



User Guide

SE Host Services API

Version 1.110.0

SERVICES Library 0.50.10

Table of Contents

Introduction	7
SERVICES Library Change log	8
SERVICES Library Pairings.....	11
Known Limitations	12
A32	12
SE-UART Spamming	12
SE Host Services summary	12
SE Host Services Delivery Components	15
Pre-requisites	15
Building SE Services – Windows / LINUX	16
Building with ARM GNU C.....	17
Building with ARM CLANG	17
SERVICES Library Dependencies	17
CMSIS Package	17
JSON Configurations	18
Power Example	18
Power Example Use Cases	19
BASIC1 (No XIP)	19
BASIC2 (XIP).....	19
BASIC3 (No XIP)	19
BASIC4 (No XIP)	19
BASIC5 (No XIP)	19
BASIC6	19
BASIC7 (No XIP)	19
BASIC8 to BASIC13 Clock configuration examples (No XIP)	19
BASIC14 GET request examples (No XIP)	19
BASIC15 Clock Source Cycling (No XIP)	20
BASIC16 M55s run in TCM not retained (No XIP)	20
BASIC17 PD5=OFF and Wake up	20
BASIC18 Raw power services	20
Power Consumption Examples	20
SES Power Policies.....	21
Power Example Running – Debugging common issues	21
ISP Not responding.....	21

- Cannot use UpdateSystemPackage 21
- PPU Interrupt Spamming 21
- Power Examples Limitations 22
- SES Clock Policies 22
- SERVICES test harness example 23
 - Examples customization options. 23
- Changing CMSIS Packs 23
- Building and running the Examples 24
 - Building the M55 HE Example - run from TCM. 24
 - Building the M55 HE Example - run from MRAM. 24
 - Building and running the M55_HE Power Example (ARM Clang) 25
 - Building and running the M55_HE Power Example (ARM GNUC) 25
 - Building and running the M55_HP Power Example (ARM Clang) 25
 - Building and running the M55_HP Power Example (ARM GNUC) 25
 - Building and running the STOC update example 26
 - Building SE Host Services – LINUX..... 26
 - Installing examples..... 27
 - Installing examples to a different location 27
 - Running with SERVICES Debug disabled 29
 - Building the M55 Host Example under ARM-DS..... 30
- Adding ALIF SE SERVICES to your Application code 31
- SE Firmware OTA Updates 32
 - Update Packages..... 32
 - Update Packages Header 33
 - Full Update Packages 34
 - Partial Update Packages 34
 - Update Packages acceptance policy 34
 - Expected usage case 35
- SE Firmware JTAG Provisioning..... 36
 - Development and Production Flow for provisioning..... 36
 - Overview of JTAG Provisioning 37
 - Using JTAG Provisioning..... 37
- SE Host Services Library API 38
 - Host Services Library Interface API Porting Layer..... 38
 - SERVICES_wait_ms..... 38

SERVICES_send_mhu_message_to_se 38

Host Services Library API Layer 39

 SERVICES_initialize 39

 SERVICES_send_request 39

 SERVICES_send_msg_acked_callback 40

 SERVICES_rx_msg_callback 40

SERVICES Based Provisioning 41

SE Host SERVICES Library 42

 SE Host SERVICES Library - Anatomy of a SERVICE Call 42

 SE Host Service Library Internal implementation 42

 SE Host Service Library Transport Protocol details 43

 SE Host Service Library Transport Error Codes 43

 SE Host Services Library Error Handling 44

 SE Host Services Library Memory handling 44

SE Host Services API 45

Miscellaneous 45

 SERVICES_initialize 45

 SERVICES_version 47

 SERVICES_register_channel 48

 SERVICES_prepare_packet_buffer 49

Maintenance Services 50

 SERVICES_heartbeat 50

 SERVICES_synchronize_with_se 51

System Management 52

 SERVICES_system_set_services_debug 52

 SERVICES_system_read_otp 53

 SERVICES_system_write_otp 54

 SERVICES_system_get_otp_data 54

 SERVICES_system_get_toc_data 55

 SERVICES_system_get_toc_number 57

 SERVICES_system_get_toc_version 58

 SERVICES_system_get_toc_via_name 59

 SERVICES_system_get_toc_via_cpuid 60

 SERVICES_system_get_device_part_number 62

 SERVICES_system_get_device_data 63

SERVICES_system_get_eui_extension 65

SERVICES_system_get_device_id64 65

SERVICES_system_get_ecc_public_key 66

SERVICES_get_se_revision 66

Application Services 68

 SERVICES_uart_write 68

 SERVICES_pinmux 69

 SERVICES_padcontrol 69

 SERVICES_application_ospi_write_key 70

 SERVICES_application_verify_image 71

 SERVICES_application_dmpu 73

Power Services 74

 SERVICES_power_stop_mode_request 74

 SERVICES_power_ewic_config 75

 SERVICES_power_wakeup_config 76

 SERVICES_power_mem_retention_config 78

 SERVICES_power_m55_he_vtor_save 80

 SERVICES_power_m55_hp_vtor_save 82

 SERVICES_corestone_standby_mode 84

 SERVICES_power_memory_req 86

 SERVICES_get_run_cfg 87

 SERVICES_set_run_cfg 88

 SERVICES_get_off_cfg 89

 SERVICES_set_off_cfg 90

 SERVICES_power_dcdc_voltage_control 91

 SERVICES_power_ldo_voltage_control 92

 SERVICES_power_setting_configure 93

 SERVICES_power_setting_get 94

 SERVICES_power_stop_mode_raw_req 95

 SERVICES_power_ewic_config_raw 96

 SERVICES_power_wakeup_config_raw 97

 SERVICES_power_mem_retention_config_raw 98

 SERVICES_power_m55_he_vtor_save_raw 99

Clocks Services 100

 Clock frequency definitions 100

SERVICES_clocks_select_osc_source 101

SERVICES_clocks_select_pll_source 103

SERVICES_clocks_enable_clock 104

SERVICES_clocks_set_ES0_frequency..... 105

SERVICES_clocks_set_ES1_frequency..... 105

SERVICES_clocks_select_a32_source 106

SERVICES_clocks_select_aclk_source 107

SERVICES_clocks_set_divider..... 108

SERVICES_clocks_get_clocks..... 109

SERVICES_clocks_get_apb_frequency - OBSOLETE 110

SERVICES_clocks_get_refclk_frequency - OBSOLETE 111

SERVICES_clocks_setting_get 111

SERVICES_clocks_set_aclk 112

SERVICES_pll_xtal_start 113

SERVICES_pll_xtal_stop..... 113

SERVICES_pll_xtal_is_started..... 114

SERVICES_pll_clkpll_start..... 114

SERVICES_pll_clkpll_stop 115

SERVICES_pll_clkpll_is_locked 115

SERVICES_pll_initialize 115

SERVICES_pll_deinit 116

Boot Services..... 116

 SERVICES_boot_process_toc_entry..... 117

 SERVICES_boot_cpu..... 117

 SERVICES_boot_set_vtor 119

 SERVICES_boot_reset_cpu..... 119

 SERVICES_boot_release_cpu 120

 SERVICES_boot_reset_soc 121

Crypto Services..... 122

 SERVICES_cryptocell_get_rnd..... 122

 SERVICES_cryptocell_get_lcs 123

 MbedTLS Services 124

Update Services 131

 SERVICES_update_stoc 131

External System Services 132

Boot arguments	132
SERVICES_Boot_Net_Proc.....	133
SERVICES_Shutdown_Net_Proc.....	134
Document History	135

Introduction

The Fusion product series is a scalable SoC solution for IoT Edge Computing platforms.

SERVICES provide a method for the Application CPUs (M55_HE, HP, A32) to communicate with the Secure Enclave. This secure communication path is achieved using the MHU (Message Handling Unit) hardware block.

The SERVICES library consists of C code that interfaces with an MHU driver to facilitate this communication.

Services fall into the following categories:

- Non-Power
 - o Maintenance Services
 - o Crypto Services
 - o Update Services
 - o Secure Debug Service
 - o Application Services
- Power
 - o BASIC Use case examples
 - o Power number user case examples

The library source code is provided along with a test harness showing the invocation of each SERVICE library call.

The example (test harness) can be used as a framework to copy for integrating SERVICES into your application code. You only need to include the SERVICE header files and link with the SERVICE libraries.

The examples can be built for all Application CPUs in XIP and Non-XIP mode. Sample ATOC configuration files are also provided.

ARM Clang and ARM GNU CC are both supported.

SERVICES Library Change log

0.50.10

Defaulting example build to use CMSIS 2.0.4
Adding POWER_SETTING_ANA_PERIPH_EN to power_setting_t
Adding SERVICES_application_dmpu
Services power examples updated to deal with RTC pre-scaling for EAGLE REV_A1
CMSIS V2.0.4 update, default for SERVICES examples.
Check for MAX_RND_LENGTH in examples
SERVICES_boot_process_toc_entry() should only process ATOC, not STOC

0.50.9

Adding SERVICES_application_verify_image()
net_proc_boot_svc_t additional configuration field for BLE LPA/HPA support
Defaulting build to use CMSIS 2.0.3
services/examples/power/services_basic_9.c fix
Adding SERVICES_aclk_get
Adding SERVICES_aclk_set
Defaulting build to use CMSIS 2.0.2

0.50.8

BUG services_aipm_stop_modes.c incorrect register name for ARM CMSISV6.1.0)
aipm.h – adding ENSEMBLE_SOC_E1C guard
Defaulting build to use CMSIS V2.0.0-rc5
Defaulting build to use CMSIS V2.0.0-rc4

0.50.7

Adding services_benchmark.c
Adding basic_19 example (for EAGLE ONLY)

0.50.6

Adding SERVICES_system_get_ecc_public_key
Adding SERVICES_power_stop_mode_raw_req
Adding SERVICES_power_ewic_config_raw(
Adding SERVICES_power_wakeup_config_raw
Adding SERVICES_power_mem_retention_config_raw
Adding SERVICES_power_m55_he_vtor_save_raw
Removal of unused / unnecessary OTP bit fields from services_lib_api.h
Fix for mfr_data_t bitfields
Updating SERVICES_version_data_t structure to take new fields for Alternate boot data
Rename CLKEN_USB to CLKEN_CLK_20M
Adding new clock for ENSEMBLE E4.E6 and E8
CMSIS V2.0.0 updates (NOTE: This is no longer compatible with previous CMSIS releases)

- M55_HE_E1C build option added
- aipm.h
 - Changes required due to CMSIS namespace collisions with ip_clock_gating_t fields
USB and USB_MASK
 - USB → USB_CLK (a better name is needed?)
 - USB_MASK → USB_CLK_GATE_MASK
 - Power examples
 - Updated to replace USB, USB_MASK to new enumeration names

- LPRTC0_IRQ_IRQn instead of LPRTC_IRQ_IRQn
 - Addition of #include "sys_utils.h"
- 0.50.5
 - Adding SERVICES_system_get_device_id64
 - Updated documentation stating SERVICES_update_stoc does not support MRAM (it does)
- 0.50.4
 - Doxygen updates across all SERVICES files
 - Update the get_eui_extension service to the latest Manufacturing data formats
- 0.50.3
 - Corrected Retention bit field masks
 - Ignore XIP build flag
 - UPD example changes
- 0.50.2
 - SERVICES_toc_info_t updated with extra fields.
 - adding SERVICES_system_gettoc_version API implementation for SE version
 - adding SERVICES_ERROR_INVALID_PARAMETER
 - nvds data added field for ESO internal clock selection field
 - SERVICES switch to CMSIS V1.3.0.
 - Updated Scatter file for ARM-DS
- 0.50.1 Updated contributed power examples
- 0.50.0 API for getting HFOSC and EXTSYS0/1 frequency
- 0.49.0 Deprecating SERVICES_system_get_toc_via_name
- 0.49.1 Updated examples for ALIF Update Package support.
- 0.48.0 Switch to external CMSIS source builds
- 0.0.47 Add Power setting Get/Configure API
- 0.0.46 Adding UPDATE STOC Service and test
- 0.0.45 Adding STOP, STANDBY Cycle tests
 - Adding SES Full update Service
- 0.0.44 Example test changes
- 0.0.43 SERVICES switch to CMSIS V1.0.0
- 0.0.42 Reduce the size of the packet buffer in the services examples
- 0.0.41
- 0.0.40 [aiPM] Define the parameter VDD_IOFLEX_3V3 as an enum
- 0.0.39 SERVICES switch to CMSIS V0.9.4
- 0.0.38 SERVICES switch to CMSIS V0.9.3
- 0.0.37 Scalable HFRC and HXTAL frequencies
- 0.0.36 SERVICES switch to CMSIS V0.9.1
 - arm clang startup issues
- 0.0.35 SERVICES build switch to CMake
 - addition of aiPM Service API - RUN
- 0.0.34 Addition of amPM Service API - OFF
 - retrieve full revision info
- 0.0.33 edits for simulation testing
- 0.0.32 Add a new service call SERVICES_boot_set_vtor
 - Use defines to support different power test variations
- 0.0.31 Warnings fixes
- 0.0.30 Fix services-he-hp-a32-xip.json HP address
- 0.0.29 Consistent Error handling

- Memory Power On Off
- Add boot services and clock tests
- 0.0.28 Add new warnings and do cleanup
- 0.0.27 Add new warnings and do cleanup
- 0.0.26 Retention fine grained control
 - CMSIS update build options
 - Add SERVICES_INVALID_ADDRESS error code
- 0.0.25 Clocks apis
 - Retention APIs
 - TEST Services initialize polling
- 0.0.24 GCC build for m55_power
 - Query TOC by cpu id returns N entries
- 0.0.23 CMSIS integration
- 0.0.22 Simplify error code usage at the services transport layer
- 0.0.21 Added build support for SPARK
- 0.0.20 Updated ALIF License
- 0.0.19 Adding CMake build
- 0.0.18 Stop mode power profile
- 0.0.17 Fixed unimplemented function warnings in GCC build
- 0.0.16 Update to RTSS V0.4.1
- 0.0.15 Added examples installation, fixed build flags
- 0.0.14 Addition of XIP examples
- 0.0.13 Examples common directory
- 0.0.12 get otp data (for real!)
- 0.0.11 get all toc info
 - get otp data
- 0.0.10 standardized variables for send/resp
- 0.0.9 bounds checks for UART prints
- 0.0.8 Added firmware version id
- 0.0.7 mbed TLS accelerators
- 0.0.6 Added enable / disable debug status
- 0.0.5 Service API error code added
- 0.0.4 Service BOOT reset added
- 0.0.3 RPC Parameter changes
- 0.0.2 First re-factoring
- 0.0.1 Concept + realization - First implementation

SERVICES Library Pairings

SERVICES Version	SES Version	CMSIS Version	NOTES
0.50.10	1.110.0		
0.50.9	1.109.0	2.0.3	CMSIS update
0.50.8	1.108.0		SES support for SERVICES_aclk_get/set
0.50.7	1.107.0		
0.50.6	1.106.0		SES support for SERVICES_system_get_ecc_public_key SES support for SERVICES_version_data_t new field retrieval
0.50.5	1.105.0		
0.50.4	1.104.0		
0.50.0	1.103.0		
0.50.0	1.102.0		

Known Limitations

A32

A32 bare-metal support is not provided by the ALIF CMSIS packs.

SE-UART Spamming

When using CONTINUOUS mode in the test harness via the SE-UART print SERVICE the amount of print traffic can be very heavy and make the system unresponsive. We recommend you use a local UART (UART 2 or 4) dedicated to the M55_HE or M55_HP. Note that there are mechanisms to stall the spamming within SES.

The SERVICES print capability using the SE-UART was added as a convenience to save having to set up external UARTs.

SE Host Services summary

Service Group		Notes
Maintenance		
	SERVICES_heartbeat	Health status
System Management		
	SERVICES_system_get_toc_data	
	SERVICES_system_get_toc_number	
	SERVICES_system_get_toc_via_name	N/I
	SERVICES_system_get_toc_version	N/I
	SERVICES_system_get_toc_via_cpuid	
	SERVICES_system_get_device_part_number	
	SERVICES_system_get_device_data	
	SERVICES_system_set_services_debug	Debug toggle
	SERVICES_system_get_otp_data	N/I
	SERVICES_system_read_otp	
	SERVICES_get_se_revision	
	SERVICES_system_get_eui_extension	
	SERVICES_system_get_ecc_public_key	
	Application / Pin mux management	
SERVICES_pinmux		
SERVICES_padcontrol		
SERVICES_uart_write		
SERVICES_application_ospi_write_key		
SERVICES_application_verify_image		
SERVICES_application_dmpu		

Power		
	SERVICES_power_stop_mode_request	
	SERVICES_power_ewic_config	
	SERVICES_power_wakeup_config	
	SERVICES_power_mem_retention_config	
	SERVICES_power_m55_he_vtor_save	
	SERVICES_power_m55_hp_vtor_save	
	SERVICES_power_memory_req	
	SERVICES_global_standby_mode	
	SERVICES_get_run_cfg	aiPM
	SERVICES_set_run_cfg	aiPM
	SERVICES_get_off_cfg	aiPM
	SERVICES_set_off_cfg	aiPM
	SERVICES_power_ldo_voltage_control	
	SERVICES_power_dcdc_voltage_control	
	SERVICES_power_setting_configure	
	SERVICES_power_setting_get	
	SERVICES_power_mem_retention_config_raw	
	SERVICES_power_ewic_config_raw	
	SERVICES_power_wakeup_config_raw	
	SERVICES_power_stop_mode_raw_req	
	SERVICES_power_m55_he_vtor_save_raw	
Security /Crypto		
	SERVICES_cryptocell_get_lcs	
	SERVICES_cryptocell_get_rnd	TRNG
	SERVICES_cryptocell_mbedtls_aes_init	
	SERVICES_cryptocell_mbedtls_aes_set_key	
	SERVICES_cryptocell_mbedtls_aes_crypt	
	SERVICES_cryptocell_mbedtls_sha_starts	
	SERVICES_cryptocell_mbedtls_sha_process	
	SERVICES_cryptocell_mbedtls_sha_update	
	SERVICES_cryptocell_mbedtls_sha_finish	
	SERVICES_cryptocell_mbedtls_ccm_gcm_set_key	
	SERVICES_cryptocell_mbedtls_ccm_gcm_crypt	
	SERVICES_cryptocell_mbedtls_ccm_gcm_chachapoly_crypt	
	SERVICES_cryptocell_mbedtls_ccm_gcm_poly1305_crypt	
	SERVICES_cryptocell_mbedtls_cmac_init_setkey	
	SERVICES_cryptocell_mbedtls_cmac_update	
	SERVICES_cryptocell_mbedtls_cmac_finish	
	SERVICES_cryptocell_mbedtls_cmac_reset	
Boot		
	SERVICES_boot_process_toc_entry	
	SERVICES_boot_cpu	
	SERVICES_set_vtor	

	<u>SERVICES boot reset cpu</u>	
	<u>SERVICES boot release cpu</u>	
	<u>SERVICES boot reset soc</u>	
Clock		
	<u>SERVICES clocks select osc source</u>	
	<u>SERVICES clocks select pll source</u>	
	<u>SERVICES clocks enable clock</u>	
	<u>SERVICES_clocks_set_ES0_frequency</u>	
	<u>SERVICES_clocks_set_ES1_frequency</u>	
	<u>SERVICES_clocks_select_a32_source</u>	
	<u>SERVICES_clocks_select_aclk_source</u>	
	<u>SERVICES_clocks_set_divider</u>	
	<u>SERVICES_clocks_get_clocks</u>	
	<u>SERVICES_clocks_get_apb_frequency</u>	
	<u>SERVICES_clocks_get_refclk_frequency</u>	
	<u>SERVICES_pll_initialize</u>	
	<u>SERVICES_pll_deinit</u>	
	<u>SERVICES_pll_xtal_start</u>	
	<u>SERVICES_pll_xtal_stop</u>	
	<u>SERVICES_pll_xtal_is_started</u>	
	<u>SERVICES_pll_clkpll_start</u>	
	<u>SERVICES_pll_clkpll_stop</u>	
	<u>SERVICES_pll_clkpll_is_locked</u>	
Update		
	<u>SERVICES_update_stoc</u>	

SE Host Services Delivery Components

A SERVICES release package from ALIF consists of the following components:

- Source code SERVICES library
 - Public header files
- CMAKE files for ARM Clang and ARM GNU C builds
- Example ports for Bare metal
 - Example SERVICE library initializations.
- Example use cases for M55_HE, M55_HP, A32 and M55_HE+M55_HP
 - Example runs a test program calling all available SERVICES API.
 - Output is sent via SE-UART to save having to install extra UART debug ports.
- Example use cases for Low Power
 - BASIC tests.

