SUIT
Internet-Draft
Intended status: Informational
Expires: April 23, 2020

B. Moran
H. Tschofenig
Arm Limited
H. Birkholz
Fraunhofer SIT

October 21, 2019

SUIT CBOR manifest serialisation format
draft-ietf-suit-manifest-00

Abstract

This specification describes the format of a manifest. A manifest is a bundle of metadata about the firmware for an IoT device, where to find the firmware, the devices to which it applies, and cryptographic information protecting the manifest.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 23, 2020.

Table of Contents

1. Introduction

A firmware update mechanism is an essential security feature for IoT devices to deal with vulnerabilities. While the transport of firmware images to the devices themselves is important there are already various techniques available, such as the Lightweight Machine-to-Machine (LwM2M) protocol offering device management of IoT devices. Equally important is the inclusion of meta-data about the

conveyed firmware image (in the form of a manifest) and the use of end-to-end security protection to detect modifications and (optionally) to make reverse engineering more difficult. End-to-end security allows the author, who builds the firmware image, to be sure that no other party (including potential adversaries) can install firmware updates on IoT devices without adequate privileges. This authorization process is ensured by the use of dedicated symmetric or asymmetric keys installed on the IoT device: for use cases where only integrity protection is required it is sufficient to install a trust anchor on the IoT device. For confidentiality protected firmware images it is additionally required to install either one or multiple symmetric or asymmetric keys on the IoT device. Starting security protection at the author is a risk mitigation technique so firmware images and manifests can be stored on untrusted respositories; it also reduces the scope of a compromise of any repository or intermediate system to be no worse than a denial of service.

It is assumed that the reader is familiar with the high-level firmware update architecture [Architecture]. This document is structured as follows: In Section 8 we describe the main building blocks of the manifest and Section 12 contains the description of the CBOR of the manifest.

The SUIT manifest is heavily optimised for consumption by constrained devices. This means that it is not constructed as a conventional descriptive document, as described in [Behaviour]. This means that a user viewing the contents of the document will require tooling to view the contents in a more descriptive way.

2. Conventions and Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

- SUIT: Sofware Update for the Internet of Things, the IETF working group for this standard.

- Payload: A piece of information to be delivered. Typically Firmware for the purposes of SUIT.

- Resource: A piece of information that is used to construct a payload.

- Manifest: A piece of information that describes one or more payloads, one or more resources, and the processors needed to transform resources into payloads.

- Update: One or more manifests that describe one or more payloads.

- Update Authority: The owner of a cryptographic key used to sign updates, trusted by recipient devices.

- Recipient: The system, typically an IoT device, that receives a manifest.

- Condition: A test for a property of the Recipient or its components.

- Directive: An action for the Recipient to perform.

- Command: A Condition or a Directive.

3. SUIT digest container

RFC 8152 [RFC8152] provides containers for signature, MAC, and encryption, but no basic digest container. The container needed for a digest requires a type identifier and a container for the raw digest data. Some forms of digest may require additional parameters. These can be added following the digest. Algorithm identifiers defined in RFC 6920 [RFC6920] are reused for this digest container. This structure is described by the following CDDL:

```
SUIT_Digest = [
 suit-digest-algorithm-id : $suit-digest-algorithm-ids,
 suit-digest-bytes : bytes,
 ? suit-digest-parameters : any
]

; Named Information Hash Algorithm Identifiers
digest-algorithm-ids /= algorithm-id-sha256
digest-algorithm-ids /= algorithm-id-sha256-128
digest-algorithm-ids /= algorithm-id-sha256-120
digest-algorithm-ids /= algorithm-id-sha256-96
digest-algorithm-ids /= algorithm-id-sha256-64
digest-algorithm-ids /= algorithm-id-sha256-32
digest-algorithm-ids /= algorithm-id-sha384
digest-algorithm-ids /= algorithm-id-sha512
digest-algorithm-ids /= algorithm-id-sha3-224
digest-algorithm-ids /= algorithm-id-sha3-256
digest-algorithm-ids /= algorithm-id-sha3-384
digest-algorithm-ids /= algorithm-id-sha3-512
```

4. Distributing firmware

Distributing firmware in a multi-party environment is a difficult operation. Each party requires a different subset of data. Some data may not be accessible to all parties. Multiple signatures may be required from parties with different authorities. This topic is covered in more depth in [Architecture]

5. Workflow of a device applying a firmware update

The manifest is designed to work with a pull parser, where each section of the manifest is used in sequence. The expected workflow for a device installing an update can be broken down into 5 steps:

1. Verify the signature of the manifest

2. Verify the applicability of the manifest

3. Resolve dependencies

4. Fetch payload(s)

5. Install payload(s)

When installation is complete, similar information can be used for validating and running images in a further three steps:

1. Verify image(s)

2. Load image(s)

3. Run image(s)

When multiple manifests are used for an update, each manifest's steps occur in a lockstep fashion; all manifests have dependency resolution performed before any manifest performs a payload fetch, etc.

6. SUIT manifest goals

The manifest described in this document is intended to simplify the construction of constrained device firmware update solutions. It is also intended to allow update authors to describe complex update processes for complex devices.

Manifests implemented as descriptive documents require changes to the parser and the information model whenever a new feature is added. This is particularly accentuated when the parser is a fixed-function minimal parser (or a pull parser) such as the type that is typically

used in a bootloader or in a constrained client. The issue is not as significant in devices that can use general purpose parsers.

The manifest detailed in this document aims to address these and more problems by changing the processing model from a piece of software that loads a manifest, interprets the data, then performs some actions, into a model in which the software performs exactly the operations stated in the manifest, in order. This allows the manifest to encode data in a way that matches precisely with what the parser expects. It also makes inflexible code, like a bootloader, more flexible in what it can do; because the manifest defines part of the "program," the manifest's execution defines part of the behaviour of the system. Further detail on this approach is covered in [Behaviour]

The SUIT manifest can be used for a variety of purposes throughout its lifecycle. The manifest allows:

1. the Firmware Author to reason about releasing a firmware.

2. the Network Operator to reason about compatibility of a firmware.

3. the Device Operator to reason about the impact of a firmware.

4. the Device Operator to manage distribution of firmware to devices.

5. the Plant Manager to reason about timing and acceptance of firmware updates.

6. the device to reason about the authority & authenticity of a firmware prior to installation.

7. the device to reason about the applicability of a firmware.

8. the device to reason about the installation of a firmware.

9. the device to reason about the authenticity of a firmware at boot.

Each of these uses happens at a different stage of the manifest lifecycle, so each has different requirements.

To verify authenticity at boot time, only the smallest portion of the manifest is required. This core part of the manifest describes only the fully installed firmware and any of its dependencies.

7. SUIT manifest design overview

In order to provide flexible behaviour to constrained devices, while still allowing more powerful devices to use their full capabilities, the SUIT manifest takes a new approach, encoding the required behaviour of a Recipient device, instead of just presenting the information used to determine that behaviour. This gives benefits equivalent to those provided by a scripting language or byte code, with two substantial differences. First, the language is extremely high level, consisting of only the operations that a device may perform during update and secure boot of a firmware image. The language specifies behaviours in a linearised form, without branches or loops. Conditional processing is supported, and parallel and out-of-order processing may be performed by sufficiently capable devices.

By structuring the data in this way, the manifest processor becomes a very simple engine that uses a pull parser to interpret the manifest. This pull parser invokes a series of command handlers that evaluate a Condition or execute a Directive. Most data is structured in a highly regular pattern, which simplifies the parser.

The results of this allow a Recipient with minimal functionality to perform complex updates with reduced overhead. Conditional execution of commands allows a simple device to perform important decisions at validation-time, such as which differential update to download for a given current version, or which hash to check, based on the installation address.

Dependency handling is vastly simplified as well. Dependencies function like subroutines of the language. When a manifest has a dependency, it can invoke that dependency's commands and modify their behaviour by setting parameters. Because some parameters come with security implications, the dependencies also have a mechanism to reject modifications to parameters on a fine-grained level.

Developing a robust permissions system works in this model too. The Recipient can use a simple ACL that is a table of Identities and Component Identifier permissions to ensure that only manifests authenticated by the appropriate identity have access to define a component.

Capability reporting is similarly simplified. A Recipient can report the Commands and Parameters that it supports. This is sufficiently precise for a manifest author to create a manifest that the Recipient can accept.

The simplicity of design in the Recipient due to all of these benefits allows even a highly constrained platform to use advanced update capabilities.

7.1. Severable Elements

Because the manifest can be used by different actors at different times, some parts of the manifest can be removed without affecting later stages of the lifecycle. This is called "Severing." Severing of information is achieved by separating that information from the signed container so that removing it does not affect the signature. This means that ensuring authenticity of severable parts of the manifest is a requirement for the signed portion of the manifest. Severing some parts makes it possible to discard parts of the manifest that are no longer necessary. This is important because it allows the storage used by the manifest to be greatly reduced. For example, no text size limits are needed if text is removed from the manifest prior to delivery to a constrained device.

Elements are made severable by removing them from the manifest, encoding them in a bstr, and placing a SUIT_Digest of the bstr in the manifest so that they can still be authenticated. The SUIT_Digest typically consumes 4 bytes more than the size of the raw digest, therefore elements smaller than (Digest Bits)/8 + 4 SHOULD never be severable. Elements larger than (Digest Bits)/8 + 4 MAY be severable, while elements that are much larger than (Digest Bits)/8 + 4 SHOULD be severable.

7.2. Conventions

The map indices in this encoding are reset to 1 for each map within the structure. This is to keep the indices as small as possible. The goal is to keep the index objects to single bytes (CBOR positive integers 1-23).

Wherever enumerations are used, they are started at 1. This allows detection of several common software errors that are caused by uninitialised variables. Positive numbers in enumerations are reserved for IANA registration. Negative numbers are used to identify application-specific implementations.

CDDL names are hyphenated and CDDL structures follow the convention adopted in COSE [RFC8152]: SUIT_Structure_Name.

7.3. Payloads

Payloads can take many forms, for example, binary, hex, s-record, elf, binary diff, PEM certificate, CBOR Web Token, serialised configuration. These payloads fall into two broad categories: those that require installation-time unpacking and those that do not. Binary, PEM certificate, and CBOR Web Token do not require installation-time unpacking. Hex, s-record, elf, and serialised configuration require installation-time unpacking.

Some payloads cannot be directly converted to a writable binary stream. Hex, s-record, and elf may contain gaps and they have no guarantee of monotonic increase of address, which makes pre-processing them into a binary stream difficult on constrained platforms. Serialised configuration may be unpacked into a configuration database, which makes it impossible to preprocess into a binary stream, suitable for direct writing.

Where a specialised unpacking algorithm is needed, a digest is not always calculable over an installed payload. For example, an elf, s-record or hex file may contain gaps that can contain any data, while not changing whether or not an installed payload is valid. Serialised configuration may update only some device data rather than all of it. This means that the digest cannot always be calculated over an installed payload when a specialised installer is used.

This presents two problems for the manifest: first, it must indicate that a specialised installer is needed and, second, it cannot provide a hash of the payload that is checkable after installation. These two problems are resolved in two ways:

1. Payloads that need a specialised installer must indicate this in suit-payload-info-unpack.

2. Payloads that need specialised verification must indicate this in the SUIT_Payload section or SUIT_Parameter_Image_Digest by indicating a SUIT_Digest algorithm that correctly validates their information.

8. Manifest Structure

The manifest is divided into several sections in a hierarchy as follows:

1. The outer wrapper

    1. The authentication wrapper

2. The manifest
    1. Critical Information
    2. List of dependencies
    3. List of payloads
    4. List of payloads in dependencies
    5. Common list of conditions, directives
    6. Dependency resolution Reference or list of conditions, directives
    7. Payload fetch Reference or list of conditions, directives
    8. Installation Reference or list of conditions, directives
    9. Verification conditions/directives
    10. Load conditions/directives
    11. Run conditions/directives
    12. Text / Reference
    13. COSWID / Reference
3. Dependency resolution conditions/directives
4. Payload fetch conditions/directives
5. Installation conditions/directives
6. Text
7. COSWID / Reference
8. Intermediate Certificate(s) / CWTs
9. Small Payload(s)

8.1. Outer wrapper

This object is a container for the other pieces of the manifest to provide a common mechanism to find each of the parts. All elements of the outer wrapper are contained in bstr objects. Wherever the manifest references an object in the outer wrapper, the bstr is included in the digest calculation.

