SUIT
Internet-Draft
Intended status: Standards Track
Expires: April 25, 2019

B. Moran
H. Tschofenig
Arm Limited

October 22, 2018

A CBOR-based Firmware Manifest Serialisation Format
draft-moran-suit-manifest-03

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 25, 2019.

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] and with the information model specification [Information], which motivates various elements in the manifest. In Section 6 we describe the main building blocks of the manifest and Section 7 contains the description of the CBOR of the manifest. Examples are found in Section 9.

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 RFC 2119 [RFC2119].

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

- Payload: A piece of information, typically Firmware, to be delivered.

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

- Processor: A component that transforms one or more Resources into another resource or into 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.

3. COSE Digest Container

RFC 8152 [RFC8152] provides containers for signature, MAC, and encryption, but no basic digest container. The container needed for a digest is identical to a COSE_Mac0 structure, so no new container is defined.

```
COSE_Digest_Tagged = #6.19(COSE_Digest)
COSE_Digest = COSE_Mac0
```

N.B. The value 19 is a placeholder and needs to be registered.

3.1. Computing and Verifying a Digest

In order to get a consistent encoding of the data to be digested, the Digest_structure is used to have a canonical form. The Digest_structure is a CBOR array. The fields of the Digest_structure in order are:

1. A text string that identifies the structure that is being encoded. This string is "Digest".

2. The protected attributes from the COSE_Digest structure. If there are no protected attributes, a zero-length bstr is used.

3. The protected attributes from the application encoded as a bstr type. If this field is not supplied, it defaults to a zero-length binary string. (See RFC 8152 [RFC8152], Section 4.3 for application UUIDance on constructing this field.)

4. The payload to be digested encoded in a bstr type. The payload is placed here independent of how it is transported.

The CDDL fragment that corresponds to the above text is:

```
Digest_structure = [
    context : "Digest",
    protected : empty_or_serialized_map,
    external_aad : bstr,
    payload : bstr
]
```

The steps to compute a Digest are:

1. Create a Digest_structure and populate it with the appropriate fields.

2. Create the value ToBeDigested by encoding the Digest_structure to a byte stream, using the encoding described in RFC 8152 [RFC8152], Section 14.

3. Call the Digest creation algorithm passing in alg (the algorithm to Digest with), and ToBeDigested (the value to compute the digest on).

4. Place the resulting Digest in the 'tag' field of the COSE_Digest structure.

The steps to verify a Digest are:

1. Create a Digest_structure object and populate it with the appropriate fields.

2. Create the value ToBeDigested by encoding the Digest_structure to a byte stream, using the encoding described in RFC 8152 [RFC8152], Section 14.

3. Call the digest creation algorithm passing in alg (the algorithm to digest with), and ToBeDigested (the value to compute the Digest on).

4. Compare the digest value to the 'tag' field of the COSE_Digest structure.

Name	Value
SHA-224	40
SHA-256	41
SHA-384	42
SHA-512	43
SHA3-224	44
SHA3-256	45
SHA3-384	46
SHA3-512	47

N.B. Values are provisional, pending review.

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 authority. 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 a manifest is as follows:

1. Verify the signature of the manifest

2. Verify the applicability of the manifest (verify PreConditions)

3. Verify that all installation processors are available

4. Verify that all dependencies are met

5. Run PreInstalation Directives

6. Load the descriptor for the next payload to be installed

7. Load installation descriptor for the next payload to be installed

8. Install the payload

9. While there are more payloads to install, go to 5.

10. Validate PostInstallation Conditions

11. Run PostInstallation Directives

When multiple manifests are used for an update, the pull parser is not possible to orchestrate in the same manner.

6. The SUIT Manifest

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.

6.1. 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 COSE_Digest of the bstr in the manifest so that they can still be authenticated. The COSE_Digest typically consumes 10 bytes more than the size of the raw digest, therefore elements smaller than (Digest Bits)/8 + 10 SHOULD never be severable. Elements larger than (Digest Bits)/8 + 10 MAY be severable, while elements that are much larger than (Digest Bits)/8 + 10 SHOULD be severable.

6.2. 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 lowerCamelCase and CDDL structures are UpperCamelCase so that these names can be directly transcribed into most common programming languages, whereas hyphens require translation and CDDL prefers hyphens to underscores.

6.3. 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 processing and those that do not.

Binary, PEM certificate, and CBOR Web Token do not require installation-time processing. Hex, s-record, elf, and serialised configuration require installation-time processing. Binary diff is a special case that can be handled either in a pre-processing step or in an installation-time step, depending on the architectural requirements of the application.

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.

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 the payloadInstaller and payloadInfo sections, respectively.

Where a specialised installer 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.

7. 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. Pre-installation Information / Reference
        3. Payloads
        4. Installation Information / Reference

7.1. Outer wrapper

This container is just a holder for the other pieces of the manifest. The CDDL that describes the wrapper is below:

```
OuterWrapper = {
    authenticationWrapper: AuthenticationWrapper,
    manifest:              bstr .cbor Manifest,
    ? preInstallExt:       bstr .cbor PreInstallationInfo,
    ? installExt:          bstr .cbor InstallationInfo,
    ? postInstallExt:      bstr .cbor PostInstallationInfo,
    ? textInfoExt:         bstr .cbor Text,
    ? coswidExt:           bstr .cbor concise-software-identity
}
authenticationWrapper = 1
manifest = 2
preInstallExt = 3
installExt = 4
postInstallExt = 5
textExt = 6
coswidExt = 7

AuthenticationWrapper = COSE_Mac_Tagged / COSE_Sign_Tagged /
                        COSE_Mac0_Tagged / COSE_Sign1_Tagged
```

The authenticationWrapper contains a cryptographic authentication wrapper for the core part of the manifest. This is implemented as a COSE_Mac_Tagged or COSE_Sign_Tagged block. The Manifest is authenticated by this block 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 AuthenticationWrapper MUST come first in the OuterWrapper, regardless of canonical encoding of CBOR. All validators MUST reject any

OuterWrapper that begins with any element other than an AuthenticationWrapper.