Pre-requisites

The following components are required to be installed before using / building SE SERVICES:

- ALIF Ensemble RTSS Release
 - CMSIS Packs for Ensemble devices
 - Following the installation instructions for this package
- GNU Make V4.4
- Cmake V3.22.2
- Security Toolkit (SETOOLS)
 - It is not required for building, but it is required for generating the ATOC packages for the Target and programming them into MRAM.

Building SE Services – Windows / LINUX

Unpack the se-host-services-release-SE_FW_0.<version#>.000_DEV.zip

Name	Date modified	Type	Size
a32_startup	7/11/2023 4:15 PM	File folder	
build	7/7/2023 7:51 AM	File folder	
drivers	7/7/2023 7:51 AM	File folder	
example	7/11/2023 4:15 PM	File folder	
include	7/11/2023 6:01 AM	File folder	
lib	7/7/2023 7:51 AM	File folder	
RTE	7/11/2023 4:15 PM	File folder	
services_lib	7/11/2023 4:15 PM	File folder	
A32_MRAM	7/11/2023 4:15 PM	Windows Script C...	
A32_TCM	7/11/2023 4:15 PM	Windows Script C...	
CMakeLists	7/11/2023 4:15 PM	Text Document	
CMakePresets	7/11/2023 4:15 PM	JSON File	
gcc_A32_MRAM	7/11/2023 4:15 PM	LD File	
gcc_A32_TCM	7/11/2023 4:15 PM	LD File	
gcc_M55_HE_MRAM	7/11/2023 4:15 PM	LD File	
gcc_M55_HE_TCM	7/11/2023 4:15 PM	LD File	
gcc_M55_HP_MRAM	7/11/2023 4:15 PM	LD File	
gcc_M55_HP_TCM	7/11/2023 4:15 PM	LD File	
License	7/7/2023 7:51 AM	Text Document	
M55_HE_MRAM	7/11/2023 4:15 PM	Windows Script C...	
M55_HE_TCM	7/11/2023 4:15 PM	Windows Script C...	
M55_HP_MRAM	7/11/2023 4:15 PM	Windows Script C...	
M55_HP_TCM	7/11/2023 4:15 PM	Windows Script C...	
Makefile_linux	7/7/2023 7:51 AM	File	
README	7/11/2023 4:15 PM	Markdown Source...	
toolchain-armclang	7/11/2023 4:15 PM	CMake Source File	
toolchain-gnu	7/11/2023 4:15 PM	CMake Source File	

The release archive consists of the following target components.

Name	Purpose	Notes
a32_startup	A32 startup code	
Build		
include	Services header files	
lib		
services_lib	Host source code for Services	
example	SERVICES test harnesses	
README.md	instructions	

CMAKE is used.

Building with ARM GNU C

```
$ cd se-host-service-release
$ mkdir build_he_gcc_tcm
$ cd build_he_he_gcc_tcm
$ cmake .. -G "Unix Makefiles" -DENSEMBLE_CORE=M55_HE -
DCMAKE_TOOLCHAIN_FILE=toolchain-gnu.cmake
$ make install
```

```
$ cd se-host-service-release
$ mkdir build_he_power_gcc_tcm
$ cd build_he_power_gcc_tcm
$ cmake .. -G "Unix Makefiles" -DENSEMBLE_CORE=M55_HE -
DCMAKE_TOOLCHAIN_FILE=toolchain-gnu.cmake -DPOWER=ON
$ make install
```

The SERVICE library is built as part of this building process.

Building with ARM CLANG

```
$ cd se-host-services-release
$ mkdir build_he_power_clang_tcm
$ cd build_he_power_clang_tcm
$ cmake .. -G "Unix Makefiles" -DENSEMBLE_CORE=M55_HE -DPOWER=ON
DCMAKE_TOOLCHAIN_FILE=toolchain-armclang.cmake
$ make install -j 8
```

```
$ cd se-host-services-release
$ mkdir build_he_clang_tcm
$ cd build_he_clang_tcm
$ cmake .. -G "Unix Makefiles" -DENSEMBLE_CORE=M55_HE -
DCMAKE_TOOLCHAIN_FILE=toolchain-armclang.cmake
$ make install -j 8
```

NOTE: Parallel make (using -j <job#>) is available

SERVICES Library Dependencies

The SERVICES library and support file for starting SERVICES have a few dependencies.

- CMSIS Package installation location
- CMSIS Package source and include locations.

CMSIS Package

The SERVICE example uses the CMSIS startup sequences for booting the Application cores.

CMSIS packages consist of an ARM and ALIF component. The ARM components are specific for the various types of cores. The ALIF components contain specific ALIF Device and Board components.

ALIF CMSIS also provides Global Standby support for Low Power.

JSON Configurations

Examples can be built for the following Application cores:

- M55_HE
- M55_HP
- A32

Sample ATOC JSON files are provided for:

- Code running from MRAM (XIP).
- Code running from TCM memory.
- M55_HE
- M55_HP
- M55_HE + M55_HP

JSON File	TCM	MRAM
services_a32	✓	
services_a32-xip		✓
services_he	✓	
services_he_hp	✓	
services_he_hp_a32	✓	
services_he_hp_a32_xip		✓
services_he_hp_xip	✓	
services_he_xip		✓
services_hp	✓	
services_hp_xip		✓
Services-he-tcm-hp-xip	✓	✓

Power Example

The Power examples demonstrate how to use SERVICES to achieve low power states.

The SERVICES are used to configure the device and enable STOP mode.

Power Example Use Cases

Please refer to the README.md file in the SERVICES release for details of the Example use cases e.g., how to build and run.

BASIC1 (No XIP)

- Keeps device ON after waking up from STOP mode.
- M55_HE is booted on wake up.

BASIC2 (XIP)

- Keeps device ON after waking up from STOP mode.
- M55_HE is booted on wake up.

BASIC3 (No XIP)

- Continually cycles from STOP->WAKE.

BASIC4 (No XIP)

- Example
 - Make a Set RUN Config call, to change the clock settings to something different from the default.
 - Measure the CPU speed, to verify that the above call was executed successfully.
 - Make a Set OFF Config call.
 - Make the PM calls to go OFF. It seems now that is Alif CMSIS functionality.
- Can be configured (built) for continuous or limited run (10) mode.
- **NOTE:** There is an issue when measuring the CPU speeds of both M55s at the same time. It seems it is caused by the shared usage of RTC_A, which is also used for wakeup logic.

BASIC5 (No XIP)

- FASTBOOT with SE not retained.

BASIC6

- Global standby example in TCM.

BASIC7 (No XIP)

- IDLE mode example in TCM

BASIC8 to BASIC13 Clock configuration examples (No XIP)

Each of these examples perform a single aiPM call to set the RUN clock configuration of the device.

- BASIC8 – run the device off the PLL at full CPU frequencies.
- BASIC9 – run the device off the PLL at reduced M55 CPUs frequencies.

BASIC14 GET request examples (No XIP)

- Shows use of the get_off and get_run APIs.

BASIC15 Clock Source Cycling (No XIP)

- Cycles the device between PLL, HFXO and HFRC clock sources.
- Can be continuous or limited to 10 cycles.

BASIC16 M55s run in TCM not retained (No XIP)

- SES loads M55 code from MRAM to its TCM. M55-HE is not retained.
- M55 does a set_run() request
- M55 does a set_off() request
- SES puts the device in STOP mode when both m55s go OFF.
- RTC_A expiry causes wake up.
- PLL not enabled on wake up.
 - Run at RC clock speed 76 MHz.
- SES wakes up.
 - Starts PLL
 - Run at 100 Mhz
 - Initializes m55-he TCM memory.
- SES boots both m55-he and m55-hp from ATOC
 - As both CPUs are booted, you will see the chip cycle.

BASIC17 PD5=OFF and Wake up

- Turn off SES

BASIC18 Raw power services

- EWIC
- VBAT wake up
- Retention
- STOP mode

Power Consumption Examples

A series of examples are provided for various use cases.

- GO_MODES
- READY_MODES
- IDLE_MODES
- STANDBY_MODES
- STOP_MODES
- STANDBY_CYCLE
- STOP_CYCLE

These examples are designed to enable a user to obtain Power numbers like the scenarios described in the Device datasheet.

See README.md file for the build details.

NOTE: These examples use very low clock settings. In this case, the SE-UART output may not appear.

SES Power Policies

- Logic added to process ATOC and boot M55_HE and M55_HP in case the M55_HE TCM was not retained.
- SES will apply retention settings as soon as a service request is received because they cannot be applied after a subsystem goes OFF.

Power Example Running – Debugging common issues

ISP Not responding

If you have the device in a Power OFF state, be aware that commands sent from the SETOOLS maintenance command will return '[ERROR] Target not responding' as the device is turned off. ISP is not running / listening on the target when the Power is off.

Cannot use UpdateSystemPackage

- If you have programmed the device to disable the PLL then using Bulk MRAM transfer commands such as UpdateSystemPackage or app-write-mram will probably fail. The reason is that these commands automatically raise the baud rate for ISP which assumes the PLL is enabled. To use these commands

```
$ updatesystempackage -s  
$ app-write-mram -s
```

The '-s' option suppresses the baud rate increase.

To gain control of the device again you will need to enter Hard Maintenance mode.

PPU Interrupt Spamming

There can be cases where SE-UART will print a constant stream of PPU interrupt messages as seen below:

```
[SES] es0 ppu_isr=0x80
[SES] es0 ppu_aisr=0x2

[SES] es0 PPU PPU_PWRP=0x100 PPU_PWSR=0x108
[SES] es0 ppu_isr=0x80
[SES] es0 ppu_aisr=0x2

[SES] es0 PPU PPU_PWRP=0x100 PPU_PWSR=0x108
[SES] es0 ppu_isr=0x80
[SES] es0 ppu_aisr=0x2

[SES] es0 PPU PPU_PWRP=0x100 PPU_PWSR=0x108
[SES] es0 ppu_isr=0x80
[SES] es0 ppu_aisr=0x2

[SES] es0 PPU PPU_PWRP=0x100 PPU_PWSR=0x108
[SES] es0 ppu_isr=0x80
[SES] es0 ppu_aisr=0x2

[SES] es0 PPU PPU_PWRP=0x100 PPU_PWSR=0x108
[SES] es0 ppu_isr=0x80
[SES] es0 ppu_aisr=0x2
```

This can be caused by incorrect configuration of aiPM settings such as running your code from XIP (i.e. MRAM) but asking aiPM to disable the Power to MRAM.

SES will print a maximum of 50 of these messages and stop printing them. You should use Hard Maintenance mode to gain control of the Device and remove your application code from MRAM.

Power Examples Limitations

- Wakeup timers that expire before you enter STOP mode.
 - You will not enter stop mode.
- The booting of specific CPU core as per requested wake up event is not yet supported.
 - On any configured wake up event the m55-he is booted if its TCM is retained or ATOC is processed, and bootable images are booted (potentially both m55-he and m55-hp if the ATOC has bootable images for both)
- Some tests may not report on the SE-UART as the clock rates are too low.

SES Clock Policies

- SES COLD Boot
 - SES checks for presence of Application Conductor objects specifying Clock directions.
 - If no DCT object is present, SES will set the LF Clock Source to the LFXO (Default)

SERVICES test harness example

A test harness example is provided showing calls to all the SERVICES APIs.

See `example\common\services_test.c`.

There are numerous building options available. Not all tests can be run at once as they either do not return or they reboot the system.

Examples customization options.

Output of the results from the example test can be via the ARM-DS Console or the SE-UART.

In `services_test.c` there are the following defines

```
#define TEST_PRINT_ENABLE      1  /* Enable printing from Test harness */
#define PRINT_VIA_CONSOLE     0  /* Print via Debugger console      */
#define PRINT_VIA_SE_UART    1  /* Print via SE UART terminal       */
```

Flag	Meaning
TEST_PRINT_ENABLE	Turn on output from the test
PRINT_VIA_CONSOLE	Print messages to arm-ds (printf())
PRINT_VIA_SE_UART	Print messages to the SE-UART

You can enable both Console and SE-UART.

If you want to run the test from MRAM the PRINT_VIA_CONSOLE must be disabled.

Changing CMSIS Packs

The SERVICES examples use ALIF CMSIS mainly for:

- M55_HE and HP startup sequences.
- MHU interrupt vectors and numbers
- Global Standby APIs for Low Power example

The SERVICES examples build defaults to the latest ALIF CMSIS Packs.

To change CMSIS versions, do the following:

```
$ cmake .. -G "Unix Makefiles" -DENSEMBLE_CORE=M55_HE -
DCMAKE_TOOLCHAIN_FILE=toolchain-armclang.cmake -Dalifcmsis="2.0.3"
```

Building and running the Examples

Please see README.md supplied in the SERVICES examples directory

Follow the link in this file to the Doxygen API details.

Building the M55 HE Example - run from TCM.

There are two json files supplied in the Services release:

- services-he.json
 - Single Core
- services-he-hp.json
 - Dual Core

Follow the instructions to build the M55_HE or HP example.

```
$ cd <host-release directory>
$ mkdir build_he
$ cd build_he
$ cmake .. -G "Unix Makefiles" -DENSEMBLE_CORE=M55_HE -
DCMAKE_TOOLCHAIN_FILE=toolchain-armclang.cmake
$ make install
```

To boot M55_HE application CPUs you need these steps:

```
$ cd <release directory>/app-release-exec
$ ./app-gen-toc -f build/config/service-he.json
$ ./app-write-mram
```

Building the M55 HE Example - run from MRAM.

There are two json files supplied in the Services release:

- services-he-xip.json
 - Single Core
- services-he-hp-xip.json
 - Dual Core

Follow the instructions to build the M55_HE or HP example.

```
$ cd <host-release directory>
$ mkdir build_he_mram
$ cd build_he_mram
$ cmake .. -G "Unix Makefiles" -DENSEMBLE_CORE=M55_HE -
DCMAKE_TOOLCHAIN_FILE=toolchain-armclang.cmake -DXIP=1
$ make install
```

To boot M55_HE application CPU you need these steps:

```
$ cd <release directory>/app-release-exec
$ ./app-gen-toc -f build/config/service-he-xip.json
$ ./app-write-mram
```

Building and running the M55_HE Power Example (ARM Clang)

```
$ cd se-host-service-release
$ mkdir build_he_power
$ cd build_he_power
$ cmake .. -G "Unix Makefiles" -DENSEMBLE_CORE=M55_HE -
DCMAKE_TOOLCHAIN_FILE=toolchain-armclang.cmake -DPOWER=ON
$ make install

$ cd ../app-release-exec
$ ./app-gen-toc -f build/config/service-he.json
$ ./app-write-mram
```

Building and running the M55_HE Power Example (ARM GNUC)

```
$ cd se-host-service-release
$ mkdir build_he_power
$ cd build_he_power
$ cmake .. -G "Unix Makefiles" -DENSEMBLE_CORE=M55_HE -
DCMAKE_TOOLCHAIN_FILE=toolchain-gnu.cmake -DPOWER=ON
$ make install

$ cd ../app-release-exec
$ ./app-gen-toc -f build/config/service-he.json
$ ./app-write-mram
```

Building and running the M55_HP Power Example (ARM Clang)

```
$ cd se-host-service-release
$ mkdir build_hp_power
$ cd build_hp_power
$ cmake .. -G "Unix Makefiles" -DENSEMBLE_CORE=M55_HP -
DCMAKE_TOOLCHAIN_FILE=toolchain-armclang.cmake -DPOWER=ON
$ make install

$ cd ../app-release-exec
$ ./app-gen-toc -f build/config/service-hp.json
$ ./app-write-mram
```

Building and running the M55_HP Power Example (ARM GNUC)

```
$ cd se-host-service-release
$ mkdir build_hp_power
$ cd build_hp_power
$ cmake .. -G "Unix Makefiles" -DENSEMBLE_CORE=M55_HP -
DCMAKE_TOOLCHAIN_FILE=toolchain-gnu.cmake -DPOWER=ON
$ make install

$ cd ../app-release-exec
$ ./app-gen-toc -f build/config/service-hp.json
$ ./app-write-mram
```

Building and running the STOC update example

```
$ cd se-host-service-release
$ mkdir stoc_update
$ cd stoc_update
$ cmake .. -G "Unix Makefiles" -DENSEMBLE_CORE=M55_HE -
DCMAKE_TOOLCHAIN_FILE=toolchain-gnu.cmake -DEXAMPLE=UPDATE_STOC
$ make install
$ cd <app-release-dir>
$ <copy the STOC update package to build/images/STOC.bin>
$ app-gen-toc -f build/config/services-he-update-stoc.json
$ app-write-mram
```

This example shows how to update the ALIF STOC contents.

It requires an ALIF specific binary (Called an Update Package) to be passed as data to a SERVICE call which will update this image in ALIF MRAM area.

This example adds this update binary to an ATOC image so it can be loaded from MRAM into RAM for the test. In a real system this image would arrive from your OTA source.

Building SE Host Services – LINUX

Unpack the se-host-services-release-SE_FW_0.<version#>.000_DEV.zip

There is a separate makefile file for building the Services library for Linux - 'Makefile_linux', so that file should be used instead of the default 'Makefile' –

```
$ make -f Makefile_linux lib
```

By default, things are set up to use the native GCC compiler in Cygwin.