The CDDL that describes the wrapper is below

```
SUIT_Outer_Wrapper = {
    suit-authentication-wrapper => bstr .cbor
                                   SUIT_Authentication_Wrapper / nil,
    suit-manifest               => bstr .cbor Manifest,
    suit-dependency-resolution  => bstr .cbor SUIT_Command_Sequence,
    suit-payload-fetch          => bstr .cbor SUIT_Command_Sequence,
    suit-install                => bstr .cbor SUIT_Command_Sequence,
    suit-text-external          => bstr .cbor SUIT_Text_Info,
    suit-coswid-external        => bstr .cbor COSWID
}
suit-authentication-wrapper = 1
suit-manifest = 2
suit-dependency-resolution = 7
suit-payload-fetch = 8
suit-install = 9
suit-text = 13
suit-coswid = 14

SUIT_Authentication_Wrapper = [ * (COSE_Mac_Tagged / COSE_Sign_Tagged /
                                   COSE_Mac0_Tagged / COSE_Sign1_Tagged)]
```

All elements of the outer wrapper must be wrapped in a bstr to minimize the complexity of the code that evaluates the cryptographic integrity of the element and to ensure correct serialisation for integrity and authenticity checks.

The suit-authentication-wrapper contains a list of 1 or more cryptographic authentication wrappers for the core part of the manifest. These are implemented as COSE_Mac_Tagged or COSE_Sign_Tagged blocks. The Manifest is authenticated by these blocks in "detached payload" mode. The COSE_Mac_Tagged and COSE_Sign_Tagged blocks are described in RFC 8152 [RFC8152] and are beyond the scope of this document. The suit-authentication-wrapper MUST come first in the SUIT_Outer_Wrapper, regardless of canonical encoding of CBOR. All validators MUST reject any SUIT_Outer_Wrapper that begins with any element other than a suit-authentication-wrapper.

A manifest that has not had authentication information added MUST still contain the suit-authentication-wrapper element, but the content MUST be null.

suit-manifest contains a Manifest structure, which describes the payload(s) to be installed and any dependencies on other manifests.

Each of suit-dependency-resolution, suit-payload-fetch, and suit-payload-installation contain the severable contents of the identically named portions of the manifest, described in Section 8.2.

suit-text contains all the human-readable information that describes any and all parts of the manifest, its payload(s) and its resource(s).

suit-coswid contains a Concise Software Identifier. This may be discarded by the recipient if not needed.

8.2. Manifest

The manifest describes the critical metadata for the referenced payload(s). In addition, it contains:

1. a version number for the manifest structure itself

2. a sequence number

3. a list of dependencies

4. a list of components affected

5. a list of components affected by dependencies

6. a reference for each of the severable blocks.

7. a list of actions that the recipient should perform.

The following CDDL fragment defines the manifest.

```
SUIT_Manifest = {
    suit-manifest-version         => 1,
    suit-manifest-sequence-number => uint,
    ? suit-dependencies           => [ + SUIT_Dependency ],
    ? suit-components             => [ + SUIT_Component ],
    ? suit-dependency-components  => [ + SUIT_Component_Reference ],
    ? suit-common                 => bstr .cbor SUIT_Command_Sequence,
    ? suit-dependency-resolution  => Digest / bstr .cbor SUIT_Command_Sequence,
    ? suit-payload-fetch          => Digest / bstr .cbor SUIT_Command_Sequence,
    ? suit-install                => Digest / bstr .cbor SUIT_Command_Sequence
    ? suit-validate               => bstr .cbor SUIT_Command_Sequence
    ? suit-load                   => bstr .cbor SUIT_Command_Sequence
    ? suit-run                    => bstr .cbor SUIT_Command_Sequence
    ? suit-text-info              => Digest / bstr .cbor SUIT_Text_Map
    ? suit-coswid                 => Digest / bstr .cbor COSWID
}

suit-manifest-version = 1
suit-manifest-sequence-number = 2
suit-dependencies = 3
suit-components = 4
suit-dependency-components = 5
suit-common = 6
suit-dependency-resolution = 7
suit-payload-fetch = 8
suit-install = 9
suit-validate = 10
suit-load = 11
suit-run = 12
suit-text-info = 13
suit-coswid = 14
```

Several fields in the Manifest can be either a CBOR structure or a SUIT_Digest. In each of these cases, the SUIT_Digest provides for a severable field. Severable fields are RECOMMENDED to implement. In particular, text SHOULD be severable, since most useful text elements occupy more space than a SUIT_Digest, but are not needed by recipient devices. Because SUIT_Digest is a CBOR Array and each severable element is a CBOR bstr, it is straight-forward for a recipient to determine whether an element has been severed.

The suit-manifest-version indicates the version of serialisation used to encode the manifest. Version 1 is the version described in this document. suit-manifest-version is MANDATORY.

The suit-manifest-sequence-number is a monotonically increasing anti-rollback counter. It also helps devices to determine which in a set of manifests is the "root" manifest in a given update. Each manifest

MUST have a sequence number higher than each of its dependencies. Each recipient MUST reject any manifest that has a sequence number lower than its current sequence number. It MAY be convenient to use a UTC timestamp in seconds as the sequence number. suit-manifest-sequence-number is MANDATORY.

suit-dependencies is a list of SUIT_Dependency blocks that specify manifests that must be present before the current manifest can be processed. suit-dependencies is OPTIONAL.

In order to distinguish between components that are affected by the current manifest and components that are affected by a dependency, they are kept in separate lists. Components affected by the current manifest include the digest and size of the result. Components affected by a manifest only include the component identifier and the index of the manifest that fully defines the component.

suit-components is a list of SUIT_Component blocks that specify the vital information about the content a component identifier should contain following the update. These are the component identifiers that will be affected by the content of the current manifest. suit-components is OPTIONAL, but at least one manifest MUST contain a suit-components block.

suit-dependency-components is a list of SUIT_Component_Reference blocks that specify component identifiers that will be affected by the content of a dependency of the current manifest. suit-dependency-components is OPTIONAL.

suit-common is a SUIT_Command_Sequence to execute prior to executing any other command sequence. Typical actions in suit-common include setting expected device identity and image digests when they are conditional (see Section 11 for more information on conditional sequences). suit-common is OPTIONAL.

suit-dependency-resolution is a SUIT_Command_Sequence to execute in order to perform dependency resolution. Typical actions include configuring URIs of dependency manifests, fetching dependency manifests, and validating dependency manifests' contents. suit-dependency-resolution is MANDATORY when suit-dependencies is present.

suit-payload-fetch is a SUIT_Command_Sequence to execute in order to obtain a payload. Some manifests may include these actions in the suit-install section instead if they operate in a streaming installation mode. This is particularly relevant for constrained devices without any temporary storage for staging the update. suit-payload-fetch is OPTIONAL.

suit-install is a SUIT_Command_Sequence to execute in order to install a payload. Typical actions include verifying a payload stored in temporary storage, copying a staged payload from temporary storage, and unpacking a payload. suit-install is OPTIONAL.

suit-validate is a SUIT_Command_Sequence to execute in order to validate that the result of applying the update is correct. Typical actions involve image validation and manifest validation. suit-validate is MANDATORY. If the manifest contains dependencies, one process-dependency invocation per dependency or one process-dependency invocation targeting all dependencies SHOULD be present in validate.

suit-load is a SUIT_Command_Sequence to execute in order to prepare a payload for execution. Typical actions include copying an image from permanent storage into RAM, optionally including actions such as decryption or decompression. suit-load is OPTIONAL.

suit-run is a SUIT_Command_Sequence to execute in order to run an image. suit-run typically contains a single instruction: either the "run" directive for the bootable manifest or the "process dependencies" directive for any dependents of the bootable manifest. suit-run is OPTIONAL. Only one manifest in an update may contain the "run" directive.

suit-text-info is a digest that uniquely identifies the content of the Text that is packaged in the OuterWrapper. text is OPTIONAL.

suit-coswid is a digest that uniquely identifies the content of the concise-software-identifier that is packaged in the OuterWrapper. coswid is OPTIONAL.

8.3. SUIT_Dependency

SUIT_Dependency specifies a manifest that describes a dependency of the current manifest.

The following CDDL describes the SUIT_Dependency structure.

```
SUIT_Dependency = {
    suit-dependency-digest => SUIT_Digest,
    suit-dependency-prefix => SUIT_Component_Identifier,
}
```

The suit-dependency-digest specifies the dependency manifest uniquely by identifying a particular Manifest structure. The digest is calculated over the Manifest structure instead of the COSE Sig_structure or Mac_structure. This means that a digest may need to

be calculated more than once, however this is necessary to ensure that removing a signature from a manifest does not break dependencies due to missing 'body_protected' and 'body_signed' elements. This is also necessary to support the trusted intermediary use case, where an intermediary re-signs the Manifest, removing the original signature, potentially with a different algorithm, or trading COSE_Sign for COSE_Mac.

The suit-dependency-prefix element contains a SUIT_Component_Identifier. This specifies the scope at which the dependency operates. This allows the dependency to be forwarded on to a component that is capable of parsing its own manifests. It also allows one manifest to be deployed to multiple dependent devices without those devices needing consistent component hierarchy. This element is OPTIONAL.

8.4. SUIT_Component

The SUIT_Component describes an image that is uniquely defined by the current manifest. It consists of three elemnts: the component identifier that represents a component that will be affected by this manifest. This excludes components that are affected by dependencies. The following CDDL describes the SUIT_Component.

```
SUIT_Component = {
    suit-component-identifier => SUIT_Component_Identifier,
    ? suit-component-size => uint,
    ? suit-component-digest => Digest,
}
```

Because suit-component-size and suit-component-digest can be dependent on installation offset, they cannot be exclusively contained in SUIT_Component. However, since these are security critical parameters, these parameters are updated to match the contents of suit-components prior to processing suit-common. If absent, these are set to Zero and NULL, respectively. This enforces that the manifest defining a component is the only manifest that can describe its contents.

8.5. SUIT_Component_Reference

The SUIT_Component_Reference describes an image that is defined by another manifest. This is useful for overriding the behaviour of another manifest, for example by directing the recipient to look at a different URI for the image or by changing the expected format, such as when a gateway performs decryption on behalf of a constrained device. The following CDDL describes the SUIT_Component_Reference.

```
SUIT_Component_Reference = {
    suit-component-identifier => SUIT_Component_Identifier,
    suit-component-dependency-index => uint
}
```

8.6. Manifest Parameters

Many conditions and directives require additional information. That information is contained within parameters that can be set in a consistent way. Parameters MUST only be:

1. Integers
2. Byte strings
3. Booleans

This allows reduction of manifest size and replacement of parameters from one manifest to the next. Byte strings MAY contain CBOR-encoded objects.

The defined manifest parameters are described below.

Parameter Code	CBOR Type	Default	Scope	Name	Description
1	boolean	1	Global	Strict Order	Requires that the manifest is processed in a strictly linear fashion. Set to 0 to enable parallel handling of manifest directives.
2	boolean	0	Global	Coerce Condition on Failure	Coerces the success code of a command segment to success even when aborted due to a condition failure.
3	bstr	nil	Component/Global	Vendor ID	A RFC4122 UUID representing the

					vendor of the device or component
4	bstr	nil	Component/Global	Class ID	A RFC4122 UUID representing the class of the device or component
5	bstr	nil	Component/Global	Device ID	A RFC4122 UUID representing the device or component
6	bstr	nil	Component/Dependency	URI List	A CBOR encoded list of ranked URIs
7	bstr	nil	Component/Dependency	Encryption Info	A COSE object defining the encryption mode of the target
8	bstr	nil	Component	Compression Info	A SUIT_Compression_Info object
9	bstr	nil	Component	Unpack Info	A SUIT_Unpack_Info object
10	int/bstr	nil	Component	Source Component	A SUIT_Component_Identifier or Component Index
11	bstr	nil	Component/Dependency	Image Digest	A SUIT_Digest
12	bstr	nil	Component/Dependency	Image Size	Integer size
nint	int/bstr	nil	Custom	Custom Parameter	Application-defined parameter

Each parameter contains a Skip/Append flag. Append is an advanced feature that is not available on highly constrained platforms. The mechanism for setting the Append flag is TBD.

CBOR-encoded object parameters are still wrapped in a bstr. This is because it allows a parser that is aggregating parameters to reference the object with a single pointer and traverse it without understanding the contents. This is important for modularisation and division of responsibility within a pull parser. The same consideration does not apply to Conditions and Directives because those elements are invoked with their arguments immediately

8.6.1. SUIT_Parameter_Strict_Order

The Strict Order Parameter allows a manifest to govern when directives can be executed out-of-order. This allows for systems that have a sensitivity to order of updates to choose the order in which they are executed. It also allows for more advanced systems to parallelise their handling of updates. Strict Order defaults to True. It MAY be set to False when the order of operations does not matter. When arriving at the end of a command sequence, ALL commands MUST have completed, regardless of the state of SUIT_Parameter_Strict_Order. If SUIT_Parameter_Strict_Order is returned to True, ALL preceding commands MUST complete before the next command is executed.

8.6.2. SUIT_Parameter_Coerce_Condition_Failure

When executing a command sequence inside SUIT_Run_Sequence and a condition failure occurs, the manifest processor aborts the sequence. If Coerce Condition Failure is True, it returns Success. Otherwise, it returns the original condition failure. SUIT_Parameter_Coerce_Condition_Failure is scoped to the enclosing SUIT_Directive_Run_Sequence. Its value is discarded when SUIT_Directive_Run_Sequence terminates.

