RPC	Description	Intended usage	API permission
SCMI_MiscControlSet()	Set a control array to a given set of values.	Write to SoC or external peripheral registers.	SET
SCMI_MiscControlGet()	Retrieve a set of values associated with a control.	Read from SoC or external peripheral registers.	GET
SCMI_MiscControlExtSet()	Set an array of values associated with a control. An address is passed to the SM.	Write a series of values to a SoC or external SM peripheral register at a given address.	SET
SCMI_MiscControlExtGet()	Retrieve an array of values associated with a control. An address is passed to the SM.	Read a series of values from a SoC or external SM peripheral register at a given address.	GET
SCMI_MiscControlAction()	Perform an arbitrary action associated with a control. Uses an action ID and a list of	This RPC implements any complex action.	EXCLUSIVE

Table 1. SCMI API managing controls*...continued*

SCMI RPC	Description	Intended usage	API permission
	parameters. Returns a list of values.		
SCMI_MiscControlNotify()	Register an agent to be notified of an event associated with a control.	Receive notification from other logical machine on an arbitrary event.	NOTIFY
SCMI_MiscControlEvent()	Broadcast an event associated with a control. Agents registered receive a notification.	Send a notification on an arbitrary event.	None

4 Device controls vs board controls

For each i.MX supported SoC, SM provides device controls. Device controls are used to control aspects of the SoC that are not covered by the standard SCMI protocols, for example, static settings of global device configuration registers. Device controls are device-specific and provided by SM code. Their IDs can be found in the `devices/<dev>/sm/dev_sm_control.h`. The implementation of the RPC associated with the device controls can be found in `devices/<dev>/sm/dev_sm_control.c`.

For each board supported, SM provides a set of board controls. Board controls are used to expose SM-managed board-related components. For example, the SM can use a board control to expose an external GPIO expander to the SM clients. Board controls implementation can be found in `board/<board>/sm/brd_sm_control.[h|c]`.

When board controls are present, they are exposed to clients by redirecting the corresponding device control RPC to the board layer. For detailed steps, refer to the *SM porting guide*: GitHub - nxp-imx/imx-sm: System Manager firmware for i.MX processors and the following section.

4.1 Device layer redirection

SM is structured so that SCMI RPC calls used to access most of the device resources exposed to clients point to functions in the device layer. This layer is device-specific, provided by NXP, and must not be modified.

If customers must implement a specific extension to an existing SCMI protocol, the SM provides a way to redirect calls from the device layer to the board layer. This way, customer-specific implementation can reside in the board layer only.

Device layer redirection is not limited to the SM controls. However, applying this mechanism to SM control is the preferred way to extend the SM functionalities, while keeping the changes in the code contained to the board layer.

4.2 Device layer redirection workflow

An RPC call ends up calling a function in the Logical Machine Management (LMM) layer. This function either redirects to a device implementation inside the device layer (default implementation), or to a custom implementation inside the board layer.

Figure 1 shows the call flow when the LMM function is not redirected to the board layer.

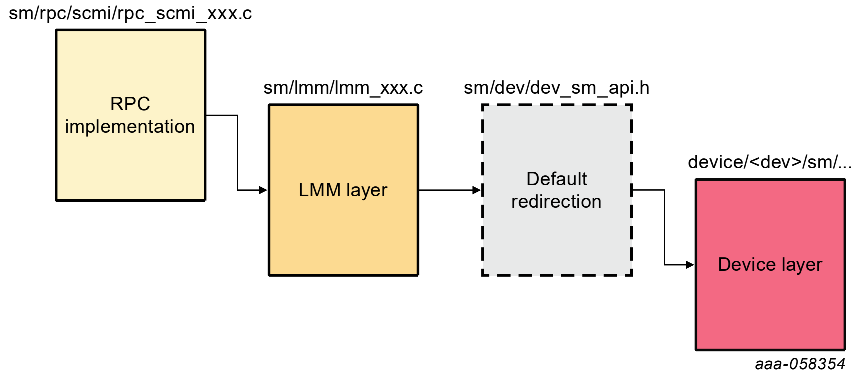

Figure 1. LMM call not redirected

Figure 2 shows the call flow when the LMM function is redirected to the board layer. If the parameters of the RPC calls are board-specific, such as accessing a board sensor or board control, the board layer implementation must perform specific actions. Otherwise, it must fall back to the device layer implementation.