To use the Alif Yocto cross compiler toolchain and generate binaries for the Alif Linux distribution, a couple of changes are needed.

- comment out the compiler definitions (like 'CC = gcc') in Makefile_linux. The Yocto toolchain provides its own definitions.

- modify the file `services_lib\services_host_handler_linux.c` and replace `'#if 0'` with `'#if 1'`, to include the Linux kernel header file for the MHU driver.

Installing examples

The examples come with supplied JSON files for A32, M55_HE and M55_HP processors including variants for MRAM (XIP) and TCM running.

There is an option to install these examples into your Application Release to enable building an ATOC for putting into MRAM. To build the ATOC you need the JSON file and the binary image for Application. These files are copied from the `se-host-services-release` into your application release.

```
$ cd se-host-services-release
```

Note the use of the `INSTALL_DIR` to specify where your application release lives.

When you unpack your application release you will get a directory structure as follows:

```
app-release-exec-windows-SE-FW_0.<version>
  + app-release-exec
    + build
      + config
      + images
```

The JSON files will be copied to the config directory and the binaries will be copied to images. This is where the ATOC generation tools will look.

A sample sequence would be as follows:

```
$ cd se-host-services-release
$ mkdir build_he
$ cd build_he
$ cmake .. -G "Unix Makefiles" -DENSEMBLE_CORE=M55_HE -
DCMAKE_TOOLCHAIN_FILE=toolchain-armclang.cmake
$ make install -j 8
```

Installing examples to a different location

An example of using the `INSTALL_DIR` is as follows:

```
-DINSTALL_DIR=<some path>
```

If you do not specify the `INSTALL_DIR` then the default is `app-release-exec`

Use the following to override the default:

```
$ cd se-host-services-release
$ mkdir build_he
```

```
$ cd build_he
$ cmake .. -G "Unix Makefiles" -DENSEMBLE_CORE=M55_HE -
DCMAKE_TOOLCHAIN_FILE=toolchain-armclang.cmake -DINSTALL_DIR=<some
location>
$ make install -j 8
```

```
$ cmake .. -G "Unix Makefiles" -DENSEMBLE_CORE=M55_HE -DCMAKE_TOOLCHAIN_FILE=toolchain-armclang.cmake -DINSTALL_DIR=./junk
-- [INFO] Version=9
-- [INFO] installation override, using ./junk
-- The C compiler identification is ARMClang 6.18.2
-- The CXX compiler identification is ARMClang 6.18.2
-- The ASM compiler identification is ARMClang
```

A message shows that installation override is enabled.

The default directory is app-release-exec which is the one used on Windows. LINUX release will have the directory name app-release-exec-linux which you would need to specify using the `INSTALL_DIR` option (or simply rename the app-release-exec-linux directory to app-release-exec).

Running with SERVICES Debug disabled

The test harness has a call to SERVICES_system_set_services_debug() which can disable or enable the debug traffic from SES for the SERVICE traffic.

With the SERVICES debug set to false:

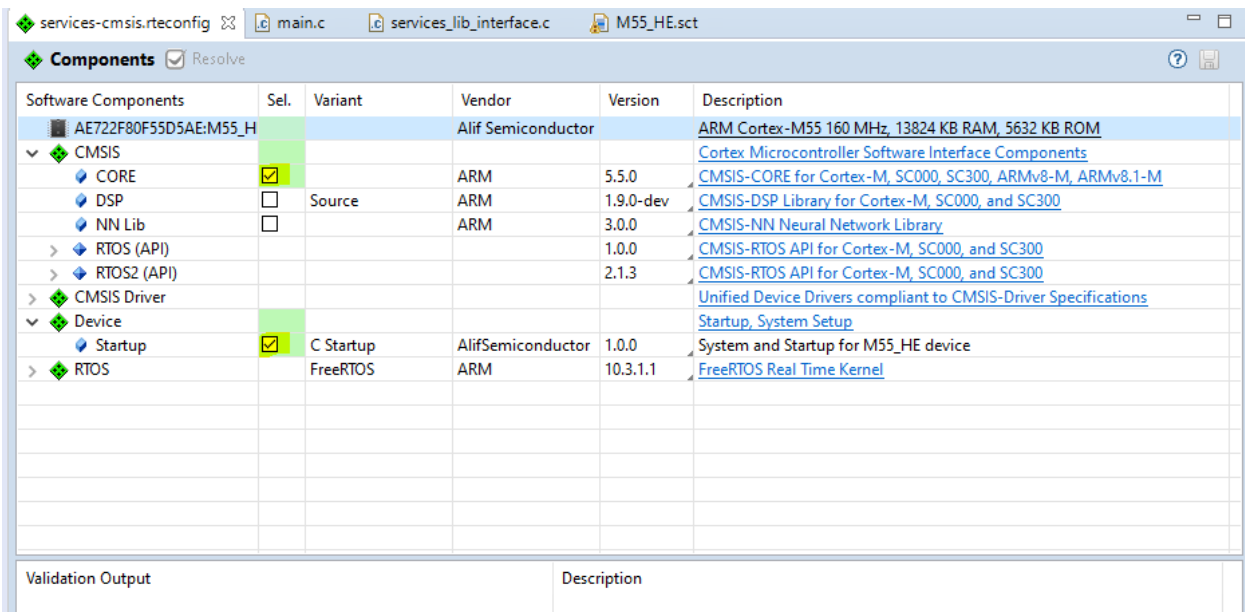
```
[SES] CM0+ frequency is 100 MHz
[SES] os Kernel 010.4.2
[SES] main Task - Looping forever...
[SRU] RK- SID= 0x0CE, Receiver ID=4, Address=0x9083FFA8
[TTV] SERVICES version 0.0.6
[TTV] ** TEST heartbeat error_code=SERVICES_REQ_SUCCESS service_resp=0x00000000
[TTV] ** TEST pinmux error_code=SERVICES_REQ_SUCCESS service_resp=0x00000000
[TTV] ** TEST padcontrol error_code=SERVICES_REQ_SUCCESS service_resp=0x00000000
[TTV] ** TEST crypto TRNG 64 error_code=SERVICES_REQ_SUCCESS 64-bit Random value = 0x1cfed9edc1501c20 service_resp=0
[TTV] ** TEST crypto TRNG 32 error_code=SERVICES_REQ_SUCCESS 32-bit Random value = 0xa479c88f service_resp=0
[TTV] ** TEST crypto TRNG 64 error_code=SERVICES_REQ_SUCCESS 64-bit Random value = 0xbc3fcf15f8479420 service_resp=0
[TTV] ** TEST crypto TRNG 32 error_code=SERVICES_REQ_SUCCESS 32-bit Random value = 0x5cfbb70c service_resp=0
[TTV] ** TEST crypto LCS error_code=SERVICES_REQ_SUCCESS LCS State 0x0 service_resp=0
[TTV] ** TEST get ATOC error_code=SERVICES_REQ_SUCCESS Application TOC number = 0 service_resp=0x00000000
[TTV] ** TEST TOC via name HE error_code=SERVICES_REQ_SUCCESS service_resp=0x00000000
[TTV] ** TEST TOC via name HP error_code=SERVICES_REQ_SUCCESS service_resp=0x00000000
[TTV] ** TEST TOC via cpuid error_code=SERVICES_REQ_SUCCESS id HE_DBG flags 63 1 0 0 service_resp=0x00000000
[TTV] ** TEST TOC via cpuid error_code=SERVICES_REQ_SUCCESS id HE_DBG flags 63 1 0 0 service_resp=0x00000000
[TTV] ** TEST soc id error_code=SERVICES_REQ_SUCCESS Device number 0xA100 service_resp=0x00000000
[TTV] ** TEST Boot TOC A32 error_code=SERVICES_REQ_SUCCESS service_resp=0x00000000
|
```

The SERVICE call to set the debug output off can be seen (the default is enabled in SES). After that, there is no SERVICE debug traffic from SES.

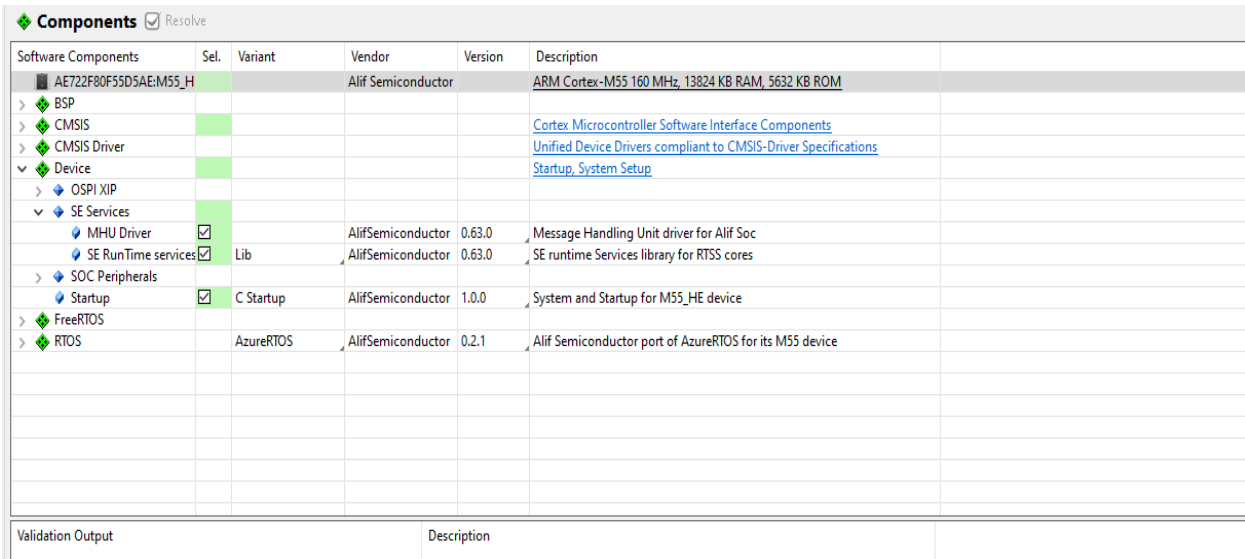
Building the M55 Host Example under ARM-DS

Before starting, ensure you have the ALIFSemiconductor CMSIS Pack installed (See [AP002 Getting Started with Bare Metal & Azure RTOS](#))

- Create a new Project -> C Project -> CMSIS C/C++ Project
- Select Device -> Alif Semiconductor -> Ensemble -> E7 -> AE722F80F55D5AE (or the other one) -> AE722F80F55AE:M55_HE
- In the Project Components window,
 - Check the following highlighted boxes,
 - Then File -> Save



Select the SE Services



Adding ALIF SE SERVICES to your Application code

Calling SERVICES from your Application requires the following:

- Include the header files:
 - `/service-release/include/services_lib_api.h`
 - `/service-release/include/aipm.h`
- Link with
 - `/service-release/lib/libservices_m55_lib.a` (or `_a32_`)
 - `/service-release/lib/libmhu_m55_lib.a` (or `_a32_`)
- Copy or create your own `service_lib_interface.c` file and add it to your build.
 - Change any interrupt sources as required.
 - Implement wait function for your environment.
 - Implement print function for your environment.

SE Firmware OTA Updates

In production applications there is usually a facility to update the product in the field using an Over the Air (OTA) strategy.

One requirement could be to update the version of the SES firmware as part of the OTA process. In development users will use the STTOOLS UpdateSystemPackage across SE-UART to perform this update, in a real product this is not practical.

For OTA solutions you can use the following packages (which are part each SETOOLS release)

- Full Update Package (FUPD)
- Partial Update Package (PUPD)

These are ALIF signed packages containing the latest SES binaries which can be stored anywhere in Memory (SRAM or MRAM) and via a SERVICE call the update can proceed.

Update Packages

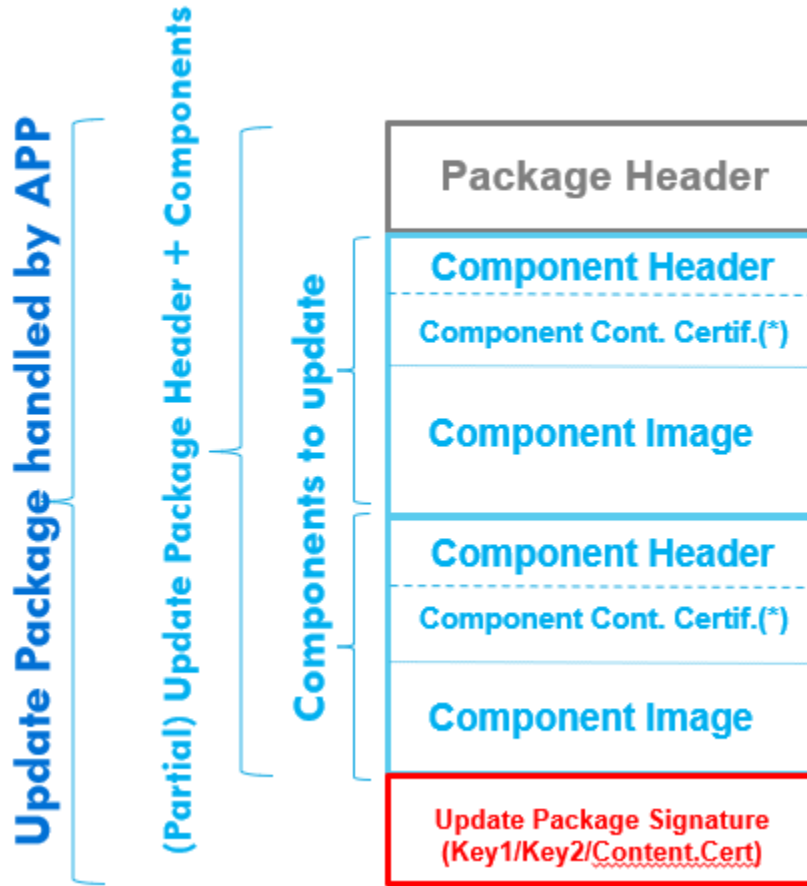
For an SES update there is not just the SES executable images, the packages contain a full STOC (System Table of Contents). This can include

- DEVICE CONFIG data
- PATCHROM (if using BALLETTTO devices)

The FUDP package format is as follows:



The PUPD package format is as follows:



These packages are wrapped with a Package Header allowing SES to identify these binary blobs as Update objects.

This package is then signed using the ALIF Key.

Update Packages Header

The header for FUPD and PUPD is as follows:

```
typedef struct
{
    uint8_t header[4];           /*!< identifier; FUPD (Full) or PUPD (Partial) */
    uint16_t version;           /*!< Update version */
    uint16_t header_size;       /*!< UPD Package Header size (in bytes) */
    uint16_t number_of_entries; /*!< Number of components (only for Partial) */
    uint16_t entry_header_size; /*!< Size of entry header (only for Partial) */
    uint32_t package_size       /* to know where the Certs.chain starts */
    uint8_t device_package[16] /* Device Package to match with Device */
} update_package_header_t;
```

For PUPD, the Component header is as follows:

```
typedef struct
{
    uint32_t object_length;    /*!< Length in bytes */
    uint32_t object_pad;      /*!< pad in bytes */
    uint32_t flags;           /*!< TOC object operators */
    uint32_t version;         /*!< Versioning for this object */
    uint8_t image_identifier[TOC_ENTRY_IMAGE_NAME_LENGTH]; /*!< Image name*/
    Uint8_t pad[8]            /* pad with 0s to complete 16-byte
alignment */
} update_component_header_t;
```

Header contains a signature that identifies this as an Update Package.

In a partial update package, there are separate component headers for each object inside the package. SES will process each of these objects to complete the update.

Full Update Packages

Full Update Packages (FUPD) are the equivalent of what is used with SETOOLS UpdateSystemPackage.

The contents include two copies of the SES image (one for each BANK supported in the ALIF section of the Nonvolatile memory).

Partial Update Packages

Partial Update Packages contain subcomponents e.g. only a single copy of the SES binary.

Update Packages acceptance policy

SES will check the integrity of the package before updating the MRAM.

The following steps are performed

- Stage#1 Check package header is an UPDate package
 - This is a check of the signature
- Stage#2 Verify UPDate package signature
 - This verifies the Package itself
- Stage#3 Check UPDPackage Part# matches this device
 - Compare the running SoC device with the Package
- Stage#4 Write MRAM

If any of the above stages fail, then an error will be returned from the SERVICE call.

The contents of the Package e.g. the SES image is separately signed and will be validated on Boot up by SEROM. ALIF guarantees that the internal contents are correct before any release is made.

If the update package was corrupted during the download, it will fail the verification stage (Stage#2) and the package will be rejected and not burnt into the MRAM.

Expected usage case

A user application will support an OTA strategy that can take binary images from an OTA source to allow the product to be updated. This application will manage the retrieval of this data and its storage on the device, which can be in SRAM or MRAM locations.

Once the data is loaded into the device it is expected that the application will have packetized this data and have a way to identify ALIF binary updates from Application updates. If the download contains an ALIF update the Application will call the following SERVICE call:

```
uint32_t SERVICES_update_stoc(uint32_t services_handle,  
                             uint32_t image_address,  
                             uint32_t image_size,  
                             uint32_t *error_code)
```

The `image_address` is where the specific ALIF download has been stored. NOTE: The `image_size` parameter is deprecated as the size is now part of the Update Package Header. SES can handle both FUPD and PUPD packages via this SERVICE call, there is no requirement for the application to filter the type of update package being sent.

Assuming the package is approved by SES then MRAM will be written with the new update ALIF package.

The next step would be for the application to reset the SoC and the new SES firmware will be executed.

There is an example test harness showing the use of this API in the SERVICES release.