8.7. SUIT_Parameter_Encryption_Info

Encryption Info defines the mechanism that Fetch or Copy should use to decrypt the data they transfer. SUIT_Parameter_Encryption_Info is encoded as a COSE_Encrypt_Tagged or a COSE_Encrypt0_Tagged, wrapped in a bstr

8.8. SUIT_Parameter_Compression_Info

Compression Info defines any information that is required for a device to perform decompression operations. Typically, this includes the algorithm identifier.

SUIT_Parameter_Compression_Info is defined by the following CDDL:

```
SUIT_Compression_Info = {
    suit-compression-algorithm => SUIT_Compression_Algorithms
    ? suit-compression-parameters => bstr
}
suit-compression-algorithm = 1
suit-compression-parameters = 2

SUIT_Compression_Algorithms /= SUIT_Compression_Algorithm_gzip
SUIT_Compression_Algorithms /= SUIT_Compression_Algorithm_bzip2
SUIT_Compression_Algorithms /= SUIT_Compression_Algorithm_deflate
SUIT_Compression_Algorithms /= SUIT_Compression_Algorithm_LZ4
SUIT_Compression_Algorithms /= SUIT_Compression_Algorithm_lzma

SUIT_Compression_Algorithm_gzip = 1
SUIT_Compression_Algorithm_bzip2 = 2
SUIT_Compression_Algorithm_deflate = 3
SUIT_Compression_Algorithm_lz4 = 4
SUIT_Compression_Algorithm_lzma = 7
```

8.9. SUIT_Parameter_Unpack_Info

SUIT_Unpack_Info defines the information required for a device to interpret a packed format, such as elf, hex, or binary diff. SUIT_Unpack_Info is defined by the following CDDL:

```
SUIT_Unpack_Info = {
    suit-unpack-algorithm => SUIT_Unpack_Algorithms
    ? suit-unpack-parameters => bstr
}
suit-unpack-algorithm  = 1
suit-unpack-parameters = 2

SUIT_Unpack_Algorithms //= SUIT_Unpack_Algorithm_Delta
SUIT_Unpack_Algorithms //= SUIT_Unpack_Algorithm_Hex
SUIT_Unpack_Algorithms //= SUIT_Unpack_Algorithm_Elf

SUIT_Unpack_Algorithm_Delta = 1
SUIT_Unpack_Algorithm_Hex = 2
SUIT_Unpack_Algorithm_Elf = 3
```

8.10. SUIT_Parameters CDDL

The following CDDL describes all SUIT_Parameters.

```
SUIT_Parameters //= SUIT_Parameter_Strict_Order
SUIT_Parameters //= SUIT_Parameter_Coerce_Condition_Failure
```

```
SUIT_Parameters //= SUIT_Parameter_Vendor_ID
SUIT_Parameters //= SUIT_Parameter_Class_ID
SUIT_Parameters //= SUIT_Parameter_Device_ID
SUIT_Parameters //= SUIT_Parameter_URI_List
SUIT_Parameters //= SUIT_Parameter_Encryption_Info
SUIT_Parameters //= SUIT_Parameter_Compression_Info
SUIT_Parameters //= SUIT_Parameter_Unpack_Info
SUIT_Parameters //= SUIT_Parameter_Source_Component
SUIT_Parameters //= SUIT_Parameter_Image_Digest
SUIT_Parameters //= SUIT_Parameter_Image_Size
SUIT_Parameters //= SUIT_Parameter_Custom

SUIT_Parameter_Strict_Order = (1 => bool)
SUIT_Parameter_Coerce_Condition_Failure = (2 => bool)
SUIT_Parameter_Vendor_ID = (3 => bstr)
SUIT_Parameter_Class_ID = (4 => bstr)
SUIT_Parameter_Device_ID = (5 => bstr)
SUIT_Parameter_URI_List = (6 => bstr .cbor SUIT_URI_List)
SUIT_Parameter_Encryption_Info = (7 => bstr .cbor SUIT_Encryption_Info)
SUIT_Parameter_Compression_Info = (8 => bstr .cbor SUIT_Compression_Info)
SUIT_Parameter_Unpack_Info = (9 => bstr .cbor SUIT_Unpack_Info)
SUIT_Parameter_Source_Component = (10 => bstr .cbor SUIT_Component_Identifier)
SUIT_Parameter_Image_Digest = (11 => bstr .cbor SUIT_Digest)
SUIT_Parameter_Image_Size = (12 => uint)
SUIT_Parameter_Custom = (nint => int/bool/bstr)

SUIT_URI_List = [ + [priority: int, uri: tstr] ]

SUIT_Encryption_Info= COSE_Encrypt_Tagged/COSE_Encrypt0_Tagged
SUIT_Compression_Info = {
    suit-compression-algorithm => SUIT_Compression_Algorithms
    ? suit-compression-parameters => bstr
}
suit-compression-algorithm = 1
suit-compression-parameters = 2

SUIT_Compression_Algorithms /= SUIT_Compression_Algorithm_gzip
SUIT_Compression_Algorithms /= SUIT_Compression_Algorithm_bzip2
SUIT_Compression_Algorithms /= SUIT_Compression_Algorithm_deflate
SUIT_Compression_Algorithms /= SUIT_Compression_Algorithm_LZ4
SUIT_Compression_Algorithms /= SUIT_Compression_Algorithm_lzma

SUIT_Compression_Algorithm_gzip = 1
SUIT_Compression_Algorithm_bzip2 = 2
SUIT_Compression_Algorithm_deflate = 3
SUIT_Compression_Algorithm_lz4 = 4
SUIT_Compression_Algorithm_lzma = 7
```

```
SUIT_Unpack_Info = {
    suit-unpack-algorithm => SUIT_Unpack_Algorithms
    ? suit-unpack-parameters => bstr
}
suit-unpack-algorithm  = 1
suit-unpack-parameters = 2

SUIT_Unpack_Algorithms //= SUIT_Unpack_Algorithm_Delta
SUIT_Unpack_Algorithms //= SUIT_Unpack_Algorithm_Hex
SUIT_Unpack_Algorithms //= SUIT_Unpack_Algorithm_Elf

SUIT_Unpack_Algorithm_Delta = 1
SUIT_Unpack_Algorithm_Hex = 2
SUIT_Unpack_Algorithm_Elf = 3
```

8.11. SUIT_Command_Sequence

A SUIT_Command_Sequence defines a series of actions that the recipient MUST take to accomplish a particular goal. These goals are defined in the manifest and include:

1. Dependency Resolution

2. Payload Fetch

3. Payload Installation

4. Image Validation

5. Image Loading

6. Run or Boot

Each of these follows exactly the same structure to ensure that the parser is as simple as possible.

Lists of commands are constructed from two kinds of element:

1. Conditions that MUST be true-any failure is treated as a failure of the update/load/boot

2. Directives that MUST be executed.

The lists of commands are logically structured into sequences of zero or more conditions followed by zero or more directives. The *logical* structure is described by the following CDDL:

```
Command_Sequence = {
    conditions => [ * Condition],
    directives => [ * Directive]
}
```

This introduces significant complexity in the parser, however, so the structure is flattened to make parsing simpler:

```
SUIT_Command_Sequence = [ + (SUIT_Condition/SUIT_Directive) ]
```

Each condition and directive is composed of:

1. A command code identifier

2. An argument block

Argument blocks are defined for each type of command.

Many conditions and directives apply to a given component, and these generally grouped together. Therefore, a special command to set the current component index is provided with a matching command to set the current manifest index. This index is a numeric index into the component ID tables defined at the beginning of the document. For the purpose of setting the index, the two component ID tables are considered to be concatenated together.

To facilitate optional conditions, a special directive is provided. It runs a new list of conditions/directives that are contained as an argument to the directive. It also contains a flag that indicates whether or not a failure of a condition should indicate a failure of the update/boot.

8.12. SUIT_Condition

Conditions are used to define mandatory properties of a system in order for an update to be applied. They can be pre-conditions or post-conditons of any directive or series of directives, depending on where they are placed in the list. Conditions include:

Condition Code	Condition Name	Argument Type
1	Vendor Identifier	RFC4122 UUID wrapped in a bstr
2	Class Identifier	RFC4122 UUID wrapped in a bstr
3	Device Identifier	RFC4122 UUID wrapped in a bstr
4	Image Match	SUIT_Digest
5	Image Not Match	SUIT_Digest
6	Use Before	Unsigned Integer timestamp
7	Minimum Battery	Unsigned Integer
8	Update Authorised	Integer
9	Version	List of Integers
10	Component Offset	Unsigned Integer
nint	Custom Condition	bstr

Each condition MUST report a success code on completion. If a condition reports failure, then the current sequence of commands MUST terminate. If a recipient encounters an unknown Condition Code, it MUST report a failure.

Positive Condition numbers are reserved for IANA registration. Negative numbers are reserved for proprietary, application-specific directives.

8.12.1. ID Conditions

There are three identifier-based conditions: SUIT_Condition_Vendor_Identifier, SUIT_Condition_Class_Identifier, and SUIT_Condition_Device_Identifier. Each of these conditions present a RFC 4122 [RFC4122] UUID that MUST be matched by the installing device in order to consider the manifest valid.

These conditions MAY be used with or without an argument. If an argument is supplied, then it must be the RFC 4122 [RFC4122] UUID

that must be matched for the condition to succeed. If no argument is supplied, then the recipient uses the ID parameter that has already been set using the Set Parameters directive. If no ID has been set, this condition fails. SUIT_Condition_Class_Identifier and SUIT_Condition_Vendor_Identifier are MANDATORY to implement. SUIT_Condition_Device_Identifier is OPTIONAL to implement.

8.12.2. SUIT_Condition_Image_Match

Verify that the current component matches the supplied digest. If no digest is specified, then the digest is verified against the digest specified in the Components list. If no digest is specified and the component is not present in the Components list, the condition fails. SUIT_Condition_Image_Match is MANDATORY to implement.

8.12.3. SUIT_Condition_Image_Not_Match

Verify that the current component does not match the supplied digest. If no digest is specified, then the digest is compared against the digest specified in the Components list. If no digest is specified and the component is not present in the Components list, the condition fails. SUIT_Condition_Image_Not_Match is OPTIONAL to implement.

8.12.4. SUIT_Condition_Use_Before

Verify that the current time is BEFORE the specified time. SUIT_Condition_Use_Before is used to specify the last time at which an update should be installed. One argument is required, encoded as a POSIX timestamp, that is seconds after 1970-01-01 00:00:00. Timestamp conditions MUST be evaluated in 64 bits, regardless of encoded CBOR size. SUIT_Condition_Use_Before is OPTIONAL to implement.

8.12.5. SUIT_Condition_Minimum_Battery

SUIT_Condition_Minimum_Battery provides a mechanism to test a device's battery level before installing an update. This condition is for use in primary-cell applications, where the battery is only ever discharged. For batteries that are charged, SUIT_Directive_Wait_Event is more appropriate, since it defines a "wait" until the battery level is sufficient to install the update. SUIT_Condition_Minimum_Battery is specified in mWh. SUIT_Condition_Minimum_Battery is OPTIONAL to implement.

8.12.6. SUIT_Condition_Update_Authorised

Request Authorisation from the application and fail if not authorised. This can allow a user to decline an update. Argument is an integer priority level. Priorities are application defined. SUIT_Condition_Update_Authorised is OPTIONAL to implement.

8.12.7. SUIT_Condition_Version

SUIT_Condition_Version allows comparing versions of firmware. Verifying image digests is preferred to version checks because digests are more precise. The image can be compared as:

- Greater
- Greater or Equal
- Equal
- Lesser or Equal
- Lesser

Versions are encoded as a CBOR list of integers. Comparisons are done on each integer in sequence.

The following CDDL describes SUIT_Condition_Version_Argument

```
SUIT_Condition_Version_Argument = [
    suit-condition-version-comparison: SUIT_Condition_Version_Comparison_Types,
    suit-condition-version-comparison: SUIT_Condition_Version_Comparison_Value
]
SUIT_Condition_Version_Comparison_Types /=
SUIT_Condition_Version_Comparison_Greater
SUIT_Condition_Version_Comparison_Types /=
SUIT_Condition_Version_Comparison_Greater_Equal
SUIT_Condition_Version_Comparison_Types /=
SUIT_Condition_Version_Comparison_Equal
SUIT_Condition_Version_Comparison_Types /=
SUIT_Condition_Version_Comparison_Lesser_Equal
SUIT_Condition_Version_Comparison_Types /=
SUIT_Condition_Version_Comparison_Lesser
SUIT_Condition_Version_Comparison_Greater = 1
SUIT_Condition_Version_Comparison_Greater_Equal = 2
SUIT_Condition_Version_Comparison_Equal = 3
SUIT_Condition_Version_Comparison_Lesser_Equal = 4
SUIT_Condition_Version_Comparison_Lesser = 5

SUIT_Condition_Version_Comparison_Value = [+int]
```

While the exact encoding of versions is application-defined, semantic versions map directly:

- 1.2.3 = [1,2,3]
- 1.2-rc3 = [1,2,-1,3]
- 1.2-beta = [1,2,-2]
- 1.2-alpha = [1,2,-3]
- 1.2-alpha4 = [1,2,-3,4]

SUIT_Condition_Version is OPTIONAL to implement.