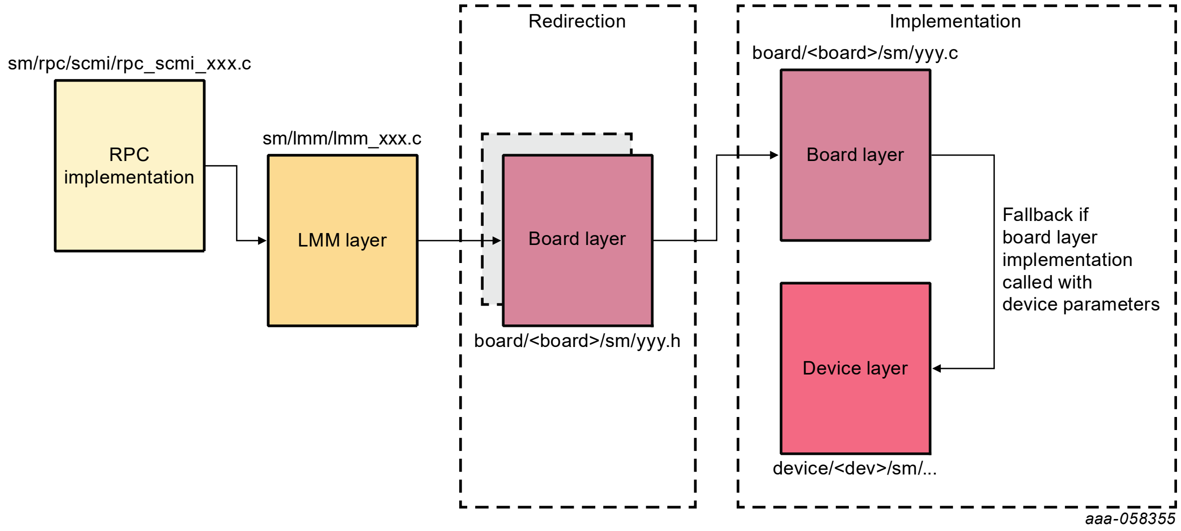

Figure 2. LMM call redirected to board layer

4.3 Example

Figure 3 shows the example for `SCMI_MiscControlSet()` in i.MX 95 SoC.