SE Firmware JTAG Provisioning

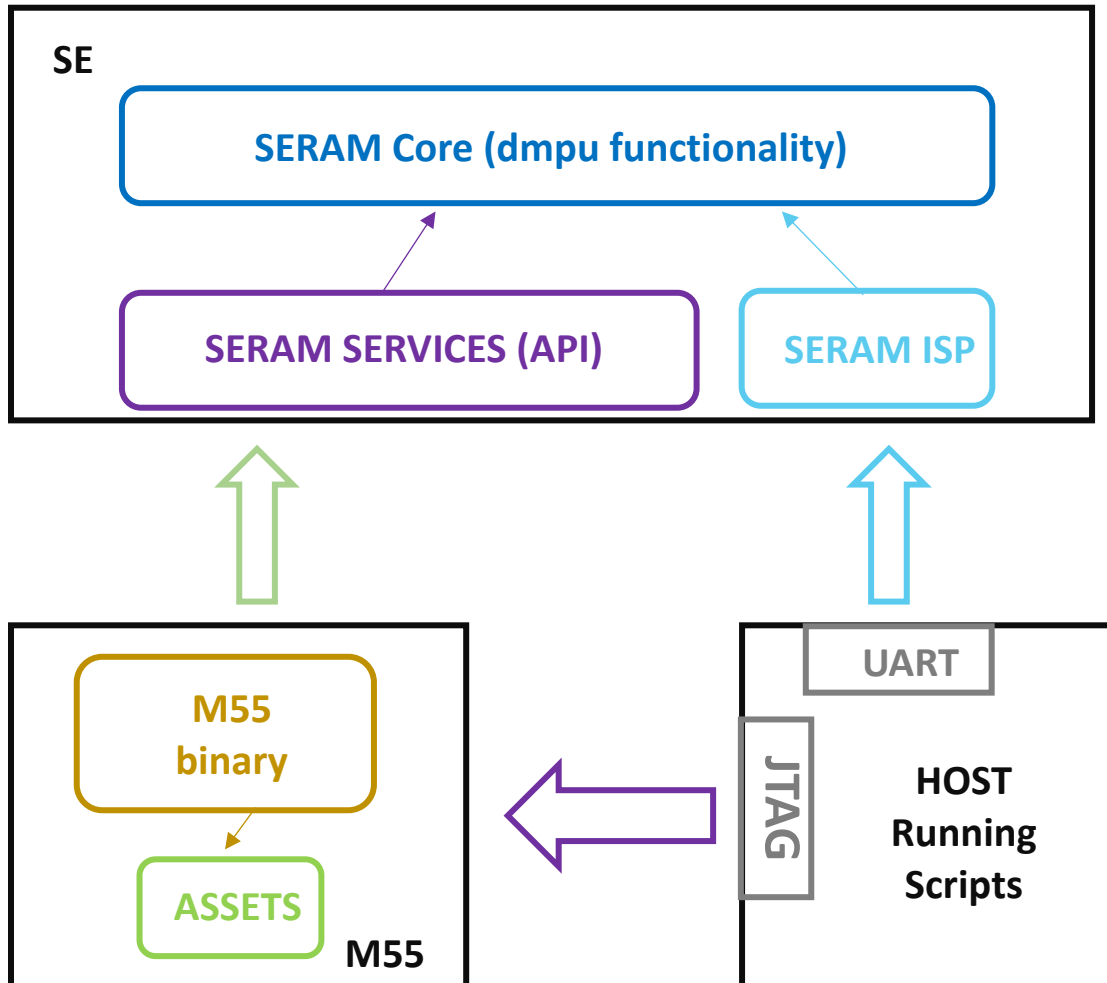
Provisioning for production usually entails that the SOC is transitioned to Life Cycle State SERCURE (LCS=SE) and other customer related data is programmed into the device e.g. customer application code is written into the NVM of the device.

In development, these steps can be achieved using the SETOOLS and SEUART to enable LCS state transition. In a production environment the use of SEUART is probably not preferred. This is the reason for having a JTAG-based provisioning method.

An executable binary, containing a call to a SERVICE routine to enable the state transition, is loaded to the target device and executed. The customer assets package is also loaded into the memory of the device, and the executable binary will use this data to send to the Secure Enclave to enable the LCS transition.

Development and Production Flow for provisioning

The following shows the flow and structure of the Provisioning process:



Overview of JTAG Provisioning

The M55_HE is loaded with two sets of data

- M55_HE executable to run the SERVICES command
- The customer assets package binary

These binaries are loaded into the TCM memory of the M55_HE.

Please refer to the SETOOLS User Guide for details on generating the assets package.

Using JTAG Provisioning

This consists of a “main.c” file (See <services release>/examples/factory_provision/main.c). The example can be built using the CMakefile provided in the examples directory.

```
$ cd <services release>
$ cmake .. -G "Unix Makefiles" -DCMAKE_BUILD_TYPE=debug -
DCMAKE_TOOLCHAIN_FILE=toolchain-gnu.cmake -DENSEMBLE_CORE=M55_HE -
DEXAMPLE=FACTORY_PROVISION
$ make install
```

In the factory_provision/bin directory you will find the binary and executable file to load to the target device using JTAG.

There is an example SEGGER J-LINK script to load the assets and the executable to the M55_HE device.

SE Host Services Library API

The Host Service API is built on a transport protocol layer. This is to facilitate changing the underlying protocol without affecting the rest of the library.

There is a porting / abstraction interface component which the user must update depending upon their operating system choice and driver interface to the Message Handling hardware (MHU).

ALIF supply completed interfaces (currently) for

- Bare metal
- FreeRTOS
- ThreadX
- Linux

The Host services library provides APIs to facilitate service requests from a host CPU to the SE. It must be set up and initialized before dispatching a Host service request to the SE. It needs access to the MHU driver functions to facilitate MHU communication.

The SERVICES library is provided in source code format with the intention to be linked into an application programme.

The Host services library also requires other generic functions:

<code>SERVICES_wait_ms(uint32_t wait_time_ms)</code>	Delay function
<code>SERVICES_send_mhu_message_to_se(uint32_t message)</code>	Interface to the MHU driver

This layer is intended for any Operating System abstraction.

Host Services Library Interface API Porting Layer

This needs to be updated by the user depending upon the operating system being used (or base metal) and the interface to the Message handling hardware. The requirements of the operating system are very light.

The file `services_lib_interface.c` is the porting interface which needs to be filled in by the user.

SERVICES_wait_ms

```
// Delay function
int wait_ms(uint32_t wait_time_ms)
```

SERVICES_send_mhu_message_to_se

```
// MHU send message to SE on MHU0 channel0
int send_mhu_message_to_se(uint32_t message)
```

The above functions must be configured in `services_init_params` structure and pass to the service library initialization function below.

Host Services Library API Layer

A Service call from an application processor looks like any other C function call, it can take parameters and return results via pass by reference parameters.

The Host Services library is responsible for taking the application Service call and communicating this to the Secure Enclave using the MHU.

SERVICES_initialize

The SERVICES library needs to be initialized before use. There are several parameters that are needed by the SERVICES library such as which MHU is being used, packet buffers etc.

Please refer to the SE Host Services API section for more details.

```
// Service library initialization
void SERVICES_initialize(services_lib_t *init_params)

SERVICES_initialize(services_lib_t *init_params);

// Service synchronization
int SERVICES_synchronize_with_se(uint32_t services_handle)

number_of_retries = SERVICES_synchronize_with_se(services_handle);
```

The M55-HE and M55-HP are started before SERAM is ready to process service calls. This function sends heartbeat requests until one of them succeeds. It returns the number of retries. The maximum number of retries is 100.

SERVICES_send_request

```
// Service request call
uint32_t SERVICES_send_request(uint32_t services_handle,
                               uint16_t service_id,
                               uint32_t service_timeout);

Error_code = SERVICES_send_request(handle, SERVICE_HEARTBEAT_ID,
DEFAULT_TIMEOUT);
```

The service request dispatches the service request to the SE. Service calls are synchronous, the function waits for the SE to send a response back and then returns an error code. This is analogous to a remote procedure call. The caller can specify the desired timeout for the service call, or specify `DEFAULT_TIMEOUT`, in which case the timeout specified when during Services library setup is used.

It needs access to the host CPUs MHU driver functions to send, receive and ACK messages over the MHU.

SERVICES_send_msg_acked_callback

```
// MHU message ACK callback function
void SERVICES_send_msg_acked_callback(void)
```

The above callback function must be passed to the MHU driver during initialization. It is called by the driver when an MHU message is ACKed by the SE. Channel clear interrupt CH_INT_ST is set when SE has received the MHU message and SE clears the channel status CH_ST bits by setting CH_CLR. This is assumed to be an ACK from SE that it has received an MHU message sent by the host CPU.

SERVICES_rx_msg_callback

```
// MHU message received callback function
void SERVICES_rx_msg_callback(uint32_t message);
```

The above callback function must be passed to the MHU driver during initialization. It is called by the driver when an MHU message is received from the SE as a response to a service request earlier to the SE

```
// Pinmux service
int PINMUX_config(Port_t port_num, Pin_t pin_num, Pinfunction_t
function);
```

SERVICES Based Provisioning

Provisioning of a device involves writing the Customer defined key into the device and advancing the Life Cycle State (LCS) of the device to SE (Secure Enable) Mode.

In development a user would use the ISP (SEUART) method to perform this operation (Please refer to the SETOOLS User Guide for more details of this process). For manufacturing using the SEUART may not be the preferred method, the more traditional way is via JTAG. As with the ISP method the user will generate an Assets Package that will be sent via the SEUART to be processed.

JTAG provisioning uses the same assets package but instead of ISP it uses a SERVICE call to perform the OTP and LCS state transition. The SERVICE call is **SERVICES_application_dmpu**.

An M55 program containing this SERVICE call is required and will be loaded via JTAG. Also loaded via JTAG will be the Assets package which is loaded into memory. An example flow would be

C Code M55

```
#define UPDATE_PACKAGE_ADDRESS 0x02000000
error_code = SERVICES_application_dmpu(services_handle,
                                       UPDATE_PACKAGE_ADDRESS,
                                       &service_error_code);
```

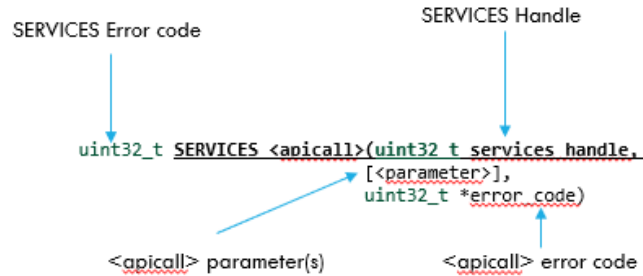
A JLINK script would have to execute

```
$ loadfile assets-app-cfg.bin <address>
$ loadfile dmpu_service.bin 0x0
$ go
```

SE Host SERVICES Library

SE Host SERVICES Library - Anatomy of a SERVICE Call

A SERVICES call takes the following format:



- SERVICES API are regular function calls taking the format SERVICES_<module>
- Returns Services error code.
 - This relates to the SERVICES transport layer.
- Other values returned are passed in the function prototype.
 - These parameters can be IN and OUT and can be variable sized.
 - Results from the SERVICE call are returned via these variables.
 - The error code return relates to the error returned from the actual SERVICE call.

SE Host Service Library Internal implementation

Each SERVICE defines a unique parameter block structure.

- See `example_service_t` in the diagram below.
- This always contains the Header and a return response error code,
- There may be passed parameters from the Caller.
- There may be return parameters to the Caller.

For each SERVICE call processed in SERAM the parameter block is dereferenced

- Sent parameters can be passed to the calling function.
- The Error code from the called function will be sent back as part of the parameter block.

```

typedef struct
{
    service_header_t header;
    volatile uint32_t send_<param>;    /*!< Send parameter */
    volatile uint32_t resp_<param>;    /*!< Return parameter */
    volatile uint32_t resp_error_code; /*!< Call error code */
} example_service_t;

void SERVICES example_call(services_req_t *service)
{
    example_service_t *p_svc =
        (example_service_t *)service->pkt buffer address; /* services request */
    uint32_t error_code;
    uint32_t local result;

    error_code = function call(p_svc->send_<param>, &local result);

    p_svc->resp_<param> = local result;
    p_svc->resp_error_code = error_code;

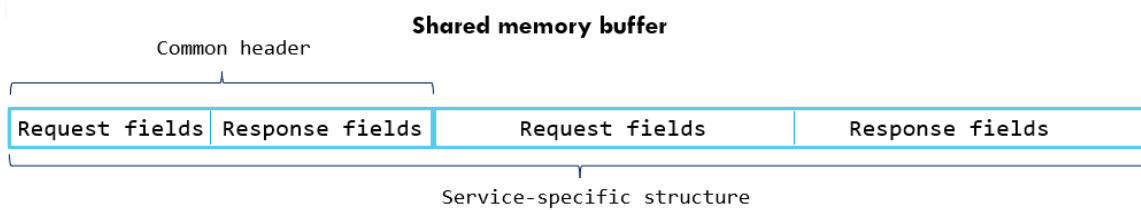
    SERVICES_send_response_code(service, SERVICES_REQ_SUCCESS);
}

```

There is no copying of data between the Host Application CPU and the Secure enclave.

SE Host Service Library Transport Protocol details

The transport protocol is as follows:



Common header format

Service ID: req
Flags: req
Error Code: resp
Padding

```

typedef struct
{
    uint16_t send_sid;
    uint16_t send_flags;
    uint16_t resp_error_code; // transport
    layer_error_code
    uint16_t none_padding;
} service_header_t;

```

Service-specific structure example

```

typedef struct
{
    service_header_t header;
    uint8_t send_port_num;
    uint8_t send_pin_num;
    uint8_t send_config_data;
    uint8_t resp_error_code; // service-specific
    error_code
} pinmux_svc_t;

```

SE Host Service Library Transport Error Codes

The following are the valid return and Error codes for the services library.

Error Code	Value	Meaning
SERVICES_REQ_SUCCESS	0x00	
SERVICES_REQ_NOT_ACKNOWLEDGE	0xFF	
SERVICES_REQ_TIMEOUT	0xFD	
SERVICES_RESP_UNKNOWN_COMMAND	0xFC	
SERVICES_RESP_INVALID_PARAMETER	0xFB	

This error relates to any operation on the transport layer of the SERVICES library. Most SERVICES library APIs have a second error code which is the error code return from the called function.

SE Host Services Library Error Handling

There are two levels of Error with a SERVICES API call,

- SERVICES Transport layer error code
- Function call error code

Valid SERVICE Error and return codes are:

```
#define SERVICES_REQ_SUCCESS           0x00
#define SERVICES_REQ_NOT_ACKNOWLEDGE  0xFF
#define SERVICES_REQ_TIMEOUT          0xFD
#define SERVICES_RESP_UNKNOWN_COMMAND 0xFC
#define SERVICE_SUCCESS                0x0
#define SERVICE_FAIL                   0x200
#define SERVICE_INVALID_PARAMETER     0x201
```

SE Host Services Library Memory handling

When passing data from the Application CPUs, the SERVICES library will perform all necessary address translations and handling.

LocalToGlobal() is used inside the SERVICE Library to pass the correct addresses to the SECURE Enclave.

SE Host Services API

The services provided by the SE via the MHU are as follows.

Miscellaneous

SERVICES_Initialize

Syntax:

```
uint32_t SERVICES_initialize(services_lib_t * init_params)
```

Description:

Initialize the services library.

A user needs to supply the following platform specific data and functions for the following operations.

- Global address of the CPU's local data memory 0x0 for A32, start of DTCMs for the M55 cores.
- Packet buffer
 - Defined in Application memory space.
 - Used by the SERVICES library.
- Send MHU message function – provided by the MHU driver.
- wait (delay) function – platform and OS specific.
- print function – platform and OS specific.

The examples source contains `service_lib_interface.c` which shows how to set up the SERVICES library. This is not part of the SERVICES Library code as it is expected to be customized by a User for their application, which is why this is included as source code in the examples.

Parameters:

`init_params` Initialization parameters

Returns:

Restrictions:

None

Example:

```
#include "services_lib_api.h" /* services_lib_t lives here */

static uint8_t
    s_packet_buffer[SERVICES_MAX_PACKET_BUFFER_SIZE] __attribute__((aligned (4)));

int SERVICES_print(const char * fmt, ...)
{
    /* To be filled in by the user */

    return 0;
}
```

```
int32_t SERVICES_wait_ms(uint32_t wait_time_ms)
{
    /* To be filled in by the user */

    return 0;
}

int main (void)
{
    uint32_t ErrorCode = SERVICES_OK;

    services_lib_t services_init_params =
    {
        .global_offset          = DTCM_GLOBAL_ADDRESS - M55_DTCM_LOCAL_OFFSET,
        .packet_buffer_address = (uint32_t)s_packet_buffer,
        .fn_send_mhu_message    = send_message,
        .fn_wait_ms             = &SERVICES_wait_ms,
        .wait_timeout           = timeout,
        .fn_print_msg           = &SERVICES_print,
    };

    ErrorCode = SERVICES_initialize(&services_init_params);

    if (ErrorCode != SERVICES_REQ_SUCCESS)
    {
        return ErrorCode;
    }
}
```

SERVICES_version

Syntax:

```
const char *SERVICES_version(void)
```

Description:

Returns the version of the Host library.

Parameters:

None

Returns:

Version string

Restrictions:

None

Example:

```
#include <services_lib_api.h>

int main (void)
{
    uint32_t ErrorCode = SERVICES_OK;

    printf("SERVICES version %s\n", SERVICES_version());
}
```

SERVICES_register_channel

Syntax:

```
uint32_t SERVICES_register_channel(uint32_t mhu_id, uint32_t channel_number);
```

Description:

Returns a handle for a specific MHU and channel, to be used in subsequent service calls.

Parameters:

mhu_id	MHU ID
channel_number	Channel number (within the MHU)

Returns:

Service channel handle

Restrictions:

The MHU ID and channel number must be valid.

The maximum number of MHU Channels is 124.

Example:

```
#include <services_lib_api.h>

#define NUM_MHU                2

#define MHU_M55_SE_MHU0       0
#define MHU_M55_SE_MHU1       1

int main (void)
{
    mhu_initialize();
    SERVICES_Setup(s_mhu_driver_out.send_message, MAXIMUM_TIMEOUT);

    uint32_t services_handle = SERVICES_register_channel(MHU_M55_SE_MHU0, 0);

    printf("SERVICES handle %d\n", services_handle);
}
```

SERVICES_prepare_packet_buffer

Syntax:

```
uintptr_t SERVICES_prepare_packet_buffer(uint32_t size)
```

Description:

Prepares a packet buffer.

Used by the SERVICES library to allocate a packet buffer from the global Packet buffer memory.

Parameters:

Size	Packet buffer size
------	--------------------

Returns:

Pointer to packet buffer.

Restrictions:**Example:**

Maintenance Services

The maintenance services provide a mechanism to maintain a reliable connection between the sender and receiver and/or request general information from the receiver. The following maintenance services are supported by SE.

SERVICES_heartbeat

Syntax:

```
uint32_t SERVICES_heartbeat (uint32_t services_handle)
```

Description:

Heartbeat request.

This service is analogous to “ping”.

It is a message sent by the sender to tell the receiver that it is alive. It can also be sent by SE to check if another core is alive and responding. When this message is ACKed by the receiver, the sender knows that the receiver is alive. This message does not warrant a response from the receiver other than ACK.

Parameters:

services_handle

Returns:**Restrictions:**

None

Example:

```
int main (void)
{
    uint32_t ErrorCode = SERVICES_OK;

    mhu_initialize();
    SERVICES_Setup(s_mhu_driver_out.send_message, MAXIMUM_TIMEOUT);

    //SERVICES wait ms(0x1000000);

    uint32_t services_handle = SERVICES_register_channel(MHU_M55_SE_MHU0, 0);

    ErrorCode = SERVICES_heartbeat(services_handle);
    if (ErrorCode != SERVICES_REQ_SUCCESS)
    {
        return ErrorCode;
    }

    return ErrorCode;
}
```

SERVICES_synchronize_with_se

Syntax:

```
int SERVICES_synchronize_with_se(uint32_t services_handle)
```

Description:

Wait for the SE to become available. In some warm boot cases the M55_HE is released by SES and so is running, this means the M55 is running before SES has fully completed booting and installing the MHU handlers. In this case the M55_HE needs to synchronize (wait) for SES to be ready.