8.12.8. SUIT_Condition_Custom

SUIT_Condition_Custom describes any proprietary, application specific condition. This is encoded as a negative integer, chosen by the firmware developer, and a bstr that encodes the parameters passed to the system that evaluates the condition matching that integer. SUIT_Condition_Custom is OPTIONAL to implement.

8.12.9. Identifiers

Many conditions use identifiers to determine whether a manifest matches a given recipient or not. These identifiers are defined to be RFC 4122 [RFC4122] UUIDs. These UUIDs are explicitly NOT human-readable. They are for machine-based matching only.

A device may match any number of UUIDs for vendor or class identifier. This may be relevant to physical or software modules. For example, a device that has an OS and one or more applications might list one Vendor ID for the OS and one or more additional Vendor IDs for the applications. This device might also have a Class ID that must be matched for the OS and one or more Class IDs for the applications.

A more complete example: A device has the following physical components: 1. A host MCU 2. A WiFi module

This same device has three software modules: 1. An operating system 2. A WiFi module interface driver 3. An application

Suppose that the WiFi module's firmware has a proprietary update mechanism and doesn't support manifest processing. This device can report four class IDs:

1. hardware model/revision
2. OS

3. WiFi module model/revision

4. Application

This allows the OS, WiFi module, and application to be updated independently. To combat possible incompatibilities, the OS class ID can be changed each time the OS has a change to its API.

This approach allows a vendor to target, for example, all devices with a particular WiFi module with an update, which is a very powerful mechanism, particularly when used for security updates.

8.12.9.1. Creating UUIDs:

UUIDs MUST be created according to RFC 4122 [RFC4122]. UUIDs SHOULD use versions 3, 4, or 5, as described in RFC4122. Versions 1 and 2 do not provide a tangible benefit over version 4 for this application.

The RECOMMENDED method to create a vendor ID is: Vendor ID = UUID5(DNS_PREFIX, vendor domain name)

The RECOMMENDED method to create a class ID is: Class ID = UUID5(Vendor ID, Class-Specific-Information)

Class-specific information is composed of a variety of data, for example:

- Model number

- Hardware revision

- Bootloader version (for immutable bootloaders)

8.12.10. SUIT_Condition CDDL

The following CDDL describes SUIT_Condition:

```
SUIT_Condition //= (nint => bstr)
SUIT_Condition //= SUIT_Condition_Vendor_Identifier
SUIT_Condition //= SUIT_Condition_Class_Identifier
SUIT_Condition //= SUIT_Condition_Device_Identifier
SUIT_Condition //= SUIT_Condition_Image_Match
SUIT_Condition //= SUIT_Condition_Image_Not_Match
SUIT_Condition //= SUIT_Condition_Use_Before
SUIT_Condition //= SUIT_Condition_Minimum_Battery
SUIT_Condition //= SUIT_Condition_Update_Authorised
SUIT_Condition //= SUIT_Condition_Version
SUIT_Condition //= SUIT_Condition_Component_Offset
SUIT_Condition //= SUIT_Condition_Custom

SUIT_Condition_Vendor_Identifier = (1 => bstr .size 16)
SUIT_Condition_Class_Identifier = (2 => bstr .size 16)
SUIT_Condition_Device_Identifier = (3 => bstr .size 16)
SUIT_Condition_Image_Match = (4 => SUIT_Digest)
SUIT_Condition_Image_Not_Match = (5 => SUIT_Digest)
SUIT_Condition_Use_Before = (6 => uint)
SUIT_Condition_Minimum_Battery = (7 => uint)
SUIT_Condition_Update_Authorised = (8 => int)
SUIT_Condition_Version = (9 => SUIT_Condition_Version_Argument)
SUIT_Condition_Component_Offset = (10 => uint)
SUIT_Condition_Custom = (nint => bstr)

SUIT_Condition_Version_Argument = [
    suit-condition-version-comparison: SUIT_Condition_Version_Comparison_Types,
    suit-condition-version-comparison: SUIT_Condition_Version_Comparison_Value
]
SUIT_Condition_Version_Comparison_Types /=
SUIT_Condition_Version_Comparison_Greater
SUIT_Condition_Version_Comparison_Types /=
SUIT_Condition_Version_Comparison_Greater_Equal
SUIT_Condition_Version_Comparison_Types /=
SUIT_Condition_Version_Comparison_Equal
SUIT_Condition_Version_Comparison_Types /=
SUIT_Condition_Version_Comparison_Lesser_Equal
SUIT_Condition_Version_Comparison_Types /=
SUIT_Condition_Version_Comparison_Lesser

SUIT_Condition_Version_Comparison_Greater = 1
SUIT_Condition_Version_Comparison_Greater_Equal = 2
SUIT_Condition_Version_Comparison_Equal = 3
SUIT_Condition_Version_Comparison_Lesser_Equal = 4
SUIT_Condition_Version_Comparison_Lesser = 5

SUIT_Condition_Version_Comparison_Value = [+int]
```

8.13. SUIT_Directive

Directives are used to define the behaviour of the recipient.
Directives include:

Directive Code	Directive Name
11	Set Component Index
12	Set Manifest Index
13	Run Sequence
14	Run Sequence Conditional
15	Process Dependency
16	Set Parameters
17	Reserved
18	Reserved
19	Override Parameters
20	Fetch
21	Copy
22	Run
23	Wait

When a Recipient executes a Directive, it MUST report a success code. If the Directive reports failure, then the current Command Sequence MUST terminate.

8.13.1. SUIT_Directive_Set_Component_Index

Set Component Index defines the component to which successive directives and conditions will apply. The supplied argument MUST be either a boolean or an unsigned integer index into the concatenation of suit-components and suit-dependency-components. If the following directives apply to ALL components, then the boolean value "True" is used instead of an index. True does not apply to dependency components. If the following directives apply to NO components, then the boolean value "False" is used. When SUIT_Directive_Set_Manifest_Index is used, SUIT_Directive_Set_Component_Index = False is implied. When SUIT_Directive_Set_Component_Index is used, SUIT_Directive_Set_Manifest_Index = False is implied.

The following CDDL describes the argument to SUIT_Directive_Set_Component_Index.

SUIT_Directive_Set_Component_Index_Argument = uint/bool

8.13.2. SUIT_Directive_Set_Manifest_Index

Set Manifest Index defines the manifest to which successive directives and conditions will apply. The supplied argument MUST be either a boolean or an unsigned integer index into the dependencies. If the following directives apply to ALL dependencies, then the boolean value "True" is used instead of an index. If the following directives apply to NO dependencies, then the boolean value "False" is used. When SUIT_Directive_Set_Component_Index is used, SUIT_Directive_Set_Manifest_Index = False is implied. When SUIT_Directive_Set_Manifest_Index is used, SUIT_Directive_Set_Component_Index = False is implied.

Typical operations that require SUIT_Directive_Set_Manifest_Index include setting a source URI, invoking "Fetch," or invoking "Process Dependency" for an individual dependency.

The following CDDL describes the argument to SUIT_Directive_Set_Manifest_Index.

SUIT_Directive_Set_Manifest_Index_Argument = uint/bool

8.13.3. SUIT_Directive_Run_Sequence

To enable conditional commands, and to allow several strictly ordered sequences to be executed out-of-order, SUIT_Run_Sequence allows the manifest processor to execute its argument as a SUIT_Command_Sequence. The argument must be wrapped in a bstr.

When a sequence is executed, any failure of a condition causes immediate termination of the sequence.

The following CDDL describes the SUIT_Run_Sequence argument.

```
SUIT_Directive_Run_Sequence_Argument = bstr .cbor SUIT_Command_Sequence
```

When SUIT_Directive_Run_Sequence completes, it forwards the last status code that occurred in the sequence. If the Coerce on Condition Failure parameter is true, then SUIT_Directive_Run_Sequence only fails when a directive in the argument sequence fails.

SUIT_Parameter_Coerce_Condition_Failure defaults to False when SUIT_Directive_Run_Sequence begins. Its value is discarded when SUIT_Directive_Run_Sequence terminates.

8.13.4. SUIT_Directive_Run_Sequence_Conditional

This command is exactly the same as SUIT_Directive_Run_Sequence, except that it initialises Coerce on Condition Failure to True.

SUIT_Parameter_Coerce_Condition_Failure defaults to True when SUIT_Directive_Run_Sequence_Conditional begins. Its value is discarded when SUIT_Directive_Run_Sequence_Conditional terminates.

8.13.5. SUIT_Directive_Process_Dependency

Execute the commands in the common section of the current dependency, followed by the commands in the equivalent section of the current dependency. For example, if the current section is "fetch payload," this will execute "common" in the current dependency, then "fetch payload" in the current dependency. Once this is complete, the command following SUIT_Directive_Process_Dependency will be processed.

If the current dependency is False, this directive has no effect. If the current dependency is True, then this directive applies to all dependencies. If the current section is "common," this directive MUST have no effect.

When SUIT_Process_Dependency completes, it forwards the last status code that occurred in the dependency.

The argument to SUIT_Directive_Process_Dependency is defined in the following CDDL.

SUIT_Directive_Process_Dependency_Argument = nil

8.13.6. SUIT_Directive_Set_Parameters

SUIT_Directive_Set_Parameters allows the manifest to configure behaviour of future directives by changing parameters that are read by those directives. When dependencies are used, SUIT_Directive_Set_Parameters also allows a manifest to modify the behaviour of its dependencies.

Available parameters are defined in Section 8.6.

If a parameter is already set, SUIT_Directive_Set_Parameters will skip setting the parameter to its argument. This provides the core

of the override mechanism, allowing dependent manifests to change the behaviour of a manifest.

The argument to SUIT_Directive_Set_Parameters is defined in the following CDDL.

```
SUIT_Directive_Set_Parameters_Argument = {+ SUIT_Parameters}
```

N.B.: A directive code is reserved for an optimisation: a way to set a parameter to the contents of another parameter, optionally with another component ID.

8.13.7. SUIT_Directive_Set_Parameter_State_Append

This command is reserved for future use. It will provide a mechanism to override the "set if unset" logic of SUIT_Directive_Set_Parameters on a per-parameter basis. This will allow certain parameters to be treated as lists, rather than fixed values. This enables a feature for an advanced device to fail over from URIs defined in one manifest to those defined in another.

8.13.8. SUIT_Directive_Override_Parameters

SUIT_Directive_Override_Parameters replaces any listed parameters that are already set with the values that are provided in its argument. This allows a manifest to prevent replacement of critical parameters.

Available parameters are defined in Section 8.6.

The argument to SUIT_Directive_Override_Parameters is defined in the following CDDL.

```
SUIT_Directive_Override_Parameters_Argument = {+ SUIT_Parameters}
```

8.13.9. SUIT_Directive_Fetch

SUIT_Directive_Fetch instructs the manifest processor to obtain one or more manifests or payloads, as specified by the manifest index and component index, respectively.

SUIT_Directive_Fetch can target one or more manifests and one or more payloads. SUIT_Directive_Fetch retrieves each component and each manifest listed in component-index and manifest-index, respectively. If component-index or manifest-index is True, instead of an integer, then all current manifest components/manifests are fetched. The current manifest's dependent-components are not automatically

fetched. In order to pre-fetch these, they MUST be specified in a component-index integer.

SUIT_Directive_Fetch typically takes no arguments unless one is needed to modify fetch behaviour. If an argument is needed, it must be wrapped in a bstr.

SUIT_Directive_Fetch reads the URI List parameter to find the source of the fetch it performs.

The behaviour of SUIT_Directive_Fetch can be modified by setting one or more of SUIT_Parameter_Encryption_Info, SUIT_Parameter_Compression_Info, SUIT_Parameter_Unpack_Info. These three parameters each activate and configure a processing step that can be applied to the data that is transferred during SUIT_Directive_Fetch.

The argument to SUIT_Directive_Fetch is defined in the following CDDL.

```
SUIT_Directive_Fetch_Argument = nil/bstr
```

8.13.10. SUIT_Directive_Copy

SUIT_Directive_Copy instructs the manifest processor to obtain one or more payloads, as specified by the component index. SUIT_Directive_Copy retrieves each component listed in component-index, respectively. If component-index is True, instead of an integer, then all current manifest components are copied. The current manifest's dependent-components are not automatically copied. In order to copy these, they MUST be specified in a component-index integer.

The behaviour of SUIT_Directive_Copy can be modified by setting one or more of SUIT_Parameter_Encryption_Info, SUIT_Parameter_Compression_Info, SUIT_Parameter_Unpack_Info. These three parameters each activate and configure a processing step that can be applied to the data that is transferred during SUIT_Directive_Copy.

N.B. Fetch and Copy are very similar. Merging them into one command may be appropriate.

SUIT_Directive_Copy reads its source from SUIT_Parameter_Source_Component.

