PSA Certified Status code API 1.0

Document number:	IHI 0097
Release Quality:	Final
Issue Number:	2
Confidentiality:	Non-confidential
Date of Issue:	14/12/2022

Contents

About this document

Release information

Prior to version 1.0.1, the definitions in this specification were released as part of *Arm® Platform Security Architecture Firmware Framework* [PSA-FFM].

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

Table 1 Document revision history

Date	Version	Confidentiality	Change
October 2022	1.0.1	Non-confidential	Definition of status codes moved to a separate specification. Incorporated some error codes from other PSA Certified APIs. Relicensed as open source under CC BY-SA 4.0.
December 2022	1.0.2	Non-confidential	Fixed whitespace error in status code definitions

For a detailed list of changes, see *Change history* on page 19.

PSA Certified Status code API

License

Text and illustrations

About the license

The language in the additional patent license is largely identical to that in section 3 of the Apache License, Version 2.0 (Apache 2.0), with two exceptions:

1. Changes are made related to the defined terms, to align those defined terms with the terminology in CC BY-SA 4.0 rather than Apache 2.0 (for example, changing "Work" to "Licensed Material").
2. The scope of the defensive termination clause is changed from "any patent licenses granted to You" to "any licenses granted to You". This change is intended to help maintain a healthy ecosystem by providing additional protection to the community against patent litigation claims.

To view the full text of the Apache 2.0 license, visit apache.org/licenses/LICENSE-2.0.

Source code

References

This document refers to the following documents.

Table 2 Documents referenced by this document

Ref	Document Number	Title
[PSA-FFM]	DEN 0063	*Arm® Platform Security Architecture Firmware Framework*. pages.arm.com/psa-apis.html
[TF-M]		trustedfirmware.org, *Trusted Firmware-M*. git.trustedfirmware.org/trusted-firmware-m.git/about/

Terms and abbreviations

This document uses the following terms and abbreviations.

Table 3 Terms and abbreviations

Term	Meaning
IMPLEMENTATION DEFINED	Behavior that is not defined by the this specification, but is defined and documented by individual implementations. Software developers can choose to depend on IMPLEMENTATION DEFINED behavior, but must be aware that their code might not be portable to another implementation.
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.
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.
Root of Trust Service (RoT Service)	A set of related security operations that are implemented in a *Root of Trust*.
RoT	See *Root of Trust*.
RoT Service	See *Root of Trust Service*.
Secure Partition Manager (SPM)	Part of *Arm® Platform Security Architecture Firmware Framework* [PSA-FFM] that is responsible for isolating software in Partitions, managing the execution of software within Partitions, and providing communication between Partitions.
SPM	See *Secure Partition Manager*.

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 example.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.

Feedback

We welcome feedback on the PSA Certified API documentation.

If you have comments on the content of this book, visit github.com/arm-software/psa-api/issues to create a new issue at the PSA Certified API GitHub project. Give:

- The title (Status code API).
- The number and issue (IHI 0097 1.0.2).
- The location in the document to which your comments apply.
- A concise explanation of your comments.

We also welcome general suggestions for additions and improvements.

1 Introduction

1.1 About Platform Security Architecture

This document is one of a set of resources provided by Arm that can help organizations develop products that meet the security requirements of PSA Certified on Arm-based platforms. The PSA Certified scheme provides a framework and methodology that helps silicon manufacturers, system software providers and OEMs to develop more secure products. Arm resources that support PSA Certified range from threat models, standard architectures that simplify development and increase portability, and open-source partnerships that provide ready-to-use software. You can read more about PSA Certified here at www.psacertified.org and find more Arm resources here at developer.arm.com/platform-security-resources.

1.2 About the Status code API

The interface described in this document is a PSA Certified API, that provides a shared set of interface definitions used by other PSA Certified APIs.

You can find additional resources relating to the Status code API here at arm-software.github.io/psa-api/status-code, and find other PSA Certified APIs here at arm-software.github.io/psa-api.

2 Status codes

2.1 Overview

The PSA Certified APIs are often implemented together in a larger framework. For example, the *Trusted Firmware-M* [TF-M] project implements all of the PSA functional APIs as Root of Trust Services within the Secure Processing Environment that it provides. Using a common definition for status and error codes enables easier integration and inter-operation of these APIs.

The PSA Certified APIs use the convention that status codes that are negative indicate an error, and zero or positive values indicate success. These are identified in the API by the `psa_status_t` type.

