PSA Firmware Update API 0.7

Document number:	IHI 0093
Release Quality:	Beta
Issue Number:	0
Confidentiality:	Non-confidential
Date of Issue:	29/04/2022

Copyright © 2020-2021, Arm Limited. All rights reserved.

Abstract

This manual defines a standard firmware interface for installing firmware updates.

Note:

This is a Beta quality release. The content is subject to change. Feedback should be sent to arm.psa-feedback@arm.com

Contents

About this document v

Release information v

Arm Non-Confidential Document Licence (“Licence”) vi

References viii

Terms and abbreviations viii

Conventions x
Typographical conventions x
Numbers x

Feedback xi
Feedback on this book xi

1 Introduction 12

2 Design goals 13

2.1 Suitable for constrained devices 13

2.2 PSA Root of Trust update 13

2.3 Application Root of Trust update 14

2.4 Flexiblility for different trust models 14

2.5 Protocol independence 14

2.6 Transport independence 14

2.7 Hardware flexibility 15

2.8 Composite devices 15

2.9 Room for different implementations 15

3 Terminology 16

3.1 Image 16

3.2 Trust anchor 16

3.3 Installer 17

3.4 Update client 18

Copyright © 2020-2021, Arm Limited or its affiliates. All rights reserved.
Non-confidential

3.5 Secure Processing Environment (SPE) 18

3.6 Staging area 18

4 Trust model and scenarios 19

5 Design overview 20

5.1 Mandatory functions 20
5.1.1 Querying installed images 20
5.1.2 Image storing 20
5.1.3 Metadata storage 21
5.1.4 Verify image 21
5.1.5 Triggering a reboot 21

5.2 Optional functions 21

5.3 State transitions for an image 22

5.4 Dependencies 22

6 Image metadata 23

6.1 Format profiles 23

6.2 Usage models 23

6.3 Example metadata 23
6.3.1 Summary 24
6.3.2 CBOR 24

7 API reference 26

7.1 Library conventions 26

7.2 Behavior on error 26

7.3 Pointer conventions 26

7.4 Macros 27
7.4.1 Library versioning 27
7.4.2 Image transfer 27
7.4.3 Digest size 27
7.4.4 Image states 27
7.4.5 Image flags 28

7.5 Types 28
7.5.1 psa_image_version_t (struct) 28
7.5.2 psa_staging_info_t (struct) 29
7.5.3 psa_image_info_t (struct) 29
7.5.4 psa_uuid_t (struct) 30
7.5.5 psa_image_id_t (type) 31

Copyright © 2020-2021, Arm Limited or its affiliates. All rights reserved.
Non-confidential

7.5.6 psa_fwu_iterator_t (type) 31
7.5.7 psa_hash_t (struct) 31

7.6 Status codes 31
7.6.1 psa_status_t (type) 31
7.6.2 PSA_SUCCESS (macro) 32
7.6.3 PSA_SUCCESS_REBOOT (macro) 32
7.6.4 PSA_SUCCESS_RESTART (macro) 32
7.6.5 PSA_SUCCESS_DEPENDENCY_NEEDED (macro) 32

7.7 Error codes 32
7.7.1 PSA_ERROR_GENERIC_ERROR (macro) 32
7.7.2 PSA_ERROR_NOT_SUPPORTED (macro) 32
7.7.3 PSA_ERROR_NOT_PERMITTED (macro) 33
7.7.4 PSA_ERROR_DOES_NOT_EXIST (macro) 33
7.7.5 PSA_ERROR_INVALID_ARGUMENT (macro) 33
7.7.6 PSA_ERROR_INSUFFICIENT_MEMORY (macro) 33
7.7.7 PSA_ERROR_INSUFFICIENT_STORAGE (macro) 33
7.7.8 PSA_ERROR_COMMUNICATION_FAILURE (macro) 34
7.7.9 PSA_ERROR_STORAGE_FAILURE (macro) 34
7.7.10 PSA_ERROR_DATA_CORRUPT (macro) 34
7.7.11 PSA_ERROR_DATA_INVALID (macro) 35
7.7.12 PSA_ERROR_HARDWARE_FAILURE (macro) 35
7.7.13 PSA_ERROR_CORRUPTION_DETECTED (macro) 36
7.7.14 PSA_ERROR_INVALID_SIGNATURE (macro) 36
7.7.15 PSA_ERROR_INSUFFICIENT_DATA (macro) 36
7.7.16 PSA_ERROR_WRONG_DEVICE (macro) 36
7.7.17 PSA_ERROR_DEPENDENCY_NEEDED (macro) 36
7.7.18 PSA_ERROR_CURRENTLY_INSTALLING (macro) 36
7.7.19 PSA_ERROR_ALREADY_INSTALLED (macro) 37
7.7.20 PSA_ERROR_INSTALL_INTERRUPTED (macro) 37
7.7.21 PSA_ERROR_FLASH_ABUSE (macro) 37
7.7.22 PSA_ERROR_INSUFFICIENT_POWER (macro) 37
7.7.23 PSA_ERROR_DECRYPTION_FAILURE (macro) 37
7.7.24 PSA_ERROR_MISSING_MANIFEST (macro) 37

7.8 Functions 37
7.8.1 psa_fwu_query (function) 37
7.8.2 psa_fwu_set_manifest (function) 38
7.8.3 psa_fwu_write (function) 39
7.8.4 psa_fwu_install (function) 40
7.8.5 psa_fwu_abort (function) 41
7.8.6 psa_fwu_request_reboot (function) 41
7.8.7 psa_fwu_request_rollback (function) 42
7.8.8 psa_fwu_accept (function) 42
7.8.9 psa_fwu_get_image_id_iterator (function) 43
7.8.10 psa_fwu_get_image_id_next (function) 43
7.8.11 psa_fwu_get_image_id_valid (function) 43
7.8.12 psa_fwu_get_image_id (function) 43

Copyright © 2020-2021, Arm Limited or its affiliates. All rights reserved.
Non-confidential

A Example header file 45

A.1 psa/update.h 45

B Example usage 48

B.1 Retrieve versions of installed images 48

B.2 Individual image update (single part operation) 48

B.3 Invividual image update (multi part operation) 49

B.4 Multiple dependent images (multi part operation) 50

C Future changes 51

C.1 Rename psa_fwu_abort 51

C.2 Init function 51

D Change history 52

Copyright © 2020-2021, Arm Limited or its affiliates. All rights reserved.
Non-confidential

About this document

Release information

The change history table lists the changes that have been made to this document.

Date	Version	Confidentiality	Change
Feb 2021	0.7 Beta 0	Non-confidential	First release at Beta quality.

Copyright © 2020-2021, Arm Limited or its affiliates. All rights reserved.
Non-confidential

PSA Firmware Update API

Copyright © 2020-2021, Arm Limited or its affiliates. All rights reserved. The copyright statement reflects the fact that some draft issues of this document have been released, to a limited circulation.

Arm Non-Confidential Document Licence ("Licence")

This Licence is a legal agreement between you and Arm Limited ("**Arm**") for the use of Arm's intellectual property (including, without limitation, any copyright) embodied in the document accompanying this Licence ("**Document**"). Arm licenses its intellectual property in the Document to you on condition that you agree to the terms of this Licence. By using or copying the Document you indicate that you agree to be bound by the terms of this Licence.

"**Subsidiary**" means any company the majority of whose voting shares is now or hereafter owner or controlled, directly or indirectly, by you. A company shall be a Subsidiary only for the period during which such control exists.

This Document is **NON-CONFIDENTIAL** and any use by you and your Subsidiaries ("Licensee") is subject to the terms of this Licence between you and Arm.

Subject to the terms and conditions of this Licence, Arm hereby grants to Licensee under the intellectual property in the Document owned or controlled by Arm, a non-exclusive, non-transferable, non-sub-licensable, royalty-free, worldwide licence to:

(i) use and copy the Document for the purpose of designing and having designed products that comply with the Document;

(ii) manufacture and have manufactured products which have been created under the licence granted in (i) above; and

(iii) sell, supply and distribute products which have been created under the licence granted in (i) above.

Licensee hereby agrees that the licences granted above shall not extend to any portion or function of a product that is not itself compliant with part of the Document.

Except as expressly licensed above, Licensee acquires no right, title or interest in any Arm technology or any intellectual property embodied therein.

THE DOCUMENT IS PROVIDED "AS IS". ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. Arm may make changes to the Document at any time and without notice. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, third party patents, copyrights, trade secrets, or other rights.

NOTWITHSTANDING ANYTHING TO THE CONTRARY CONTAINED IN THIS LICENCE, TO THE FULLEST EXTENT PERMITTED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, IN CONTRACT, TORT OR OTHERWISE, IN CONNECTION WITH THE SUBJECT MATTER OF THIS LICENCE (INCLUDING WITHOUT LIMITATION) (I) LICENSEE'S USE OF THE DOCUMENT; AND (II) THE IMPLEMENTATION OF THE DOCUMENT IN ANY PRODUCT CREATED BY LICENSEE UNDER THIS LICENCE). THE EXISTENCE OF MORE THAN ONE CLAIM OR SUIT WILL NOT ENLARGE OR EXTEND THE LIMIT. LICENSEE RELEASES ARM FROM ALL OBLIGATIONS, LIABILITY, CLAIMS OR DEMANDS IN EXCESS OF THIS LIMITATION.

This Licence shall remain in force until terminated by Licensee or by Arm. Without prejudice to any of its other rights, if Licensee is in breach of any of the terms and conditions of this Licence then Arm may terminate this Licence immediately upon giving written notice to Licensee. Licensee may terminate this Licence at any time. Upon termination of this Licence by Licensee or by Arm, Licensee shall stop using the Document and destroy all copies of the Document in its possession. Upon termination of this Licence, all terms shall survive except for the licence grants.

Any breach of this Licence by a Subsidiary shall entitle Arm to terminate this Licence as if you were the party in breach. Any termination of this Licence shall be effective in respect of all Subsidiaries. Any rights granted to any Subsidiary hereunder shall automatically terminate upon such Subsidiary ceasing to be a Subsidiary.

The Document consists solely of commercial items. Licensee shall be responsible for ensuring that any use, duplication or disclosure of the Document complies fully with any relevant export laws and regulations to assure that the Document or any portion thereof is not exported, directly or indirectly, in violation of such export laws.

This Licence may be translated into other languages for convenience, and Licensee agrees that if there is any conflict between the English version of this Licence and any translation, the terms of the English version of this Licence shall prevail.

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the

trademarks of their respective owners. No licence, express, implied or otherwise, is granted to Licensee under this Licence, to use the Arm trade marks in connection with the Document or any products based thereon. Visit Arm's website at https://www.arm.com/company/policies/trademarks for more information about Arm's trademarks.

The validity, construction and performance of this Licence shall be governed by English Law.

Copyright © 2020-2021, Arm Limited or its affiliates. All rights reserved.

Arm Limited. Company 02557590 registered in England.
110 Fulbourn Road, Cambridge, England CB1 9NJ.

Arm document reference: LES-PRE-21585 version 4.0

References

This document refers to the following documents.