The argument to SUIT_Directive_Copy is defined in the following CDDL.

```
SUIT_Directive_Copy_Argument = nil
```

8.13.11. SUIT_Directive_Run

SUIT_Directive_Run directs the manifest processor to transfer execution to the current Component Index. When this is invoked, the manifest processor MAY be unloaded and execution continues in the Component Index. Arguments provided to Run are forwarded to the executable code located in Component Index, in an application-specific way. For example, this could form the Linux Kernel Command Line if booting a linux device.

If the executable code at Component Index is constructed in such a way that it does not unload the manifest processor, then the manifest processor may resume execution after the executable completes. This allows the manifest processor to invoke suitable helpers and to verify them with image conditions.

The argument to SUIT_Directive_Run is defined in the following CDDL.

```
SUIT_Directive_Run_Argument = nil/bstr
```

8.13.12. SUIT_Directive_Wait

SUIT_Directive_Wait directs the manifest processor to pause until a specified event occurs. Some possible events include:

1. Authorisation
2. External Power
3. Network availability
4. Other Device Firmware Version
5. Time
6. Time of Day
7. Day of Week

The following CDDL defines the encoding of these events.

```
SUIT_Directive_Wait_Argument = {
    SUIT_Wait_Events
}
SUIT_Wait_Events //= (1 => SUIT_Wait_Event_Argument_Authorisation)
SUIT_Wait_Events //= (2 => SUIT_Wait_Event_Argument_Power)
SUIT_Wait_Events //= (3 => SUIT_Wait_Event_Argument_Network)
SUIT_Wait_Events //= (4 => SUIT_Wait_Event_Argument_Other_Device_Version)
SUIT_Wait_Events //= (5 => SUIT_Wait_Event_Argument_Time)
SUIT_Wait_Events //= (6 => SUIT_Wait_Event_Argument_Time_Of_Day)
SUIT_Wait_Events //= (7 => SUIT_Wait_Event_Argument_Day_Of_Week)


SUIT_Wait_Event_Argument_Authorisation = int ; priority
SUIT_Wait_Event_Argument_Power = int ; Power Level
SUIT_Wait_Event_Argument_Network = int ; Network State
SUIT_Wait_Event_Argument_Other_Device_Version = [
    other-device: bstr,
    other-device-version: [+int]
]
SUIT_Wait_Event_Argument_Time = uint ; Timestamp
SUIT_Wait_Event_Argument_Time_Of_Day = uint ; Time of Day (seconds since
00:00:00)
SUIT_Wait_Event_Argument_Day_Of_Week = uint ; Days since Sunday
```

8.13.13. SUIT_Directive CDDL

The following CDDL describes SUIT_Directive:

```
SUIT_Directive //= SUIT_Directive_Set_Component_Index
SUIT_Directive //= SUIT_Directive_Set_Manifest_Index
SUIT_Directive //= SUIT_Directive_Run_Sequence
SUIT_Directive //= SUIT_Directive_Run_Sequence_Conditional
SUIT_Directive //= SUIT_Directive_Process_Dependency
SUIT_Directive //= SUIT_Directive_Set_Parameters
SUIT_Directive //= SUIT_Directive_Override_Parameters
SUIT_Directive //= SUIT_Directive_Fetch
SUIT_Directive //= SUIT_Directive_Copy
SUIT_Directive //= SUIT_Directive_Run
SUIT_Directive //= SUIT_Directive_Wait

SUIT_Directive_Set_Component_Index = (11 => uint/bool)
SUIT_Directive_Set_Manifest_Index = (12 => uint/bool)
SUIT_Directive_Run_Sequence = (13 => bstr .cbor SUIT_Command_Sequence)
SUIT_Directive_Run_Sequence_Conditional = (14 => bstr .cbor
SUIT_Command_Sequence)
SUIT_Directive_Process_Dependency = (15 => nil)
SUIT_Directive_Set_Parameters = (16 => {+ SUIT_Parameters})
SUIT_Directive_Override_Parameters = (19 => {+ SUIT_Parameters})
SUIT_Directive_Fetch = (20 => nil/bstr)
SUIT_Directive_Copy = (21 => nil/bstr)
SUIT_Directive_Run = (22 => nil/bstr)
SUIT_Directive_Wait = (23 => { + SUIT_Wait_Events })

SUIT_Wait_Events //= (1 => SUIT_Wait_Event_Argument_Authorisation)
SUIT_Wait_Events //= (2 => SUIT_Wait_Event_Argument_Power)
SUIT_Wait_Events //= (3 => SUIT_Wait_Event_Argument_Network)
SUIT_Wait_Events //= (4 => SUIT_Wait_Event_Argument_Other_Device_Version)
SUIT_Wait_Events //= (5 => SUIT_Wait_Event_Argument_Time)
SUIT_Wait_Events //= (6 => SUIT_Wait_Event_Argument_Time_Of_Day)
SUIT_Wait_Events //= (7 => SUIT_Wait_Event_Argument_Day_Of_Week)


SUIT_Wait_Event_Argument_Authorisation = int ; priority
SUIT_Wait_Event_Argument_Power = int ; Power Level
SUIT_Wait_Event_Argument_Network = int ; Network State
SUIT_Wait_Event_Argument_Other_Device_Version = [
    other-device: bstr,
    other-device-version: [+int]
]
SUIT_Wait_Event_Argument_Time = uint ; Timestamp
SUIT_Wait_Event_Argument_Time_Of_Day = uint ; Time of Day (seconds since
00:00:00)
SUIT_Wait_Event_Argument_Day_Of_Week = uint ; Days since Sunday
```

9. Dependency processing

Dependencies need careful handling on constrained systems. A dependency tree that is too deep can cause recursive handling to overflow stack space. Systems that parse all dependencies into an object tree can easily fill up available memory. Too many dependencies can overrun available storage space.

The dependency handling system in this document is designed to address as many of these problems as possible.

Dependencies MAY be addressed in one of three ways:

1. Iterate by component

2. Iterate by manifest

3. Out-of-order

Because each manifest has a list of components and a list of components defined by its dependencies, it is possible for the manifest processor to handle one component at a time, traversing the manifest tree once for each listed component. This, however consumes significant processing power.

Alternatively, it is possible for a device with sufficient memory to accumulate all parameters for all listed component IDs. This will naturally consume more memory, but it allows the device to process the manifests in a single pass.

It is expected that the simplest and most power sensitive devices will use option 2, with a fixed maximum number of components.

Advanced devices may make use of the Strict Order parameter and enable parallel processing of some segments, or it may reorder some segments. To perform parallel processing, once the Strict Order parameter is set to False, the device may fork a process for each command until the Strict Order parameter is returned to True or the command sequence ends. Then, it joins all forked processes before continuing processing of commands. To perform out-of-order processing, a similar approach is used, except the device consumes all commands after the Strict Order parameter is set to False, then it sorts these commands into its prefered order, invokes them all, then continues processing.

10. Access Control Lists

To manage permissions in the manifest, there are three models that can be used.

First, the simplest model requires that all manifests are authenticated by a single trusted key. This mode has the advantage that only a root manifest needs to be authenticated, since all of its dependencies have digests included in the root manifest.

This simplest model can be extended by adding key delegation without much increase in complexity.

A second model requires an ACL to be presented to the device, authenticated by a trusted party or stored on the device. This ACL grants access rights for specific component IDs or component ID prefixes to the listed identities or identity groups. Any identity may verify an image digest, but fetching into or fetching from a component ID requires approval from the ACL.

A third model allows a device to provide even more fine-grained controls: The ACL lists the component ID or component ID prefix that an identity may use, and also lists the commands that the identity may use in combination with that component ID.

11. Creating conditional sequences

For some use cases, it is important to provide a sequence that can fail without terminating an update. For example, a dual-image XIP MCU may require an update that can be placed at one of two offsets. This has two implications, first, the digest of each offset will be different. Second, the image fetched for each offset will have a different URI. Conditional sequences allow this to be resolved in a simple way.

The following JSON representation of a manifest demonstrates how this would be represented. It assumes that the bootloader and manifest processor take care of A/B switching and that the manifest is not aware of this distinction.

```
{
    "structure-version" : 1,
    "sequence-number" : 7,
    "components" : [
        {
            "component-identifier" : [0],
            "component-size" : [32567],
        },
    ],
    "common" : [
        "set-component-index" : 0,
        "do-sequence" : [
            "condition-component-offset" : "<offset A>",
            "set-parameters": {
                "component-digest" : "<SHA256 A>"
            }
        ],
        "do-sequence" : [
            "condition-component-offset" : "<offset B>",
            "set-parameters": {
                "component-digest" : "<SHA256 A>"
            }
        ]
    ],
    "fetch" : [
        "set-component-index" : 0,
        "do-sequence" : [
            "condition-component-offset" : "<offset A>",
            "set-parameters": {
                "uri-list" : [[0, "<uri-A>"]]
            }
        ],
        "do-sequence" : [
            "condition-component-offset" : "<offset B>",
            "set-parameters": {
                "uri-list" : [[0, "<uri-B>"]]
            }
        ],
        "fetch" : null
    ]
}
```