This function is built on top of SERVICES_heartbeat. A heartbeat call is made (for a maximum of 100 times) until SES responds. If SES responds this function will return an integer number which is the number of retries it made until it got a response. If SES does not respond then a negative number will be returned indicating this call failed.

Parameters:

services_handle

Returns:

Restrictions:

None

Example:

```
int main (void)
{
    int retry_count;

    /* keep sending heartbeat services requests until one succeeds */
    retry_count = SERVICES_synchronize_with_se(services_handle);
}
```

System Management

SERVICES_system_set_services_debug

Syntax:

```
uint32_t SERVICES_system_set_services_debug (uint32_t services_handle,  
                                              bool debug_enable,  
                                              uint32_t *error_code)
```

Description:

Enable / Disable Service debug traffic from SES.

Parameters:

service_handle	Service Handle
debug_enable	Toggle debug output.
error_code	Service Error Code

Returns:**Restrictions:**

None

Example:

```
int main (void)  
{  
    uint32_t service_error_code;  
  
    SERVICES_system_set_services_debug(services_handle,  
                                       false, /* False = NO debug output */  
                                       &service_error_code);  
  
    if (service_error_code != SERVICES_REQ_SUCCESS)  
    {  
        /* Deal with error */  
    }  
}
```

SERVICES_system_read_otp

Syntax:

```
uint32_t SERVICES_system_read_otp (uint32_t services_handle,  
                                   uint32_t otp_offset,  
                                   uint32_t *otp_value_word,  
                                   uint32_t *error_code)
```

Description:

Read an OTP word specified by offset.

Valid OTP word offsets are from 0x70 to 0x7F, i.e., the unformatted customer area of OTP.

Parameters:

service_handle	Service Handle
otp_offset	OTP word offset to read.
otp_value_word	OTP value at otp_offset
error_code	Service Error Code

Returns:

SERVICE_SUCCESS or SERVICE_FAIL

Restrictions:**Example:**

```
int main (void)  
{  
    uint32_t ErrorCode = SERVICES_REQ_SUCCESS  
    uint32_t service_error_code;  
    uint32_t otp_value;  
  
    ErrorCode = SERVICES_system_read_otp(services_handle,  
                                        0x70, /* Offset */  
                                        &otp_value,  
                                        &service_error_code);  
  
    if (ErrorCode != SERVICES_REQ_SUCCESS)  
    {  
        return ErrorCode;  
    }  
}
```

SERVICES_system_write_otp

Syntax:

```
uint32_t SERVICES_system_write_otp (uint32_t services_handle,  
uint32_t otp_offset,  
uint32_t otp_value_word,  
uint32_t *error_code)
```

Description:

Write an OTP word specified by offset.

Valid OTP word offsets are from 0x70 to 0x7F, i.e., the unformatted customer area of OTP. Writes to OTP are only allowed in DM LCS, they are not allowed in SE LCS.

Parameters:

service_handle	Service Handle
otp_offset	OTP word offset to write.
otp_value_word	OTP value to write to otp_offset
error_code	Service Error Code

Returns:

SERVICE_SUCCESS or SERVICE_FAIL

SERVICES_system_get_otp_data

Syntax:

```
uint32_t SERVICES_system_get_otp_data (uint32_t services_handle,  
SERVICES_otp_data_t *otp_info,  
uint32_t * error_code)
```

Description:

Returns details of OTP data

Parameters:

service_handle	Service Handle
otp_info	Details of OTP contents
error_code	Service Error Code

Returns:

Restrictions: This function is deprecated. SES does not process this SERVICE command.

Example:

```
int main (void)
{
  uint32_t ErrorCode = SERVICES_OK;
  uint32_t service_error_code;
  SERVICES_otp_data_t otp_info;

  ErrorCode = SERVICES_system_get_otp_data(services_handle,
                                          &otp_info,
                                          &service_error_code);

  if (ErrorCode != SERVICES_REQ_SUCCESS)
  {
    return ErrorCode;
  }
}
```

SERVICES_system_get_toc_data

Syntax:

```
uint32_t SERVICES_system_get_toc_data (uint32_t services_handle,
                                       SERVICES_toc_data_t *toc_info,
                                       uint32_t * error_code)
```

Description:

Returns details of TOC objects in MRAM.

```
typedef struct {
  uint8_t image_identifier[TOC_NAME_LENGTH]; /*!< TOC name */
  uint32_t version; /*!< TOC Version */
  uint32_t cpu; /*!< TOC Cpu ID */
  uint32_t store_address; /*!< TOC MRAM address */
  uint32_t load_address; /*!< TOC load */
  uint32_t boot_address; /*!< TOC boot address */
  uint32_t image_size; /*!< TOC image size */
  uint32_t processing_time; /*!< TOC process time */
  uint32_t flags; /*!< TOC flag state */
  uint8_t flags_string[FLAG_STRING_SIZE]; /*!< TOC flag string */
} SERVICES_toc_info_t;
```

```
/**
 * @struct SERVICES_toc_data_t
 */
typedef struct
{
  uint32_t number_of_toc_entries;
```

```
SERVICES_toc_info_t toc_entry[SERVICES_NUMBER_OF_TOC_ENTRIES];
} SERVICES_toc_data_t;
```

The number of TOC entries found is returned followed by the TOC entry details. SERVICES_NUMBER_OF_TOC_ENTRIES is set to 15 entries.

Parameters:

service_handle	Service Handle
toc_info	Details for all TOCs found
error_code	Service Error Code

Returns:

Restrictions:

None

Example:

```
int main (void)
{
    uint32_t ErrorCode = SERVICES_OK;
    uint32_t service_error_code;
    SERVICES_toc_data_t toc_info;

    ErrorCode = SERVICES_system_get_toc_data(services_handle,
                                             &toc_info,
                                             &service_error_code);

    if (ErrorCode != SERVICES_REQ_SUCCESS)
    {
        return ErrorCode;
    }
}
```

An example output might be as follows (this is taken from the Examples)

```
[TTY] ** TEST TOC get data error_code=SERVICES_REQ_SUCCESS TOC number = 5 service_resp=0x00000000
[TTY] +-----+
[TTY] | Name | CPU | Load Address|Boot Address|Image Size| Version | Flags |
[TTY] +-----+
[TTY] | DEVICE | CM0+ | 0x00000000 | 0x00000000 | 296 | 0.5.0 | u V |
[TTY] | DEVICE | CM0+ | 0x00000000 | 0x00000000 | 372 | 0.5.0 | u V |
[TTY] | SERAM0 | CM0+ | 0x00000000 | 0x00000000 | 57612 | 1.105.0 | u s |
[TTY] | SERAM1 | CM0+ | 0x00000000 | 0x00000000 | 57612 | 1.105.0 | u s |
[TTY] | SRV-HE-T | M55_HE | 0x58000000 | 0x58000000 | 73440 | 1.0.0 | uLVB |
[TTY] +-----+
```

The load and Boot address showing 0x00000000 are Secure Enclave addresses (not the real ones). M55_HE in this example shows the load and boot address used by this application CPU.

SERVICES_system_get_toc_number

Syntax:

```
uint32_t SERVICES_system_get_toc_number(uint32_t services_handle,  
                                        uint32_t *toc_number,  
                                        uint32_t * error_code)
```

Description:

Returns the number of Table of contents entries in MRAM

Parameters:

service_handle	Service Handle
toc_number	Number of TOCs
error_code	Service Error Code

Returns:

Restrictions:

None

Example:

```
int main (void)  
{  
    uint32_t ErrorCode = SERVICES_OK;  
    uint32_t number_of_tocs;  
    uint32_t service_error_code;  
  
    ErrorCode = SERVICES_system_get_toc_number(services_handle,  
                                              &number_of_tocs,  
                                              &service_error_code);  
    if (ErrorCode != SERVICES_REQ_SUCCESS)  
    {  
        return ErrorCode;  
    }  
}
```

SERVICES_system_get_toc_version

Syntax:

```
uint32_t SERVICES_system_get_toc_version (uint32_t services_handle,  
                                           uint32_t *toc_version,  
                                           uint32_t *error_code)
```

Description:

Returns the SES Software version information.

The version can be unpacked as follows

```
/* Unpack the SE version */  
major = (version >> 24) & 0xFF;  
minor = (version >> 16) & 0xFF;  
patch = (version >> 8) & 0xFF;
```

Parameters:

service_handle	Service Handle
toc_version	version
error_code	Service Error Code

Returns:

SERVICES_SUCCESS
toc_version

Restrictions:

None

SERVICES_system_get_toc_via_name

Syntax:

```
uint32_t SERVICES_system_get_toc_via_name(uint32_t services_handle,  
                                          const uint8_t *cpu_name,  
                                          uint32_t * error_code);
```

Description:

Returns the TOC information using the CPU name.

Parameters:

service_handle	Service Handle
cpu_name	name of Application
error_code	Service Error Code

Returns:**Restrictions:**

This function is deprecated.

Use SERVICES_system_get_toc_data() to obtain all details of the TOC data or SERVICES_system_get_toc_via_cpuid() specifying the CPU ID instead of a name.

Example:

SERVICES_system_get_toc_via_cpuid

Syntax:

```
uint32_t SERVICES_system_get_toc_via_cpuid(uint32_t services_handle,
                                           SERVICE_cpuid_t cpuid,
                                           SERVICES_toc_data_t *toc_info,
                                           uint32_t * error_code);
```

Description:

Returns the TOC information for a given CPU.

Valid CPUs are

```
typedef enum {
    HOST_CPU_0 = 0,           /**< A32_0 CPU */
    HOST_CPU_1 = 1,           /**< A32_1 CPU */
    EXTSYS_0 = 2,             /**< M55 HP CPU or other CPU */
    EXTSYS_1 = 3,             /**< M55 HE CPU */
} SERVICES_cpuid_t;
```

If there is more than one TOC entry per CPUID this will be reflected in the toc_info structure returned from the SERVICE call.

Parameters:

service_handle	Service Handle
cpuid	Which Application CPU
toc_info	ATOC information
error_code	Service Error Code

Returns:

Restrictions:

Example:

```
int main (void)
{
    uint32_t ErrorCode = SERVICES_OK;
    SERVICES_toc_data_t toc_info;
    Uint32_t service_error_code;

    error_code = SERVICES_system_get_toc_via_cpuid(services_handle,
                                                  FUSION_M55_HE,
                                                  &toc_info,
                                                  &service_error_code);

    if (ErrorCode != SERVICES_REQ_SUCCESS)
    {
        return ErrorCode;
    }
}
```

```
}  
  
/* Process each TOC entry found */  
for (int each_toc = 0; each_toc < toc_info.number_of_toc_entries; each_toc++)  
{  
    SERVICES_toc_info_t *toc_entry_p;  
  
    toc_entry_p = (SERVICES_toc_info_t *)&toc_info.toc_entry[each_toc];  
  
    /* do something with the TOC information */  
}
```

SERVICES_system_get_device_part_number

Syntax:

```
uint32_t SERVICES_system_get_device_part_number(uint32_t services_handle,  
                                                uint32_t *device_part_number,  
                                                uint32_t *error_code)
```

Description:

Returns the SoC device identifier.

Parameters:

service_handle	Service Handle
device_part_number	Device id (Soc ID)
error_code	Service Error Code

Returns:

device_part_number as integer e.g., 0x0000B200

Restrictions:

None

Example:

```
int main (void)  
{  
    uint32_t ErrorCode = SERVICES_OK;  
    uint32_t device_id;  
    uint32_t service_error_code;  
  
    ErrorCode = SERVICES_system_get_device_part_number(services_handle,  
                                                       &device_part_number,  
                                                       &service_error_code);  
  
    if (ErrorCode != SERVICES_REQ_SUCCESS)  
    {  
        return ErrorCode;  
    }  
}
```

SERVICES_system_get_device_data

Syntax:

```
uint32_t SERVICES_system_get_device_data(uint32_t services_handle,
                                         SERVICES_version_data_t *device_info,
                                         uint32_t *error_code)
```

Description:

Retrieves the Device information.

The return is SERVICES_version_data_t as follows:

```
typedef struct {
    uint32_t revision_id; /*!< SoC revision */
    uint8_t version[4]; /*!< @todo deprecate */
    uint8_t ALIF_PN[16]; /*!< SoC part number */
    uint8_t HBK0[16]; /*!< ALIF Key */
    uint8_t HBK1[16]; /*!< ALIF Key */
    uint8_t HBK_FW[20]; /*!< ALIF Firmware version */
    uint8_t config[4]; /*!< Wounding data */
    uint8_t DCU[16]; /*!< DCU settings */
    uint8_t MfgData[32]; /*!< Manufacturing data */
    uint8_t SerialN[8]; /*!< SoC Serial number */
    uint8_t LCS; /*!< SoC lifecycle state */
    uint32_t external_config[4]; /*!< External mem */
    uint32_t flags2; /*!< Alt path options */
} SERVICES_version_data_t;
```

Parameters:

service_handle	Service Handle
device_info	Device info
error_code	Service Error Code

Returns:

Restrictions:

external_config[4], flags2 are valid for devices that can support Alternative Boot path. For devices that do not have this feature these fields will be '0'.

Example:

```
int main (void)
{
    uint32_t ErrorCode = SERVICES_OK;
    uint32_t device_id;
    SERVICES_version_data_t device_data;
    uint32_t service_error_code;

    ErrorCode = SERVICES_system_get_device_info(services_handle,
```

```
if (ErrorCode != SERVICES_REQ_SUCCESS)
{
    return ErrorCode;
}
}

&device_data,
&service_error_code);
```

SERVICES_system_get_eui_extension

Syntax:

```
uint32_t SERVICES_system_get_eui_extension(uint32_t services_handle,  
                                             bool is_eui48,  
                                             uint8_t *eui_extension,  
                                             uint32_t *error_code)
```

Description:

Retrieve a device-unique EUI48/64 extension value. The value is derived from the device's manufacturing data, which is 40 bits in size.

For EUI64 (`is_eui48 = false`), the returned data is 5 bytes (40 bits) and contains all 40 bits of Mfg data.

For EUI48 (`is_eui48 = true`), the returned data is 3 bytes (24 bits), i.e., some of the Mfg data bits are ignored.

Parameters:

<code>service_handle</code>	Service Handle
<code>is_eui48</code>	Which format?
<code>eui_extension</code>	Buffer for calculated extension
<code>error_code</code>	Service Error Code

SERVICES_system_get_device_id64

Syntax:

```
uint32_t SERVICES_system_get_device_id64(uint32_t services_handle,  
                                             uint8_t *device_id,  
                                             uint32_t *error_code)
```

Description:

Retrieve a 64-bit device ID that is based on the device Mfg data and the Alif Organization ID. This service combines the EUI64 extension of the device (5 bytes) and the 3-byte Alif manufacturer ID.

Parameters:

<code>service_handle</code>	Service Handle
<code>device_id</code>	Buffer for the device ID. Must be at least 8 bytes in size
<code>error_code</code>	Service Error Code

SERVICES_system_get_ecc_public_key

Syntax:

```
uint32_t SERVICES_system_get_ecc_public_key(uint32_t services_handle,  
                                             uint8_t *ecc_pubkey_buffer,  
                                             uint32_t *error_code)
```

Description:

Returns the device's unique ECC public key.

Parameters:

service_handle	Service Handle
ecc_pubkey_buffer	Buffer for the ECC public key. Must be at least 64 bytes in size
error_code	Service Error Code

SERVICES_get_se_revision

Syntax:

```
uint32_t SERVICES_get_se_revision(uint32_t services_handle,  
                                    uint8_t *revision_data,  
                                    uint32_t *error_code)
```

Description:

Retrieve the SES Banner string.

Parameters:

service_handle	Service Handle
revision_data	banner string return
error_code	Service Error Code

Returns:

String containing the banner data. Maximum size is 80 characters.

Restrictions:

None

Example:

```
int main (void)
{
    uint32_t error_code = SERVICES_REQ_SUCCESS;
    uint32_t service_error_code;
    uint8_t se_revision[80];

    error_code = SERVICES_get_se_revision(services_handle,
                                         (uint8_t*)&se_revision[0],
                                         &services_error_code);

    if (error_code != SERVICES_REQ_SUCCESS)
    {
        /* deal with error */
    }
}
```

Application Services

Application services provide mechanisms to configure certain functions. The SE can be requested to make these configuration changes.

SERVICES_uart_write

Syntax:

```
uint32_t SERVICES_uart_write(uint32_t services_handle, size_t size, const uint8_t *uart_data)
```

Description:

SE-UART write. The buffer provided is printed via the Secure enclave UART (SE-UART) port.

Parameters:

services_handle	Service handle
size	Number of bytes to write
uart_data	Buffer containing print data

None**Returns:****Restrictions:**

None

Example:

```
int main (void)
{
    uint32_t ErrorCode = SERVICES_OK;
    uint8_t buffer[256];

    ... <format print buffer>

    ErrorCode = SERVICES_uart_write(services_handle,
                                    sizeof(buffer),
                                    (uint8_t *)buffer);
    if (ErrorCode != SERVICES_REQ_SUCCESS)
    {
        return ErrorCode;
    }
}
```

SERVICES_pinmux

Syntax:

```
uint32_t SERVICES_pinmux(uint32_t services_handle, uint8_t port_number,  
                          uint8_t pin_number, uint8_t configuration_value,  
                          uint32_t * error_core)
```

Description:

Pinmux request

Parameters:

services_handle	
port_number	Port Number
pin_number	Pin Number
configuration_value	?
error_code	Service Error Code

Returns:**Restrictions:**

None

Example:

```
int main (void)  
{  
    uint32_t ErrorCode = SERVICES_OK;  
    uint32_t service_error_code;  
  
    ErrorCode = SERVICES_pinmux(services_handle, 1, 14, 0, &service_error_code);  
    if (ErrorCode != SERVICES_REQ_SUCCESS)  
    {  
        return ErrorCode;  
    }  
}
```

SERVICES_padcontrol

NOTE: Refer to document [se-mhu-pinmux-pad_configuration](#)

Syntax:

```
uint32_t SERVICES_padcontrol(uint32_t services_handle, uint8_t port_number,  
                             uint8_t pin_number, uint8_t configuration_value,  
                             uint32_t * error_core)
```

Description:

Pad control request.