Ref	Document Number	Title
[PSA-SM]	DEN 0079	*PSA Security Model*. https://pages.arm.com/psa-resources-sm.html
[PSA-TB]	DEN 0072	*PSA Trusted Boot and Firmware Update*. https://pages.arm.com/psa-resources-tbfu.html
[PSA-ATT]	IHI 0085	*PSA Attestation API*. https://pages.arm.com/psa-apis.html
[PSA-DBG]	PSA-DBG-AUTH	*PSA Debug Access Control*.
[PSA-CERT]	JSA DEN 002	*PSA Certified™ Level 2 Lightweight Protection Profile*. https://www.psacertified.org/resources/
[SUIT]		IETF, *A Concise Binary Object Representation (CBOR)-based Serialization Format for the Software Updates for Internet of Things (SUIT) Manifest*. https://tools.ietf.org/html/draft-ietf-suit-manifest-04
[SUIT-ARCH]		IETF, *A Firmware Update Architecture for Internet of Things*. https://tools.ietf.org/html/draft-ietf-suit-architecture-08#page-22
[RFC4122]		IETF, *A Universally Unique IDentifier (UUID) URN Namespace*. https://tools.ietf.org/html/rfc4122
[EBBR]	ARM DEN 0064	ARM, *Embedded Base Boot Requirements*.
[SUIT-CODE]		GitHub, *Example code to generate and parse SUIT manifests*. https://github.com/ARMmbed/suit-manifest-generator

Terms and abbreviations

This document uses the following terms and abbreviations.

Term	Meaning
Application firmware	The main application firmware for the platform, typically comprising an Operating System (OS) and application tasks. On a platform with isolation, the application firmware runs in the *NSPE*.
Application Root of Trust	This is the security domain in which additional security services are implemented. See *PSA Security Model* [PSA-SM] for details.
CBOR	Concise Binary Object Representation (CBOR). A binary data serialization format loosely based on JSON.
Cloud connector	See *Update client*.
CSP	Cloud service provider

Copyright © 2020-2021, Arm Limited or its affiliates. All rights reserved.
Non-confidential

Table 2 (continued)

Term	Meaning
IMPLEMENTATION DEFINED	Behavior that is not defined by the this specification, but is defined and documented by individual implementations. Firmware developers can choose to depend on IMPLEMENTATION DEFINED behavior, but must be aware that their code might not be portable to another implementation.
ISV	Independent software vendor
Manifest	Image metadata that is signed with a cryptographic key.
MPU	Memory Protection Unit
Non-secure Processing Environment (NSPE)	This is the security domain outside of the *Secure Processing Environment*. It is the Application domain, typically containing the *application firmware* and hardware.
NSPE	See *Non-secure Processing Environment*.
OEM	Original Equipment Manufacturer
OTA	See *Over-the-Air*.
Over-the-Air (OTA)	The procedure where a device downloads an update from a remote location ("over the air").
PROGRAMMER ERROR	An error that is caused by the misuse of a programming interface. A PROGRAMMER ERROR is in the caller of the interface, but it is detected by the implementer of the interface.
PSA	Platform Security Architecture
PSA Immutable Root of Trust	The hardware, code and data that cannot be modified following manufacturing. See *PSA Security Model* [PSA-SM] for details.
PSA Root of Trust	This defines the most trusted security domain within a PSA system. See *PSA Security Model* [PSA-SM] for details.
PSA Updateable Root of Trust	The *Root of Trust* firmware that can be updated following manufacturing. See *PSA Security Model* [PSA-SM] for details.
Root of Trust (RoT)	This is the minimal set of software, hardware and data that is implicitly trusted in the platform — there is no software or hardware at a deeper level that can verify that the Root of Trust is authentic and unmodified. See *PSA Security Model* [PSA-SM].
RoT	See *Root of Trust*.
Secure Partition	A thread of execution with protected runtime state within the *Secure Processing Environment*. Container for the implementation of one or more *RoT* services. Multiple Secure Partitions may exist on a platform.

Copyright © 2020-2021, Arm Limited or its affiliates. All rights reserved.
Non-confidential

Table 2 (continued)

Term	Meaning
Secure Partition Manager (SPM)	Part of the PSA Firmware Framework that is responsible for isolating software in Partitions, managing the execution of software within Partitions, and providing IPC between Partitions.
Secure Processing Environment (SPE)	This is the security domain that includes the *PSA Root of Trust* and the *Application Root of Trust* domains.
SPE	See *Secure Processing Environment*.
SPM	See *Secure Partition Manager*.
Trusted Boot	Trusted Boot is technology to provide a chain of trust for all the components during boot. See *PSA Trusted Boot and Firmware Update* [PSA-TB].
Update client	Software component that is responsible for downloading firmware updates to the device. The Update client is part of the *application firmware*.

Conventions

Typographical conventions

The typographical conventions are:

italic Introduces special terminology, and denotes citations.

`monospace` Used for assembler syntax descriptions, pseudocode, and source code examples.

Also used in the main text for instruction mnemonics and for references to other items appearing in assembler syntax descriptions, pseudocode, and source code examples.

SMALL CAPITALS

Used for some common terms such as IMPLEMENTATION DEFINED.

Used for a few terms that have specific technical meanings, and are included in the *Terms and abbreviations*.

Red text Indicates an open issue.

Blue text Indicates a link. This can be

- A cross-reference to another location within the document
- A URL, for example http://infocenter.arm.com

Numbers

Numbers are normally written in decimal. Binary numbers are preceded by 0b, and hexadecimal numbers by `0x`.

In both cases, the prefix and the associated value are written in a monospace font, for example `0xFFFF0000`. To improve readability, long numbers can be written with an underscore separator between every four characters, for example `0xFFFF_0000_0000_0000`. Ignore any underscores when interpreting the value of a number.

Copyright © 2020-2021, Arm Limited or its affiliates. All rights reserved.
Non-confidential

Feedback

Arm welcomes feedback on its documentation.

Feedback on this book

If you have comments on the content of this book, send an e-mail to arm.psa-feedback@arm.com. Give:

- The title (PSA Firmware Update API).
- The number and issue (IHI 0093 0.7 Beta (Issue 0)).
- The page numbers to which your comments apply.
- The rule identifiers to which your comments apply, if applicable.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

A description of the open issues is described in appendix Future changes.

Copyright © 2020-2021, Arm Limited or its affiliates. All rights reserved.
Non-confidential

1 Introduction

Firmware update support is an essential property of a PSA device. However, the mechanism to update firmware on a device differs significantly across embedded platforms. This is further complicated by different implementations of a Secure Processing Environment (SPE), which have their own set of firmware that is separate from application firmware.

Some vendors also support a range of different Update clients that may come from third parties, such as ISVs, OEMs or CSPs, in order to support different markets. Likewise, CSPs must integrate their Update clients to work with different device vendor SDKs. The sum of these aspects create a significant integration and maintenance challenge, where there are *N* device SDKs that needs to integrate with *M* Update clients.

This document describes a standard interface for updating firmware. By providing a consistent interface for firmware update, update clients and cloud connectors can be written in a more platform independent manner. The scope of the interface is primarily SPE firmware, but can be extended by implementers to cover non-SPE firmware.

The document includes:

- A rationale for the design
- Terminology
- The supported trust models
- A high level overview of the functionality
- A detailed definition of the API
- A metadata format

Copyright © 2020-2021, Arm Limited or its affiliates. All rights reserved.
Non-confidential

2 Design goals

This section describes the main goals of the interface and the rationale.

2.1 Suitable for constrained devices

The interface is suitable for a range of embedded devices: from very simple microcontrollers with one or two flat images, to richer devices that have images for multiple subsystems and separated applications.

Consequentially, the interface is scalable and modular:

- *Scalable*: devices only need to implement the functionality that they will use.
- *Modular*: larger devices can implement more aspects of the same interface, rather than different interfaces.

A device is assumed to have enough storage for downloading updates over-the-air (OTA). The device is also assumed to have a backup or recovery capability in the event of a failed update. An implementation without such a capability may still choose to implement this interface.

This document does not cover manual reprogramming of a device using a debug interface, such as JTAG or SWD. For more information, see *PSA Debug Access Control* [PSA-DBG].

In general, these constrained devices are expected to run either bare metal programs or a real-time OS. A device that is not constrained should implement the *Embedded Base Boot Requirements* [EBBR] specification, which prescribes the UEFI Capsule Update interface. The EBBR specification defines requirements for embedded systems to enable inter-operability between SoCs, hardware platforms, firmware implementations, and operating system distributions. The aim is to establish consistent boot ABIs and behaviour so that supporting new hardware platforms does not require custom engineering work.

2.2 PSA Root of Trust update

The PSA Security Model (SM) requires all of the Mutable PSA Root of Trust firmware to be updateable. This may include bootloaders, SPM, Trusted OS, and runtime services. In some implementations, the PSA RoT may be built using a trusted subsystem with its own isolated and updateable firmware.

The PSA SM requirements for firmware update are also reflected in certifications like NIST IR 8259, ETSI EN 303 645 and PSA Certified. The PSA Certified Protection Profiles describes the following objectives, where the Target of Evaluation (TOE) refers to the PSA RoT:

- The TOE verifies the integrity and authenticity of the TOE update prior to performing the update.
- The TOE also rejects attempts of firmware downgrade.
- This security function mitigates T.UPDATE_ABUSE by preventing installation of firmware from unknown sources or installation of obsolete firmware.

T.UPDATE_ABUSE is defined as:

Copyright © 2020-2021, Arm Limited or its affiliates. All rights reserved.
Non-confidential

> “An attacker exploits a flaw in the firmware update mechanisms of the TOE, for instance by sending malformed parameters, by altering an authentic firmware update, by installing an old version of the firmware or by bypassing security checks, and installs a flawed version of the PSA updateable root of trust.”

2.3 Application Root of Trust update

In addition to the PSA Root of Trust firmware, unprivileged applications that run in the SPE require updates. The applications may be bundled as a single image or they may be separate images. This is an OEM and supply chain decision.

In some instances, the authority who signs this firmware might be different from the PSA Root of Trust vendor.

2.4 Flexiblility for different trust models

Supply chains dictate a particular trust model for a product. A device may have to support firmware updates from multiple, mutually distrustful firmware vendors.

Some regulations may also require certain implementations to use Certificate Authorities and PKI.

Furthermore, the firmware signer might not be the operator of a device. An operator of a device may have their own security policy that is complimentary to the firmware author’s policy.

The interface must be flexible enough to support a trust model needed by a particular products requirements, without imposing unnecessary requirements on constrained devices.

2.5 Protocol independence

Different protocols are used to communicate with a device depending on the industry and application context. This includes open protocols, such as LWM2M, and propietary protocols from cloud service providers. These protocols serve the specific needs of their respective markets.

Some of the protocols have metadata that is separate from the images themselves. This is taken into account.

The interface must be independent of the protocol used to recieve an update.

2.6 Transport independence

Embedded devices may recieve firmware updates OTA over different transport media depending on the industry and the application. This may include, but not limited, to Wi-Fi, LTE, LoRa, and commercial low-power wide-area networks (LPWAN).

Some devices might not be directly connected to a network but may recieve updates through a physical interface from an adjacent device, such as UART or a CAN bus.

Firmware installation can take a long time for small devices with very low bandwidth. The device may reboot several times while downloading an update.

The interface must be independent of the transport used to receieve an update.

Copyright © 2020-2021, Arm Limited or its affiliates. All rights reserved.
Non-confidential

2.7 Hardware flexibility

The interface is designed to be reasonably efficient to implement on different SoC architectures, while providing a consistent interface for Update clients or cloud connectors to target.