12. Full CDDL

In order to create a valid SUIT Manifest document the structure of the corresponding CBOR message MUST adhere to the following CDDL data definition.

```
SUIT_Outer_Wrapper = {
    suit-authentication-wrapper => bstr .cbor SUIT_Authentication_Wrapper / nil,
    suit-manifest               => bstr .cbor SUIT_Manifest,
    suit-dependency-resolution  => bstr .cbor SUIT_Command_Sequence,
    suit-payload-fetch          => bstr .cbor SUIT_Command_Sequence,
    suit-install                => bstr .cbor SUIT_Command_Sequence,
    suit-text                   => bstr .cbor SUIT_Text_Map,
    suit-coswid                 => bstr .cbor concise-software-identity
}
suit-authentication-wrapper = 1
suit-manifest = 2
suit-dependency-resolution = 7
suit-payload-fetch = 8
suit-install = 9
suit-text = 13
suit-coswid = 14

SUIT_Authentication_Wrapper = [ * (
    COSE_Mac_Tagged /
    COSE_Sign_Tagged /
    COSE_Mac0_Tagged /
    COSE_Sign1_Tagged)]


COSE_Mac_Tagged = any
COSE_Sign_Tagged = any
COSE_Mac0_Tagged = any
COSE_Sign1_Tagged = any
COSE_Encrypt_Tagged = any
COSE_Encrypt0_Tagged = any

SUIT_Digest = [
  suit-digest-algorithm-id : $suit-digest-algorithm-ids,
  suit-digest-bytes : bytes,
  ? suit-digest-parameters : any
]

; Named Information Hash Algorithm Identifiers
suit-digest-algorithm-ids /= algorithm-id-sha256
suit-digest-algorithm-ids /= algorithm-id-sha256-128
suit-digest-algorithm-ids /= algorithm-id-sha256-120
suit-digest-algorithm-ids /= algorithm-id-sha256-96
suit-digest-algorithm-ids /= algorithm-id-sha256-64
suit-digest-algorithm-ids /= algorithm-id-sha256-32
suit-digest-algorithm-ids /= algorithm-id-sha384
suit-digest-algorithm-ids /= algorithm-id-sha512
suit-digest-algorithm-ids /= algorithm-id-sha3-224
suit-digest-algorithm-ids /= algorithm-id-sha3-256
```

```
suit-digest-algorithm-ids /= algorithm-id-sha3-384
suit-digest-algorithm-ids /= algorithm-id-sha3-512

SUIT_Manifest = {
    suit-manifest-version         => 1,
    suit-manifest-sequence-number => uint,
    ? suit-dependencies           => [ + SUIT_Dependency ],
    ? suit-components             => [ + SUIT_Component ],
    ? suit-dependency-components  => [ + SUIT_Component_Reference ],
    ? suit-common                 => bstr .cbor SUIT_Command_Sequence,
    ? suit-dependency-resolution  => SUIT_Digest / bstr .cbor
SUIT_Command_Sequence,
    ? suit-payload-fetch          => SUIT_Digest / bstr .cbor
SUIT_Command_Sequence,
    ? suit-install                => SUIT_Digest / bstr .cbor
SUIT_Command_Sequence
    ? suit-validate               => bstr .cbor SUIT_Command_Sequence
    ? suit-load                   => bstr .cbor SUIT_Command_Sequence
    ? suit-run                    => bstr .cbor SUIT_Command_Sequence
    ? suit-text-info              => SUIT_Digest / bstr .cbor SUIT_Text_Map
    ? suit-coswid                 => SUIT_Digest / bstr .cbor concise-software-
identity
}

suit-manifest-version = 1
suit-manifest-sequence-number = 2
suit-dependencies = 3
suit-components = 4
suit-dependency-components = 5
suit-common = 6
suit-dependency-resolution = 7
suit-payload-fetch = 8
suit-install = 9
suit-validate = 10
suit-load = 11
suit-run = 12
suit-text-info = 13
suit-coswid = 14

concise-software-identity = any

SUIT_Dependency = {
    suit-dependency-digest => SUIT_Digest,
    suit-dependency-prefix => SUIT_Component_Identifier,
}

suit-dependency-digest = 1
suit-dependency-prefix = 2
```

```
SUIT_Component_Identifier =  [* bstr]

SUIT_Component = {
```

```
    suit-component-identifier => SUIT_Component_Identifier,
    ? suit-component-size => uint,
    ? suit-component-digest => SUIT_Digest,
}

suit-component-identifier = 1
suit-component-size = 2
suit-component-digest = 3

SUIT_Component_Reference = {
    suit-component-identifier => SUIT_Component_Identifier,
    suit-component-dependency-index => uint
}

suit-component-dependency-index = 2

SUIT_Command_Sequence = [ + { SUIT_Condition // SUIT_Directive //
SUIT_Command_Custom} ]

SUIT_Command_Custom = (nint => bstr)

SUIT_Condition //= (SUIT_Condition_Vendor_Identifier => RFC4122_UUID) ;
SUIT_Condition_Vendor_Identifier
SUIT_Condition //= (2 => RFC4122_UUID) ; SUIT_Condition_Class_Identifier
SUIT_Condition //= (3 => RFC4122_UUID) ; SUIT_Condition_Device_Identifier
SUIT_Condition //= (4 => SUIT_Digest) ; SUIT_Condition_Image_Match
SUIT_Condition //= (5 => SUIT_Digest) ; SUIT_Condition_Image_Not_Match
SUIT_Condition //= (6 => uint) ; SUIT_Condition_Use_Before
SUIT_Condition //= (7 => uint) ; SUIT_Condition_Minimum_Battery
SUIT_Condition //= (8 => int) ;  SUIT_Condition_Update_Authorised
SUIT_Condition //= (9 => SUIT_Condition_Version_Argument) ;
SUIT_Condition_Version
SUIT_Condition //= (10 => uint) ; SUIT_Condition_Component_Offset
SUIT_Condition //= (nint => bstr) ; SUIT_Condition_Custom

SUIT_Condition_Vendor_Identifier = 1
RFC4122_UUID = bstr .size 16

SUIT_Condition_Version_Argument = [
    suit-condition-version-comparison: SUIT_Condition_Version_Comparison_Types,
    suit-condition-version-comparison: SUIT_Condition_Version_Comparison_Value
]
SUIT_Condition_Version_Comparison_Types /=
SUIT_Condition_Version_Comparison_Greater
SUIT_Condition_Version_Comparison_Types /=
SUIT_Condition_Version_Comparison_Greater_Equal
SUIT_Condition_Version_Comparison_Types /=
SUIT_Condition_Version_Comparison_Equal
SUIT_Condition_Version_Comparison_Types /=
```

```
SUIT_Condition_Version_Comparison_Lesser_Equal
SUIT_Condition_Version_Comparison_Types /=
SUIT_Condition_Version_Comparison_Lesser

SUIT_Condition_Version_Comparison_Greater = 1
SUIT_Condition_Version_Comparison_Greater_Equal = 2
SUIT_Condition_Version_Comparison_Equal = 3
```

```
SUIT_Condition_Version_Comparison_Lesser_Equal = 4
SUIT_Condition_Version_Comparison_Lesser = 5

SUIT_Condition_Version_Comparison_Value = [+int]

SUIT_Directive //= (11 => uint/bool) ; SUIT_Directive_Set_Component_Index
SUIT_Directive //= (12 => uint/bool) ; SUIT_Directive_Set_Manifest_Index
SUIT_Directive //= (13 => bstr .cbor SUIT_Command_Sequence) ;
SUIT_Directive_Run_Sequence
SUIT_Directive //= (14 => bstr .cbor SUIT_Command_Sequence) ;
SUIT_Directive_Run_Sequence_Conditional
SUIT_Directive //= (15 => nil) ; SUIT_Directive_Process_Dependency
SUIT_Directive //= (16 => {+ SUIT_Parameters}) ; SUIT_Directive_Set_Parameters
SUIT_Directive //= (19 => {+ SUIT_Parameters}) ;
SUIT_Directive_Override_Parameters
SUIT_Directive //= (20 => nil/bstr) ; SUIT_Directive_Fetch
SUIT_Directive //= (21 => nil/bstr) ; SUIT_Directive_Copy
SUIT_Directive //= (22 => nil/bstr) ; SUIT_Directive_Run
SUIT_Directive //= (23 => { + SUIT_Wait_Events }) ; SUIT_Directive_Wait

SUIT_Wait_Events //= (1 => SUIT_Wait_Event_Argument_Authorisation)
SUIT_Wait_Events //= (2 => SUIT_Wait_Event_Argument_Power)
SUIT_Wait_Events //= (3 => SUIT_Wait_Event_Argument_Network)
SUIT_Wait_Events //= (4 => SUIT_Wait_Event_Argument_Other_Device_Version)
SUIT_Wait_Events //= (5 => SUIT_Wait_Event_Argument_Time)
SUIT_Wait_Events //= (6 => SUIT_Wait_Event_Argument_Time_Of_Day)
SUIT_Wait_Events //= (7 => SUIT_Wait_Event_Argument_Day_Of_Week)


SUIT_Wait_Event_Argument_Authorisation = int ; priority
SUIT_Wait_Event_Argument_Power = int ; Power Level
SUIT_Wait_Event_Argument_Network = int ; Network State
SUIT_Wait_Event_Argument_Other_Device_Version = [
    other-device: bstr,
    other-device-version: [+int]
]
SUIT_Wait_Event_Argument_Time = uint ; Timestamp
SUIT_Wait_Event_Argument_Time_Of_Day = uint ; Time of Day (seconds since
00:00:00)
SUIT_Wait_Event_Argument_Day_Of_Week = uint ; Days since Sunday

SUIT_Parameters //= (1 => bool) ; SUIT_Parameter_Strict_Order
SUIT_Parameters //= (2 => bool) ; SUIT_Parameter_Coerce_Condition_Failure
SUIT_Parameters //= (3 => bstr) ; SUIT_Parameter_Vendor_ID
SUIT_Parameters //= (4 => bstr) ; SUIT_Parameter_Class_ID
SUIT_Parameters //= (5 => bstr) ; SUIT_Parameter_Device_ID
SUIT_Parameters //= (6 => bstr .cbor SUIT_URI_List) ; SUIT_Parameter_URI_List
SUIT_Parameters //= (7 => bstr .cbor SUIT_Encryption_Info) ;
```

```
SUIT_Parameter_Encryption_Info
SUIT_Parameters //= (8 => bstr .cbor SUIT_Compression_Info) ;
SUIT_Parameter_Compression_Info
SUIT_Parameters //= (9 => bstr .cbor SUIT_Unpack_Info) ;
SUIT_Parameter_Unpack_Info
SUIT_Parameters //= (10 => bstr .cbor SUIT_Component_Identifier) ;
SUIT_Parameter_Source_Component
SUIT_Parameters //= (11 => bstr .cbor SUIT_Digest) ; SUIT_Parameter_Image_Digest
```

```
SUIT_Parameters //= (12 => uint) ; SUIT_Parameter_Image_Size
SUIT_Parameters //= (nint => int/bool/bstr) ; SUIT_Parameter_Custom

SUIT_URI_List = [ + [priority: int, uri: tstr] ]

SUIT_Encryption_Info = COSE_Encrypt_Tagged/COSE_Encrypt0_Tagged
SUIT_Compression_Info = {
    suit-compression-algorithm => SUIT_Compression_Algorithms
    ? suit-compression-parameters => bstr
}
suit-compression-algorithm = 1
suit-compression-parameters = 2

SUIT_Compression_Algorithms /= SUIT_Compression_Algorithm_gzip
SUIT_Compression_Algorithms /= SUIT_Compression_Algorithm_bzip2
SUIT_Compression_Algorithms /= SUIT_Compression_Algorithm_lz4
SUIT_Compression_Algorithms /= SUIT_Compression_Algorithm_lzma

SUIT_Compression_Algorithm_gzip = 1
SUIT_Compression_Algorithm_bzip2 = 2
SUIT_Compression_Algorithm_deflate = 3
SUIT_Compression_Algorithm_lz4 = 4
SUIT_Compression_Algorithm_lzma = 7

SUIT_Unpack_Info = {
    suit-unpack-algorithm => SUIT_Unpack_Algorithms
    ? suit-unpack-parameters => bstr
}
suit-unpack-algorithm  = 1
suit-unpack-parameters = 2

SUIT_Unpack_Algorithms /= SUIT_Unpack_Algorithm_Delta
SUIT_Unpack_Algorithms /= SUIT_Unpack_Algorithm_Hex
SUIT_Unpack_Algorithms /= SUIT_Unpack_Algorithm_Elf

SUIT_Unpack_Algorithm_Delta = 1
SUIT_Unpack_Algorithm_Hex = 2
SUIT_Unpack_Algorithm_Elf = 3

SUIT_Text_Map = {int => tstr}
```

13. Examples

The following examples demonstrate a small subset of the functionality of the manifest. However, despite this, even a simple manifest processor can execute most of these manifests.

None of these examples include authentication. This is provided via RFC 8152 [RFC8152], and is omitted for clarity.

13.1. Example 0:

Secure boot only.

The following JSON shows the intended behaviour of the manifest.

```
{
    "structure-version": 1,
    "sequence-number": 1,
    "components": [
        {
            "id": ["Flash",78848],
            "digest": "00112233445566778899aabbccddeeff"
                      "0123456789abcdeffedcba9876543210",
            "size": 34768
        }
    ],
    "run-image": [
        {"directive-set-component": 0},
        {"condition-image": null},
        {"directive-run": null}
    ]
}
```

Converted into the SUIT manifest, this produces:

```
{
    / auth object / 1 : None
    / manifest / 2 : h'a4010102010481a3018245466c61736843003401021987'
        h'd00382015820001122334455667788 99aabbccddeeff0123456789abcdef'
        h'fedcba98765432100c4a83a10b00a104f6a116f6' \
    {
        / structure-version / 1 : 1
        / sequence-number / 2 : 1
        / components / 4 : [
            {
                / component-identifier / 1 : [h'466c617368', h'003401'],
                / component-size / 3 : 34768
                / component-digest / 2 : [
                    / sha-256 / 1,
                    h'00112233445566778899aabbccddeeff0123456789abcdef'
                    h'fedcba9876543210'],
            }
        ],
        / run-image / 12 : [
            {/ set-component-index / 11 : 0}
            {/ condition-image / 4 : None}
            {/ run / 22 : None}
        ],
    }
}
```

Total size of outer wrapper without COSE authentication object: 79

Outer:

```
a201f6025849a4010102010481a3018245466c617368430034010219 87d00382015820
00112233445566778899aabbccddeeff0123456789abcdeffedcba98765432100c4a83
a10b00a104f6a116f6
```

13.2. Example 1:

Simultaneous download and installation of payload.

The following JSON shows the intended behaviour of the manifest.

```
{
    "structure-version": 1,
    "sequence-number": 2,
    "components": [
        {
            "id": ["Flash",78848],
            "digest": "00112233445566778899aabbccddeeff"
                      "0123456789abcdeffedcba9876543210",
            "size": 34768
        }
    ],
    "apply-image": [
        {"directive-set-component": 0},
        {"directive-set-var": {
            "uris": [[ 0, "http://example.com/file.bin"]]
        }},
        {"directive-fetch": null}
    ]
}
```

Converted into the SUIT manifest, this produces:

```
{
    / auth object / 1 : None
    / manifest / 2 : h'a4010102020481a3018245466c61736843003401021987'
        h'd00382015820001122334455667788 99aabbccddeeff0123456789abcdef'
        h'fedcba987654321009582d83a10b00a110a1065820818200781b68747470'
        h'3a2f2f6578616d706c652e636f6d2f66696c652e62696ea114f6' \
    {
        / structure-version / 1 : 1
        / sequence-number / 2 : 2
        / components / 4 : [
            {
                / component-identifier / 1 : [h'466c617368', h'003401'],
                / component-size / 3 : 34768
                / component-digest / 2 : [
                    / sha-256 / 1,
                    h'00112233445566778899aabbccddeeff'
                    h'0123456789abcdeffedcba9876543210'
                ],
            }
        ],
        / apply-image / 9 : [
            {/ set-component-index / 11 : 0}
            {/ set-vars / 16 : {
                / uris / 6 : h'818200781b687474703a2f2f6578616d706c'
                    h'652e636f6d2f66696c652e62696e' /
                        [[0, 'http://example.com/file.bin']] /
            }},
            {/ fetch / 20 : None}
        ],
    }
}
```

Total size of outer wrapper without COSE authentication object: 115

Outer:

```
a201f602586da4010102020481a3018245466c617368430034010219 87d00382015820
00112233445566778899aabbccddeeff0123456789abcdeffedcba987654321009582d
83a10b00a110a1065820818200781b687474703a2f2f6578616d706c652e636f6d2f66
696c652e62696ea114f6
```

13.3. Example 2:

Compatibility test, simultaneous download and installation, and secure boot.