Every other element 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.

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

PreInstallationInfo provides all the information that a device needs in order to decide whether and when to install an update

InstallationInfo provides all the information that a device needs in order to process one or more resources into one or more payloads.

PostInstallationInfo provides the information that a device needs to verify that a payload has been installed correctly, any instructions for what to do after the payload has been installed, for example migration tools.

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

7.2. 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 payloads

5. a reference for each of the severable blocks.

The following CDDL fragment defines the manifest.

```
Manifest = {
    manifestVersion : 1,
    sequence        : SequenceNumber,
    ? preInstall    : PreInstallationInfo / COSE_Digest,
    ? dependencies  : [* DependencyInfo],
    ? payloads      : [* PayloadInfo],
    ? install       : InstallationInfo / COSE_Digest,
    ? postInstall   : PostInstallationInfo / COSE_Digest,
    ? text          : TextInfo / COSE_Digest,
    ? coswid        : concise-software-identity / COSE_Digest
}

manifestVersion = 1
sequence        = 2
preInstall      = 3
dependencies    = 4
payloads        = 5
install         = 6
postInstall     = 7
text            = 8
coswid          = 9

SequenceNumber = uint
```

Several fields in the Manifest can be either a CBOR structure or a COSE_Digest. In each of these cases, the COSE_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 COSE_Digest, but are not needed by recipient devices. Because COSE_Digest is a CBOR Array and each severable element is a CBOR Map, it is straight-forward for a recipient to determine whether an element has been severed.

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

The sequence number is an 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. It MAY be convenient to use a UTC timestamp in seconds as the sequence number. The SequenceNumber is MANDATORY.

preInstall is a digest that uniquely identifies the content of the PreInstallationInfo that is packaged in the OuterWrapper. preInstall is OPTIONAL within a given Manifest. There MUST be one preInstall in at least one Manifest within an Update because PreInstallationInfo

contains the conditions that define the applicability of the Update to specific hardware/firmware versions. preInstall MAY be severable.

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

payloads is a list of PayloadInfo blocks that describe the payloads to be installed. payloads is OPTIONAL.

install is a digest that uniquely identifies the content of the InstallationInfo that is packaged in the OuterWrapper. install is OPTIONAL. install MAY

postInstall is a digest that uniquely identifies the content of the PostInstallationInfo that is packaged in the OuterWrapper. postInstall is OPTIONAL.

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

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

7.3. DependencyInfo

DependencyInfo specifies a manifest that describes one or more dependencies of the current manifest.

The following CDDL describes the DependencyInfo structure.

```
DependencyInfo = {
    depDigest  : COSE_Digest,         ; digest of the resource
    depScope   : ComponentIdentifier, ; where the dependency's
                                      ; payloads will be applied
    ? depUris  : UriList              ; where to find the resource
}
depDigest = 1
depScope  = 2
depUris   = 3

UriList = [ + [priority: int, uri: tstr] ]
ComponentIdentifier = [* bstr]
```

The depDigest 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 depUris element describes one or more indications of where to find the dependency. This element is OPTIONAL when the fetch location for a dependency is known implicitly.

The depScope element contains a ComponentIdentifier. 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 MANDATORY.

7.4. PayloadInfo

Payload Info describes a payload that is ready for installation. When representing a payload that requires a specialised installer, the Update Authority can provide information to regenerate a digest.

The following CDDL describes the PayloadInfo structure.

```
PayloadInfo = {
    payloadComponent:       ComponentIdentifier,
    payloadSize:            uint / nil,
    payloadDigest:          COSE_Digest,
    ? regenInfo : {
        regenDigest:        COSE_Digest
        regenType:          int
        ? regenParameters:  bstr
    },
}
payloadComponent = 1
payloadSize      = 2
payloadDigest    = 3
regenInfo        = 4
regenDigest      = 5
regenType        = 6
regenParameters  = 7

RegenType = LocationLengthRegenType /
            FileListRegenType /
            KeyListRegenType /
            CustomRegenType

LocationLengthRegenType = 1
FileListRegenType       = 2
KeyListRegenType        = 3
CustomRegenType         = nint

RegenParameters = LocationLengthRegenParameters /
                  FileListRegenParameters /
                  KeyListRegenParameters /
                  CustomRegenParameters
LocationLengthRegenParameters = [ * [ location: uint, length: uint ] ]
FileListRegenParameters       = [ * file: tstr ]
KeyListRegenParameters        = [ * key: tstr ]
CustomRegenParameters         = bstr
```

The payloadComponent element contains a ComponentIdentifier. This specifies the module/component/location in which the payload should be installed. The meaning of ComponentIdentifier is application-specific. In general, the last bstr in the ComponentIdentifier defines where to store a payload within a given storage subsystem in a Heterogeneous Storage Architecture device, the remainder of the elements in the ComponentIdentifier define which storage subsystem to use to store the payload. payloadComponent is MANDATORY. When used on a single-image device payloadComponent MAY contain 0 elements. On multi-image devices, payloadComponent MUST contain at least one element.

payloadSize contains a positive integer that describes the size of the ready-to-install payload. Where the payload requires a specialised installer, this is the payload prior to installation. This element is MANDATORY.

payloadDigest contains a digest of the payload, prior to installation. For payloads that do not require a specialised installer, this is the also the post-installation digest. This element is MANDATORY.

regenInfo describes the mechanism for recreating a message digest of payload that requires a specialised installer. This element is OPTIONAL. This element is OPTIONAL TO IMPLEMENT.

regenDigest is a Digest that contains the message digest that an application should regenerate to verify the installed payload. This element is MANDATORY when regenInfo is present.

regenType is an int that identifies a particular mechanism for creating the regenDigest. This element is MANDATORY when regenInfo is present.

regenParameters is a bstr that provides any additional arguments needed by the specialised installer. This element is OPTIONAL.