For example, the design should be compatible with the following types of system:

- Armv8-M based SoCs that use TrustZone, or equivalent security IP, to protect the SPE.
- SoCs using multiple CPUs, providing an isolated CPU and flash for the SPE and another for the NSPE.
- Armv7-M, Armv7-R and Armv8-R based SoCs, that use an MPU to protect the SPE.
- Armv7-A or Armv8-A based SoCs, using TrustZone to protect the SPE.

In addition to the SoC components, board level features provided by OEMs are also considered. This includes peripherals, personalization, and various storage options.

In some system designs, the application firmware may not be able to update itself without interacting with the SPE.

To enable compile-time and design-time optimization, the interface places no requirement on binary compatibility. The interface is therefore described as an Application Programming Interface (API) instead of a binary interface.

2.8 Composite devices

Some platforms have specific subsystems that are isolated from the main application or OS. These subystems have their own firmware that need updating depending on their criticality. For example, this may include radios, secure elements, secure enclaves, or other kinds of microcontroller.

An implementation should be able to support update of these using the interface if this is desired.

2.9 Room for different implementations

The interface is architectural and does not define a single implementation. For example, some implementations can:

- offer a more robust solution while others optimise for device cost.
- optimise for bandwidth efficiency while others optimise for simplicity
- provide fine grained update of personalization data while others perform monoloithic updates of all code and data
- provide enhanced security for stricter markets.

An implementation chooses what features to support. The interface may also be ported to systems without an SPE for compatibility benefits.

Copyright © 2020-2021, Arm Limited or its affiliates. All rights reserved.
Non-confidential

3 Terminology

This section describes important concepts and consistent terms.

3.1 Image

A firmware update consists of one or more images. An image can be:

- A complete image or a partial diff.
- Compressed.
- Encrypted. For example, using a pre-provisioned device key.
- Encoded.
- Code, data, or both.
- A single component or multiple components that are linked or packaged together.

An implementation may only support the properties that it needs.

An image is always associated with some signed metadata. The signed metadata describes:

- The intended device, which might be a specific instance or class.
- The intended device component. For example, the PSA RoT component.
- The digest of the image.
- Anti-rollback information.
- Any dependencies.

Additional metadata can be included to provide hints or explicit instructions on how to decrypt, decompress or install an image. It may also describe whether the component needs to be restarted.

3.2 Trust anchor

A device contains one or more **trust anchors**. A trust anchor is used to check if an image and its metadata are signed by a signing authority that the device trusts.

Each trust anchor is pre-provisioned on the device. A trust anchor can be implemented in many ways, but typically takes the form of a public key or certificate chain, depending on the complexity of the trust model.

The management and provisioning of trust anchors is not within the scope of this document.

An example of a trust anchor is the Root of Trust Public Key (ROTPK), which is defined in the Trusted Base System Architecture.

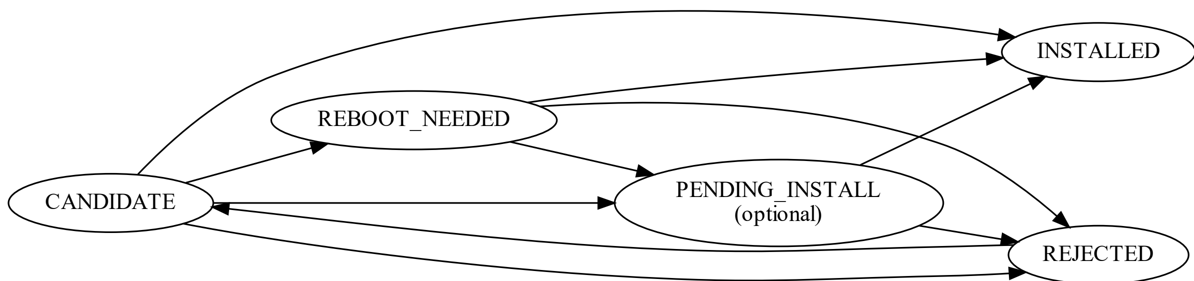
Copyright © 2020-2021, Arm Limited or its affiliates. All rights reserved.
Non-confidential

3.3 Installer

An installer is a part of the device that processes an image. An image can be in different states when it is on a device:

- When an image is not installed, the image is referred to as a **candidate image**.
- When an image is ready to run, it is installed, and the image is referred to as an **installed image**. The process of converting a candidate image to an installed image is as follows:
  1) An installer verifies that the image metadata is signed by an appropriate trust anchor.
  2) An installer verifies that all installation conditions and dependencies are satisfied.
  3) An installer takes the necessary steps to apply the update. A reboot might be needed during the installation.
- If an image requires a reboot to complete installation, then the image is set to a **reboot needed** state.
- When an image fails installation, it is referred to as a **rejected image**. If the severity of the failure is recoverable, the implementation may choose to turn the rejected image into a candidate image again. A rejected image might be marked as invalid and should be erased or overwritten.
- An implementation can optionally support a state where an image is tested for functional correctness before it is fully installed. This state is referred to as a **pending install** state. An implementation can also choose to support this state for selected types of image. This state must meet all the same security conditions as an installed image.

The architectural states and the permitted transitions are shown in the following diagram:



Implementation note:

For example, an installer might be a secure bootloader or runtime software. An installer can also be split across multiple components.

Some devices require protection against failure of a new image by retention of a known good image, normally the current image. This implies sufficient NVM to store two images. The simplest case is when both images might be stored on the device in eFlash, in which case the eFlash has to be dimensioned for two image slots, a primary slot and a secondary slot. The same principle can be applied for external flash. A system that uses two slots with a hardware banking mechanism might contain installed images in both slots.

Copyright © 2020-2021, Arm Limited or its affiliates. All rights reserved.
Non-confidential

3.4 Update client

A software component that downloads images. It may initiate the download (pull model) or wait until it recieves a notification to fetch (push model).

The Update client runs as part of the *application firmware*.

It may report device identity and installation state to a remote party using the *PSA Attestation API* [PSA-ATT], which returns an Entity Attestation Token (EAT). For example, the reported installation state can include the versions of installed images and error information of images that did not install successfully.

When it downloads an image, it transfers it to the installer using the interface described in this document.

3.5 Secure Processing Environment (SPE)

An isolated environment that hosts the PSA Root of Trust and Application Root of Trust. It is isolated from the Non-secure Processing Environment (NSPE).

The SPE protects the trust anchors.

The SPE is an installer and installs SPE images. In some constrained implementations, it may also install NSPE images.

The SPE also contains storage that is protected from the NSPE and from physical snooping.

The SPE also contains the secure bootloader needed for Trusted Boot, see *PSA Trusted Boot and Firmware Update* [PSA-TB].

The SPE has a means of recovery if a newly installed image fails after an update.

3.6 Staging area

A staging area is an area of memory used to verify an image. The staging area might be in NVM or RAM and thus not guaranteed to be non-volatile. The choice of memory depends on resource constraints and the nature of an update.

The staging area might also be protected by a Secure Processing Environment or resident on another processor.

The size of a staging area is pre-determined by the implementation. This avoids contention of system resources.

An implementation might have one staging area per type of image.

The *Update client* is responsible for deciding when to erase the staging area.

Copyright © 2020-2021, Arm Limited or its affiliates. All rights reserved.
Non-confidential

4 Trust model and scenarios

The following actor definitions are taken from the IETF SUIT architecture draft:

- Author: The author is the entity that creates the firmware image. There may be multiple authors in a system either when a device consists of multiple microcontrollers or when the the final firmware image consists of software components from multiple companies.
- Device Operator: The actor responsible for the day-to-day operation of a fleet of devices.
- Network Operator: The actor responsible for the operation of a network to which devices connect.
- Status Tracker: The status tracker offers device management functionality to retrieve information about the installed firmware on a device and other device characteristics (including free memory and hardware components), to obtain the state of the firmware update cycle the device is currently in, and to trigger the update process. The deployment of status trackers is flexible and they may be used in cloud-based servers, on-premise servers, embedded in edge computing device, etc.

The implementor of the interface described within this document always:

- verifies that the image metadata is signed by a trusted author before installation
- verifies that the image metadata complies with the platform's security policy
- verifies the image itself using information from the metadata
- trusts authors for specific purposes only (e.g. the NSPE author cannot directly overwrite images in the SPE)

The *Update client*, the consumer of the interface, is only trusted for the following purposes:

- Downloading images from an approved image repository decided by the Device Operator.
- Selecting the images that the Status Tracker wants installed.
- Obeying the restrictions of the Network Operator.

Depending on the threat model, the following should also be considered:

- If an image is tested before it is permanently installed then a component in the system needs to approve or reject images if they detect a fault during testing. Typically, only a single component can vouch for system wide functional correctness. The update client might be trusted to make decisions during a test.
- Certain images might contain confidential code or data that must never be exposed to the NSPE. Therefore, some images might be encrypted with a key that is private to the SPE and when decryption occurs the data is stored in storage that is inaccessible from the NSPE.

Copyright © 2020-2021, Arm Limited or its affiliates. All rights reserved.
Non-confidential

5 Design overview

The document *A Firmware Update Architecture for Internet of Things* [SUIT-ARCH] describes an example flow and the expected device side actions:

- Query image information
- Validate image metadata
- Store image
- Verify image
- Trigger reboot

The design of the interface offers functions for these actions. The interface contains three major classes of error codes: storage errors, security errors, and dependency errors. The caller must be prepared to handle these.

The interface is expected to be provided by an implementation that runs in the *NSPE*. However, it may be necessary to protect certain functions using the SPE on some platforms.

5.1 Mandatory functions

The interface supports these actions using the following calls.

5.1.1 Querying installed images

The caller calls `psa_fwu_query` to fetch information about firmware images on the device. This includes state for installed images, rejected images, and candidate images. This information is expected to be passed to a remote operator or status tracker. A local *Update client* may also use this information to make a local decision. For example, the data may be used to avoid fetching images unnecessarily if they are already on the device.

To satisfy Section 2.1, the query function uses an iterator to minimize memory footprint. The caller may relay this information to a remote status tracker. Eventually, new firmware is downloaded to the device by the caller.

Each image has its own local identifier that represents the type of image. This is known as an image ID. Queries are based on an image ID.

The image ID of each updatable image can be discovered using the provided iterator functions. The query function returns global identification information about the image, such as the associated Vendor ID, Class ID, and the hash of the public signing key.

5.1.2 Image storing

Each image has its own image ID that represents the type of image. Each image has its own staging area. The image ID is used by the implementation to determine the appropriate staging area.

Copyright © 2020-2021, Arm Limited or its affiliates. All rights reserved.
Non-confidential

The caller uses psa_fwu_write to write a candidate image to its staging area. To satisfy Section 2.1 the caller can use this operation to write an image incrementally in blocks. On a rich device with plenty of memory, the caller uses this as a single-part function. A number of errors can be detected and returned by this call, and can depend on the qualities of the implementation.

A staging area can be erased using psa_fwu_abort.

5.1.3 Metadata storage

If the image does not embed metadata, then a standalone metadata object can be associated with the same image by calling psa_fwu_set_manifest with the same image ID.

5.1.4 Verify image

The caller finishes the firmware update process with psa_fwu_install.

On success, the implementation checks any necessary preconditions and prepares installation.

On some implementations, this will starts an integrity check on the firmware image based on the metadata. On other implementations, the validation of the metadata is deferred to the bootloader when the platform next resets.