The following JSON shows the intended behaviour of the manifest.

```
{
    "structure-version": 1,
    "sequence-number": 3,
    "components": [
        {
            "id": [
                "Flash",
                78848
            ],
            "digest": "00112233445566778899aabbccddeeff"
                "0123456789abcdeffedcba9876543210",
            "size": 34768
        }
    ],
    "common": [
        {"condition-vendor-id": "fa6b4a53-d5ad-5fdf-be9d-e663e4d41ffe"},
        {"condition-class-id": "1492af14-2569-5e48-bf42-9b2d51f2ab45"}
    ],
    "apply-image": [
        {"directive-set-component": 0},
        {"directive-set-var": {
            "uris": [[ 0, "http://example.com/file.bin" ]]
        }},
        {"directive-fetch": null}
    ],
    "run-image": [
        {"directive-set-component": 0},
        {"condition-image": null},
        {"directive-run": null}
    ]
}
```

Converted into the SUIT manifest, this produces:

```
{
    / auth object / 1 : None
    / manifest / 2 : h'a6010102030481a3018245466c61736843003401021987'
        h'd0038201582000112233445566778899aabbccddeeff0123456789abcdef'
        h'fedcba98765432100658278 2a10150fa6b4a53d5ad5fdfbe9de663e4d41f'
        h'fea102501492af1425695e48bf429b2d51f2ab4509582d83a10b00a110a1'
        h'065820818200781b687474703a2f2f6578616d706c652e636f6d2f66696c'
        h'652e62696ea114f60c4a83a10b00a104f6a116f6' \
    {
        / structure-version / 1 : 1
        / sequence-number / 2 : 3
        / components / 4 : [
            {
                / component-identifier / 1 : [h'466c617368', h'003401'],
                / component-size / 3 : 34768
                / component-digest / 2 : [
                    / sha-256 / 1,
                    h'00112233445566778899aabbccddeeff'
                    h'0123456789abcdeffedcba9876543210'
                ],
            }
        ],
        / common / 6 : [
            {/ vendor-id / 1 : h'fa6b4a53d5ad5fdfbe9de663e4d41ffe' \
                fa6b4a53-d5ad-5fdf-be9d-e663e4d41ffe},
            {/ class-id / 2 : h'1492af1425695e48bf429b2d51f2ab45' \
                1492af14-2569-5e48-bf42-9b2d51f2ab45}
        ],
        / apply-image / 9 : [
            {/ set-component-index / 11 : 0}
            {/ set-vars / 16 : {
                / uris / 6 : h'818200781b687474703a2f2f6578616d706c65'
                    h'2e636f6d2f66696c652e62696e' /
                    [[0, 'http://example.com/file.bin']] /
            }},
            {/ fetch / 20 : None}
        ],
        / run-image / 12 : [
            {/ set-component-index / 11 : 0}
            {/ condition-image / 4 : None}
            {/ run / 22 : None}
        ],
    }
}
```

Total size of outer wrapper without COSE authentication object: 169

Outer:

```
a201f60258a3a6010102030481a3018245466c617368430034010219 87d00382015820
00112233445566778899aabbccddeeff0123456789abcdeffedcba 9876543210065827
82a10150fa6b4a53d5ad5fdfbe9de663e4d41ffea102501492af1425695e48bf429b2d
51f2ab4509582d83a10b00a110a1065820818200781b687474703a2f2f6578616d706c
652e636f6d2f66696c652e62696ea114f60c4a83a10b00a104f6a116f6
```

13.4. Example 3:

Compatibility test, simultaneous download and installation, load from external storage, and secure boot.

The following JSON shows the intended behaviour of the manifest.

```
{
    "structure-version": 1,
    "sequence-number": 4,
    "components": [
        {
            "id": ["Flash",78848],
            "digest": "00112233445566778899aabbccddeeff"
                      "0123456789abcdeffedcba9876543210",
            "size": 34768
        },
        {
            "id": ["RAM",1024],
            "digest": "00112233445566778899aabbccddeeff"
                      "0123456789abcdeffedcba9876543210",
            "size": 34768
        }
    ],
    "common": [
        {"condition-vendor-id": "fa6b4a53-d5ad-5fdf-be9d-e663e4d41ffe"},
        {"condition-class-id": "1492af14-2569-5e48-bf42-9b2d51f2ab45"}
    ],
    "apply-image": [
        {"directive-set-component": 0},
        {"directive-set-var": {
            "uris": [[0, "http://example.com/file.bin"]]
        }},
        {"directive-fetch": null}
    ],
    "run-image": [
        {"directive-set-component": 0},
        {"condition-image": null},
        {"directive-set-component": 1},
        {"directive-set-var": {
            "source-index": 0
        }},
        {"directive-fetch": null},
        {"condition-image": null},
        {"directive-run": null}
    ]
}
```

Converted into the SUIT manifest, this produces:

```
{
    / auth object / 1 : None
    / manifest / 2 : h'a6010102040482a3018245466c61736843003401021987'
        h'd00382015820001122334455667788 99aabbccddeeff0123456789abcdef'
        h'fedcba9876543210a301824352414d420004021987d00382015820001122'
```

```
        h'33445566778899aabbccddeeff0123456789abcdeffedcba987654321006'
        h'582782a10150fa6b4a53d5ad5fdfbe9de663e4d41ffea102501492af1425'
        h'695e48bf429b2d51f2ab4509582d83a10b00a110a1065820818200781b68'
        h'7474703a2f2f6578616d706c652e636f6d2f66696c652e62696ea114f60c'
        h'581887a10b00a104f6a10b01a110a10a00a114f6a104f6a116f6' \
    {
        / structure-version / 1 : 1
        / sequence-number / 2 : 4
        / components / 4 : [
            {
                / component-identifier / 1 : [h'466c617368', h'003401'],
                / component-size / 3 : 34768
                / component-digest / 2 : [
                    / sha-256 / 1,
                    h'00112233445566778899aabbccddeeff'
                    h'0123456789abcdeffedcba9876543210'
                ],
            },
            {
                / component-identifier / 1 : [h'52414d', h'0004'],
                / component-size / 3 : 34768
                / component-digest / 2 : [
                    / sha-256 / 1,
                    h'00112233445566778899aabbccddeeff'
                    h'0123456789abcdeffedcba9876543210'
                ],
            }
        ],
        / common / 6 : [
            {/ vendor-id / 1 : h'fa6b4a53d5ad5fdfbe9de663e4d41ffe' \
                fa6b4a53-d5ad-5fdf-be9d-e663e4d41ffe}
            {/ class-id / 2 : h'1492af1425695e48bf429b2d51f2ab45' \
                1492af14-2569-5e48-bf42-9b2d51f2ab45}
        ],
        / apply-image / 9 : [
            {/ set-component-index / 11 : 0}
            {/ set-vars / 16 : {
                / uris / 6 : h'818200781b687474703a2f2f6578616d706c65'
                             h'2e636f6d2f66696c652e62696e' /
                    [[0, 'http://example.com/file.bin']] /
            }},
            {/ fetch / 20 : None}
        ],
        / run-image / 12 : [
            {/ set-component-index / 11 : 0}
            {/ condition-image / 4 : None}
            {/ set-component-index / 11 : 1}
            {/ set-vars / 16 : {
```

```
                / source-component / 10 : 0
            }},
            {/ fetch / 20 : None}
            {/ condition-image / 4 : None}
            {/ run / 22 : None}
        ],
    }
}
```

Total size of outer wrapper without COSE authentication object: 235

Outer:

```
a201f60258e5a6010102040482a3018245466c617368430034010219870d00382015820
00112233445566778899aabbccddeeff0123456789abcdeffedcba9876543210a30182
4352414d420004021987d00382015820001122334455667788 99aabbccddeeff012345
6789abcdeffedcba98765432100658278 2a10150fa6b4a53d5ad5fdfbe9de663e4d41f
fea102501492af1425695e48bf429b2d51f2ab4509582d83a10b00a110a10658208182
00781b687474703a2f2f6578616d706c652e636f6d2f66696c652e62696ea114f60c58
1887a10b00a104f6a10b01a110a10a00a114f6a104f6a116f6
```

13.5. Example 4:

Compatibility test, simultaneous download and installation, load and decompress from external storage, and secure boot.

The following JSON shows the intended behaviour of the manifest.

```
{
    "structure-version": 1,
    "sequence-number": 5,
    "components": [
        {
            "id": ["Flash",78848],
            "digest": "00112233445566778899aabbccddeeff"
                      "0123456789abcdeffedcba9876543210",
            "size": 34768
        },
        {
            "id": ["RAM",1024],
            "digest": "0123456789abcdeffedcba9876543210"
                      "00112233445566778899aabbccddeeff",
            "size": 34768
        }
    ],
    "common": [
        {"condition-vendor-id": "fa6b4a53-d5ad-5fdf-be9d-e663e4d41ffe"},
        {"condition-class-id": "1492af14-2569-5e48-bf42-9b2d51f2ab45"}
    ],
    "apply-image": [
        {"directive-set-component": 0},
        {"directive-set-var": {
            "uris": [[ 0, "http://example.com/file.bin" ]]
        }},
        {"directive-fetch": null}
    ],
    "load-image": [
        {"directive-set-component": 0},
        {"condition-image": null},
        {"directive-set-component": 1},
        {"directive-set-var": {
            "source-index": 0,
            "compression-info": {
                "algorithm": "gzip"
            }
        }},
        {"directive-copy": null}
    ],
    "run-image": [
        {"condition-image": null},
        {"directive-run": null}
    ]
}
```

Converted into the SUIT manifest, this produces:

```
{
    / auth object / 1 : None
    / manifest / 2 : h'a7010102050482a3018245466c61736843003401021987'
        h'd00382015820001122334455667788 99aabbccddeeff0123456789abcdef'
        h'fedcba9876543210a301824352414d420004021987d00382015820012345'
        h'6789abcdeffedcba98765432100011223344556677889 9aabbccddeeff06'
        h'582782a10150fa6b4a53d5ad5fdfbe9de663e4d41ffea102501492af1425'
        h'695e48bf429b2d51f2ab4509582d83a10b00a110a1065820818200781b68'
        h'7474703a2f2f6578616d706c652e636f6d2f66696c652e62696ea114f60b'
        h'5585a10b00a104f6a10b01a110a20841f60a00a115f60c4782a104f6a116'
        h'f6' \
    {
        / structure-version / 1 : 1
        / sequence-number / 2 : 5
        / components / 4 : [
            {
                / component-identifier / 1 : [h'466c617368', h'003401'],
                / component-size / 3 : 34768
                / component-digest / 2 : [
                    / sha-256 / 1,
                    h'00112233445566778899aabbccddeeff'
                    h'0123456789abcdeffedcba9876543210'
                ],
            },
            {
                / component-identifier / 1 : [h'52414d', h'0004'],
                / component-size / 3 : 34768
                / component-digest / 2 : [
                    / sha-256 / 1,
                    h'0123456789abcdeffedcba9876543210'
                    h'00112233445566778899aabbccddeeff'
                ],
            }
        ],
        / common / 6 : [
            {/ vendor-id / 1 : h'fa6b4a53d5ad5fdfbe9de663e4d41ffe' \
                fa6b4a53-d5ad-5fdf-be9d-e663e4d41ffe},
            {/ class-id / 2 : h'1492af1425695e48bf429b2d51f2ab45' \
                1492af14-2569-5e48-bf42-9b2d51f2ab45}
        ],
        / apply-image / 9 : [
            {/ set-component-index / 11 : 0}
            {/ set-vars / 16 : {
                / uris / 6 : h'818200781b687474703a2f2f6578616d706c65'
                             h'2e636f6d2f66696c652e62696e' /
                    [[0, 'http://example.com/file.bin']] /
            }},
            {/ fetch / 20 : None}
```

```
        ],
        / load-image / 11 : [
            {/ set-component-index / 11 : 0}
            {/ condition-image / 4 : None}
            {/ set-component-index / 11 : 1}
            {/ set-vars / 16 : {
                / unknown / 8 : b'\xf6'
                / source-component / 10 : 0
            }},
            {/ copy / 21 : None}
        ],
        / run-image / 12 : [
            {/ condition-image / 4 : None}
            {/ run / 22 : None}
        ],
    }
}
```

Total size of outer wrapper without COSE authentication object: 240

Outer:

```
a201f60258eaa7010102050482a3018245466c617368430034010219 87d00382015820
00112233445566778899aabbccddeeff0123456789abcdeffedcba9876543210a30182
4352414d420004021987d00382015820 0123456789abcdeffedcba9876543210001122
33445566778899aabbccddeeff06582782a10150fa6b4a53d5ad5fdfbe9de663e4d41f
fea102501492af1425695e48bf429b2d51f2ab4509582d83a10b00a110a10658208182
00781b687474703a2f2f6578616d706c652e636f6d2f66696c652e62696ea114f60b55
85a10b00a104f6a10b01a110a20841f60a00a115f60c4782a104f6a116f6
```

13.6. Example 5:

Compatibility test, download, installation, and secure boot.