When message digest regeneration is in place, regenType implies a regenParameters structure, as described in the following table:

regenType	RegenParameters	Description
0	-	Reserved
1	[* [location: uint, length: uint]]	Lists a series of regions to include in the digest
2	[* file: tstr]	Lists a series of files to digest
3	[* key: tstr]	Lists a series of keys, whose values should be digested

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

7.5. PreInstallationInfo

The recipient processes the PreInstallationInfo in order to determine whether the manifest is applicable to it. This check is only needed once, so the PreInstallationInfo is severable.

The following CDDL describes the PreInstallationInfo structure.

```
PreInstallationInfo = {
    ? preConditions : [ * PreCondition ],
    ? preDirectives : [ * PreDirective ]
}
preConditions = 1
preDirectives = 2
```

preConditions contains a list of 0 or more PreCondition structures.

preDirectives contains a list of 0 or more PreDirective structures.

7.6. PreCondition

PreCondition structures describe conditions that must be true in order for a manifest to be installed. The target device MUST check these conditions before any installation is performed. The target device MAY check these conditions prior to fetching any dependency manifests.

All updates MUST contain either a device IdCondition or both a vendor IdCondition and a class IdCondition. This is to ensure that firmware is only ever delivered to compatible devices.

The following CDDL describes the PreCondition structure.

```
PreCondition    = IdCondition /
                  TimeCondition /
                  ImageCondition /
                  BatteryLevelCondition /
                  CustomCondition
IdCondition     = [ vendor : 1, id: Uuid ] /
                  [ class  : 2, id: Uuid ] /
                  [ device : 3, id: Uuid ]
Uuid = bstr .size 16

TimeCondition   = [useBy: 4,
                   time:       Timestamp]
ImageCondition  = [ currentContent :    6 ,
                    digest: COSE_Digest / nil,
                    location: ComponentIdentifier ] /
                  [ notCurrentContent : 7 ,
                    digest: COSE_Digest / nil,
                    location: ComponentIdentifier ]
BatteryLevelCondition = [ batteryLevel: 8,
                          level: uint ]
CustomCondition = [nint,
                   parameters: bstr]

Timestamp       = uint
```

All PreConditions are serialised as a list of one integer and one or more parameters. The type of the parameters is dictated by the value of the integer. An update that has contradictory preConditions MUST be rejected.

IdCondition describes three conditions: the vendor ID condition, the class ID condition, and the device ID condition. 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.

A device MUST have at least one vendor ID and one class ID. A device MAY have one or more device IDs, more than one vendor ID, and/or more than one class ID.

TimeCondition describes one condition: the useBy condition, which can be used to specify the last time at which an update should be installed. The timestamp is 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.

ImageCondition describes two conditions: the currentContent and the notCurrentContent conditions. Both of these conditions specify a

storage identifier and a digest that the contents of that storage identifier should match.

BatteryLevelCondition 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, BatteryLevelDirective is more appropriate, since it defines a "wait" until the battery level is sufficient to install the update. BatteryLevelCondition is specified in mWh.

CustomCondition 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 system that evaluates the condition matching that integer.

Positive Condition numbers are reserved for IANA registration. Negative numbers are reserved for proprietary, application-specific directives. When a negative number is used, the parameters MUST be wrapped in a bstr.

7.7. 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 does not 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.

7.7.1. Creating UUIDs

UUIDs MUST be created according to RFC 4122 [RFC4122]. UUIDs SHOULD use versions 3, 4, or 5, as described in RFC 4122. 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)

7.8. PreDirective

PreDirective structures describe operations that a device MUST execute prior to installing an update. For example, shut down monitored equipment, enter safe mode, sync cached files to disk, wait for another device to be updated, or wait until a specific time. Some PreDirectives may appear similar to PreConditions, however there is one difference: a PreCondition is evaluated at one time. A PreDirective can include a "wait" instruction, that means that the evaluation of the manifest does not immediately fail if the condition in the PreDirective is not met. Instead, the PreDirective remains active, waiting for its condition to be met.

For example, suppose two devices, A and B. Device B has an "other device firmware version" condition, requiring Device A to be at Rev 2. If both devices are updated from Rev 1 to Rev 2 simultaneously, then Device B may fail the PreCondition check if Device A has not finished its installation. If a PreDirective is used instead, then it can be a "wait for other device firmware version" directive. Then, Device B will postpone its update until Device A has finished updating.

The following CDDL describes the PreDirective structure.

```
PreDirective = WaitUntilDirective /
               DayOfWeekDirective /
               TimeOfDayDirective /
               BatteryLevelDirective /
               ExternalPowerDirective /
               CustomDirective

WaitUntilDirective     = [ 1,
                           timestamp: uint ]
DayOfWeekDirective     = [ 2, day: 0..6 ]
TimeOfDayDirective     = [ 3, hours: 0..23,
                              ? minutes: 0..59,
                              ? seconds: 0..59 ]
BatteryLevelDirective  = [ 4, level: uint]
ExternalPowerDirective = [ 5 ]
NetworkDisconnectDirective = [ 6 ]
CustomDirective        = [ nint,
                           ? parameters: bstr ]
```

WaitUntilDirective instructs the target device to wait until a specific time to install the update. The timestamp is 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.

DayOfWeekDirective instructs the target device to wait until a specific day of the week to install the update. The day is encoded as days since Sunday, with Sunday being day 0 and Saturday being day 6.