The implementation indicates whether a reboot is required to complete installation by using a return value.

The implementation indicates whether a dependency is missing by using a set of output parameters.

If a dependency is missing, the dependency is specified in two output parameters, the image ID of the missing image and the required version. The caller then fetches the required image and repeats the process.

5.1.5 Triggering a reboot

If a reboot is required to complete the firmware update process then the caller uses psa_fwu_request_reboot to restart the platform. The caller chooses when is an appropriate time to reboot the platform.

5.2 Optional functions

The following image management functions must be implemented if a PENDING_INSTALL state is supported:

- A function, psa_fwu_request_rollback, is provided for *application firmware* to request the device to roll back the recently applied updates. This is for scenarios where the newly updated firmware detects a fatal problem with the update. The implementation may deny this request if this is prevented by security policy (e.g. rollback protection). An implementation can also choose not to support this.
- A function, psa_fwu_accept, is provided for *application firmware* to indicate whether the recently applied updates are working correctly. This is for scenarios where the newly installed firmware must be tested before it is permanently installed. An example of a test is a built in self-test.

Copyright © 2020-2021, Arm Limited or its affiliates. All rights reserved.
Non-confidential

5.3 State transitions for an image

The permitted state transitions are shown in the following diagram:

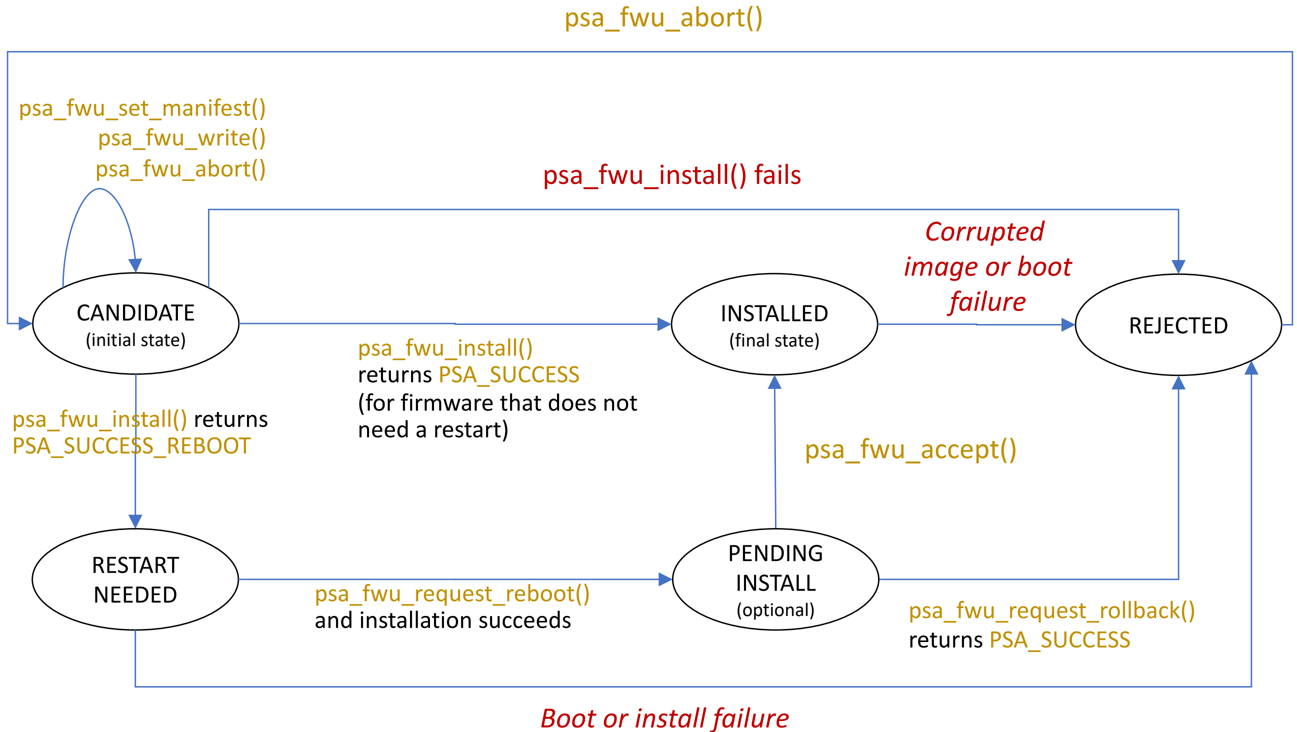


Figure 1 Permitted state transitions for an image using the API.

Every firmware update image begins in the CANDIDATE state. All successful firmware update images reach the INSTALLED state.

The state of an image can be queried using `psa_fwu_query`.

5.4 Dependencies

An image can have a dependency on another image. When an image has a dependency it cannot be installed until all of its dependencies are satisfied. This means that all images must be written prior to calling `psa_fwu_install`. Dependencies are described within image metadata.

If a dependency is not explicit then the implementation is not guaranteed to detect compatibility issues. For instance, if an installation contains two images, A and B, and A depends on B, then the system will not be able to detect incompatibilities if B is upgraded in isolation. If mutual dependency is required, then A should specify a dependency on B and B should specify a dependency on A.

A dependency consists of an image ID and version.

Copyright © 2020-2021, Arm Limited or its affiliates. All rights reserved.
Non-confidential

6 Image metadata

This section describes the format options, usage models, and a metadata example.

6.1 Format profiles

The choice of metadata format depends on the software development practices and interoperability requirements of a particular deployment.

At least one of the following formats must be supported by the interface:

Table 3 Supported formats per segment

Profile ID	Profile name	Description
`0x1`	IETF SUIT CBOR	The full specification for the format is described in *A Concise Binary Object Representation (CBOR)-based Serialization Format for the Software Updates for Internet of Things (SUIT) Manifest* [SUIT]. The manifest contains an authentication header that is signed using CBOR Object Signing and Encryption (COSE). An example is provided in the following sections. Arm recommends this profile.
`0x2`	Implementation specified	A proprietary implementation. A description of the format should be made publicly available. This option is not preferable because it provides no interoperability.

It is expected that implementations are built to only support a single profile. However, an implementation can choose to support multiple profiles if necessary.

All Profile IDs not specified here are reserved for future use.

6.2 Usage models

The metadata can be supplied using one of two methods, depending on the protocol used:

- **Embedded method**: The metadata is prepended to each image.
- **Standalone method**: The metadata is standalone and separate from images. The metadata can describe metadata for multiple images. In this case, `psa_fwu_set_manifest` is used.

The decision is made by the caller depending on the firmware update protocol.

6.3 Example metadata

The manifest format described contains many optional and extensible fields covering the goals of this specification.

Copyright © 2020-2021, Arm Limited or its affiliates. All rights reserved.
Non-confidential

The following example shows an example of a single image specified as a SUIT CBOR manifest. Further examples are provided in at this link.

6.3.1 Summary

The example manifest contains two major subparts:

- Manifest authentication header. The authentication header authenticates the manifest data. It includes the following fields:
  - Signature information (`alg` and `payload`).
  - The raw signature (`signature`).
- Manifest content. It includes the following fields:
  - Manifest format version (`manifest-version`).
  - Manifest Sequence number (`manifest-sequence-number`).
  - Class UUID and Vendor UUID (`class-id` and `vendor-id`).
  - Image digest information (e.g. `image-size`, `algorithm-id`, `digest-bytes`)
  - Installation instructions (in this example there is only the `condition-image-match` and `directive-run` directive)

Implementation note:

An example open-source tool for generating SUIT manifests is available at :*Example code to generate and parse SUIT manifests* [SUIT-CODE]. The project also contains a small parser that can be included in small microcontrollers.

6.3.2 CBOR

The example manifest below conforms to IETF SUIT and is written in CBOR diagnostic language for illustrative purposes.

```
{
    / authentication-wrapper / 2:h'81d28443a10126a058248202582064d8094
da3ef71c5971b7b84e7f4be1f56452c32fdde7bc1c70889112f1d5d9958407d637397e
12abdd41bc026a8e8a22f0f902a5b972e7786d570a37ac43c370b64a6946b0311f059c
a01d40f74d88d6fd7193baa36f5cf20aa57c46a0411a6b704' / [
        18([
                / protected / h'a10126' / {
                    / alg / 1:-7 / ES256 /,
                } /,
                / unprotected / {
                },
                / payload / h'8202582064d8094da3ef71c5971b7b84e7f4be1f
56452c32fdde7bc1c70889112f1d5d99' / [
                    / algorithm-id / 2 / sha256 /,
                    / digest-bytes /
h'64d8094da3ef71c5971b7b84e7f4be1f56452c32fdde7bc1c70889112f1d5d99'
                ] /,
```

(continues on next page)

Copyright © 2020-2021, Arm Limited or its affiliates. All rights reserved.
Non-confidential

(continued from previous page)

```
                / signature / h'7d637397e12abdd41bc026a8e8a22f0f902a5b
972e7786d570a37ac43c370b64a6946b0311f059ca01d40f74d88d6fd7193baa36f5cf
20aa57c46a0411a6b704'
            ])
    ] /,
    / manifest / 3:h'a50101020103585ea20244818141000458548614a40150fa6
b4a53d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab4503820
2582000112233445566778899aabbccddeeff0123456789abcdeffedcba98765432100
e1987d001f602f60a438203f60c438217f6' / {
        / manifest-version / 1:1,
        / manifest-sequence-number / 2:1,
        / common / 3:h'a20244818141000458548614a40150fa6b4a53d5ad5fdfb
e9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab450382025820001122334
45566778899aabbccddeeff0123456789abcdeffedcba98765432100e1987d001f602f
6' / {
            / components / 2:h'81814100' / [
                [h'00']
            ] /,
            / common-sequence / 4:h'8614a40150fa6b4a53d5ad5fdfbe9de663
e4d41ffe02501492af1425695e48bf429b2d51f2ab4503820258200011223344556677
8899aabbccddeeff0123456789abcdeffedcba98765432100e1987d001f602f6' / [
                / directive-override-parameters / 20,{
                    / vendor-id /
1:h'fa6b4a53d5ad5fdfbe9de663e4d41ffe' / fa6b4a53-d5ad-5fdf-be9d-e663e4d41ffe /,
                    / class-id / 2:h'1492af1425695e48bf429b2d51f2ab45'
/ 1492af14-2569-5e48-bf42-9b2d51f2ab45 /,
                        / image-digest / 3:[
                        / algorithm-id / 2 / sha256 /,
                        / digest-bytes /
h'00112233445566778899aabbccddeeff0123456789abcdeffedcba9876543210'
                    ],
                    / image-size / 14:34768,
                } ,
                / condition-vendor-identifier / 1,F6 / nil / ,
                / condition-class-identifier / 2,F6 / nil /
            ] /,
        } /,
        / validate / 10:h'8203f6' / [
            / condition-image-match / 3,F6 / nil /
        ] /,
        / run / 12:h'8217f6' / [
            / directive-run / 23,F6 / nil /
        ] /,
    } /,
}
```

The total size of this example with COSE authentication object is estimated to be 231 bytes.

Copyright © 2020-2021, Arm Limited or its affiliates. All rights reserved.
Non-confidential

7 API reference

The API is defined in the C language. The API make use of standard C data types as defined in the ISO C99 specification.