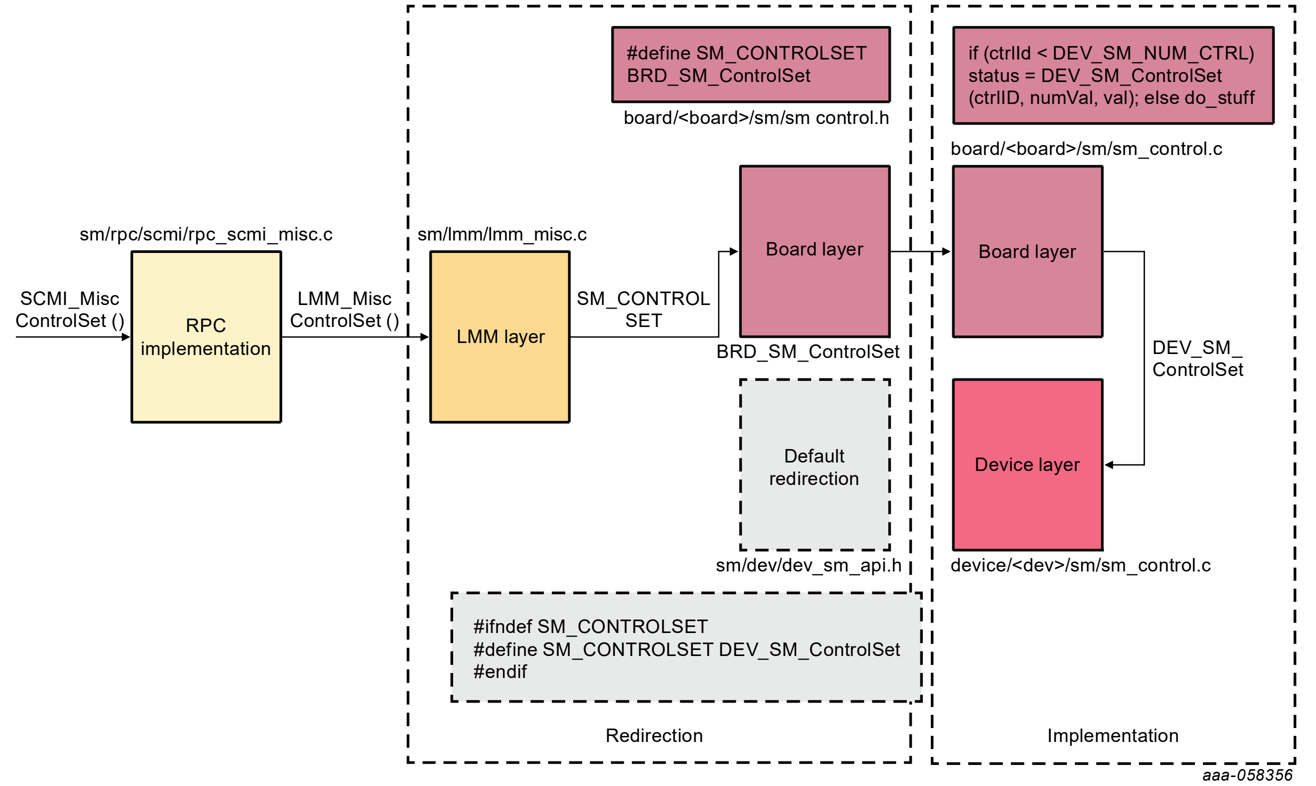

Figure 3. SCMI_MiscControlSet()

- A call to `SCMI_MiscControlSet()` from the client executes the `LMM_MiscControlSet()` function defined in `sm/lmm/lmm_misc.c`.
- `LMM_MiscControlSet()` calls `SM_CONTROLSET`.
- `SM_CONTROLSET` is defined in `sm/dev/dev_sm_api.h` as follows:

```
#ifndef SM_CONTROLSET
/*! Redirector (device/board) to write a control */
#define SM_CONTROLSET              DEV_SM_ControlSet
#endif
```

- i.MX 95 EVK board layer redefines `SM_CONTROLSET` to point to a board layer implementation, in `boards/mcimx95evk/sm/brd_sm_control.h`:

```
#define SM_CONTROLSET          BRD_SM_ControlSet
```

- The board control IDs and their numbers are also defined in `boards/mcimx95evk/sm/brd_sm_control.h`:

```
/*! Number of board controls */
#define BRD_SM_NUM_CTRL  7UL
/*! Total number of controls */
#define SM_NUM_CTRL  (DEV_SM_NUM_CTRL + BRD_SM_NUM_CTRL)
/*!
@name BRD_SM control domain indexes
*/
/** @{ */
#define BRD_SM_CTRL_SD3_WAKE    (DEV_SM_NUM_CTRL + 0U)
…
#define BRD_SM_CTRL_PCA2131     (DEV_SM_NUM_CTRL + 6U)
```

- Therefore, RPC ends up calling `BRD_SM_ControlSet`. The corresponding implementation is provided in `boards/mcimx95evk/sm/brd_sm_control.c`. If the control ID is a device control ID, it falls back to the device implementation. If the control ID is any other control ID, it provides a board implementation.

```
int32_t BRD_SM_ControlSet(uint32_t ctrlId, uint32_t numVal, const uint32_t
 *val)
{
    int32_t status = SM_ERR_SUCCESS;
    /* Check to see if ctrlId is within bounds*/
    if (ctrlId < SM_NUM_CTRL)
    {
        /* Check if device or board */
        if (ctrlId < DEV_SM_NUM_CTRL)
        {
            status = DEV_SM_ControlSet(ctrlId, numVal, val);
        }
        else
        {
            /* Check the ctrlId and do what’s needed */
        }
    }
    else
    {
        status = SM_ERR_NOT_FOUND;
    }
    /* Return status */
    return status;
}
```

- For a client to act on the new board control, permissions must be granted to access it. Those permissions are granted in the board configuration file `configs/mx95evk.cfg`.
  Note: *The permissions are granted control by control.*

```
#===============================================================#
# M7 EENV                                                                  #
#============================== =================================#
LM1    name="M7", rpc=scmi, boot=2, skip=1, did=4, safe=seenv
…
# API
…
BRD_SM_CTRL_PCA2131       ALL
…
```

5 Board control implementation

This section shows the procedure to implement a new board control. It gives a step-by-step outline of the implementation, from the control ID definition to the necessary modifications in the SM board layer.