TimeOfDayDirective instructs the target device to wait until a specific time each day to install the update. When combined with DayOfWeekDirective, this can specify a particular time on a particular day of the week to install an update. Leap seconds are not allowed in the TimeOfDayDirective.

BatteryLevelDirective defines a directive to wait until the battery is above the specified value. This is for use in rechargeable battery and energy harvesting devices because it instructs the device to wait for a minimum charge. BatteryLevelCondition should be used in discharge-only devices. BatteryLevelDirective is specified in mWh. Battery Levels MUST be evaluated in 16 bits or more. 32 bit evaluation MUST be used for high battery capacity devices (over 65535 mWh capacity)

ExternalPowerDirective defines a directive to the device to wait until it is connected to an external power source before installing the update.

NetworkDisconnectDirective defines a directive to the device to disconnect from the network before installing the update.

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

7.9. InstallationInfo

InstallationInfo contains the information that a device needs in order to install a payload. As described in [Payloads], some payloads require specialised installers. Where a specialised installer is needed, the InstallationInfo block must represent this requirement.

InstallationInfo is described by the following CDDL.

```
InstallationInfo = {
    payloadInstallationInfo : [ * PayloadInstallationInfo ],
}
payloadInstallationInfo = 1
```

installationInfo contains a list of 0 or more PayloadInstallationInfo blocks. PayloadInstallationInfo is described by the following CDDL

```
PayloadInstallationInfo = {
    installComponent :          ComponentIdentifier
    payloadProcessors :         [ * Processor ],
    ? allowOverride :    bool,
    ? payloadInstaller:  {
        payloadInstallerID:           [ * int ],
        ? payloadInstallerParameters: bstr,
    }
}
installComponent = 1
payloadProcessors = 2
allowOverride = 3
payloadInstaller = 4
payloadInstallerID = 5
payloadInstallerParameters = 6
```

installComponent defines the component identifier of the component to update. This includes both the storage subsystem designator and the path within the storage subsystem as the final element of the component identifier. This element is MANDATORY.

payloadProcessors defines where to obtain a resource and how to transform it into a payload. This element is MANDATORY. Processors MUST be instantiated in a parent-last order.

The payloadInstaller contains a map of elements that are only needed when a specialised payload installer is used. This element is OPTIONAL TO IMPLEMENT.

payloadInstallerID contains an integer that defines which payload installer will be used. Positive integers are reserved for IANA registration. Negative integers are reserved for application-specific payload installers. Default payload installers are listed below. This element is MANDATORY when payloadInstaller is defined.

payloadInstallerID	Definition
[0]	Binary
[5, 2]	Intel Hex
[5, 3]	Motorola S-Record
[6, 1]	Executable and Linkable Format (ELF)
[7, 1]	CBOR-encoded data

Note that specialised installer 0 (binary) is typically not necessary and SHOULD ONLY be used when one of the other members of the payloadInstaller structure is required for a particular application.

These IDs are chosen to match those chosen for ProcessorIDs.

payloadInstallerParameters contains a bstr that provides any additional arguments needed by the specialised installer. This element is OPTIONAL.

7.10. Processor

Processors define one operation performed in order to modify a Resource in one step towards reconstructing the payload.

All Processors are OPTIONAL to implement.

```
Processor       = {
    processorId:       ProcessorID
    parameters:        Digest / COSE_Encrypt / COSE_Encrypt0 /
                       int / tstr / bstr / nil,
    inputs:            UriList / ComponentIdentifier /
                       ProcessorDependencies
}
ProcessorID = [ * int ]
ProcessorDependencies = {int => int}
```

The form of parameters and inputs depends on the processorId.

ProcessorDependencies is an interger-indexed map of integers. Each processor defines its inputs as integers-these are the indicies of the map. The inputs to the processor are other processors, identified by index in the Processors list. Processors that use the ProcessorDependencies input form MUST have an index in the Processors list greater than any index listed in ProcessorDependencies. The last processor listed in Processors is the processor that generates the payload to be installed in the ComponentIdentifier in PayloadInstallationInfo.

processorID contains a list of ints. This is conceptually similar to an OID, however, unlike an OID, this list is context-sensitive, encoded as a CBOR list, and supports negative numbers. The reasons for these distinctions are as follows. Contextual IDs are smaller because their use is correlated with their context. CBOR is already in use, so it reduces the number of CODECs required. Negative numbers allow for non-standard extension of IDs.

Devices are expected to compare processorIDs, bytewise, as binary blobs.

The first integer represents a broad classification of the processor, as defined in the following table.

ID[0]	Type	Description
0	Reserved	Do not use.
1	Resource	Indicates that the processor sources data by reading a resource.
2	Cipher	Encrypts or Decrypts data.
3	Compress	Compresses or Decompresses data.
4	Relocate	Reserved for Relocation.
5	BinText	Packs or Unpacks Binary-to-text encoding formats.
6	Object	Reserved for object formats, such as elf.

Each of these classifications has a subset

7.10.1. Resource

A resource can be either local or remote. Local resources fetch ComponentIdentifiers. Remote Resources fetch from URIs.

ID	Type	Parameters	Inputs	Description
[1, 1]	Remote	Digest	UriList	Fetch a resource from a remote location
[1, 2]	Local	Digest	ComponentIdentifier	Fetch a resource from a local location

7.10.2. Cipher

Ciphers are typically implemented using a cryptographic container, such as a COSE_Encrypt structure. In the context of SUIT Ciphers are typically used in decrypt mode, so this is the default behaviour. If encrypt mode is needed, then this can be achieved by extending the ID as shown in the table below.

Only one input is used, specified using ProcessorDependencies[0], a positive integer index of the data to be processed in the tree.

Only two Cipher modes are defined, the COSE_Encrypt and COSE_Encrypt0 Cipher mode2.