7.1 Library conventions

Almost all functions return a status indication of type `psa_status_t`. This is an enumeration of integer values, with the value 0 (`PSA_SUCCESS`) or greater indicating successful operation and other values indicating errors.

Unless specified otherwise, if multiple error conditions apply, an implementation is free to return any of the applicable error codes. The choice of error code is considered an implementation quality issue. Different implementations can make different choices, for example to favor code size over ease of debugging or vice versa.

If the behavior is undefined, for example, if a function receives an invalid pointer as a parameter, this specification makes no guarantee that the function will return an error. Implementations are encouraged to return an error or halt the application in a manner that is appropriate for the platform if the undefined behavior condition can be detected. However, application developers need to be aware that undefined behavior conditions cannot be detected in general.

7.2 Behavior on error

All function calls must be implemented atomically:

- When a function returns a type other than psa_status_t, the requested action has been carried out.
- When a function returns the status `PSA_SUCCESS` or `PSA_SUCCESS_xxx`, the requested action has been carried out.
- When a function returns another status of type psa_status_t, no action has been carried out. The content of the output parameters is undefined, but otherwise the state of the system has not changed, except as described below.

Unless otherwise documented, the content of output parameters is not defined when a function returns a status other than `PSA_SUCCESS`. It is recommended that implementations set output parameters to safe defaults to avoid limit risk, in case the caller does not properly handle all errors.

7.3 Pointer conventions

Unless explicitly stated in the documentation of a function, all pointers must be valid pointers to an object of the specified type.

Copyright © 2020-2021, Arm Limited or its affiliates. All rights reserved.
Non-confidential

7.4 Macros

7.4.1 Library versioning

PSA_FWU_API_VERSION_MAJOR (macro)

The major version of this implementation of the API.

```
#define PSA_FWU_API_VERSION_MAJOR 0
```

PSA_FWU_API_VERSION_MINOR (macro)

The minor version of this implementation of the API.

```
#define PSA_FWU_API_VERSION_MINOR 7
```

7.4.2 Image transfer

PSA_FWU_MAX_BLOCK_SIZE (macro)

The maximum permitted size for `block` in `psa_fwu_write`, in bytes. The specific value is IMPLEMENTATION DEFINED and must be greater than `0`.

```
#define PSA_FWU_MAX_BLOCK_SIZE IMPDEF
```

Note:

Implementations that aim to support post quantum security are recommended to provide a minimum of 4096, especially if there is an intention to support stateful hash-based signatures.

7.4.3 Digest size

PSA_FWU_MAX_DIGEST_SIZE (macro)

The maximum size of an image digest, in bytes. This is dependent on the hash algorithm used. The value is IMPLEMENTATION DEFINED.

```
#define PSA_FWU_MAX_DIGEST_SIZE IMPDEF
```

7.4.4 Image states

PSA_IMAGE_UNDEFINED (macro)

```
#define PSA_IMAGE_UNDEFINED 0
```

PSA_IMAGE_CANDIDATE (macro)

Copyright © 2020-2021, Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
#define PSA_IMAGE_CANDIDATE 1
```

PSA_IMAGE_INSTALLED (macro)

```
#define PSA_IMAGE_INSTALLED 2
```

PSA_IMAGE_REJECTED (macro)

```
#define PSA_IMAGE_REJECTED 3
```

PSA_IMAGE_PENDING_INSTALL (macro)

```
#define PSA_IMAGE_PENDING_INSTALL 4
```

PSA_IMAGE_REBOOT_NEEDED (macro)

```
#define PSA_IMAGE_REBOOT_NEEDED 5
```

7.4.5 Image flags

PSA_IMAGE_FLAG_VOLATILE_STAGING (macro)

```
#define PSA_IMAGE_FLAG_VOLATILE_STAGING (0x00000001)
```

If set then image data written to the staging area will not maintained across a system reset.

If not set then image data written to the staging area is guaranteed to exist after a system reset.

PSA_IMAGE_FLAG_ENCRYPTION (macro)

```
#define PSA_IMAGE_FLAG_ENCRYPTION (0x00000002)
```

This flag describes whether an image is encrypted or not during an update.

If set then this type of image must be provided in encrypted form when installing.

If not set then this type of image must be provided in unencrypted form when installing.

7.5 Types

7.5.1 `psa_image_version_t` (struct)

Version information about an image

Copyright © 2020-2021, Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
typedef struct psa_image_version_t {
    uint16_t major;
    uint16_t minor;
    uint16_t patch;
} psa_image_version_t;
```

Fields

`major`	The major version of an image.
`minor`	The minor version of an image. If the image has no minor version then this field is set to 0.
`patch`	The patch version of an image. If the image has no patch version then this field is set to 0.

7.5.2 psa_staging_info_t (struct)

A container with information about how to update the image.

```
typedef struct psa_staging_info_t {
    uint8_t flags;
    uint8_t metadata_format;
    size_t max_size;
    psa_hash_t key_id;
} psa_staging_info_t;
```

Fields

`flags`	A set of flags that describe extra information about the staging area. All unused flag values are reserved for future use by this specification and must be zero. See PSA_IMAGE_FLAG_VOLATILE_STAGING (macro) for the flag values.
`metadata_format`	The expected format of the image's signed metadata.
`max_size`	The maximum possible size of the image in bytes.
`key_id`	If `PSA_IMAGE_FLAG_ENCRYPTION` is set in `flags` then this field contains the key identifier. The key identifier is a cryptographic hash of the decryption key. If `PSA_IMAGE_FLAG_ENCRYPTION` is not set in `flags` then this field must be zero.

7.5.3 psa_image_info_t (struct)

A container containing status information about an image.

```
typedef struct psa_image_info_t {
    struct psa_image_id_t image_id;
    psa_uuid_t vendor_id;
    psa_uuid_t class_id;
```

(continues on next page)

Copyright © 2020-2021, Arm Limited or its affiliates. All rights reserved.
Non-confidential

(continued from previous page)

```
    struct psa_image_version_t version;
    uint8_t state;
    psa_staging_info_t staging;
    uint32_t error;
    uint8_t digest[PSA_FWU_MAX_DIGEST_SIZE];
} psa_image_info_t;
```

Fields

`image_id`
The assigned image ID.

`vendor_id`
The expected vendor ID from the image metadata.

`class_id`
The expected class ID from the image metadata.

`version`
The version of the image.

`state`
An integer that describes the current state of an image. Contains one of the following values:

- `PSA_IMAGE_UNDEFINED`
- `PSA_IMAGE_CANDIDATE`
- `PSA_IMAGE_INSTALLED`
- `PSA_IMAGE_REJECTED`
- `PSA_IMAGE_PENDING_INSTALL`

> **Note:**
> I think we want this information to be richer and extensible. For example, we could define a range of values defined by Arm and a range of values for implementers. For example. An implementation can provide extra information for debug reasons.

`staging`
A container with information about how to update the image.

`error`
An application-specific error that caused the firmware to roll back. If `psa_fwu_request_rollback` is not supported, then this field must be 0.

> **Note:**
> What about error codes from bootloader? Are there use cases for a bootloader sending a dynamic amount of data?

`digest`
The digest of the image.

7.5.4 `psa_uuid_t` (struct)

A 128-bit universally unique identifier. UUIDs MUST be created according to *A Universally Unique IDentifier (UUID) URN Namespace* [RFC4122]. UUIDs should use versions 3, 4, or 5.

Copyright © 2020-2021, Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
typedef struct psa_uuid_t {
    uint32_t uuid[4];
} psa_uuid_t;
```

Fields

`uuid`

7.5.5 psa_image_id_t (type)

A fixed size integer for the type of image that needs installing. This ID can be returned by a dependency error from `psa_fwu_install`. The caller must use the ID to download or locate the appropriate image to install.

```
typedef uint32_t psa_image_id_t;
```

7.5.6 psa_fwu_iterator_t (type)

An iterator object. Used with `psa_fwu_query` to iterate through all the image information.

The definition is IMPLEMENTATION DEFINED.

```
typedef /*...*/ psa_fwu_iterator_t;
```

7.5.7 psa_hash_t (struct)

A cryptographic hash.

```
struct psa_hash_t {
    uint8_t value[PSA_FWU_MAX_DIGEST_SIZE];
};
```

Fields

`value`

7.6 Status codes

7.6.1 psa_status_t (type)

Function return status.

```
typedef int32_t psa_status_t;
```

This is either `PSA_SUCCESS`, which is zero, indicating success; or a small negative value indicating that an error occurred. Errors are encoded as one of the `PSA_ERROR_xxx` values defined here.

Copyright © 2020-2021, Arm Limited or its affiliates. All rights reserved.
Non-confidential

7.6.2 PSA_SUCCESS (macro)

The action was completed successfully.

```
#define PSA_SUCCESS ((psa_status_t)0)
```

7.6.3 PSA_SUCCESS_REBOOT (macro)

The action was completed successfully and requires a system reboot to complete installation.

```
#define PSA_SUCCESS_REBOOT ((psa_status_t)+1)
```

7.6.4 PSA_SUCCESS_RESTART (macro)

The action was completed successfully and requires a restart of the component to complete installation.

```
#define PSA_SUCCESS_RESTART ((psa_status_t)+2)
```

7.6.5 PSA_SUCCESS_DEPENDENCY_NEEDED (macro)

The action was completed successfully and requires the installation of a dependency to complete installation.

```
#define PSA_SUCCESS_DEPENDENCY_NEEDED ((psa_status_t)+3)
```

7.7 Error codes

The following are the possible error codes that can be returned to the caller.

7.7.1 PSA_ERROR_GENERIC_ERROR (macro)

An error occurred that does not correspond to any defined failure cause.

```
#define PSA_ERROR_GENERIC_ERROR ((psa_status_t)-132)
```

Implementations can use this error code if none of the other standard error codes are applicable.

7.7.2 PSA_ERROR_NOT_SUPPORTED (macro)

The requested operation or a parameter is not supported by this implementation.

```
#define PSA_ERROR_NOT_SUPPORTED ((psa_status_t)-134)
```

If a combination of parameters is recognized and identified as not valid, return `PSA_ERROR_INVALID_ARGUMENT` instead.

Copyright © 2020-2021, Arm Limited or its affiliates. All rights reserved.
Non-confidential

7.7.3 PSA_ERROR_NOT_PERMITTED (macro)

The requested action is denied by a policy.

```
#define PSA_ERROR_NOT_PERMITTED ((psa_status_t)-133)
```

It is recommended that implementations return this error code when the parameters are recognized as valid and supported, and a policy explicitly denies the requested operation.

If a subset of the parameters of a function call identify a forbidden operation, and another subset of the parameters are not valid or not supported, it is unspecified whether the function returns PSA_ERROR_NOT_PERMITTED, PSA_ERROR_NOT_SUPPORTED or PSA_ERROR_INVALID_ARGUMENT.

7.7.4 PSA_ERROR_DOES_NOT_EXIST (macro)

Asking for an item that doesn't exist.

```
#define PSA_ERROR_DOES_NOT_EXIST ((psa_status_t)-140)
```

7.7.5 PSA_ERROR_INVALID_ARGUMENT (macro)

The parameters passed to the function are invalid.

```
#define PSA_ERROR_INVALID_ARGUMENT ((psa_status_t)-135)
```

Implementations can return this error any time a parameter or combination of parameters are recognized as invalid.

7.7.6 PSA_ERROR_INSUFFICIENT_MEMORY (macro)

There is not enough runtime memory.