Status codes `-129` to `-248` are for use by PSA Certified API specifications. These codes are defined in the current PSA specifications, or are reserved for future PSA specifications. Status codes in this range are used in the following ways:

- A set of standard error codes that cover failure conditions that are common to more than one PSA Certified API.
- Error codes that are specific to an individual PSA Certified API.

Status codes in this range must only be used as defined in a PSA specification.

In the context of an implementation of *Arm® Platform Security Architecture Firmware Framework* [PSA-FFM]:

- The *Secure Partition Manager* (SPM) implementation can define error codes in the range `-249` to `-256` for IMPLEMENTATION DEFINED purposes.
- A *Root of Trust Service* (RoT Service) can define additional error codes in the ranges `-1` to `-128` and `-257` to `MIN_INT32` for RoT Service-specific error conditions.

Table 4 defines the common error codes and reserved ranges for the PSA Certified APIs. See the error code macros and function definitions in *API Reference* on page 9 for details on their usage.

Table 4 Standard error codes

Status code name	Value	Condition
Success	`>= 1`	API-specific status code.
`PSA_SUCCESS`	`0`	General success status code.
API-specific error	`-1` to `-128`	API-specific error code.
`PSA_ERROR_PROGRAMMER_ERROR`	`-129`	Connection dropped due to PROGRAMMER ERROR.
`PSA_ERROR_CONNECTION_REFUSED`	`-130`	Connection to the service is not permitted.
`PSA_ERROR_CONNECTION_BUSY`	`-131`	Connection to the service is not possible.
`PSA_ERROR_GENERIC_ERROR`	`-132`	An error not related to a specific failure cause.

Table 4 (continued)

Status code name	Value	Condition
PSA_ERROR_NOT_PERMITTED	-133	The operation is denied by a policy.
PSA_ERROR_NOT_SUPPORTED	-134	The operation or a parameter is not supported.
PSA_ERROR_INVALID_ARGUMENT	-135	One or more parameters are invalid.
PSA_ERROR_INVALID_HANDLE	-136	A handle parameter is not valid.
PSA_ERROR_BAD_STATE	-137	The operation is not valid in the current state.
PSA_ERROR_BUFFER_TOO_SMALL	-138	An output buffer parameter is too small.
PSA_ERROR_ALREADY_EXISTS	-139	An identifier or index is already in use.
PSA_ERROR_DOES_NOT_EXIST	-140	An identified resource does not exist.
PSA_ERROR_INSUFFICIENT_MEMORY	-141	There is not enough runtime memory.
PSA_ERROR_INSUFFICIENT_STORAGE	-142	There is not enough persistent storage.
PSA_ERROR_INSUFFICIENT_DATA	-143	A data source has insufficient capacity left.
PSA_ERROR_SERVICE_FAILURE	-144	Failure within the service.
PSA_ERROR_COMMUNICATION_FAILURE	-145	Communication failure with another component.
PSA_ERROR_STORAGE_FAILURE	-146	Storage failure that may have led to data loss.
PSA_ERROR_HARDWARE_FAILURE	-147	General hardware failure.
Reserved	-148	Reserved for PSA Certified APIs.
PSA_ERROR_INVALID_SIGNATURE	-149	A signature, MAC or hash is incorrect.
Reserved	-150	Reserved for PSA Certified APIs.
PSA_ERROR_CORRUPTION_DETECTED	-151	Internal data has been tampered with.
PSA_ERROR_DATA_CORRUPT	-152	Stored data has been corrupted.
PSA_ERROR_DATA_INVALID	-153	Data read from storage is not valid.
Reserved	-154 to -247	Reserved for PSA Certified APIs.
PSA_OPERATION_INCOMPLETE	-248	The requested operation is not finished.
SPM Implementation error	-249 to -256	Reserved for the SPM implementation.
API-specific error	<= -257	API-specific error code.

2.2 API Reference

These are common status and error codes for all PSA Certified APIs, and for SPM and RoT Service APIs. See *Overview* on page 7 for a summary of the status codes.

The API elements described in the following sections §2.2.1 to §2.2.4, must be defined in a header file `psa/error.h`. See *Reference header file* on page 17 for a reference version of this header file.

It is permitted for these API elements to also be defined in header files that are part of an implementation of another PSA Certified API, for example, in `psa/crypto.h`.

> **Implementation note**
>
> In an implementation of any PSA Certified API, it essential that the status code macros are defined precisely as shown in the API specifications and reference header files. In particular, there is no white-space in the definition.
>
> The C language only permits a macro definition to be repeated within a compilation, if every definition is identical, including the white-space separation.