5.1 Control ID

To prevent conflicts between device control ID and board control ID, the SM RPC layer uses bit 15 of the control ID to discriminate between board control and device control.

- If bit 15 of the control ID is not set, then the control ID is passed to the (possibly redirected) device layer without modification.
- If bit 15 of the control ID is set, then the control ID `(input_control_id & ~0x8000) + DEV_SM_NUM_CTRL` is passed to the (possibly redirected) device layer.

The consequences of using bit 15 as a marker for board controls are as follows:

- Even if the CtrlID field is a 32 bits unsigned int, use only control IDs lower than 0xFFFF. This means 32768 device control IDs are from 0x0000 to 0x7FFF and 32768 board control IDs are from 0x8000 to 0xFFFF.
- In the SM code, the board control ID must be defined and used as follows:

```
#define MY_ID (DEV_SM_NUM_CTRL + MY_INDEX)
```

- In the client code, the board control ID that must be passed to the SCMI RPC is `MY_INDEX | MISC_CTRL_FLAG_BRD`. NXP client software defines the `MISC_CTRL_FLAG_BRD` as `0x8000`.

5.2 Implementation outline

A custom board control can be implemented by redirecting the device implementation of the MiscControlXXX() functions.

To implement a custom board control, perform the following steps:

- Redirect the requests `SCMI_MiscControlxxx()` API from the device layer to the board layer. Redefine the `SM_CONTROLXXX` symbol in the board layer code.
- Add a new control ID in the board control layer, define `BRD_SM_MY_ID (DEV_SM_NUM_CTRL + BRD_SM_INDEX)`, and increment `BRD_SM_NUM_CTRL` accordingly.
- Grant clients access to the new control. Add the required API permissions to `BDR_SM_MY_ID` in the impacted agent.
- Provide an implementation of the redirected function or extend the existing implementation. To detect your new control is targeted, check for `BRD_SM_MY_ID`.

5.2.1 Client-side implementation

Call the necessary SCMI access function passing `BRD_SM_INDEX | MISC_CTRL_FLAG_BRD` as CtrlID.

An example of a client-side call to MiscControlAction for a control action passing action=0, no input arguments is as follows:

```
err = SCMI_MiscControlAction(SCMI_A2P, BRD_SM_INDEX | SCMI_MISC_CTRL_FLAG_BRD,
 0, 0, NULL, &numRtn,  &rtn);
```

5.3 Caveats

Board controls and device layer redirections can give any client the possibility to run arbitrary code in an SM context. Before writing such code, one must be aware of the following caveats:

1. Limit board layer code processing time. SM is mono-threaded. If an SCMI call is being served, other concurrent SCMI calls have to wait until it returns. This process can stall other clients for an unacceptable amount of time. A few hundreds microseconds or less is a good order of magnitude for a board layer call in a production context. Accessing a device register or performing a single fast I^2C read is probably okay. Dumping many I^2C registers from a slow device is not. To assess their impact on SM responsiveness, ensure to profile your board control implementation.
2. Bugs in board layer code can hang or crash the SM, resulting in a full platform reset. Board control manipulation code must be carefully audited. Inputs to board control functions (especially ControlAction) must be sanitized and checked to prevent a buggy or malicious client from impacting SM and therefore the full platform in unwanted ways.