Parameters:

services_handle	
port_number	Port Number
pin_number	Pin Number
configuration_value	?
error_code	Service Error Code

Returns:**Restrictions:**

None

Example:

```
int main (void)  
{  
    uint32_t ErrorCode = SERVICES_OK;  
    uint32_t service_error_code;  
  
    ErrorCode = SERVICES_padcontrol(services_handle, 1, 14, 0, &service_error_code);  
    if (ErrorCode != SERVICES_REQ_SUCCESS)  
    {  
        return ErrorCode;  
    }  
}
```

[SERVICES_application_ospi_write_key](#)

Syntax:

```
uint32_t SERVICES_application_ospi_write_key(uint32_t services_handle, uint32_t command, uint8_t  
*key, uint32_t * error_code)
```

Description:

Write an AES decryption key to the OSPI registers. The command field indicates whether to use an externally provided key or a key stored in the OTP, and which OSPI to apply it to – OSPI0 or OSPI1.

```
#define OSPI_WRITE_OTP_KEY_OSPI0          0
#define OSPI_WRITE_OTP_KEY_OSPI1          1
#define OSPI_WRITE_EXTERNAL_KEY_OSPI0     2
#define OSPI_WRITE_EXTERNAL_KEY_OSPI1     3
```

Parameters:

services_handle	Service handle
command	Indicates OSPI0/OSPI1 and external/OTP key
key	Buffer containing print data
error_code	Service error code

Returns:**Restrictions:**

None

Example:

[SERVICES_application_verify_image](#)

Syntax:

```
uint32_t SERVICES_application_verify_image(uint32_t services_handle, uint32_t image_address,
uint32_t cert_chain_address, uint32_t * error_code)
```

Description:

Verify an image signed with an Arm certificate chain (Key1-Key2-Content). The image and the certificate chain can be located anywhere in the address space. The content certificate in the chain must use the 'verify in flash' verification scheme.

Parameters:

services_handle	Service handle
image_address	Image address
cert_chain_address	Certificate chain address
error_code	Service error code

Returns:**Restrictions:**



None

SERVICES_application_dmpu

Syntax:

```
uint32_t SERVICES_application_dmpu(uint32_t services_handle,  
    uint32_t assets_address,  
    uint32_t *error_code);
```

Description:

IMPORTANT: This is an internal service not intended to be called from user application code!

This service is called from a factory provisioning tool to provision the OTP of an SoC for transitioning to the Secure Lifecycle State.

The assets package contains various data to be programmed into the OTP (keys, etc.).

Parameters:

services_handle	Service handle
assets_address	Address of the assets package to be processed by the service
error_code	Service error code

Returns:

Restrictions:

None

Power Services

Please see other documentation on how the Ensemble series implements Power modes. The following is details regarding the SERVICES APIs not a deep dive in how they are implemented.

SERVICES_power_stop_mode_request

Syntax:

```
uint32_t SERVICES_power_stop_mode_request(uint32_t services_handle)
```

Description:

Request the Secure Enclave to enter stop mode.

Parameters:

services_handle

Returns:

ErrorCode - SERVICES_REQ_SUCCESS, SERVICES_REQ_CANNOT_EXECUTE_SERVICE

Restrictions:

Example:

```
int main (void)
{
    uint32_t ErrorCode = SERVICES_OK;

    error_code = SERVICES_power_stop_mode_request(services_handle);
    if (ErrorCode != SERVICES_REQ_SUCCESS)
    {
        return ErrorCode;
    }
}
```

SERVICES_power_ewic_config

Syntax:

```
uint32_t SERVICES_power_ewic_config(uint32_t services_handle,  
                                     uint32_t ewic_source);
```

Description:

Configure the EWIC

Parameters:

services_handle

ewic_source EWIC source

Returns:

ErrorCode - SERVICES_REQ_SUCCESS, SERVICES_REQ_CANNOT_EXECUTE_SERVICE

Restrictions:**Example:**

```
int main (void)  
{  
    uint32_t error_code = SERVICES_REQ_SUCCESS;  
    uint32_t ewic_config;  
  
    ewic_config &= (1 << 6);  
    error_code = SERVICES_power_ewic_config(services_handle,  
                                           ewic_config);  
  
    if (error_code != SERVICES_REQ_SUCCESS)  
    {  
        return error_code;  
    }  
}
```

SERVICES_power_wakeup_config

Syntax:

```
uint32_t SERVICES_power_wakeup_config(uint32_t services_handle,
                                     uint32_t vbat_wakeup_source,
                                     services_power_profile_t power_profile)
```

Description:

Configure the wake up source

Parameters:

services_handle

vbat_wakeup_source Wake up source

```
typedef enum
{
    VBAT_WAKEUP_MDM                      = 0x1,                      // bit0
    VBAT_WAKEUP_RTC_SE                  = 0x10,                     // bit4
    VBAT_WAKEUP_RTC_A                   = 0x20,                     // bit5
    VBAT_WAKEUP_LPCMP                   = 0x40,                     // bit6
    VBAT_WAKEUP_BROWN_OUT               = 0x80,                     // bit7
    VBAT_WAKEUP_LPTIMER                 = 0XF00,                    // bit11:8
    VBAT_WAKEUP_LPGPIO                  = 0XFF0000,               // bit23:16
} SERVICES_wakeup_cfg_t;
```

power_profile Power profile

```
typedef enum
{
    LOWEST_POWER_PROFILE = 0,            /**< LOWEST_POWER_PROFILE */
    HIGH_PERFORMANCE_POWER_PROFILE, /**< HIGH_PERFORMANCE_POWER_PROFILE */
    USER_SPECIFIED_PROFILE,            /**< USER_SPECIFIED_PROFILE */
    DEFAULT_POWER_PROFILE,             /**< DEFAULT_POWER_PROFILE */
    NUMBER_OF_POWER_PROFILES          /**< NUMBER_OF_POWER_PROFILES */
} services_power_profile_t;
```

Returns:

ErrorCode - SERVICES_REQ_SUCCESS, SERVICES_REQ_CANNOT_EXECUTE_SERVICE

Restrictions:

Example:

```
int main (void)
{
```

```
uint32_t error_code = SERVICES_REQ_SUCCESS;

error_code = SERVICES_power_wakeup_config(services_handle,
VBAT_WAKEUP_RTC_SE
| VBAT_WAKEUP_RTC_A,
LOWEST_POWER_PROFILE);

if (error_code != SERVICES_REQ_SUCCESS)
{
    return error_code;
}
}
```

SERVICES_power_mem_retention_config

Syntax:

uint32_t

```
SERVICES_power_mem_retention_config(uint32_t services_handle,
                                     uint32_t mem_retention,
                                     services_power_profile_t power_profile)
```

Description:

Configure memory retention.

Parameters:

services_handle

mem_retention Memory to be retained.

```
// Memory retention bit encoding for mem_retention_enable
#define POWER_MEM_RET_FIREWALL_RAM            0x01UL
#define POWER_MEM_RET_SE_SRAM                0x02UL
#define POWER_MEM_RET_BACKUP_RAM_4KB        0x04UL
// M55-HE TCM RET1: ITCM 0-128kb; DTCM 0-128kb
#define POWER_MEM_RET_ES1_TCM_RET1           0x08UL
// M55-HE TCM RET1: ITCM 128-256kb; DTCM 128-256kb
#define POWER_MEM_RET_ES1_TCM_RET2           0x10UL
// XTENSA TCM RET1: ITCM 128-512kb
#define POWER_MEM_RET_XTENSA_TCM_RET1       0x20UL
// XTENSA TCM RET1: ITCM 64-128kb
#define POWER_MEM_RET_XTENSA_TCM_RET2       0x40UL
// XTENSA TCM RET1: ITCM 0-64kb
#define POWER_MEM_RET_XTENSA_TCM_RET3       0x80UL
// M55-M TCM RET1: ITCM 1MB; DTCM 384kb
#define POWER_MEM_RET_M55_M_TCM_RET1       0x100UL
#define POWER_MEM_RET_MODEM_BACKUP_RAM_16KB 0x200UL
```

power_profile Power profile

```
typedef enum
{
    LOWEST_POWER_PROFILE = 0,                /**< LOWEST_POWER_PROFILE */
    HIGH_PERFORMANCE_POWER_PROFILE,        /**< HIGH_PERFORMANCE_POWER_PROFILE */
    USER_SPECIFIED_PROFILE,                /**< USER_SPECIFIED_PROFILE */
    DEFAULT_POWER_PROFILE,                 /**< DEFAULT_POWER_PROFILE */
    NUMBER_OF_POWER_PROFILES               /**< NUMBER_OF_POWER_PROFILES */
} services_power_profile_t;
```

Returns:

ErrorCode - SERVICES_REQ_SUCCESS, SERVICES_REQ_CANNOT_EXECUTE_SERVICE

Restrictions:**Example:**

```
int main (void)
{
    uint32_t error_code = SERVICES_REQ_SUCCESS;

    error_code = SERVICES_power_mem_retention_config(services_handle,
                                                    POWER_MEM_RETENTION_SE_RAM,
                                                    LOWEST POWER PROFILE);

    if (error_code != SERVICES_REQ_SUCCESS)
    {
        return error_code;
    }
}
```

SERVICES_power_m55_he_vtor_save

Syntax:

```
SERVICES_power_m55_he_vtor_save(uint32_t services_handle,  
                                uint32_t ns_vtor_addr,  
                                uint32_t se_vtor_addr,  
                                services_power_profile_t power_profile)
```

Description:

m55-he VTOR value save for wake up

Parameters:

services_handle	
ns_vtor_addr	Non-secure VTOR address
se_vtor_addr	Secure VTOR address
power_profile	Power profile

typedef enum

```
{  
    LOWEST_POWER_PROFILE = 0,          /**< LOWEST_POWER_PROFILE */  
    HIGH_PERFORMANCE_POWER_PROFILE, /**< HIGH_PERFORMANCE_POWER_PROFILE */  
    USER_SPECIFIED_PROFILE,          /**< USER_SPECIFIED_PROFILE */  
    DEFAULT_POWER_PROFILE,          /**< DEFAULT_POWER_PROFILE */  
    NUMBER_OF_POWER_PROFILES        /**< NUMBER_OF_POWER_PROFILES */  
} services_power_profile_t;
```

Returns:

ErrorCode - **SERVICES_REQ_SUCCESS**, **SERVICES_REQ_CANNOT_EXECUTE_SERVICE**

Restrictions:**Example:**

```
int main (void)  
{  
    uint32_t error_code = SERVICES_REQ_SUCCESS;  
  
    error_code = SERVICES_power_m55_he_vtor_save(services_handle,  
                                                0x0,  
                                                0x0,  
                                                LOWEST_POWER_PROFILE);
```

```
if (error_code != SERVICES_REQ_SUCCESS)
{
    return error_code;
}
}
```

SERVICES_power_m55_hp_vtor_save

Syntax:

```
SERVICES_power_m55_hp_vtor_save(uint32_t services_handle,  
                                uint32_t ns_vtor_addr,  
                                uint32_t se_vtor_addr,  
                                services_power_profile_t power_profile)
```

Description:

m55-hp VTOR value save for wake up

Parameters:

services_handle	
ns_vtor_addr	Non-secure VTOR address
se_vtor_addr	Secure VTOR address
power_profile	Power profile

typedef enum

```
{  
    LOWEST_POWER_PROFILE = 0,          /**< LOWEST_POWER_PROFILE */  
    HIGH_PERFORMANCE_POWER_PROFILE, /**< HIGH_PERFORMANCE_POWER_PROFILE */  
    USER_SPECIFIED_PROFILE,          /**< USER_SPECIFIED_PROFILE */  
    DEFAULT_POWER_PROFILE,          /**< DEFAULT_POWER_PROFILE */  
    NUMBER_OF_POWER_PROFILES       /**< NUMBER_OF_POWER_PROFILES */  
} services_power_profile_t;
```

Returns:

ErrorCode - **SERVICES_REQ_SUCCESS**, **SERVICES_REQ_CANNOT_EXECUTE_SERVICE**

Restrictions:**Example:**

```
int main (void)  
{  
    uint32_t error_code = SERVICES_REQ_SUCCESS;  
  
    error_code = SERVICES_power_m55_hp_vtor_save(services_handle,  
                                                0x0,  
                                                0x0,  
                                                LOWEST_POWER_PROFILE);  
}
```

```
if (error_code != SERVICES_REQ_SUCCESS)
{
    return error_code;
}
}
```

SERVICES_corestone_standby_mode

Syntax:

```
SERVICES_corestone_standby_mode (uint32_t services_handle,  
                                host_cpu_clus_pwr_req_t host_cpu_clus_pwr_req,  
                                bsys_pwr_req_t bsys_pwr_req,  
                                uint32_t *error_code)
```

Description:

Function to configure corestone standby mode

Parameters:

services_handle

host_cpu_clus_pwr_req Host CPU cluster power state request configuration

bsys_pwr_req Base system power request configuration

power_profile Power profile

Returns:

ErrorCode - SERVICES_REQ_SUCCESS, SERVICES_REQ_CANNOT_EXECUTE_SERVICE

Restrictions:**Example:**

```
int main (void)  
{  
    uint32_t error_code = SERVICES_REQ_SUCCESS;  
    host_cpu_clus_pwr_req_t host_cpu_clus_pwr_req;  
    bsys_pwr_req_t bsys_pwr_req;  
  
    host_cpu_clus_pwr_req.word = 0;  
    host_cpu_clus_pwr_req.bits.mem_ret_req = 0;  
    host_cpu_clus_pwr_req.bits.pwr_req = 1;  
  
    bsys_pwr_req.word = 0;  
    bsys_pwr_req.bits.systop_pwr_req = 1;  
    bsys_pwr_req.bits.dbgtop_pwr_req = 0;  
    bsys_pwr_req.bits.refclk_req = 1;  
    bsys_pwr_req.bits.wakeup_en = 0;  
  
    error_code = SERVICES_corestone_standby_mode(services_handle,  
                                                host_cpu_clus_pwr_req,
```

```
bsys_pwr_req,  
&service_error_code);  
  
if (error_code != SERVICES_REQ_SUCCESS)  
{  
    return error_code;  
}  
}
```

SERVICES_power_memory_req

Syntax:

```
uint32_t SERVICES_power_memory_req(uint32_t services_handle,  
                                   uint32_t memory_request,  
                                   uint32_t *error_code)
```

Description:

Function to disable power to SERAM0, SERAM1 or MRAM

The following are Memory requests:

```
POWER_MEM_SRAM_0_ENABLE  
POWER_MEM_SRAM_1_ENABLE  
POWER_MEM_SRAM_0_ISOLATION_ENABLE  
POWER_MEM_SRAM_1_ISOLATION_ENABLE  
POWER_MEM_MRAM_ENABLE
```

NOTE: This is subject to change

Parameters:

services_handle

memory_request, Which Memory to deal with

Returns:

ErrorCode - SERVICES_REQ_SUCCESS, SERVICES_REQ_CANNOT_EXECUTE_SERVICE

Restrictions:

MRAM may not be able to be disabled directly (To be checked on real device).

Example:

```
int main (void)  
{  
    error_code = SERVICES_power_memory_req(services_handle,  
                                           (POWER_MEM_SRAM_0_ENABLE |  
                                            POWER_MEM_SRAM_1_ISOLATION_ENABLE),  
                                           &return_error_code);  
  
    if (error_code != SERVICES_REQ_SUCCESS)  
    {  
        return error_code;  
    }  
}
```

SERVICES_get_run_cfg

Syntax:

```
uint32_t SERVICES_get_run_cfg(uint32_t services_handle,  
                              run_profile_t *pp,  
                              uint32_t *error_code);
```

Description:

Retrieve the current RUN mode status.

Parameters:

services_handle

pp Run mode parameter block.

error_code Return error code

Returns:

ErrorCode - SERVICES_REQ_SUCCESS, SERVICES_REQ_CANNOT_EXECUTE_SERVICE

Restrictions:

Example:

```
int main (void)  
{  
    uint32_t error_code = SERVICES_REQ_SUCCESS;  
    run_profile_t runp;  
  
    error_code = SERVICES_get_run_cfg(services_handle, &runp,  
                                     &service_error_code);  
  
    if (error_code != SERVICES_REQ_SUCCESS)  
    {  
        return error_code;  
    }  
}
```


SERVICES_get_off_cfg

Syntax:

```
uint32_t SERVICES_get_off_cfg(uint32_t services_handle,  
                              off_profile_t *wp,  
                              uint32_t *error_code);
```

Description:

Retrieved the current OFF mode parameters.

Parameters:

services_handle

wp Off mode parameter block

error_code Return error code

Returns:

ErrorCode - SERVICES_REQ_SUCCESS, SERVICES_REQ_CANNOT_EXECUTE_SERVICE

Restrictions:**Example:**

```
int main (void)  
{  
    uint32_t error_code = SERVICES_REQ_SUCCESS;  
    ewic_config);  
    off_profile_t off_mode;  
  
    error_code = SERVICES_set_off_cfg(services_handle,  
                                     &off_mode,  
                                     &service_error_code);  
  
    if (error_code != SERVICES_REQ_SUCCESS)  
    {  
        return error_code;  
    }  
}
```

SERVICES_set_off_cfg

Syntax:

```
uint32_t SERVICES_set_off_cfg(uint32_t services_handle,  
                             off_profile_t *wp,  
                             uint32_t *error_code);
```

Description:

Set the OFF-mode parameters.

Parameters:

services_handle

wp off mode parameter block

error_code Return error

Returns:

ErrorCode - SERVICES_REQ_SUCCESS, SERVICES_REQ_CANNOT_EXECUTE_SERVICE

Restrictions:

Example:

```
int main (void)  
{  
    uint32_t error_code = SERVICES_REQ_SUCCESS;  
    off_profile_t off_mode;  
  
    error_code = SERVICES_get_off_cfg(services_handle,  
                                     &rnp,  
                                     &service_error_code);  
  
    if (error_code != SERVICES_REQ_SUCCESS)  
    {  
        return error_code;  
    }  
}
```

SERVICES_power_dcdc_voltage_control

Syntax:

```
uint32_t SERVICES_power_dcdc_voltage_control(uint32_t services_handle,  
                                             uint32_t dcdc_vout_sel,  
                                             uint32_t dcdc_vout_trim,  
                                             uint32_t *error_code)
```

Description:

Sets the DCDC voltage and trimming.