2.2.1 Status type

`psa_status_t` (typedef)

A status code type used for all PSA Certified APIs.

```
/* Prevent multiple definitions of psa_status_t, if PSA_SUCCESS is already
   defined in an external header
 */
#ifndef PSA_SUCCESS
typedef int32_t psa_status_t;
#endif
```

A zero or positive value indicates success, the interpretation of the value depends on the specific operation.

A negative integer value indicates an error.

> **Implementation note**
>
> This definition is permitted to be present in multiple header files that are included in a single compilation.
>
> To prevent a compilation error from duplicate definitions of `psa_status_t`, the definition of `psa_status_t` must be guarded, by testing for an exiting definition of `PSA_SUCCESS`, in any header file that defines `psa_status_t`.
>
> The definition of `psa_status_t` above shows the recommended form of the guard.

2.2.2 Success code

PSA_SUCCESS (macro)

A status code to indicate general success.

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

This is a general return value to indicate success of the operation.

2.2.3 Error codes

PSA_ERROR_PROGRAMMER_ERROR (macro)

A status code that indicates a PROGRAMMER ERROR in the client.

```
#define PSA_ERROR_PROGRAMMER_ERROR ((psa_status_t)-129)
```

This error indicates that the function has detected an abnormal call, which typically indicates a programming error in the caller, or an abuse of the API.

This error has a specific meaning in an implementation of *Arm® Platform Security Architecture Firmware Framework* [PSA-FFM].

PSA_ERROR_CONNECTION_REFUSED (macro)

A status code that indicates that the caller is not permitted to connect to a Service.

```
#define PSA_ERROR_CONNECTION_REFUSED ((psa_status_t)-130)
```

This message has a specific meaning in an implementation of *Arm® Platform Security Architecture Firmware Framework* [PSA-FFM].

PSA_ERROR_CONNECTION_BUSY (macro)

A status code that indicates that the caller cannot connect to a service.

```
#define PSA_ERROR_CONNECTION_BUSY ((psa_status_t)-131)
```

This message has a specific meaning in an implementation of *Arm® Platform Security Architecture Firmware Framework* [PSA-FFM].

PSA_ERROR_GENERIC_ERROR (macro)

A status code that indicates an error that does not correspond to any defined failure cause.

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

Functions can return this error code if none of the other standard error codes are applicable.

Note:

For new APIs, it is recommended that additional error codes are defined by the API for important error conditions which do not correspond to an existing status code.

PSA_ERROR_NOT_PERMITTED (macro)

A status code that indicates that the requested action is denied by a policy.

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

It is recommended that a function returns 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 with PSA_ERROR_NOT_PERMITTED, PSA_ERROR_NOT_SUPPORTED, or PSA_ERROR_INVALID_ARGUMENT.

PSA_ERROR_NOT_SUPPORTED (macro)

A status code that indicates that the requested operation or a parameter is not supported.

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

This error code is recommended for indicating that optional functionality in an API specification is not provided by the implementation.

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

PSA_ERROR_INVALID_ARGUMENT (macro)

A status code that indicates that the parameters passed to the function are invalid.

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

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

It is recommended that a function returns a more specific error code where applicable, for example PSA_ERROR_INVALID_HANDLE, PSA_ERROR_DOES_NOT_EXIST, or PSA_ERROR_ALREADY_EXISTS.

PSA_ERROR_INVALID_HANDLE (macro)

A status code that indicates that a handle parameter is not valid.

```
#define PSA_ERROR_INVALID_HANDLE ((psa_status_t)-136)
```

A function can return this error any time a handle parameter is invalid.

PSA_ERROR_BAD_STATE (macro)

A status code that indicates that the requested action cannot be performed in the current state.

```
#define PSA_ERROR_BAD_STATE ((psa_status_t)-137)
```

It is recommended that a function returns this error when an operation is requested out of sequence.

PSA_ERROR_BUFFER_TOO_SMALL (macro)

A status code that indicates that an output buffer parameter is too small.

```
#define PSA_ERROR_BUFFER_TOO_SMALL ((psa_status_t)-138)
```

A function can return this error if an output parameter is too small for the requested output data.

It is recommended that a function only returns this error code in cases where performing the operation with a larger output buffer would succeed. However, a function can also return this error if a function has invalid or unsupported parameters in addition to an insufficient output buffer size.

PSA_ERROR_ALREADY_EXISTS (macro)

A status code that indicates that an identifier or index is already in use.

```
#define PSA_ERROR_ALREADY_EXISTS ((psa_status_t)-139)
```

A function can return this error if the call is attempting to reuse an identifier or a resource index that is already allocated or in use.