ID	Type	Parameters	Inputs	Description
[2, 1]	COSE_Encrypt	COSE_Encrypt	{0 : dataIdx}	Decrypt data enveloped by a COSE_Encrypt structure
[2, 1, 2]	COSE_Encrypt	COSE_Encrypt	{0 : dataIdx}	Encrypt data enveloped by a COSE_Encrypt structure
[2, 2]	COSE_Encrypt0	COSE_Encrypt0	{0 : dataIdx}	Decrypt data enveloped by a COSE_Encrypt0 structure
[2, 2, 2]	COSE_Encrypt0	COSE_Encrypt0	{0 : dataIdx}	Encrypt data enveloped by a COSE_Encrypt0 structure

Mode 1 (decrypt) is implied when mode 2 (encrypt) is not specified.

7.10.3. Compress

A compression/decompression algorithm. Typically, this means that the input should contain a valid compression container. Compression algorithms are typically used in decompress mode, so this is the default behaviour.

Only one input is used, specified using ProcessorDependencies[0], a positive integer index of the data to be processed in the tree.

ID	Type	Parameters	Inputs	Description
[3, 1]	gzip	nil	{0 : dataIdx}	Decompress using gzip
[3, 1, 2]	gzip	nil	{0 : dataIdx}	Compress using gzip
[3, 2]	bzip2	nil	{0 : dataIdx}	Decompress using bzip2
[3, 2, 2]	bzip2	nil	{0 : dataIdx}	Compress using bzip2
[3, 4]	lz4	nil	{0 : dataIdx}	Decompress using lz4
[3, 4, 2]	lz4	nil	{0 : dataIdx}	Compress using lz4
[3, 7]	lzma	nil	{0 : dataIdx}	Decompress using lzma
[3, 7, 2]	lzma	nil	{0 : dataIdx}	Compress using lzma

Mode 1 (decompress) is implied when mode 2 (compress) is not specified.

7.10.4. Relocate

Relocation is reserved for future use.

7.10.5. BinText

Packs or unpacks a binary-to-text format.

WARNING: Some binary-to-text formats can cause significant difficulty for a resource-constrained device. They also dramatically increase bandwidth over equivalent binary formats, with the worst being hex encoding at a 2:1 inflation. The best is base64 at a 4:3 inflation.

binary-to-text formats are typically used in decode mode, so this is the default behaviour.

Only one input is used, specified using ProcessorDependencies[0], a positive integer index of the data to be processed in the tree.

ID	Type	Parameters	Inputs	Description
[5, 1]	base64	uint / tstr	{ 0 : dataIdx }	Decode base64 data in one of several encodings
[5, 1, 2]	base64	uint / tstr	{ 0 : dataIdx }	Decode base64 data in one of several encodings
[5, 2]	hex	nil	{ 0 : dataIdx }	Decode Intel Hex data
[5, 2, 2]	hex	nil	{ 0 : dataIdx }	Encode Intel Hex data
[5, 3]	srecord	nil	{ 0 : dataIdx }	Decode S-Record data
[5, 3, 2]	srecord	nil	{ 0 : dataIdx }	Encode S-Record data

Mode 1 (decode) is implied when mode 2 (encode) is not specified.

When base64 is specified, several choices of parameter are available:

Type	Value	Description
uint	1	RFC 4648 [RFC4648] standard base64
uint	2	base64url
tstr	base64 characterset	Arbitrary base64, as specified by the character set.

7.10.6. Object

Object packing and unpacking is reserved for future use.

7.11. PostInstallationInfo

PostInstallationInfo contains information that the recipient needs in order to determine whether an installation has completed successfully and whether anything needs to be done after completion. These checks and instructions are only needed once, so PostInstallationInfo is severable.

The following CDDL describes the PostInstallationInfo structure.

```
PostInstallationInfo = {
    ? postConditions : [ * PostCondition ],
    ? postDirectives : [ * PostDirective ]
}
postConditions = 1
postDirectives = 2

PostCondition   = ImageCondition / CustomCondition
PostDirective   = CustomDirective
```

postConditions contains a list of 0 or more PostCondition structures. postConditions is OPTIONAL and OPTIONAL to implement.

postDirectives contains a list of 0 or more PostDirective structures. postDirectives is OPTIONAL and OPTIONAL to implement.

PostConditions are used to specify conditions that must be true after an update has completed. The ImageCondition specifies a digest of an image that must match after application of an update.

PostDirectives can be used to specify an action taken by the recipient after application of an update is complete, such as:

- Reboot after application

- Restart designated component when installation is complete

8. Complete CDDL

A small portion of RFC 8152 [RFC8152] is reproduced in this CDDL so that COSE_Digest can be fully defined.

```
OuterWrapper = {
    authenticationWrapper: AuthenticationWrapper,
```

```
      manifest:              bstr .cbor Manifest,
      ? preInstallExt:       bstr .cbor PreInstallationInfo,
      ? installExt:          bstr .cbor InstallationInfo,
      ? postInstallExt:      bstr .cbor PostInstallationInfo,
      ? textInfoExt:         bstr .cbor Text,
      ? coswidExt:           bstr .cbor concise-software-identity
  }
  authenticationWrapper = 1
  manifest = 2
  preInstallExt = 3
  installExt = 4
  postInstallExt = 5
  textExt = 6
  coswidExt = 7

  AuthenticationWrapper = COSE_Mac_Tagged / COSE_Sign_Tagged /
                          COSE_Mac0_Tagged / COSE_Sign1_Tagged

  concise-software-identity = any
  AuthenticationWrapper = 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 = any
  COSE_Encrypt0 = any

  COSE_Mac0 = [
        Headers,
        payload : bstr / nil,
        tag : bstr,
     ]

  Headers = (
      protected : empty_or_serialized_map,
      unprotected : header_map
  )

  header_map = {
      Generic_Headers,
      * label => values
  }

  empty_or_serialized_map = bstr .cbor header_map / bstr .size 0

  Generic_Headers = (
```

```
    ? 1 => int / tstr,  ; algorithm identifier
    ? 2 => [+label],    ; criticality
    ? 3 => tstr / int,  ; content type
    ? 4 => bstr,        ; key identifier
    ? 5 => bstr,        ; IV
    ? 6 => bstr,        ; Partial IV
    ? 7 => COSE_Signature / [+COSE_Signature] ; Counter signature
  )
  COSE_Digest = COSE_Mac0

  Manifest = {
      manifestVersion : 1,
      sequence        : SequenceNumber,
      ? preInstall    : PreInstallationInfo / COSE_Digest,
      ? dependencies  : [* DependencyInfo],
      ? payloads      : [* PayloadInfo],
      ? install       : InstallationInfo / COSE_Digest,
      ? postInstall   : PostInstallationInfo / COSE_Digest,
      ? text          : TextInfo / COSE_Digest,
      ? coswid        : concise-software-identity / COSE_Digest
  }

  manifestVersion = 1
  sequence        = 2
  preInstall      = 3
  dependencies    = 4
  payloads        = 5
  install         = 6
  postInstall     = 7
  text            = 8
  coswid          = 9

  SequenceNumber = uint

  DependencyInfo = {
      depDigest  : COSE_Digest,         ; digest of the resource
      depScope   : ComponentIdentifier, ; where the dependency's
                                        ; payloads will be applied
      ? depUris  : UriList              ; where to find the resource
                                       ; applied
  }
  depDigest = 1
  depScope  = 2
  depUris   = 3

  UriList = [ + [priority: int, uri: tstr] ]
  ComponentIdentifier = [* bstr]
```

```
  PayloadInfo = {
      payloadComponent:       ComponentIdentifier,
      payloadSize:            uint / nil,
      payloadDigest:          COSE_Digest,
      ? regenInfo : {
          regenDigest:        COSE_Digest
          regenType:          int
          ? regenParameters:  bstr
      },
  }
  payloadComponent = 1
  payloadSize      = 2
  payloadDigest    = 3
  regenInfo        = 4
  regenDigest      = 5
  regenType        = 6
  regenParameters  = 7

  RegenType = LocationLengthRegenType /
              FileListRegenType /
              KeyListRegenType /
              CustomRegenType

  LocationLengthRegenType = 1
  FileListRegenType       = 2
  KeyListRegenType        = 3
  CustomRegenType         = nint

  RegenParameters = LocationLengthRegenParameters /
                    FileListRegenParameters /
                    KeyListRegenParameters /
                    CustomRegenParameters
  LocationLengthRegenParameters = [ * [ location: uint, length: uint ] ]
  FileListRegenParameters       = [ * file: tstr ]
  KeyListRegenParameters        = [ * key: tstr ]
  CustomRegenParameters         = bstr

  PreInstallationInfo = {
      ? preConditions : [ * PreCondition ],
      ? preDirectives : [ * PreDirective ]
  }
  preConditions = 1
  preDirectives = 2

  PreCondition    = IdCondition /
                    TimeCondition /
                    ImageCondition /
                    BatteryLevelCondition /
```

```
                    CustomCondition

  IdCondition     = [ vendor : 1, id: Uuid ] /
                    [ class  : 2, id: Uuid ] /
                    [ device : 3, id: Uuid ]
  Uuid = bstr .size 16

  TimeCondition   = [useBy: 4,
                     time:       Timestamp]
  ImageCondition  = [ currentContent : 6 ,
                     digest: COSE_Digest / nil,
                     location: ComponentIdentifier ] /
                    [ notCurrentContent : 7 ,
                     digest: COSE_Digest / nil,
                     location: ComponentIdentifier ]
  BatteryLevelCondition = [ batteryLevel: 8,
                            level: uint ]
  CustomCondition = [nint,
                     parameters: bstr]

  Timestamp       = uint

  PreDirective = WaitUntilDirective /
                 DayOfWeekDirective /
                 TimeOfDayDirective /
                 BatteryLevelDirective /
                 ExternalPowerDirective /
                 CustomDirective

  WaitUntilDirective     = [ 1,
                             timestamp: uint ]
  DayOfWeekDirective     = [ 2, day: 0..6 ]
  TimeOfDayDirective     = [ 3, hours: 0..23,
                                ? minutes: 0..59,
                                ? seconds: 0..59 ]
  BatteryLevelDirective  = [ 4, level: uint]
  ExternalPowerDirective = [ 5 ]
  NetworkDisconnectDirective = [ 6 ]
  CustomDirective        = [ nint,
                             ? parameters: bstr ]

  InstallationInfo = {
     payloadInstallationInfo : [ * PayloadInstallationInfo ],
  }
  payloadInstallationInfo = 1

  PayloadInstallationInfo = {
     installComponent :          ComponentIdentifier
```

```
    payloadProcessors :         [ * Processor ],
    ? allowOverride :    bool,
    ? payloadInstaller:  {
        payloadInstallerID:           [ * int ],
        ? payloadInstallerParameters: bstr,
    }
}
installComponent = 1
payloadProcessors = 2
allowOverride = 3
payloadInstaller = 4
payloadInstallerID = 5
payloadInstallerParameters = 6

Processor       = {
    processorId:       ProcessorID
    parameters:        COSE_Digest / COSE_Encrypt / COSE_Encrypt0 /
                       int / tstr / bstr / nil,
    inputs:            UriList / ComponentIdentifier / {int => int}
}
ProcessorID = [ * int ]

PostInstallationInfo = {
    ? postConditions : [ * PostCondition ],
    ? postDirectives : [ * PostDirective ]
}
postConditions = 1
postDirectives = 2

PostCondition   = ImageCondition / CustomCondition
PostDirective   = CustomDirective

Text = {
    * int => tstr
}
```

NOTE: COSE structures are specified as "any" to enable CDDL tooling to process this CDDL without including all of the COSE specification. The same consideration applies to concise-software-identifier.