```
#define PSA_ERROR_INSUFFICIENT_MEMORY ((psa_status_t)-141)
```

If the action is carried out across multiple security realms, this error can refer to available memory in any of the security realms.

7.7.7 PSA_ERROR_INSUFFICIENT_STORAGE (macro)

There is not enough persistent storage.

```
#define PSA_ERROR_INSUFFICIENT_STORAGE ((psa_status_t)-142)
```

Functions that modify the key storage return this error code if there is insufficient storage space on the host media. In addition, many functions that do not otherwise access storage might return this error code if the implementation requires a mandatory log entry for the requested action and the log storage space is full.

Copyright © 2020-2021, Arm Limited or its affiliates. All rights reserved.
Non-confidential

7.7.8 PSA_ERROR_COMMUNICATION_FAILURE (macro)

There was a communication failure inside the implementation.

```
#define PSA_ERROR_COMMUNICATION_FAILURE ((psa_status_t)-145)
```

This can indicate a communication failure between the application and an external processor or between the processor and an external volatile or persistent memory. A communication failure can be transient or permanent depending on the cause.

Warning: If a function returns this error, it is undetermined whether the requested action has completed. Returning PSA_SUCCESS is recommended on successful completion whenever possible, however functions can return PSA_ERROR_COMMUNICATION_FAILURE if the requested action was completed successfully in an external processor but there was a breakdown of communication before the processor could report the status to the application.

7.7.9 PSA_ERROR_STORAGE_FAILURE (macro)

There was a storage failure that might have led to data loss.

```
#define PSA_ERROR_STORAGE_FAILURE ((psa_status_t)-146)
```

This error indicates that some persistent storage could not be read or written by the implementation. It does not indicate the following situations, which have specific error codes:

- A corruption of volatile memory - use PSA_ERROR_CORRUPTION_DETECTED.
- A communication error between the processor and its external storage - use PSA_ERROR_COMMUNICATION_FAILURE.
- When the storage is in a valid state but is full - use PSA_ERROR_INSUFFICIENT_STORAGE.
- When the storage or stored data is corrupted - use PSA_ERROR_DATA_CORRUPT.
- When the stored data is not valid - use PSA_ERROR_DATA_INVALID.

A storage failure does not indicate that any data that was previously read is invalid. However this previously read data might no longer be readable from storage.

When a storage failure occurs, it is no longer possible to ensure the global integrity of the keystore. Depending on the global integrity guarantees offered by the implementation, access to other data might fail even if the data is still readable but its integrity cannot be guaranteed.

It is recommended to only use this error code to report a permanent storage corruption. However application writers must keep in mind that transient errors while reading the storage might be reported using this error code.

7.7.10 PSA_ERROR_DATA_CORRUPT (macro)

Stored data has been corrupted.

Copyright © 2020-2021, Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
#define PSA_ERROR_DATA_CORRUPT ((psa_status_t)-152)
```

This error indicates that some persistent storage has suffered corruption. It does not indicate the following situations, which have specific error codes:

- A corruption of volatile memory - use PSA_ERROR_CORRUPTION_DETECTED.
- A communication error between the processor and its external storage - use PSA_ERROR_COMMUNICATION_FAILURE.
- When the storage is in a valid state but is full - use PSA_ERROR_INSUFFICIENT_STORAGE.
- When the storage fails for other reasons - use PSA_ERROR_STORAGE_FAILURE.
- When the stored data is not valid - use PSA_ERROR_DATA_INVALID.

Note that a storage corruption does not indicate that any data that was previously read is invalid. However this previously read data might no longer be readable from storage.

When a storage failure occurs, it is no longer possible to ensure the global integrity of the keystore. Depending on the global integrity guarantees offered by the implementation, access to other data might fail even if the data is still readable but its integrity cannot be guaranteed.

It is recommended to only use this error code to report when a storage component indicates that the stored data is corrupt, or fails an integrity check.

7.7.11 PSA_ERROR_DATA_INVALID (macro)

Data read from storage is not valid for the implementation.

```
#define PSA_ERROR_DATA_INVALID ((psa_status_t)-153)
```

This error indicates that some data read from storage does not have a valid format. It does not indicate the following situations, which have specific error codes:

- When the storage or stored data is corrupted - use PSA_ERROR_DATA_CORRUPT.
- When the storage fails for other reasons - use PSA_ERROR_STORAGE_FAILURE.
- An invalid argument to the API - use PSA_ERROR_INVALID_ARGUMENT.

This error is typically a result of an integration failure, where the implementation reading the data is not compatible with the implementation that stored the data.

It is recommended to only use this error code to report when data that is successfully read from storage is invalid.

7.7.12 PSA_ERROR_HARDWARE_FAILURE (macro)

A hardware failure was detected.

```
#define PSA_ERROR_HARDWARE_FAILURE ((psa_status_t)-147)
```

A hardware failure can be transient or permanent depending on the cause.

Copyright © 2020-2021, Arm Limited or its affiliates. All rights reserved.
Non-confidential

7.7.13 PSA_ERROR_CORRUPTION_DETECTED (macro)

A tampering attempt was detected.

```
#define PSA_ERROR_CORRUPTION_DETECTED ((psa_status_t)-151)
```

If an application receives this error code, there is no guarantee that previously accessed or computed data was correct and remains confidential. In this situation, it is recommended that applications perform no further security functions and enter a safe failure state.

Implementations can return this error code if they detect an invalid state that cannot happen during normal operation and that indicates that the implementation's security guarantees no longer hold. Depending on the implementation architecture and on its security and safety goals, the implementation might forcibly terminate the application.

This error indicates an attack against the application. Implementations must not return this error code as a consequence of the behavior of the application itself.

7.7.14 PSA_ERROR_INVALID_SIGNATURE (macro)

The signature, MAC or hash is incorrect.

```
#define PSA_ERROR_INVALID_SIGNATURE ((psa_status_t)-149)
```

Verification functions return this error if the verification calculations completed successfully, and the value to be verified was determined to be incorrect.

If the value to verify has an invalid size, implementations can return either PSA_ERROR_INVALID_ARGUMENT or PSA_ERROR_INVALID_SIGNATURE.

7.7.15 PSA_ERROR_INSUFFICIENT_DATA (macro)

Return this error when there's insufficient data when attempting to read from a resource.

```
#define PSA_ERROR_INSUFFICIENT_DATA ((psa_status_t)-143)
```

7.7.16 PSA_ERROR_WRONG_DEVICE (macro)

```
#define PSA_ERROR_WRONG_DEVICE ((psa_status_t)-155)
```

7.7.17 PSA_ERROR_DEPENDENCY_NEEDED (macro)

```
#define PSA_ERROR_DEPENDENCY_NEEDED ((psa_status_t)-156)
```

7.7.18 PSA_ERROR_CURRENTLY_INSTALLING (macro)

```
#define PSA_ERROR_CURRENTLY_INSTALLING ((psa_status_t)-157)
```


Copyright © 2020-2021, Arm Limited or its affiliates. All rights reserved.
Non-confidential

7.7.19 PSA_ERROR_ALREADY_INSTALLED (macro)

```
#define PSA_ERROR_ALREADY_INSTALLED ((psa_status_t)-158)
```

7.7.20 PSA_ERROR_INSTALL_INTERRUPTED (macro)

```
#define PSA_ERROR_INSTALL_INTERRUPTED ((psa_status_t)-159)
```

7.7.21 PSA_ERROR_FLASH_ABUSE (macro)

```
#define PSA_ERROR_FLASH_ABUSE ((psa_status_t)-160)
```

7.7.22 PSA_ERROR_INSUFFICIENT_POWER (macro)

```
#define PSA_ERROR_INSUFFICIENT_POWER ((psa_status_t)-161)
```

7.7.23 PSA_ERROR_DECRYPTION_FAILURE (macro)

```
#define PSA_ERROR_DECRYPTION_FAILURE ((psa_status_t)-162)
```

7.7.24 PSA_ERROR_MISSING_MANIFEST (macro)

```
#define PSA_ERROR_MISSING_MANIFEST ((psa_status_t)-163)
```

7.8 Functions

7.8.1 psa_fwu_query (function)

Returns information for an image of a particular image ID.

```
psa_status_t psa_fwu_query(psa_image_id_t image_id,
                           psa_image_info_t *info);
```

Parameters

image_id	The image ID of the image to query.
info	Output parameter for image information related to the image ID.

Returns: psa_status_t

PSA_SUCCESS	Image information has been returned.
PSA_ERROR_NOT_PERMITTED	The caller is not authorized to access platform version information.

Copyright © 2020-2021, Arm Limited or its affiliates. All rights reserved.
Non-confidential

7.8.2 `psa_fwu_set_manifest` (function)

Stores a manifest object and associates it with a particular image ID.

This function is optional, and if not implemented shall return PSA_ERROR_NOT_SUPPORTED.

```
psa_status_t psa_fwu_set_manifest(psa_image_id_t image_id,
                                  const void *manifest,
                                  size_t manifest_size,
                                  psa_hash_t *manifest_dependency);
```

Parameters

image_id	The identifier of the image type.
manifest	A pointer to a buffer containing a manifest object.
manifest_size	The size of the manifest parameter.
manifest_dependency	Output parameter containing the hash of a required manifest when PSA_ERROR_DEPENDENCY_NEEDED is returned.

Returns: psa_status_t

PSA_SUCCESS	The manifest is persisted.
PSA_ERROR_NOT_PERMITTED	The manifest is too old to be installed. If the image metadata contains a timestamp, and it has expired, then this error is also returned.
PSA_ERROR_WRONG_DEVICE	The manifest is not intended for this device.
PSA_ERROR_INVALID_SIGNATURE	The manifest signature is not valid.
PSA_ERROR_DEPENDENCY_NEEDED	A different manifest is needed.
PSA_ERROR_INVALID_ARGUMENT	Parameter size is 0 or a pointer parameter is NULL.
PSA_ERROR_COMMUNICATION_FAILURE	The system could not communicate with the installer.
PSA_ERROR_NOT_SUPPORTED	This function is not implemented.
PSA_ERROR_CURRENTLY_INSTALLING	An existing manifest for image ID is currently being installed and is locked from writing. For example, psa_fwu_install is currently executing.
PSA_ERROR_GENERIC_ERROR	A fatal error occured.

Description

Note:

Rationale

It is better to have a separate function for manifest related data rather than overload existing functions. Otherwise, we burden non-manifest users with configuring parameters correctly.

Similarly, if a system does not support a manifest, this function can not be supported using PSA_ERROR_NOT_SUPPORTED.

Copyright © 2020-2021, Arm Limited or its affiliates. All rights reserved.
Non-confidential

7.8.3 psa_fwu_write (function)

Writes an image to its staging area.

If the image size is less than or equal to PSA_FWU_MAX_BLOCK_SIZE, the caller can send the entire image in one call.

If the image size is greater than PSA_FWU_MAX_BLOCK_SIZE, the caller must send parts of the image by calling psa_fwu_write multiple times with different data blocks.

Once complete, the caller calls psa_fwu_install to install the candidate image.

```
psa_status_t psa_fwu_write(psa_image_id_t image_id,
                           size_t image_offset,
                           const void *block,
                           size_t block_size);
```

Parameters

image_id	The identifier of the image type.
image_offset	The offset of the image being passed into block, in bytes.
block	A buffer containing a block of image data. This might be a complete image or a subset.
block_size	Size of block. The size must not be greater than PSA_FWU_MAX_BLOCK_SIZE.