It is recommended that this error code is not used for a handle or index that is invalid. For these situations, return PSA_ERROR_PROGRAMMER_ERROR, PSA_ERROR_INVALID_HANDLE, or PSA_ERROR_INVALID_ARGUMENT.

PSA_ERROR_DOES_NOT_EXIST (macro)

A status code that indicates that an identified resource does not exist.

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

A function can return this error if a request identifies a resource that has not been created or is not present.

It is recommended that this error code is not used for a handle or index that is invalid. For these situations, return PSA_ERROR_PROGRAMMER_ERROR, PSA_ERROR_INVALID_HANDLE, or PSA_ERROR_INVALID_ARGUMENT.

PSA_ERROR_INSUFFICIENT_MEMORY (macro)

A status code that indicates that there is not enough runtime memory.

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

A function can return this error if runtime memory required for the requested operation cannot be allocated.

If the operation involves multiple components, this error can refer to available memory in any of the components.

PSA_ERROR_INSUFFICIENT_STORAGE (macro)

A status code that indicates that there is not enough persistent storage.

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

A function can return this error if the operation involves storing data in non-volatile memory, and when there is insufficient space on the host media.

Operations that do not directly store persistent data can also return this error code if the implementation requires a mandatory log entry for the requested action and the log storage space is full.

PSA_ERROR_INSUFFICIENT_DATA (macro)

A status code that indicates that a data source has insufficient capacity left.

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

A function can return this error if the operation attempts to extract data from a source which has been exhausted.

PSA_ERROR_SERVICE_FAILURE (macro)

A status code that indicates an error within the service.

```
#define PSA_ERROR_SERVICE_FAILURE ((psa_status_t)-144)
```

A function can return this error if it unable to operate correctly. For example, if an essential initialization operation failed.

For failures that are related to hardware peripheral errors, it is recommended that the function returns `PSA_ERROR_COMMUNICATION_FAILURE` or `PSA_ERROR_HARDWARE_FAILURE`.

PSA_ERROR_COMMUNICATION_FAILURE (macro)

A status code that indicates a communication failure between the function and another service or component.

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

A function can return this error if there is a fault in the communication between the implementation and another service or peripheral used to provide the requested service. A communication failure may 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, a function can return `PSA_ERROR_COMMUNICATION_FAILURE` if the requested action was completed successfully in an external component, but there was a breakdown of communication before this was reported to the application.

PSA_ERROR_STORAGE_FAILURE (macro)

A status code that indicates a storage failure that may have led to data loss.

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

A function can return this error to indicate that some persistent storage could not be read or written. 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 the storage hardware — 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 may no longer be readable from storage.

It is recommended to only use this error code to report a permanent storage corruption. However, transient errors while reading the storage can also be reported using this error code.

PSA_ERROR_HARDWARE_FAILURE (macro)

A status code that indicates that a hardware failure was detected.

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

A function can return this error to report a general hardware fault. A hardware failure may be transient or permanent depending on the cause.

PSA_ERROR_INVALID_SIGNATURE (macro)

A status code that indicates that a signature, MAC or hash is incorrect.

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

A function can return this error to report when a verification calculation completes successfully, and the value to be verified is incorrect.

PSA_ERROR_CORRUPTION_DETECTED (macro)

A status code that indicates that internal data has been tampered with.

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

A function can return this error if it detects 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 should not be used to indicate a hardware failure that merely makes it impossible to perform the requested operation, instead use PSA_ERROR_COMMUNICATION_FAILURE, PSA_ERROR_STORAGE_FAILURE, PSA_ERROR_HARDWARE_FAILURE, or other applicable error code.

This error should not be used to report modification of application state, or misuse of the API.

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.

PSA_ERROR_DATA_CORRUPT (macro)

A status code that indicates that stored data has been corrupted.

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

A function can return this error to indicate 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.

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.

PSA_ERROR_DATA_INVALID (macro)

A status code that indicates that 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.

2.2.4 Unfinished operation code

PSA_OPERATION_INCOMPLETE (macro)

A status code that indicates that the requested operation is interruptible, and still has work to do.

```
#define PSA_OPERATION_INCOMPLETE ((psa_status_t)-248)
```

This status code does not mean that the operation has failed or that it has succeeded. The operation must be repeated until it completes with either success or failure.