Parameters:

services_handle

dcdc_vout_sel Selection shift

dcdc_vout_trim Trimming shift

error_code Return error

Returns:

ErrorCode - SERVICES_REQ_SUCCESS, SERVICES_REQ_CANNOT_EXECUTE_SERVICE

Restrictions:**Example:**

SERVICES_power_ldo_voltage_control

Syntax:

```
uint32_t SERVICES_power_ldo_voltage_control(uint32_t services_handle,  
                                             uint32_t ret_ldo_voltage,  
                                             uint32_t aon_ldo_voltage,  
                                             uint32_t *error_code)
```

Description:

LDO voltage control

Parameters:

services_handle

ret_ldo_voltage retention voltage

aon_ldo_voltage AON voltage

error_code Return error

Returns:

ErrorCode - SERVICES_REQ_SUCCESS, SERVICES_REQ_CANNOT_EXECUTE_SERVICE

SERVICES_power_setting_configure

Syntax:

```
uint32_t SERVICES_power_setting_configure(uint32_t services_handle,  
                                         power_setting_t setting_type,  
                                         uint32_t value,  
                                         uint32_t *error_code)
```

```
typedef enum {  
    POWER_SETTING_BOR_EN,  
    POWER_SETTING_SCALED_CLK_FREQ,  
    POWER_SETTING_ANA_PERIPH_EN  
} power_setting_t;
```

Description:

Configure a power-related setting. The following settings are supported –

BOR_EN	Enable/disable the brown out reset detector
SCALED_CLK_FREQ	Scale the frequency of the HFRC or HFXO
ANA_PERIPH_EN	Enable/disable the analog peripherals band gap and LDO

Parameters:

services_handle
setting_type Power setting type
value Setting value
error code Service error code

Returns:

Transport layer error code

SERVICES_power_setting_get

Syntax:

```
uint32_t SERVICES_power_setting_get(uint32_t services_handle,  
                                     power_setting_t setting_type,  
                                     uint32_t *value,  
                                     uint32_t *error_code)
```

Description:

Get a power-related setting. See SERVICES_power_setting_configure for information about the supported settings.

Parameters:

services_handle	
setting_type	Power setting type
value	Setting value
error code	Service error code

Returns:

Transport layer error code

SERVICES_power_stop_mode_raw_req

Syntax:

```
uint32_t SERVICES_power_stop_mode_raw_req(uint32_t services_handle,  
                                           uint32_t * error_code);
```

Description:

Put the chip in STOP mode immediately. User is responsible for configuring wake up events, clocks etc before going into STOP mode.

Parameters:

services_handle

error code Service error code

Returns:

Transport layer error code

SERVICES_power_ewic_config_raw

Syntax:

```
uint32_t SERVICES_power_ewic_config_raw(uint32_t services_handle,  
                                         uint32_t ewic_source,  
                                         uint32_t *error_code);
```

Description:

Configures EWIC immediately. Overwrites current configuration.

Parameters:

services_handle

ewic_source

error code Service error code

Returns:

Transport layer error code

SERVICES_power_wakeup_config_raw

Syntax:

```
uint32_t SERVICES_power_wakeup_config_raw(uint32_t services_handle,  
                                           uint32_t vbat_wakeup_source,  
                                           uint32_t *error_code);
```

Description:

Configures VBAT wake up immediately. Overwrites current configuration.

Parameters:

services_handle

vbat_wakeup_source

error code Service error code

Returns:

Transport layer error code

SERVICES_power_mem_retention_config_raw

Syntax:

uint32_t

```
SERVICES_power_mem_retention_config_raw(uint32_t services_handle,  
                                         uint32_t mem_retention,  
                                         uint32_t *error_code);
```

Description:

Applies retention settings immediately. Overwrites current configuration.

Parameters:

services_handle

mem_retention

error code Service error code

Returns:

Transport layer error code

SERVICES_power_m55_he_vtor_save_raw

Syntax:

int32_t

```
SERVICES_power_m55_he_vtor_save_raw(uint32_t services_handle,  
                                     uint32_t ns_vtor_addr,  
                                     uint32_t se_vtor_addr,  
                                     uint32_t *error_code);
```

Description:

Applies new VTOR immediately. Overwrites current configuration.

Parameters:

services_handle

ns_vtor_addr

se_vtor_addr

error code Service error code

Returns:

Transport layer error code

Clocks Services

Services to control the Clock, PLL and XTAL (High-frequency external oscillator) settings.

Clock frequency definitions

```
typedef enum {
    CLOCK_FREQUENCY_800MHZ,
    CLOCK_FREQUENCY_400MHZ,
    CLOCK_FREQUENCY_300MHZ,
    CLOCK_FREQUENCY_200MHZ,
    CLOCK_FREQUENCY_160MHZ,
    CLOCK_FREQUENCY_120MHZ,
    CLOCK_FREQUENCY_80MHZ,
    CLOCK_FREQUENCY_60MHZ,
    CLOCK_FREQUENCY_100MHZ,
    CLOCK_FREQUENCY_50MHZ,
    CLOCK_FREQUENCY_20MHZ,
    CLOCK_FREQUENCY_10MHZ,
    CLOCK_FREQUENCY_76_8_RC_MHZ,
    CLOCK_FREQUENCY_38_4_RC_MHZ,
    CLOCK_FREQUENCY_76_8_XO_MHZ,
    CLOCK_FREQUENCY_38_4_XO_MHZ,
    CLOCK_FREQUENCY_DISABLED
} clock_frequency_t;

typedef enum {
    SCALED_FREQ_RC_ACTIVE_76_8_MHZ = 0,
    SCALED_FREQ_RC_ACTIVE_38_4_MHZ,
    SCALED_FREQ_RC_ACTIVE_19_2_MHZ,
    SCALED_FREQ_RC_ACTIVE_9_6_MHZ,
    SCALED_FREQ_RC_ACTIVE_4_8_MHZ,
    SCALED_FREQ_RC_ACTIVE_2_4_MHZ,
    SCALED_FREQ_RC_ACTIVE_1_2_MHZ,
    SCALED_FREQ_RC_ACTIVE_0_6_MHZ,

    SCALED_FREQ_RC_STDBY_76_8_MHZ = 8,
    SCALED_FREQ_RC_STDBY_38_4_MHZ,
    SCALED_FREQ_RC_STDBY_19_2_MHZ,
    SCALED_FREQ_RC_STDBY_4_8_MHZ,
    SCALED_FREQ_RC_STDBY_1_2_MHZ,
    SCALED_FREQ_RC_STDBY_0_6_MHZ,
    SCALED_FREQ_RC_STDBY_0_3_MHZ,
    SCALED_FREQ_RC_STDBY_0_075_MHZ,

    SCALED_FREQ_XO_LOW_DIV_38_4_MHZ = 16,
    SCALED_FREQ_XO_LOW_DIV_19_2_MHZ,
```

```

SCALED_FREQ_XO_LOW_DIV_9_6_MHZ,
SCALED_FREQ_XO_LOW_DIV_4_8_MHZ,
SCALED_FREQ_XO_LOW_DIV_2_4_MHZ,
SCALED_FREQ_XO_LOW_DIV_1_2_MHZ,
SCALED_FREQ_XO_LOW_DIV_0_6_MHZ,
SCALED_FREQ_XO_LOW_DIV_0_3_MHZ,

SCALED_FREQ_XO_HIGH_DIV_38_4_MHZ = 24,
SCALED_FREQ_XO_HIGH_DIV_19_2_MHZ,
SCALED_FREQ_XO_HIGH_DIV_9_6_MHZ,
SCALED_FREQ_XO_HIGH_DIV_2_4_MHZ,
SCALED_FREQ_XO_HIGH_DIV_0_6_MHZ,
SCALED_FREQ_XO_HIGH_DIV_0_3_MHZ,
SCALED_FREQ_XO_HIGH_DIV_0_15_MHZ,
SCALED_FREQ_XO_HIGH_DIV_0_0375_MHZ,
SCALED_FREQ_NONE

```

```
} scaled_clk_freq_t;
```

[SERVICES_clocks_select_osc_source](#)

Syntax:

```

uint32_t SERVICES_clocks_select_osc_source (
    uint32_t services_handle,
    oscillator_source_t source,
    oscillator_target_t target,
    uint32_t * error_code)

```

Description:

Selects between RC or XTAL clock source for various modules (HF or LF). The selected clock is referred to as the 'OSC' clock.

Type definitions:

```

typedef enum {
    OSCILLATOR_SOURCE_RC, // use RC as oscillator clock
    OSCILLATOR_SOURCE_XTAL // use XTAL as oscillator clock
} oscillator_source_t;

typedef enum {
    OSCILLATOR_TARGET_SYS_CLOCKS, // various system clocks
    OSCILLATOR_TARGET_PERIPH_CLOCKS, // clock for peripherals
    OSCILLATOR_TARGET_S32K_CLOCK // 32K low frequency clock
} oscillator_target_t;

```

Parameters:

```

services_handle
source          RC or XTAL (either HF or LF, depending on the target)
target         SYS clocks, PERIPH clocks, S32K clock

```



error code Service error code

Returns:

Transport layer error code

SERVICES_clocks_select_pll_source

Syntax:

```
uint32_t SERVICES_clocks_select_pll_source(  
    uint32_t services_handle,  
    pll_source_t source,  
    pll_target_t target,  
    uint32_t * error_code)
```

Description:

Select OSC or PLL as the source clock for various modules.

Type definitions:

```
typedef enum {  
    PLL_SOURCE_PLL, // use the PLL clocks  
    PLL_SOURCE_OSC // use the OCS clocks (can be RC or XTAL)  
} pll_source_t;
```

```
typedef enum {  
    PLL_TARGET_SYSREFCLK,  
    PLL_TARGET_SYSCLK,  
    PLL_TARGET_UART,  
    PLL_TARGET_ES0,  
    PLL_TARGET_ES1,  
    PLL_TARGET_SECENC,  
    PLL_TARGET_PD4_SRAM  
} pll_target_t;
```

Parameters:

services_handle	
source	OSC or PLL
target	SYSREFCLK, SYSCLK, ES0, ES1
error code	Service error code

Returns:

Transport layer error code

SERVICES_clocks_enable_clock

Syntax:

```
uint32_t SERVICES_clocks_enable_clock(  
    uint32_t services_handle,  
    clock_enable_t clock,  
    bool enable,  
    uint32_t * error_code)
```

Description:

Enable or disable a clock.

Type definitions:

```
typedef enum {  
    CLKEN_SYSPLL,  
    CLKEN_CPUPLL,  
    CLKEN_ES0,  
    CLKEN_ES1,  
    CLKEN_HFXO_OUT,  
    CLKEN_CLK_160M,  
    CLKEN_CLK_100M,  
    CLKEN_USB,  
    CLKEN_HFOSC,  
    CLKEN_SRAM0,  
    CLKEN_SRAM1  
} clock_enable_t;
```

Parameters:

services_handle	
clock	Clock to enable or disable
enable	Enable/disable flag
error code	Service error code

Returns:

Transport layer error code

SERVICES_clocks_set_ES0_frequency

Syntax:

```
uint32_t SERVICES_clocks_set_ES0_frequency(  
    uint32_t services_handle,  
    clock_frequency_t frequency,  
    uint32_t * error_code)
```

Description:

Set the frequency of External System 0 (M55-HP).

Parameters:

services_handle	
frequency	Frequency to set
error code	Service error code

Returns:

Transport layer error code

SERVICES_clocks_set_ES1_frequency

Syntax:

```
uint32_t SERVICES_clocks_set_ES1_frequency(  
    uint32_t services_handle,  
    clock_frequency_t frequency,  
    uint32_t * error_code)
```

Description:

Set the frequency of External System 1 (M55-HE).

Parameters:

services_handle	
frequency	Frequency to set
error code	Service error code

Returns:

Transport layer error code

SERVICES_clocks_select_a32_source

Syntax:

```
uint32_t SERVICES_clocks_select_a32_source (  
    uint32_t services_handle,  
    a32_source_t source,  
    uint32_t * error_code)
```

Description:

Selects the clock source for the A32 CPU cores.

Type definitions:

```
typedef enum {  
    A32_CLOCK_GATE = 0,  
    A32_REFCLK = 1,  
    A32_SYSPLL = 2,  
    A32_CPUPLL = 4  
} a32_source_t;
```

Parameters:

services_handle
source Clock source – CPUPLL, SYSPLL, REFCLK, Clock gate
error code Service error code

Returns:

Transport layer error code

SERVICES_clocks_select_aclk_source

Syntax:

```
uint32_t SERVICES_clocks_select_aclk_source (  
    uint32_t services_handle,  
    aclk_source_t source,  
    uint32_t * error_code)
```

Description:

Selects the clock source for the AXI bus.

Type definitions:

```
typedef enum {  
    ACLK_CLOCK_GATE = 0,  
    ACLK_REFCLK = 1,  
    ACLK_SYSPLL = 2  
} aclk_source_t;
```

Parameters:

services_handle
source Clock source – SYSPLL, REFCLK, Clock gate
error code Service error code

Returns:

Transport layer error code

SERVICES_clocks_set_divider

Syntax:

```
uint32_t SERVICES_clocks_set_divider (  
    uint32_t services_handle,  
    clock_divider_t divider,  
    uint32_t value, uint32_t * error_code)
```

Description:

Selects the value of a clock divider.

Type definitions:

```
typedef enum {  
    DIVIDER_CPUPLL,  
    DIVIDER_SYSPLL,  
    DIVIDER_ACLK,  
    DIVIDER_HCLK,  
    DIVIDER_PCLK  
} clock_divider_t;
```

Parameters:

services_handle
divider Which divider to set – CPUPLL, SYSPLL, ACLK (Corstone), HCLK, PCLK (Alif)
value Divider value. 0x0 to 0x1F for Corstone dividers, 0x0 to 0x2 for Alif divider
error code Service error code

Returns:

Transport layer error code

SERVICES_clocks_get_clocks

Syntax:

```
uint32_t SERVICES_clocks_get_clocks(  
    uint32_t services_handle,  
    clk_get_clocks_svc_t ** pp_svc,  
    scaled_clk_freq_t * scaled_clk_freq,  
    uint32_t * error_code)
```

Description:

Get the values of the clocks registers.

Parameters:

services_handle	
pp_svc	Return values of clocks registers
error code	Service error code

Returns:

Transport layer error code

SERVICES_clocks_get_apb_frequency - OBSOLETE

Syntax:

```
uint32_t SERVICES_clocks_get_apb_frequency(uint32_t services_handle, uint32_t * frequency, uint32_t * error_code)
```

Description:

Get the values of the APB clocks frequency.

This function is now OBSOLETE and will be removed. Use SERVICES_clocks_setting_get() instead.

Parameters:

services_handle	
frequency	Return frequency
error code	Service error code

Returns:

Transport layer error code

SERVICES_clocks_get_refclk_frequency - OBSOLETE

Syntax:

```
uint32_t SERVICES_clocks_get_refclk_frequency(uint32_t services_handle, uint32_t * frequency,  
uint32_t * error_code)
```

Description:

Get the values of the REFCLK frequency.

This function is now OBSOLETE and will be removed. Use SERVICES_clocks_setting_get() instead.

Parameters:

services_handle	
frequency	Return frequency
error code	Service error code

Returns:

Transport layer error code

SERVICES_clocks_setting_get

Syntax:

```
uint32_t SERVICES_clocks_setting_get(uint32_t services_handle, clocks_setting_t setting_type, uint32_t  
*value, uint32_t *error_code)
```

Description:

Get a clock-related setting. Currently, the following settings are supported –

```
typedef enum {  
    CLOCK_SETTING_HFOSC_FREQ,  
    CLOCK_SETTING_EXTSYS0_FREQ,  
    CLOCK_SETTING_EXTSYS1_FREQ,  
    CLOCK_SETTING_AXI_FREQ,  
    CLOCK_SETTING_AHB_FREQ,  
    CLOCK_SETTING_APB_FREQ,  
    CLOCK_SETTING_SYSREF_FREQ,  
    CLOCK_SETTING_ACLK_FORCE_EN,  
    CLOCK_SETTING_ACLK_ENTRY_DELAY,  
} clock_setting_t;
```

Parameters:

services_handle	
setting_type	Power setting type
value	Setting value

error code Service error code

Returns:

Transport layer error code

`SERVICES_clocks_set_aclk`

Syntax:

```
uint32_t SERVICES_clocks_set_aclk(uint32_t services_handle, uint32_t * aclk_entry_delay, uint32_t *  
aclk_force_en, uint32_t *error_code)
```

Description:

Set ACLK clock gating configuration – entry delay and force flag.

Parameters:

<code>services_handle</code>	
<code>aclk_entry_delay</code>	ACLK delay
<code>aclk_force_en</code>	ACLK force enable
<code>error code</code>	Service error code

Returns:

Transport layer error code

SERVICES_pll_xtal_start

Syntax:

```
uint32_t SERVICES_pll_xtal_start(uint32_t services_handle, bool faststart, bool boost, uint32_t delay_count, uint32_t * error_code)
```

Description:

Start the external HF crystal.

Parameters:

services_handle	
faststart	Enable 'fast start' mode
boost	Enable 'boost' mode
delay_count	Timeout to wait for crystal startup
error code	Service error code

Returns:

Transport layer error code

SERVICES_pll_xtal_stop

Syntax:

```
uint32_t SERVICES_pll_xtal_stop(uint32_t services_handle, uint32_t * error_code)
```

Description:

Stop the external HF crystal.

Parameters:

services_handle	
error code	Service error code

Returns:

Transport layer error code

SERVICES_pll_xtal_is_started

Syntax:

```
uint32_t SERVICES_pll_xtal_is_started(uint32_t services_handle, bool * is_started, uint32_t * error_code)
```

Description:

Check if the external HF Crystal is started.

Parameters:

services_handle
is_started External HF Crystal started status
error code Service error code

Returns:

Transport layer error code

SERVICES_pll_clkpll_start

Syntax:

```
uint32_t SERVICES_pll_clkpll_start(uint32_t services_handle, bool faststart, uint32_t delay_count, uint32_t * error_code)
```

Description:

Start the PLL.

Parameters:

services_handle
faststart Enable 'fast start' mode
delay_count Timeout to wait for PLL lock
error code Service error code

Returns:

Transport layer error code

SERVICES_pll_clkpll_stop

Syntax:

```
uint32_t SERVICES_pll_clkpll_stop(uint32_t services_handle, uint32_t * error_code)
```

Description:

Stop the PLL.

Parameters:

services_handle
error code Service error code

Returns:

Transport layer error code

SERVICES_pll_clkpll_is_locked

Syntax:

```
uint32_t SERVICES_pll_clkpll_is_locked(uint32_t services_handle, bool * is_locked, uint32_t * error_code)
```

Description:

Check if the PLL is started and locked.

Parameters:

services_handle
is_locked PLL locked status
error code Service error code

Returns:

Transport layer error code

SERVICES_pll_initialize

Syntax:

```
uint32_t SERVICES_pll_initialize(uint32_t services_handle, uint32_t * error_code)
```

Description:

Initialize the device to enable XTAL and PLL and switch all clocks to PLL.

Parameters:

services_handle
error code Service error code

Returns:

Transport layer error code

SERVICES_pll_deinit

Syntax:

```
uint32_t SERVICES_pll_deinit(uint32_t services_handle, uint32_t * error_code)
```

Description:

De-initialize the device – switch all clocks to TC and disable XTAL and PLL.

Parameters:

services_handle

error code Service error code

Returns:

Transport layer error code

Boot Services

Most Services in this group have a cpu_id parameter. The supported CPU ids are –

HOST_CPU_0	A32_0
HOST_CPU_1	A32_1
EXTSYS_0	M55 HP CPU
EXTSYS_1	M55 HE CPU

SERVICES_boot_process_toc_entry

Syntax:

```
uint32_t SERVICES_boot_process_toc_entry(uint32_t services_handle, const uint8_t * image_id,
uint32_t *error_code)
```

Description:

Request to process a TOC entry. Depending on the information in the TOC entry, this could result in the booting of a CPU core. The TOC entry should also be in a DEFERRED state which means on Boot up it is not automatically booted by SES. This SERVICE call will un-defer the TOC entry.