Returns: psa_status_t

PSA_SUCCESS	The data in block has been successfully stored.
PSA_ERROR_INVALID_ARGUMENT	One of the following error conditions occured: • The parameter size is greater than PSA_FWU_MAX_BLOCK_SIZE. • The parameter size is 0. • The combination of offset and size is out of bounds.
PSA_ERROR_INSUFFICIENT_MEMORY	There is not enough memory to process the update.
PSA_ERROR_INSUFFICIENT_STORAGE	There is not enough storage to process the update.
PSA_ERROR_COMMUNICATION_FAILURE	The system could not communicate with the installer.
PSA_ERROR_FLASH_ABUSE	The system is defending against quick flash exhaustion and is in a time-out period. The time-out period is IMPLEMENTATION DEFINED.
PSA_ERROR_DATA_CORRUPT	Data corruption has been detected. For example, an implementation that uses stream signatures may be able to detect corruption early.
PSA_ERROR_INSUFFICIENT_POWER	There is not enough power to complete the operation.
PSA_ERROR_GENERIC_ERROR	A fatal error occured.
PSA_ERROR_CURRENTLY_INSTALLING	The image is currently locked for writing. For example, psa_fwu_install is currently executing.

Description

Copyright © 2020-2021, Arm Limited or its affiliates. All rights reserved.
Non-confidential

It is the caller's responsibility to account for how much data is written at which offset. If no persistent storage is directly available for the caller to perform accounting, then the caller can use a different storage mechanism, such as the PSA Storage API.

Note:

Open point

Should we provide a mechanism to read firmware? For example psa_fwu_read?

7.8.4 `psa_fwu_install` (function)

Starts the installation of an image.

The authenticity and integrity of the image is checked during installation. If a reboot is required to complete installation then the implementation can choose to defer the authenticity checks to that point.

While this function executes, calls to `psa_fwu_write` with the image ID or a dependent image ID are rejected.

If an image dependency is missing then an error is returned.

Concurrent calls to this function with the same or dependent image IDs are not permitted and return an error.

```
psa_status_t psa_fwu_install(psa_image_id_t image_id,
                             psa_image_id_t *dependency_image_id,
                             psa_image_version_t *dependency_version);
```

Parameters

`image_id`	The identifier of the image to install.
`dependency_image_id`	If `PSA_SUCCESS_DEPENDENCY_NEEDED` is returned, this parameter is filled with dependency information.
`dependency_version`	If `PSA_SUCCESS_DEPENDENCY_NEEDED` is returned, this parameter is filled with the minimum required version for the dependency.

Returns: `psa_status_t`

`PSA_SUCCESS`	The image was successfully installed. The platform does not require a reboot.
`PSA_SUCCESS_REBOOT`	A system reboot is needed to finish installation. Use `psa_fwu_request_reboot`.
`PSA_SUCCESS_RESTART`	A restart of the updated component is required to complete the update. The restart mechanism is component specific.
`PSA_SUCCESS_DEPENDENCY_NEEDED`	Another image needs to be installed to finish installation. The caller must begin the firmware update process with the image specified in `dependency`.
`PSA_ERROR_INVALID_SIGNATURE`	The signature is incorrect.

Copyright © 2020-2021, Arm Limited or its affiliates. All rights reserved.
Non-confidential

PSA_ERROR_NOT_PERMITTED	The image is too old to be installed. If the image metadata contains a timestamp, and it has expired, then this error is also returned.
PSA_ERROR_DATA_CORRUPT	The image is corrupt.
PSA_ERROR_INSUFFICIENT_DATA	The image is smaller than expected.
PSA_ERROR_NOT_SUPPORTED	The image ID is either unknown or not supported.
PSA_ERROR_WRONG_DEVICE	The image is not intended for this device instance. For example, the image
PSA_ERROR_DEPENDENCY_NEEDED	A different image requires installation first.
PSA_ERROR_STORAGE_FAILURE	Some persistent storage could not be read or written by the implementation.
PSA_ERROR_MISSING_MANIFEST	A manifest is needed for this image ID. It must be set using psa_fwu_set_manifest.
PSA_ERROR_DECRYPTION_FAILURE	The key used to decrypt the data is unknown or decryption failed.
PSA_ERROR_INSUFFICIENT_POWER	There is not enough power to complete the operation.
PSA_ERROR_CURRENTLY_INSTALLING	The implementation is busy installing the requested image ID.
PSA_ERROR_ALREADY_INSTALLED	The storage item associated with image ID has already been installed.
PSA_ERROR_INSTALL_INTERRUPTED	Installation was interrupted or aborted.
PSA_ERROR_GENERIC_ERROR	A fatal error occured. This error is returned if not covered by other errors.

Description

7.8.5 psa_fwu_abort (function)

Aborts an ongoing installation and erases the staging area of the image.

```
psa_status_t psa_fwu_abort(psa_image_id_t image_id);
```

Parameters

image_id	The identifier of the image to abort installation.

Returns: psa_status_t

PSA_SUCCESS	Installation of the provided image ID has been aborted.
PSA_ERROR_INVALID_ARGUMENT	No image with the provided image ID is currently being installed.
PSA_ERROR_NOT_PERMITTED	The caller is not authorized to to abort an installation.
PSA_ERROR_STORAGE_FAILURE	Some persistent storage could not be erased.

7.8.6 psa_fwu_request_reboot (function)

Requests the platform to reboot. On success, the platform initiates a reboot, and might not return to the caller.

```
psa_status_t psa_fwu_request_reboot(void);
```


Copyright © 2020-2021, Arm Limited or its affiliates. All rights reserved.
Non-confidential

Returns: psa_status_t

PSA_SUCCESS	The platform will reboot soon.
PSA_ERROR_NOT_PERMITTED	The caller is not authorized to reboot the platform.

7.8.7 psa_fwu_request_rollback (function)

Requests the platform to roll back the firmware belonging to the caller and any other image that is dependent on that firmware. This is only used when the caller detects a fatal error after an update.

The platform may reject the request due to security policy.

```
psa_status_t psa_fwu_request_rollback(uint32_t error);
```

Parameters

error	An application-specific error code chosen by the application. If a specific error does not need to be reported, the value should be 0. On success, this error is recorded in the error field of the psa_image_info_t structure corresponding to this image.

Returns: psa_status_t

PSA_SUCCESS	The current firmware, and dependent images, will be rolled back on reboot.
PSA_ERROR_DOES_NOT_EXIST	The image ID is not recognized.
PSA_ERROR_NOT_PERMITTED	Either the caller is not authorized to roll back the platform or the current image cannot be rolled back due to security policy.
PSA_ERROR_NOT_SUPPORTED	This call is not implemented or rollback support is not implemented.

Description

On success, the current image becomes a rejected image, and the client will need to call psa_fwu_request_reboot to start the rollback process.

7.8.8 psa_fwu_accept (function)

Indicates to the implementation that the upgrade was successful. This changes the image state of a firmware image, and its dependencies, from PSA_IMAGE_PENDING_INSTALL to PSA_IMAGE_INSTALLED. For more information about image states, see the definition.

```
psa_status_t psa_fwu_accept(void);
```

Returns: psa_status_t

PSA_SUCCESS	The image and its dependencies have transitioned into a PSA_IMAGE_INSTALLED state.
PSA_ERROR_DOES_NOT_EXIST	The image ID is not recognized.
PSA_ERROR_NOT_SUPPORTED	The image ID is recognized but is not supported in this implementation.
PSA_ERROR_NOT_PERMITTED	The caller is not permitted to make this call.

Copyright © 2020-2021, Arm Limited or its affiliates. All rights reserved.
Non-confidential

Description

This function is optional and only required if an implementation supports a testing state.

If the image is in a PSA_IMAGE_PENDING_INSTALL state and this function is not called, then the image is replaced with an older image on reset.

7.8.9 psa_fwu_get_image_id_iterator (function)

Gets an iterator for use with psa_fwu_get_image_id. An iterator always returns at least one image ID.

```
void psa_fwu_get_image_id_iterator(psa_fwu_iterator_t *iterator);
```

Parameters

iterator

Returns: void

7.8.10 psa_fwu_get_image_id_next (function)

Advances an initialized iterator object. Used with psa_fwu_get_image_id.

```
bool psa_fwu_get_image_id_next(psa_fwu_iterator_t *iterator);
```

Parameters

iterator	An initialized iterator object.

Returns: bool

0	Returns 0 if there are no more images to report.
1	Returns 1 if there are more images to report.

Description

7.8.11 psa_fwu_get_image_id_valid (function)

Determines whether an iterator object is valid or not. Used with psa_fwu_get_image_id.

```
bool psa_fwu_get_image_id_valid(psa_fwu_iterator_t *iterator);
```

Parameters

iterator	An initialized iterator object

Returns: bool

0	Returns 0 if the iterator is invalid
1	Returns 1 if the iterator is valid.

7.8.12 psa_fwu_get_image_id (function)

Returns the image ID.

Copyright © 2020-2021, Arm Limited or its affiliates. All rights reserved.
Non-confidential

```
psa_status_t psa_fwu_get_image_id(psa_fwu_iterator_t *iterator,
                                  psa_image_id_t *image_id);
```

Parameters

iterator

image_id

Returns: psa_status_t

PSA_SUCCESS	Image information has been returned.
PSA_ERROR_INVALID_ARGUMENT	A parameter is not valid.
PSA_ERROR_NOT_PERMITTED	The caller is not authorized to access platform version information.

Copyright © 2020-2021, Arm Limited or its affiliates. All rights reserved.
Non-confidential

Appendix A: Example header file

Each implementation of the PSA Firmware Update API must provide a header file named `psa/update.h`, in which the API elements in this specification are defined.

This appendix provides a example of the `psa/update.h` header file with all of the API elements. This can be used as a starting point or reference for an implementation.