Usage

This is an example of how this status code can be used:

```
psa_status_t r = start_operation();

if (r == PSA_SUCCESS) {
   do {
      r = complete_operation();
   } while (r == PSA_OPERATION_INCOMPLETE);
}

if (r == PSA_SUCCESS) {
   /* Handle success */
} else {
   /* Handle errors */
}
```

Appendix A: Reference header file

The following is an example of the standard header files for the Status code API.

An implementation of other PSA Certified APIs will provide an instance of the `psa/error.h` source file.

psa/error.h

```
// This file is a reference template for implementation of the PSA Certified Status code API

#ifndef PSA_ERROR_H
#define PSA_ERROR_H

#include <stddef.h>
#include <stdint.h>

#ifdef __cplusplus
extern "C" {
#endif

/* Prevent multiple definitions of psa_status_t, if PSA_SUCCESS is already
   defined in an external header
 */
#ifndef PSA_SUCCESS
typedef int32_t psa_status_t;
#endif
#define PSA_SUCCESS ((psa_status_t)0)
#define PSA_ERROR_PROGRAMMER_ERROR ((psa_status_t)-129)
#define PSA_ERROR_CONNECTION_REFUSED ((psa_status_t)-130)
#define PSA_ERROR_CONNECTION_BUSY ((psa_status_t)-131)
#define PSA_ERROR_GENERIC_ERROR ((psa_status_t)-132)
#define PSA_ERROR_NOT_PERMITTED ((psa_status_t)-133)
#define PSA_ERROR_NOT_SUPPORTED ((psa_status_t)-134)
#define PSA_ERROR_INVALID_ARGUMENT ((psa_status_t)-135)
#define PSA_ERROR_INVALID_HANDLE ((psa_status_t)-136)
#define PSA_ERROR_BAD_STATE ((psa_status_t)-137)
#define PSA_ERROR_BUFFER_TOO_SMALL ((psa_status_t)-138)
#define PSA_ERROR_ALREADY_EXISTS ((psa_status_t)-139)
#define PSA_ERROR_DOES_NOT_EXIST ((psa_status_t)-140)
#define PSA_ERROR_INSUFFICIENT_MEMORY ((psa_status_t)-141)
#define PSA_ERROR_INSUFFICIENT_STORAGE ((psa_status_t)-142)
#define PSA_ERROR_INSUFFICIENT_DATA ((psa_status_t)-143)
#define PSA_ERROR_SERVICE_FAILURE ((psa_status_t)-144)
#define PSA_ERROR_COMMUNICATION_FAILURE ((psa_status_t)-145)
#define PSA_ERROR_STORAGE_FAILURE ((psa_status_t)-146)
#define PSA_ERROR_HARDWARE_FAILURE ((psa_status_t)-147)
#define PSA_ERROR_INVALID_SIGNATURE ((psa_status_t)-149)
#define PSA_ERROR_CORRUPTION_DETECTED ((psa_status_t)-151)
#define PSA_ERROR_DATA_CORRUPT ((psa_status_t)-152)
```

(continues on next page)

(continued from previous page)

```
#define PSA_ERROR_DATA_INVALID ((psa_status_t)-153)
#define PSA_OPERATION_INCOMPLETE ((psa_status_t)-248)

#ifdef __cplusplus
}
#endif

#endif // PSA_ERROR_H
```

Appendix B: Change history

B.1 Changes between version 1.0.1 and version 1.0.2

- Removed the whitespace within the definition of some of the status codes. The whitespace was erroneously introduced during the separation from the *Arm® Platform Security Architecture Firmware Framework* [PSA-FFM]. This change is necessary to ensure that multiple definitions of the same status code are identical, as required by the C language.

B.2 Changes between version 1.0.0 and version 1.0.1

- Moved the specification of the common error codes into a separate specification.
- Relicensed the document under Attribution-ShareAlike 4.0 International with a patent license derived from Apache License 2.0. See *License* on page iii.
- Generalized the definitions of the error codes to better fit all PSA Certified APIs.
- Added definitions from other PSA Certified APIs:
  - `PSA_ERROR_CORRUPTION_DETECTED`
  - `PSA_ERROR_DATA_CORRUPT`
  - `PSA_ERROR_DATA_INVALID`
- Added `PSA_OPERATION_INCOMPLETE` to indicate that the requested operation is unfinished. This can be used to break long-running operations into smaller pieces.

B.3 Changes prior to version 1.0.0

The definition of the common status codes was incorporated in the *Arm® Platform Security Architecture Firmware Framework* [PSA-FFM] specification up until version 1.0.0.