9. Examples

Note: Line-breaks have been introduced to meet the character line limit.

9.1. Unsigned Manifest with One Payload

```
OuterWrapper = {
    / authenticationWrapper / 1 : F6 / null /,
    / manifest / 2: h'a3010102020581a3018141300218250384444a
                     1011829a0f658208caf9283b13666ca4e50f7
                     a1eee86ba40b5e6a1d2ca39f7498b6a6a7be8d8d67' /
    {
        \ manifestVersion \  1 : 1,
        \ sequence \ 2: 1,
        \ payloads \ 5: [
            {
                \ payloadComponent \ 1: [h'30'],
                \ payloadSize \ 2: 37,
                \ payloadDigest \ 3: [
                    \ protected \ "a1011829" \ {
                        \ alg \ 1 : 41 \ sha-256 \
                    } \ ,
                    \ unprotected \ {},
                    \ payload \ F6 \ null,
                    \ tag \ h'8caf9283b13666ca4e50f7a1eee86ba40
                             b5e6a1d2ca39f7498b6a6a7be8d8d67'
                ]
            }
        ]
    } /
}
```

```
Raw OuterWrapper: 62 bytes

a102583aa3010102020581a3018141300218250384444a1011829a0f658208c
af9283b13666ca4e50f7a1eee86ba40b5e6a1d2ca39f7498b6a6a7be8d8d67
```

9.2. ECDSA secp256r1-signed Manifest with One Payload

A manifest with payload description only, authenticated by an ECDSA signature. The signing key is identified by the Subject Key Identifier.

```
  OuterWrapper = {
      / authenticationWrapper / 1: #98([
          / protected / h'A103182A' / {
              \ content type \ 3 : 42 \ application octet-stream \
              } / ,
          / unprotected / {},
          / payload / null,
          / signatures / [
              [
                  / protected / h'A10126' / {
                      \ alg \ 1 : -7 \ ECDSA 256 \
                  },
                  / unprotected / {
                      / kid / 4 : h'537ac93ac909e79990914caa00fe87ee
                                    ea637ef89b5512e5cb6e558a136ff98d'
                  } / ,
                  / signature / h'304502201d65938ec454354a6e866b468e9
                                  808db4ef36e97de09f98fda92e9c0e3302f
                                  c8022100aff871fe581d3f6b831d74e46f9
                                  acd7a015e5548770b2a437970be9272a7fbaa'
              ]
          ]
      ])
      / manifest / 2: h'a3010102020581a30181413002182503844a1011829a
                        0f658208caf9283b13666ca4e50f7a1eee86ba40b5e6a
                        1d2ca39f7498b6a6a7be8d8d67' /
      {
          \ manifestVersion \  1 : 1,
          \ sequence \ 2: 1,
          \ payloads \ 5: [
              {
                  \ payloadComponent \ 1: [h'30'],
                  \ payloadSize \ 2: 37,
                  \ payloadDigest \ 3: [
                      \ protected \ "a1011829" \ {
                          \ alg \ 1 : 41 \ sha-256 \
                      } \ ,
                      \ unprotected \ {},
                      \ payload \ F6 \ null,
                      \ tag \ h'8caf9283b13666ca4e50f7a1eee86ba4
                                0b5e6a1d2ca39f7498b6a6a7be8d8d67'
                  ]
              }
          ]
      } /
  }
```

Raw OuterWrapper: 188 bytes

```
   a201d8628444a103182aa0f6818343a10126a1045820537ac93ac909e79990
   914caa00fe87eeea637ef89b5512e5cb6e558a136ff98d5847304502201d65
   938ec454354a6e866b468e9808db4ef36e97de09f98fda92e9c0e3302fc802
   2100aff871fe581d3f6b831d74e46f9acd7a015e5548770b2a437970be9272
   a7fbaa02583aa3010102020581a3018141300218250384 44a1011829a0f658
   208caf9283b13666ca4e50f7a1eee86ba40b5e6a1d2ca39f7498b6a6a7be8d8d67
```