A.1 psa/update.h

```
typedef /*...*/ psa_fwu_iterator_t;
typedef uint32_t psa_image_id_t;
typedef int32_t psa_status_t;
struct psa_hash_t {
    uint8_t value[PSA_FWU_MAX_DIGEST_SIZE];
};
typedef struct psa_image_info_t {
    struct psa_image_id_t image_id;
    psa_uuid_t vendor_id;
    psa_uuid_t class_id;
    struct psa_image_version_t version;
    uint8_t state;
    psa_staging_info_t staging;
    uint32_t error;
    uint8_t digest[PSA_FWU_MAX_DIGEST_SIZE];
} psa_image_info_t;
typedef struct psa_image_version_t {
    uint16_t major;
    uint16_t minor;
    uint16_t patch;
} psa_image_version_t;
typedef struct psa_staging_info_t {
    uint8_t flags;
    uint8_t metadata_format;
    size_t max_size;
    psa_hash_t key_id;
} psa_staging_info_t;
typedef struct psa_uuid_t {
    uint32_t uuid[4];
} psa_uuid_t;
#define PSA_ERROR_ALREADY_INSTALLED ((psa_status_t)-158)
#define PSA_ERROR_COMMUNICATION_FAILURE ((psa_status_t)-145)
#define PSA_ERROR_CORRUPTION_DETECTED ((psa_status_t)-151)
#define PSA_ERROR_CURRENTLY_INSTALLING ((psa_status_t)-157)
#define PSA_ERROR_DATA_CORRUPT ((psa_status_t)-152)
#define PSA_ERROR_DATA_INVALID ((psa_status_t)-153)
#define PSA_ERROR_DECRYPTION_FAILURE ((psa_status_t)-162)
#define PSA_ERROR_DEPENDENCY_NEEDED ((psa_status_t)-156)
```

(continues on next page)

Copyright © 2020-2021, Arm Limited or its affiliates. All rights reserved.
Non-confidential

(continued from previous page)

```
#define PSA_ERROR_DOES_NOT_EXIST ((psa_status_t)-140)
#define PSA_ERROR_FLASH_ABUSE ((psa_status_t)-160)
#define PSA_ERROR_GENERIC_ERROR ((psa_status_t)-132)
#define PSA_ERROR_HARDWARE_FAILURE ((psa_status_t)-147)
#define PSA_ERROR_INSTALL_INTERRUPTED ((psa_status_t)-159)
#define PSA_ERROR_INSUFFICIENT_DATA ((psa_status_t)-143)
#define PSA_ERROR_INSUFFICIENT_MEMORY ((psa_status_t)-141)
#define PSA_ERROR_INSUFFICIENT_POWER ((psa_status_t)-161)
#define PSA_ERROR_INSUFFICIENT_STORAGE ((psa_status_t)-142)
#define PSA_ERROR_INVALID_ARGUMENT ((psa_status_t)-135)
#define PSA_ERROR_INVALID_SIGNATURE ((psa_status_t)-149)
#define PSA_ERROR_MISSING_MANIFEST ((psa_status_t)-163)
#define PSA_ERROR_NOT_PERMITTED ((psa_status_t)-133)
#define PSA_ERROR_NOT_SUPPORTED ((psa_status_t)-134)
#define PSA_ERROR_STORAGE_FAILURE ((psa_status_t)-146)
#define PSA_ERROR_WRONG_DEVICE ((psa_status_t)-155)
#define PSA_FWU_API_VERSION_MAJOR 0
#define PSA_FWU_API_VERSION_MINOR 7
#define PSA_FWU_MAX_BLOCK_SIZE IMPDEF
#define PSA_FWU_MAX_DIGEST_SIZE IMPDEF
#define PSA_IMAGE_CANDIDATE 1
#define PSA_IMAGE_FLAG_ENCRYPTION (0x00000002)
#define PSA_IMAGE_FLAG_VOLATILE_STAGING (0x00000001)
#define PSA_IMAGE_INSTALLED 2
#define PSA_IMAGE_PENDING_INSTALL 4
#define PSA_IMAGE_REBOOT_NEEDED 5
#define PSA_IMAGE_REJECTED 3
#define PSA_IMAGE_UNDEFINED 0
#define PSA_SUCCESS ((psa_status_t)0)
#define PSA_SUCCESS_DEPENDENCY_NEEDED ((psa_status_t)+3)
#define PSA_SUCCESS_REBOOT ((psa_status_t)+1)
#define PSA_SUCCESS_RESTART ((psa_status_t)+2)
psa_status_t psa_fwu_abort(psa_image_id_t image_id);
psa_status_t psa_fwu_accept(void);
psa_status_t psa_fwu_get_image_id(psa_fwu_iterator_t *iterator,
                                  psa_image_id_t *image_id);
void psa_fwu_get_image_id_iterator(psa_fwu_iterator_t *iterator);
bool psa_fwu_get_image_id_next(psa_fwu_iterator_t *iterator);
bool psa_fwu_get_image_id_valid(psa_fwu_iterator_t *iterator);
psa_status_t psa_fwu_install(psa_image_id_t image_id,
                             psa_image_id_t *dependency_image_id,
                             psa_image_version_t *dependency_version);
psa_status_t psa_fwu_query(psa_image_id_t image_id,
                           psa_image_info_t *info);
psa_status_t psa_fwu_request_reboot(void);
psa_status_t psa_fwu_request_rollback(uint32_t error);
psa_status_t psa_fwu_set_manifest(psa_image_id_t image_id,
                                  const void *manifest,
                                  size_t manifest_size,
                                  psa_hash_t *manifest_dependency);
psa_status_t psa_fwu_write(psa_image_id_t image_id,
                           size_t image_offset,
```

(continues on next page)

Copyright © 2020-2021, Arm Limited or its affiliates. All rights reserved.
Non-confidential

(continued from previous page)

```
                         const void *block,
                         size_t block_size);
```


Copyright © 2020-2021, Arm Limited or its affiliates. All rights reserved.
Non-confidential

Appendix B: Example usage

Warning: These examples are for illustrative purposes only and are not guaranteed to compile. Many error codes are not handled in order to keep the examples brief. A real implementation will need to initialize variables appropriately and handle failures as they see fit.

B.1 Retrieve versions of installed images

```
#include <psa/update.h>
#include <stddef.h> /* not necessarily required */

void example_get_installation_info() {

    psa_status_t rc;
    psa_fwu_iterator_t iter;
    psa_image_id_t id;
    psa_image_info_t image_info;

    psa_fwu_get_image_id_iterator(&iter);

    do {
        psa_fwu_get_image_id(&iter, &id);

        rc = psa_fwu_query(id, &image_info);

        if (image_info.state == PSA_IMAGE_INSTALLED && rc == PSA_SUCCESS) {
            specific_protocol_report(image_info.image_id, image_info.version);
        }

    } while (psa_fwu_get_image_id_next());
}
```

B.2 Individual image update (single part operation)

A single image with no dependencies.

```
#include <psa/update.h>
#include <stddef.h> /* not necessarily required */

/* Single image update */
void example_install_single_image(void *image, size_t image_size, psa_image_id_t id) {

    psa_status_t rc;
    psa_image_id_t needed_image;
    psa_image_version_t needed_version;
```

(continues on next page)

Copyright © 2020-2021, Arm Limited or its affiliates. All rights reserved.
Non-confidential

(continued from previous page)

```
    rc = psa_fwu_write(id, 0, image, image_size, &needed_image, &needed_version);
    rc = psa_fwu_install(id, &needed_image, &needed_version);

    if (rc == PSA_SUCCESS_REBOOT) {
        /* do other things and then eventually... */
        psa_fwu_request_reboot();
    } else {
        /* handle error */
    }
}
```

B.3 Invividual image update (multi part operation)

```
#include <psa/update.h>
#include <stddef.h> /* not necessarily required */

/* Single image update (multi-part) */
void example_install_single_image_multipart(size_t total_image_size, psa_image_id_t id) {

    psa_status_t rc;
    psa_image_id_t needed_image;
    psa_image_version_t needed_version;
    size_t offset = 0;
    size_t amount_to_send = PSA_FWU_MAX_BLOCK_SIZE;
    void *image;

    while (offset < total_image_size)
    {
        /* Unrealistic example, fetches malloc'd piece of image of size PSA_FWU_MAX_BLOCK_SIZE */
        image = fetch_next_part_of_image(id);

        if ((total_image_size - offset) <= PSA_FWU_MAX_BLOCK_SIZE) {
            amount_to_send = total_image_size - offset;
        }

        rc = psa_fwu_write(id, offset, image[offset], amount_to_send,
                           &needed_image, &needed_version);

        free(image);
        offset += amount_to_send;
    }

    rc = psa_fwu_install(id, &needed_image, &needed_version);

    if (rc == PSA_SUCCESS_REBOOT) {
        /* do other things and then eventually... */
        psa_fwu_request_reboot();
    } else if (rc == PSA_SUCCESS) {
        /* Success */
    } else {
```

(continues on next page)

Copyright © 2020-2021, Arm Limited or its affiliates. All rights reserved.
Non-confidential

(continued from previous page)

```
        /* Handle error */
    }
}
```

B.4 Multiple dependent images (multi part operation)

```
#include <psa/update.h>
#include <stddef.h> /* not necessarily required */

void example_install_multiple_images(void *image, size_t image_size, psa_image_id_t id) {

    psa_fwu_ctx_t ctx;
    psa_status_t rc;
    psa_image_id_t needed_image;
    psa_image_version_t needed_version;
    size_t offset = 0;
    size_t amount_to_send = PSA_FWU_MAX_BLOCK_SIZE;
    void *image_part;

    while (offset < image_size)
    {
        if (image_size - offset) <= PSA_FWU_MAX_BLOCK_SIZE) {
            amount_to_send = (image_size - offset));

            rc = psa_fwu_write(id, offset, image[offset], amount_to_send,
                               &needed_image, &needed_version);
            offset += amount_to_send;
        }

        rc = psa_fwu_install(id, &needed_image, &needed_version);

        if (rc == PSA_SUCCESS_DEPENDENCY_NEEDED) {
            /* Image might need download or might already been downloaded */
            int new_image_size = 0;
            void *new_image = retrieve_image_from_wherever(&needed_image, &needed_version,
                              &new_image_size);
            example_install_multiple_images(new_image, new_image_size, &needed_image);
        }

        if (rc == PSA_SUCCESS_REBOOT) {
            /* do other things and then eventually... */
            psa_fwu_request_reboot();
        } else if (rc == PSA_SUCCESS) {
            /* other success */
        } else {
            /* handle failures /*
        }
    }
}
```


Copyright © 2020-2021, Arm Limited or its affiliates. All rights reserved.
Non-confidential

Appendix C: Future changes

We appreciate feedback from the technical community on this document. Feedback can be sent by e-mail to the following address arm.psa-feedback@arm.com.

There are a number of anticipated changes that may affect future versions of this document. Feedback or preference on the open issues below would be appreciated.

C.1 Rename psa_fwu_abort

The function `psa_fwu_abort` aborts any ongoing installation for the specified staging area and erases it. This may not be descriptive enough to applications that expect erase functions.

It may be more intuitive to either:

- Rename `psa_fwu_abort` to `psa_fwu_erase`
- Split the functionality into two functions: `psa_fwu_abort` and `psa_fwu_erase`

C.2 Init function

The current version of the API assumes that RAM has been allocated to the implementation. This may not be ideal in a simple library implementation, particularly if there is no *SPE* and the caller expects to manage all RAM usage.

There are at least a couple of potential options:

- `psa_fwu_init(void)` could be introduced to initialize the library, where applications must call before using any other function.
- `psa_fwu_init(psa_fwu_ctx_t * context)` could be introduced to initialize the library, where applications must call before using any other function. This option allows the caller to specify where the library's working RAM is allocated. The structure is an opauque one, allowing for different implementations. This option is the most intrusive change because the context variable would need to be added to the parameters of all the other API functions.

Copyright © 2020-2021, Arm Limited or its affiliates. All rights reserved.
Non-confidential

Appendix D: Change history

This section describes detailed changes between past versions.

- PSA_FWU_API_VERSION_MINOR has increased from 6 to 7
- psa_image_id_t is now defined as a 32-bit integer. Functions no longer have a pointer type for this parameter.
- UUID concept dropped from function names and parameters.
- Added Vendor ID and Class ID to psa_image_info_t structure.
- Added Future changes section
- Added error code and success code definitions
- Fixed mistake: psa_fwu_abort return type changed from void to psa_status_t
- Clarifications to the text
- Replaced PSA_ERROR_ROLLBACK_DETECTED with PSA_ERROR_NOT_PERMITTED
- Remove standardized image IDs until we get more feedback
- Improvements to the Design Overview text

Copyright © 2020-2021, Arm Limited or its affiliates. All rights reserved.
Non-confidential