This is a higher-level function compared to the other Boot services and is a convenient way to boot a CPU core.

Parameters:

services_handle	
image_id	ID of the TOC entry to process. The 'entry_id' field is 8 bytes in size, matching the corresponding TOC entry field 'image_identifier'.
error_code	Service Error Code

Returns:**Restrictions:**

None

SERVICES_boot_cpu

Syntax:

```
uint32_t SERVICES_boot_cpu(uint32_t services_handle, uint32_t cpu_id, uint32_t address, uint32_t *
error_code)
```

Description:

Request to boot a CPU core.

This service does not perform image loading, verification, etc., it just boots the core, specifying the boot address. You would need to use an ATOC to achieve these.

For the M55 cores, there are cases in which this service does not work. The currently known case is the M55-HP core in FUSION REV_Bx devices, where resetting the core also invalidates its TCM content. For that reason, it is recommended that the M55 cores are booted using one of the following methods –

- SERVICES_boot_process_toc_entry().
- SERVICES_set_vtor(), SERVICES_reset_cpu(), and SERVICES_release_cpu(), described in the next sections.

Parameters:

services_handle	
cpu_id	ID of the CPU to boot
address	Boot address for the CPU
error_code	Service Error Code

Returns:**Restrictions:**

None

SERVICES_boot_set_vtor

Syntax:

```
uint32_t SERVICES_boot_set_vtor(uint32_t services_handle, uint32_t cpu_id, uint32_t address, uint32_t * error_code)
```

Description:

Request to initialize the VTOR value for a M55 CPU core.

Note that the address value is stored in a Global register, not in the CPU's internal VTOR register. To transfer the address to the internal VTOR, call SERVICES_reset_cpu() after this call.

Parameters:

services_handle	
cpu_id	ID of the CPU to boot
address	The address to be stored in the VTOR
error_code	Service Error Code

Returns:**Restrictions:**

EXT_SYS0 is not a valid operation on Ensemble devices.

SERVICES_boot_reset_cpu

Syntax:

```
uint32_t SERVICES_boot_reset_cpu(uint32_t services_handle, uint32_t cpu_id, uint32_t * error_code)
```

Description:

Request to reset a CPU core, which effectively stops the core. For M55 cores, it also transfers the VTOR value from the Global VTOR register to the CPU's internal VTOR.

Parameters:

services_handle	
cpu_id	ID of the CPU to boot
error_code	Service Error Code

Returns:**Restrictions:**

None

SERVICES_boot_release_cpu

Syntax:

```
uint32_t SERVICES_boot_release_cpu(uint32_t services_handle, uint32_t cpu_id, uint32_t * error_code)
```

Description:

Request to release a CPU core. This service does not perform image loading, verification, etc., and does not reset the CPU or specify the boot address, it just releases the core.

If the CPU is not running, this function can be called to release it.

If the CPU is running, SERVICES_boot_reset_cpu() must be called before this function to stop the core.

Notes on releasing M55 cores –

- in some cases, resetting the core also invalidates its TCM. A known case is the M55-HP core in Ensemble devices. Because of that, after calling SERVICES_boot_reset_cpu() to stop the core, the image in the TCM must be reloaded, before calling SERVICES_boot_release_cpu() to start the core.
- If the VTOR value of the core needs to be changed, that too requires calling SERVICES_boot_reset_cpu(), to transfer the new address value to the core's internal VTOR. So, the call order of services in this case is –
 1. SERVICES_boot_set_vtor(),
 2. SERVICES_boot_reset_cpu(),
 3. load the image in the TCM, 3. SERVICES_boot_release_cpu().

Parameters:

services_handle	
cpu_id	ID of the CPU to boot
error_code	Service Error Code

Returns:**Restrictions:**

SERVICES_boot_reset_soc

Syntax:

```
uint32_t SERVICES_boot_reset_soc(uint32_t services_handle)
```

Description:

Request to reset the entire SoC.

Parameters:

services_handle

Returns:

Restrictions:

None

Crypto Services

The SE provides several crypto services to other cores as detailed below.

SERVICES_cryptocell_get_rnd

Syntax:

```
uint32_t SERVICES_cryptocell_get_rnd(uint32_t services_handle, uint16_t rnd_length, void * rnd_value,
uint32_t * error_code)
```

Description:

Request random number.

The service SERVICES_cryptocell_get_rnd returns a random vector generated by the cryptocell-rt library using the MbedTLS API call mbedtls_ctr_drbg_random().

The desired length of the vector to generate is passed as an input parameter. Currently, the maximum supported vector length is 128 bytes.

Parameters:

services_handle

rnd_length Length of random number vector

rnd_value returned Random number

error_code Service Error Code

None

Returns:

Restrictions:

None

Example:

```
int main (void)
{
    uint32_t ErrorCode = SERVICES_OK;
    uint64_t rnd_value;
    uint32_t service_error_code;

    ErrorCode = SERVICES_cryptocell_get_rnd(services_handle,
    _____ sizeof(uint64_t), /* random number/vector length in bytes*/
    &rnd_value,
```

```
        &service_error_code);  
    if (ErrorCode != SERVICES_REQ_SUCCESS)  
    {  
        return ErrorCode;  
    }  
}
```

SERVICES_cryptocell_get_lcs

Syntax:

```
uint32_t SERVICES_cryptocell_get_lcs(uint32_t services_handle, uint32_t *lcs_state, uint32_t *  
error_code)
```

Description:

The service SERVICES_cryptocell_get_lcs returns the current Life Cycle State.

Parameters:

services_handle

lcs_state Life cycle state

error_code Service Error Code

Returns:

Restrictions:

None

Example:

```
int main (void)  
{  
    uint32_t ErrorCode = SERVICES_OK;  
    uint32_t lcs_state;  
    uint32_t service_error_code  
  
    ErrorCode = SERVICES_cryptocell_get_lcs(services_handle, &lcs_state,  
&service_error_code);  
    if (ErrorCode != SERVICES_REQ_SUCCESS)  
    {  
        return ErrorCode;  
    }  
}
```

MbedTLS Services

These services expose the hardware accelerated functionality provided by the Arm CryptoCell-RT library in SES.

Arm has chosen to use MbedTLS as the public API to that functionality. For that reason, the exposed Services correspond to MbedTLS public APIs.

IMPORTANT: These Services are not intended to be used directly by applications. Instead, they should be used by a client-side MbedTLS library implementation in which hardware acceleration is done by calling the Services.

To simplify the Services APIs and to avoid introducing MbedTLS types into them, all parameters of the MbedTLS functions are passed as `uint32_t`. The client-side MbedTLS implementation must convert them to the appropriate types.

Also, to reduce the number of Service APIs, some of them cover multiple MbedtTLS API functions.

Please refer to the MbedTLS documentation for more information on these APIs, usage and parameters.

Many MbedTLS API functions have a 'context' parameter of the appropriate type for the API. E.g. `mbedtls_aes_context` for the AES functions. As an alternative, the corresponding Services APIs also accept arrays of type `uint32_t` of the same size as the MbedTSLI context structure.

SERVICES_cryptocell_mbedtls_hardware_poll

Syntax:

```
uint32_t SERVICES_cryptocell_mbedtls_hardware_poll(uint32_t services_handle,
                                                    uint32_t * error_code,
                                                    uint32_t data,
                                                    uint32_t output,
                                                    uint32_t len,
                                                    uint32_t olen)
```

Description:

Service API replacement for `mbedtls_hardware_poll()`.

This function is a callback that the `mbedtls` library calls internally.

Alif devices have built-in TRNG hardware. To get random values generated by it, please use the Service API `SERVICES_cryptocell_get_rnd()`.

Parameters:

<code>data</code>	This parameter is in the MbedTLS API function signature, but it is not used by the CC312 library code.
<code>output, len</code>	Buffer to store the data returned from the call. <u>The CC312 hardware requires that the buffer is at least 144 bytes in size.</u>

olen Set by the CC312 library to indicate the length of the returned data.

SERVICES_cryptocell_mbedtls_aes_init

Syntax:

```
uint32_t SERVICES_cryptocell_mbedtls_aes_init(uint32_t services_handle,  
                                              uint32_t * error_code,  
                                              uint32_t ctx)
```

Description:

Service API replacement for mbedtls_aes_init()

SERVICES_cryptocell_mbedtls_aes_set_key

Syntax:

```
uint32_t SERVICES_cryptocell_mbedtls_aes_set_key(uint32_t services_handle,  
                                                  uint32_t * error_code,  
                                                  uint32_t ctx,  
                                                  uint32_t key,  
                                                  uint32_t keybits,  
                                                  uint32_t dir)
```

Description:

Service API replacement for mbedtls_aes_set_key_enc() and mbedtls_aes_set_key_dec()

SERVICES_cryptocell_mbedtls_aes_crypt

Syntax:

```
uint32_t SERVICES_cryptocell_mbedtls_aes_crypt(uint32_t services_handle,  
                                                uint32_t * error_code,  
                                                uint32_t ctx,  
                                                uint32_t crypt_type,  
                                                uint32_t mode,  
                                                uint32_t length,  
                                                uint32_t iv,  
                                                uint32_t input,  
                                                uint32_t output)
```

Description:

Service API replacement for the mbedtls_aes_crypt_XXX functions. The following AES modes are supported – ECB, CBC, CTR, OFB. Depending on the mode, some of the parameters are not used.

mode Used only in ECB and CBC modes. The value to provide is the same as the value provided in the 'dir' parameter of SERVICES_cryptocell_mbedtls_aes_set_key()

iv

Not used in ECB mode. CTR mode uses a 'nonce' instead of an initialization vector. The nonce should be passed to this parameter.

SERVICES_cryptocell_mbedtls_sha_starts

Syntax:

```
uint32_t SERVICES_cryptocell_mbedtls_sha_starts(uint32_t services_handle,  
                                                uint32_t * error_code,  
                                                uint32_t ctx,  
                                                uint32_t sha_type)
```

Description:

Service API replacement for mbedtls_sha_starts()

*SERVICES_cryptocell_mbedtls_sha_process***Syntax:**

```
uint32_t SERVICES_cryptocell_mbedtls_sha_process(uint32_t services_handle,  
                                                uint32_t * error_code,  
                                                uint32_t ctx,  
                                                uint32_t sha_type,  
                                                uint32_t data)
```

Description:

Service API replacement for mbedtls_sha_process()

*SERVICES_cryptocell_mbedtls_sha_update***Syntax:**

```
uint32_t SERVICES_cryptocell_mbedtls_sha_update(uint32_t services_handle,  
                                                uint32_t * error_code,  
                                                uint32_t ctx,  
                                                uint32_t sha_type,  
                                                uint32_t data,  
                                                uint32_t data_length)
```

Description:

Service API replacement for mbedtls_sha_update()

*SERVICES_cryptocell_mbedtls_sha_finish***Syntax:**

```
uint32_t SERVICES_cryptocell_mbedtls_sha_finish(uint32_t services_handle,  
                                                uint32_t * error_code,  
                                                uint32_t ctx,  
                                                uint32_t sha_type,  
                                                uint32_t data)
```

Description:

Service API replacement for mbedtls_sha_finish()

*SERVICES_cryptocell_mbedtls_ccm_gcm_set_key***Syntax:**

```
uint32_t SERVICES_cryptocell_mbedtls_ccm_gcm_set_key(uint32_t services_handle,  
    uint32_t * error_code,  
    uint32_t context_addr,  
    uint32_t key_type,  
    uint32_t cipher,  
    uint32_t key_addr,  
    uint32_t key_bits)
```

Description:

Service API replacement for mbedtls_ccm_set_key() and mbedtls_gcm_set_key(). Only integrated operations are supported.

*SERVICES_cryptocell_mbedtls_ccm_gcm_crypt***Syntax:**

```
uint32_t SERVICES_cryptocell_mbedtls_ccm_gcm_crypt(uint32_t services_handle,  
    uint32_t * error_code,  
    uint32_t context_addr,  
    uint32_t crypt_type,  
    uint32_t length,  
    uint32_t iv_addr,  
    uint32_t iv_length,  
    uint32_t add_addr,  
    uint32_t add_length,  
    uint32_t input_addr,  
    uint32_t output_addr,  
    uint32_t tag_addr,  
    uint32_t tag_length)
```

Description:

Service API replacement for the mbedtls CCM and GCM crypto functions. Only AES is supported as crypto type. Only integrated operations are supported.

*SERVICES_cryptocell_mbedtls_chacha20_crypt***Syntax:**

```
uint32_t SERVICES_cryptocell_mbedtls_chacha20_crypt(uint32_t services_handle,  
    uint32_t * error_code,  
    uint32_t key_addr,  
    uint32_t nonce_addr,  
    uint32_t counter,  
    uint32_t data_len,  
    uint32_t input_addr,  
    uint32_t output_addr)
```

Description:

Service API replacement for `mbedtls_chacha20_crypt()`. Only integrated operations are supported.

SERVICES_cryptocell_mbedtls_chachapoly_crypt

Syntax:

```
uint32_t SERVICES_cryptocell_mbedtls_chachapoly_crypt(uint32_t services_handle,  
    uint32_t * error_code,  
    uint32_t context_addr,  
    uint32_t crypt_type,  
    uint32_t length,  
    uint32_t nonce_addr,  
    uint32_t aad_addr,  
    uint32_t aad_len,  
    uint32_t tag_addr,  
    uint32_t input_addr,  
    uint32_t output_addr)
```

Description:

Service API replacement for the `mbedtls_chachapoly_encrypt` and `mbedtls_chachapoly_decrypt` functions. Only integrated operations are supported.

SERVICES_cryptocell_mbedtls_poly1305_crypt

Syntax:

```
uint32_t SERVICES_cryptocell_mbedtls_poly1305_crypt(uint32_t services_handle,  
    uint32_t * error_code,  
    uint32_t key_addr,  
    uint32_t input_addr,  
    uint32_t ilen,  
    uint32_t mac_addr)
```

Description:

Service API replacement for `mbedtls_poly1305_crypt()`. Only integrated operations are supported.

SERVICES_cryptocell_mbedtls_cmac_init_setkey

Syntax:

```
uint32_t SERVICES_cryptocell_mbedtls_cmac_init_setkey(uint32_t services_handle,  
    uint32_t * error_code,  
    uint32_t context_addr,  
    uint32_t key_addr,  
    uint32_t key_bits)
```

Description:

Service API replacement for `mbedtls_cmac_init_setkey()`

*SERVICES_cryptocell_mbedtls_cmhac_update***Syntax:**

```
uint32_t SERVICES_cryptocell_mbedtls_cmhac_update(uint32_t services_handle,  
    uint32_t * error_code,  
    uint32_t context_addr,  
    uint32_t input_addr,  
    uint32_t input_length)
```

Description:

Service API replacement for mbedtls_cmhac_update()

*SERVICES_cryptocell_mbedtls_cmhac_finish***Syntax:**

```
uint32_t SERVICES_cryptocell_mbedtls_cmhac_finish(uint32_t services_handle,  
    uint32_t * error_code,  
    uint32_t context_addr,  
    uint32_t output_addr)
```

Description:

Service API replacement for mbedtls_cmhac_finish()

*SERVICES_cryptocell_mbedtls_cmhac_reset***Syntax:**

```
uint32_t SERVICES_cryptocell_mbedtls_cmhac_reset(uint32_t services_handle,  
    uint32_t * error_code,  
    uint32_t context_addr)
```

Description:

Service API replacement for mbedtls_cmhac_finish()

Update Services

SERVICES_update_stoc

Syntax:

```
uint32_t SERVICES_update_stoc(uint32_t services_handle,
                              uint32_t image_address,
                              uint32_t image_size,
                              uint32_t *error_code)
```

Description:

Update the ALIF STOC using the provided image.

This image uses the ALIF Update Package format (as this is used to update the ALIF System FW). Packages are either FULL or PARTIAL.

Parameters:

services_handle	
image_address	Address of the source of the update image.
Image_size	Size of the update image, only used with UNSIGNED Packages
error_code	

Returns:

BL_ERROR_ENTRY_NOT_SIGNED	0x17	
BL_ERROR_UPD_SIGNATURE_INCORRECT	0x21	Package is not in ALIF UPD format

Restrictions:

The UPDATE Package is supplied (and signed) by ALIF.

This SERVICE is not intended to be used for Application Over-the-air updates but for updating the SES firmware in an OTA environment.

External System Services

These SERVICES are only relevant for the BALLETO (Blue Tooth) Products.

They will return SERVICES_RESP_UNKNOWN_COMMAND when used with ENSEMBLE devices.

Boot arguments

```
// ESO CPU clock frequencies
#define ESO_CLOCK_16MHZ 0
#define ESO_CLOCK_24MHZ 4
#define ESO_CLOCK_48MHZ 0xC

// ExtSys0 Boot arguments
typedef struct {
    uint32_t nvds_src_addr;
    uint32_t nvds_dst_addr;
    uint32_t nvds_copy_len;
    uint32_t trng_dst_addr;
    uint32_t trng_len;
    uint32_t es0_clock_select;
} net_proc_boot_args_t;
```

Used to encapsulate the boot arguments for the External System.

- NVDS data is supplied by the main application.
- TRNG (Truly random number) destination address and number of bytes to be generated.
- ESO Clock select, can be 16 (default), 24 or 48Mhz.

SERVICES_Boot_Net_Proc

Syntax:

```
uint32_t SERVICES_Boot_Net_Proc(uint32_t services_handle,  
                                net_proc_boot_args_t* boot_args,  
                                uint32_t *error_code)
```

Description:

Packages the boot arguments for External system (ExtSys0).

The following steps are performed in this SERVICE:

- *#1 Enable Power to the ExtSys0
- *#2 Load PatchROM from MRAM (which is in Deferred state)
- *#3 Copy the NVDS Data from supplied address
- *#4 Change External Sys0 CPU frequency
- *#4 Generate TRNG value size based on SERVICE call provided width
- * #5 Clear EXTSYS1 status
- * #6 Release EXTSYS0 CPU

Parameters:

services_handle

net_proc_boot_args_t External System boot arguments
error_code

Returns:

SERVICES_SUCCESS

SERVICES_RESP_UNKNOWN_COMMAND If used with ENSEMBLE devices.

BL_ERROR_* Boot loader error, should PATCHROM fail to load.

Restrictions:

SERVICES_Shutdown_Net_Proc

Syntax:

```
uint32_t SERVICES_Shutdown_Net_Proc(uint32_t services_handle,  
                                     uint32_t *error_code)
```

Description:

Powers off the External System 0

Parameters:

services_handle
error_code

Returns:

SERVICES_SUCCESS

Restrictions:

Document History

Version	Change Log
0.41.1	Editing version derived from v0.41 engineering input