The following JSON shows the intended behaviour of the manifest.

```
{
    "structure-version": 1,
    "sequence-number": 6,
    "components": [
        {
            "id": [ "ext-Flash", 78848 ],
            "digest": "00112233445566778899aabbccddeeff"
                      "0123456789abcdeffedcba9876543210",
            "size": 34768
        },
        {
            "id": ["Flash",1024],
            "digest": "0123456789abcdeffedcba9876543210"
                      "00112233445566778899aabbccddeeff",
            "size": 34768
        }
    ],
    "common": [
        {"condition-vendor-id": "fa6b4a53-d5ad-5fdf-be9d-e663e4d41ffe"},
        {"condition-class-id": "1492af14-2569-5e48-bf42-9b2d51f2ab45"}
    ],
    "apply-image": [
        {"directive-set-component": 0},
        {"directive-set-var": {
            "uris": [[0, "http://example.com/file.bin"]]
        }},
        {"directive-fetch": null}
    ],
    "load-image": [
        {"directive-run-conditional": [
            {"directive-set-component": 1},
            {"condition-not-image": null},
            {"directive-set-component": 0},
            {"condition-image": null},
            {"directive-set-component": 1},
            {"directive-set-var": {
                "source-index": 0
            }},
            {"directive-fetch": null}
        ]}
    ],
    "run-image": [
        {"directive-set-component": 1},
        {"condition-image": null},
        {"directive-run": null}
    ]
}
```

Converted into the SUIT manifest, this produces:

```
{
    / auth object / 1 : None
    / manifest / 2 : h'a7010102060482a30182496578742d466c617368430034'
        h'01021987d00382015820001122334455667788 99aabbccddeeff01234567'
        h'89abcdeffedcba9876543210a3018245466c617368420004021987d00382'
        h'0158200123456789abcdeffedcba9876543210001122334455667788 99aa'
        h'bbccddeeff06582782a10150fa6b4a53d5ad5fdfbe9de663e4d41ffea102'
        h'501492af1425695e48bf429b2d51f2ab4509582d83a10b00a110a1065820'
        h'818200781b687474703a2f2f6578616d706c652e636f6d2f66696c652e62'
        h'696ea114f60b581d81a10e581887a10b01a105f6a10b00a104f6a10b01a1'
        h'10a10a00a114f60c4a83a10b01a104f6a116f6' \
    {
        / structure-version / 1 : 1
        / sequence-number / 2 : 6
        / components / 4 : [
            {
                / component-identifier / 1 : [
                    h'6578742d466c617368',
                    h'003401'
                ],
                / component-size / 3 : 34768
                / component-digest / 2 : [
                    / sha-256 / 1,
                    h'00112233445566778899aabbccddeeff'
                    h'0123456789abcdeffedcba9876543210'
                ],
            }
            {
                / component-identifier / 1 : [h'466c617368', h'0004'],
                / component-size / 3 : 34768
                / component-digest / 2 : [
                    / sha-256 / 1,
                    h'0123456789abcdeffedcba9876543210'
                    h'00112233445566778899aabbccddeeff'
                ],
            }
        ],
        / common / 6 : [
            {/ vendor-id / 1 : h'fa6b4a53d5ad5fdfbe9de663e4d41ffe' \
                fa6b4a53-d5ad-5fdf-be9d-e663e4d41ffe}
            {/ class-id / 2 : h'1492af1425695e48bf429b2d51f2ab45' \
                1492af14-2569-5e48-bf42-9b2d51f2ab45}
        ],
        / apply-image / 9 : [
            {/ set-component-index / 11 : 0}
            {/ set-vars / 16 : {
```

```
                / uris / 6 : h'818200781b687474703a2f2f6578616d706c65'
                             h'2e636f6d2f66696c652e62696e' /
                    [[0, 'http://example.com/file.bin']] /
            }},
            {/ fetch / 20 : None}
        ],
        / load-image / 11 : [
            / conditional-sequence / 14 : [
                {/ set-component-index / 11 : 1}
                {/ condition-not-image / 5 : None}
                {/ set-component-index / 11 : 0}
                {/ condition-image / 4 : None}
                {/ set-component-index / 11 : 1}
                {/ set-vars / 16 : {
                    / source-component / 10 : 0
                }},
                {/ fetch / 20 : None}
            ],
        ],
        / run-image / 12 : [
            {/ set-component-index / 11 : 1}
            {/ condition-image / 4 : None}
            {/ run / 22 : None}
        ],
    }
}
```

Total size of outer wrapper without COSE authentication object: 258

Outer:

```
a201f60258fca7010102060482a30182496578742d466c6173684300340102198 7d003
8201582000112233445566778899aabbccddeeff0123456789abcdeffedcba98765432
10a3018245466c6173684200402198 7d00382015820012345 6789abcdeffedcba9876
5432100011223344556677 8899aabbccddeeff06582782a10150fa6b4a53d5ad5fdfbe
9de663e4d41ffea102501492af1425695e48bf429b2d51f2ab4509582d83a10b00a110
a106582081820078 1b687474703a2f2f6578616d706c652e636f6d2f66696c652e6269
6ea114f60b581d81a10e581887a10b01a105f6a10b00a104f6a10b01a110a10a00a114
f60c4a83a10b01a104f6a116f6
```

13.7. Example 6:

Compatibility test, 2 images, simultaneous download and installation, and secure boot.

The following JSON shows the intended behaviour of the manifest.

```
{
    "structure-version": 1,
    "sequence-number": 7,
    "components": [
        {
            "id": ["Flash",78848],
            "digest": "00112233445566778899aabbccddeeff"
                      "0123456789abcdeffedcba9876543210",
            "size": 34768
        },
        {
            "id": ["Flash",132096],
            "digest": "0123456789abcdeffedcba9876543210"
                      "00112233445566778899aabbccddeeff",
            "size": 76834
        }
    ],
    "common": [
        {"condition-vendor-id": "fa6b4a53-d5ad-5fdf-be9d-e663e4d41ffe"},
        {"condition-class-id": "1492af14-2569-5e48-bf42-9b2d51f2ab45"}
    ],
    "apply-image": [
        {"directive-set-component": 0},
        {"directive-set-var": {
            "uris": [[ 0, "http://example.com/file1.bin" ]]
        }},
        {"directive-set-component": 1},
        {"directive-set-var": {
            "uris": [[ 0, "http://example.com/file2.bin" ]]
        }},
        {"directive-set-component": true},
        {"directive-fetch": null}
    ],
    "run-image": [
        {"directive-set-component": true},
        {"condition-image": null},
        {"directive-set-component": 0},
        {"directive-run": null}
    ]
}
```

Converted into the SUIT manifest, this produces:

```
{
    / auth object / 1 : None
    / manifest / 2 : h'a6010102070482a3018245466c61736843003401021987'
        h'd00382015820001122334455667788 99aabbccddeeff0123456789abcdef'
        h'fedcba9876543210a3018245466c61736843000402021a00012c22038201'
```

```
        h'58200123456789abcdeffedcba9876543210001122334455667788 99aabb'
        h'ccddeeff06582782a10150fa6b4a53d5ad5fdfbe9de663e4d41ffea10250'
        h'1492af1425695e48bf429b2d51f2ab4509585b86a10b00a110a106582181'
        h'8200781c687474703a2f2f6578616d706c652e636f6d2f66696c65312e62'
        h'696ea10b01a110a1065821818200781c687474703a2f2f6578616d706c65'
        h'2e636f6d2f66696c65322e62696ea10bf5a114f60c4d84a10bf5a104f6a1'
        h'0b00a116f6' \
    {
        / structure-version / 1 : 1
        / sequence-number / 2 : 7
        / components / 4 : [
            {
                / component-identifier / 1 : [h'466c617368', h'003401'],
                / component-size / 3 : 34768
                / component-digest / 2 : [
                    / sha-256 / 1,
                    h'00112233445566778899aabbccddeeff'
                    h'0123456789abcdeffedcba9876543210'
                ],
            }
            {
                / component-identifier / 1 : [h'466c617368', h'000402'],
                / component-size / 3 : 76834
                / component-digest / 2 : [
                    / sha-256 / 1,
                    h'0123456789abcdeffedcba9876543210'
                    h'00112233445566778899aabbccddeeff'
                ],
            }
        ],
        / common / 6 : [
            {/ vendor-id / 1 : h'fa6b4a53d5ad5fdfbe9de663e4d41ffe' \
                fa6b4a53-d5ad-5fdf-be9d-e663e4d41ffe}
            {/ class-id / 2 : h'1492af1425695e48bf429b2d51f2ab45' \
                1492af14-2569-5e48-bf42-9b2d51f2ab45}
        ],
        / apply-image / 9 : [
            {/ set-component-index / 11 : 0}
            {/ set-vars / 16 : {
                / uris / 6 : h'818200781c687474703a2f2f6578616d706c'
                             h'652e636f6d2f66696c65312e62696e' /
                                [[0, 'http://example.com/file1.bin']] /
            }},
            {/ set-component-index / 11 : 1}
            {/ set-vars / 16 : {
                / uris / 6 : h'818200781c687474703a2f2f6578616d706c
                             h'652e636f6d2f66696c65322e62696e' /
                                [[0, 'http://example.com/file2.bin']] /
```

```
            }},
            {/ set-component-index / 11 : True}
            {/ fetch / 20 : None}
        ],
        / run-image / 12 : [
            {/ set-component-index / 11 : True}
            {/ condition-image / 4 : None}
            {/ set-component-index / 11 : 0}
            {/ run / 22 : None}
        ],
    }
}
```

Total size of outer wrapper without COSE authentication object: 275

Outer:

```
a201f60259010ca6010102070482a3018245466c617368430034010219 87d003820158
2000112233445566778899aabbccddeeff0123456789abcdeffedcba9876543210a301
8245466c61736843000402021a00012c22038201582001234567 89abcdeffedcba9876
5432100011223344556677 8899aabbccddeeff06582782a10150fa6b4a53d5ad5fdfbe
9de663e4d41ffea102501492af1425695e48bf429b2d51f2ab4509585b86a10b00a110
a1065821818200781c687474703a2f2f6578616d706c652e636f6d2f66696c65312e62
696ea10b01a110a1065821818200781c687474703a2f2f6578616d706c652e636f6d2f
66696c65322e62696ea10bf5a114f60c4d84a10bf5a104f6a10b00a116f6
```

14. IANA Considerations

Several registries will be required for:

- standard Commands
- standard Parameters
- standard Algorithm identifiers
- standard text values

15. Security Considerations

This document is about a manifest format describing and protecting firmware images and as such it is part of a larger solution for offering a standardized way of delivering firmware updates to IoT devices. A more detailed discussion about security can be found in the architecture document [Architecture] and in [Information].

16. Mailing List Information

The discussion list for this document is located at the e-mail address suit@ietf.org [1]. Information on the group and information on how to subscribe to the list is at https://www1.ietf.org/mailman/listinfo/suit [2]

Archives of the list can be found at: https://www.ietf.org/mail-archive/web/suit/current/index.html [3]

17. Acknowledgements

We would like to thank the following persons for their support in designing this mechanism:

- Milosch Meriac
- Geraint Luff
- Dan Ros
- John-Paul Stanford
- Hugo Vincent
- Carsten Bormann
- Oeyvind Roenningstad
- Frank Audun Kvamtroe
- Krzysztof Chruściński
- Andrzej Puzdrowski
- Michael Richardson
- David Brown
- Emmanuel Baccelli

18. References

18.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005, <https://www.rfc-editor.org/info/rfc4122>.

[RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <https://www.rfc-editor.org/info/rfc8152>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

18.2. Informative References

[Architecture]
Moran, B., "A Firmware Update Architecture for Internet of Things Devices", January 2019, <https://tools.ietf.org/html/draft-ietf-suit-architecture-02>.

[Behaviour]
Moran, B., "An Information Model for Behavioural Description of Firmware Update and Related Operations", March 2019, <https://datatracker.ietf.org/doc/draft-moran-suit-behavioural-manifest/>.

[Information]
Moran, B., "Firmware Updates for Internet of Things Devices - An Information Model for Manifests", January 2019, <https://tools.ietf.org/html/draft-ietf-suit-information-model-02>.

[RFC6920] Farrell, S., Kutscher, D., Dannewitz, C., Ohlman, B., Keranen, A., and P. Hallam-Baker, "Naming Things with Hashes", RFC 6920, DOI 10.17487/RFC6920, April 2013, <https://www.rfc-editor.org/info/rfc6920>.

18.3. URIs

[1] mailto:suit@ietf.org

[2] https://www1.ietf.org/mailman/listinfo/suit

[3] https://www.ietf.org/mail-archive/web/suit/current/index.html

Authors' Addresses

Brendan Moran
Arm Limited

EMail: Brendan.Moran@arm.com

Hannes Tschofenig
Arm Limited

EMail: hannes.tschofenig@arm.com

Henk Birkholz
Fraunhofer SIT

EMail: henk.birkholz@sit.fraunhofer.de