5.4 Examples of board control usage

1. Read a PF09 PMIC register:
   - Create a `BRD_SM_CTRL_PMIC_ACCESS` control ID.
   - Extend the board implementation of `SCMI_MiscControlExtGet(uint32_t ctrlId, uint32_t addr, uint32_t numRtn, uint32_t *rtn)` to call `BRD_SM_PmicRead(BOARD_PF09_DEV_ADDR, addr, rtn)` when called with the `BRD_SM_CTRL_PMIC_ACCESS` ID.
     `numRtn` must be 1 and `rtn` must be a valid pointer.
   - Any client can then call the function as follows:

```
SCMI_MiscControlExtGet(SCMI_A2P, BRD_SM_CTRL_PMIC_ACCESS |
 SCMI_MISC_CTRL_FLAG_BRD, pmic_reg_address, 1, &rtn);
```

2. Write a PF09 PMIC register:
   - Use the same control ID as above.
   - Extend the board implementation of `SCMI_MiscControlExtSet()` to write to the PMIC register when called with the `BRD_SM_CTRL_PMIC_ACCESS` ID.

6 Device and board control client support

The i.MX MCUXpresso SDK v16_00 for i.MX 95 supports the full SCMI_MiscControl API set, except `SCMI_MiscControlExtSet()` and `SCMI_MiscControlExtGet()`.

The NXP Linux kernel 6.6.36_2.1.0 only supports `SCMI_MiscControlSet`, `SCMI_MiscControlGet`, and `SCMI_MiscControlNotify` API.

7 Control specificities

This section describes parameters and usage specific to some SM controls.

SCMI_MiscCtrlNotify()/SCMI_MiscCtrlEvent()

Any agent can call `SCMI_MiscCtrlNotify()` to request notifications from a controlID. Any other agent can then use `SCMI_MiscCtrlEvent()` to broadcast notifications associated with this control ID.

`SCMI_MiscCtrlNotify()` and `SCMI_MiscCtrlEvent()` take the control ID and a 32-bits flag as argument. The notification sent by `SCMI_MiscCtrlEvent()` is delivered to all agents registered for notifications for the relevant control ID and who have at least one flag bit matching the flag set in `SCMI_MiscCtrlNotify()`.

The device layer call associated with `SCMI_MiscCtrlNotify()` is `DEV_SM_ControlFlagSet()`. It can be redirected the usual way. The i.MX 95 board port uses this mechanism to notify agents of incoming interrupts from GPIO expanders (see `BRD_SM_ControlHandler()` function).

8 References

The references used to supplement this document are as follows:

- Arm System Control and Management Interface Platform Design Document v.3.2: https://developer.arm.com/documentation/den0056
- System manager GitHub repository: GitHub - nxp-imx/imx-sm: System Manager firmware for i.MX processors

9 Acronyms

Table 2 lists the acronyms used in this document.

Table 2. Acronyms

Term	Description
LMM	Logical machine management
RPC	Remote procedure call
SCMI	System control and management interface
SM	System manager
SoC	System-on-chip

10 Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

11 Revision history

Table 3 summarizes the revisions to this document.

Table 3. Revision history

Document ID	Release date	Description
AN14478 v.1.0	25 November 2024	Initial public release

Legal information

Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors’ aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer’s own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer’s sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer’s applications and products planned, as well as for the planned application and use of customer’s third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer’s applications or products, or the application or use by customer’s third party customer(s). Customer is responsible for doing all necessary testing for the customer’s applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer’s third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at https://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer’s general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors’ warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors’ specifications such use shall be solely at customer’s own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors’ standard warranty and NXP Semiconductors’ product specifications.

HTML publications — An HTML version, if available, of this document is provided as a courtesy. Definitive information is contained in the applicable document in PDF format. If there is a discrepancy between the HTML document and the PDF document, the PDF document has priority.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer’s applications and products. Customer’s responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer’s applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

Contents

1 Introduction ... 2
2 System manager controls ... 2
3 RPC for managing controls ... 2
4 Device controls vs board controls ... 3
4.1 Device layer redirection ... 3
4.2 Device layer redirection workflow ... 3
4.3 Example ... 4
5 Board control implementation ... 6
5.1 Control ID ... 6
5.2 Implementation outline ... 7
5.2.1 Client-side implementation ... 7
5.3 Caveats ... 7
5.4 Examples of board control usage ... 8
6 Device and board control client support ... 8
7 Control specificities ... 8
8 References ... 8
9 Acronyms ... 8
10 Note about the source code in the document ... 9
11 Revision history ... 9
Legal information ... 10

Please be aware that important notices concerning this document and the product(s) described herein, have been included in section 'Legal information'.