9.3. A ECDSA-signed Raw Binary Payload with Conditions, Text, and InstallationInfo

```
 OuterWrapper = {
     / authenticationWrapper / 1: #98([
         / protected / h'A103182A' / {
             \ content type \ 3 : 42 \ application octet-stream \
             } / ,
         / unprotected / {},
         / payload / null,
         / signatures / [
             [
                 / protected / h'A10126' / {
                     \ alg \ 1 : -7 \ ECDSA 256 \
                 },
                 / unprotected / {
                     / kid / 4 : h'537ac93ac909e79990914caa00
                                   fe87eeea637ef89b5512e5cb6e
                                   558a136ff98d'
                 } / ,
                 / signature /"3045022100830cf142cc4adf563392dc7e043
                              0000158bf3720b28b7cea388b0f1a5f8918a8
                              02201def2df34d6abd3b17c3425573ff2b7ca
                              cae3dd085e11dfc23bf0c60be51b7da"
             ]
         ]
     ])
     / manifest / 2: h'a60101020203a10182820150fa6b4a53d5ad5fdfbe9de
                      663e4d41ffe8202506e04d3c2488759e4a597b5e7cd49
                      76530581a3018141300218250384 44a1011829a0f6582
                      08caf9283b13666ca4e50f7a1eee86ba40b5e6a1d2ca3
                      9f7498b6a6a7be8d8d6706a10181a2018141300281a20
                      1820101038200766874747 03a2f2f666f6f2e6261722f
                      62617a2e62696e088444a1011829a0f658204e2714598
                      479d8b6634805df5019ef3420edff0329894acc91de8c
                      8de16fb0cf' /
     {
         \ manifestVersion \  1 : 1,
         \ sequence \ 2 : 2,
         \ preInstall \ 3 : {
             \ preConditions 1 : [
```

```
                [ \ vendorId \ 1, h'fa6b4a53d5ad5fdfbe9de663e4d41ffe'],
                [ \ classId \ 2, h'6e04d3c2488759e4a597b5e7cd497653']
            ]
        },
        \ payloads \ 5: [
            {
                \ payloadComponent \ 1: [h'30'],
                \ payloadSize \ 2: 37,
                \ payloadDigest \ 3: [
                    \ protected \ "a1011829" \ {
                        \ alg \ 1 : 41 \ sha-256 \
                    } \ ,
                    \ unprotected \ {},
                    \ payload \ F6 \ null,
                    \ tag \ h'8caf9283b13666ca4e50f7a1eee86ba4
                              0b5e6a1d2ca39f7498b6a6a7be8d8d67'
                ]
            }
        ],
        \ install \ 6 : {
            \ payloadInstallationInfo \ 1 : [
                {
                    \ installComponent \ 1 : [h'30'],
                    \ payloadProcessors \ 2 : [
                        {
                            \ processorId \ 1 :
                                 [1,1] \ remote resource \ ,
                            \ inputs \ 3 : [
                                0, "http://foo.bar/baz.bin"
                            ]
                        }
                    ]
                }
            ]
        },
        \ textInfo \ 8 : [
            \ protected \ "a1011829" \ {
                \ alg \ 1 : 41 \ sha-256 \
            } \ ,
            \ unprotected \ {},
            \ payload \ F6 \ null,
            \ tag \ "4e2714598479d8b6634805df5019ef342
                     0edff0329894acc91de8c8de16fb0cf"
        ]
    }
    / textInfoExt / 6 : h'a10178c84c6f72656d20697073756d20646f6c6f7
                          22073697420616d65742c20636f6e736563746574
                          75722061646970 697363696e6720656c69742e20
```

```
                          4e756e63207365642074696e636964756e7420616
                          e74652c206120736f64616c6573206c6967756c61
                          2e205068617365 6c6c757320756c6c616d636f72
                          706572206f64696f20636f6d6d6f646f206970737
                          56d20656765737461732c207669746165206c6163
                          696e6961206c656f206f726e6172652e205375737
                          0656e646973736520706f7375657265207365642e' /
    {
        \ updateDescription \ 1 : "Lorem ipsum dolor sit amet,
                                   consectetur adipiscing elit.
                                   Nunc sed tincidunt ante, a
                                   sodales ligula. Phasellus
                                   ullamcorper odio commodo ipsum
                                   egestas, vitae lacinia leo ornare.
                                   Suspendisse posuere sed."
    } /
}
```

Raw OuterWrapper: 522 bytes

```
a301d8628444a103182aa0f6818343a10126a1045820537ac93ac909e79990914ca
a00fe87eeea637ef89b5512e5cb6e558a136ff98d58473045022100830cf142cc4a
df563392dc7e0430000158bf3720b28b7cea388b0f1a5f8918a802201def2df34d6
abd3b17c3425573ff2b7cacae3dd085e11dfc23bf0c60be51b7da0658cca10178c8
4c6f72656d20697073756d20646f6c6f722073697420616d65742c20636f6e73656
3746574757220616469706973636963696e6720656c69742e204e756e63207365642074
696e636964756e7420616e74652c206120736f64616c6573206c6967756c612e205
0686173656c6c757320756c6c616d636f72706572206f64696f20636f6d6d6f646f
20697073756d20656765737461732c207669746165206c6163696e6961206c656f2
06f726e6172652e2053757370656e646973736520706f7375657265207365642e02
58b9a60101020203a10182820150fa6b4a53d5ad5fdfbe9de663e4d41ffe8202506
e04d3c2488759e4a597b5e7cd4976530581a3018141300218250384444a1011829a0
f658208caf9283b13666ca4e50f7a1eee86ba40b5e6a1d2ca39f7498b6a6a7be8d8
d6706a10181a2018141300281a2018201010382007668747470 3a2f2f666f6f2e62
61722f62617a2e62696e088444a1011829a0f658204e2714598479d8b6634805df5
019ef3420edff0329894acc91de8c8de16fb0cf
```

Text severed (textInfoExt deleted from OuterWrapper): 315 bytes

```
a201d8628444a103182aa0f6818343a10126a1045820537ac93ac909e79990914ca
a00fe87eeea637ef89b5512e5cb6e558a136ff98d58473045022100830cf142cc4a
df563392dc7e0430000158bf3720b28b7cea388b0f1a5f8918a802201def2df34d6
abd3b17c3425573ff2b7cacae3dd085e11dfc23bf0c60be51b7da0258b9a6010102
0203a10182820150fa6b4a53d5ad5fdfbe9de663e4d41ffe8202506e04d3c248875
9e4a597b5e7cd4976530581a3018141300218250384444a1011829a0f658208caf92
83b13666ca4e50f7a1eee86ba40b5e6a1d2ca39f7498b6a6a7be8d8d6706a10181a
2018141300281a2018201010382007668747470 3a2f2f666f6f2e6261722f62617a
```

2e62696e088444a1011829a0f658204e2714598479d8b6634805df5019ef3420edf
f0329894acc91de8c8de16fb0cf

10. IANA Considerations

Several registries will be required for:

- standard Conditions
- standard Directives
- standard Processors
- standard text values

Editor's Note: A few registries would be good to allow easier allocation of new features.

11. 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].

12. 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]

13. 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

- Carsten Bormann

- Henk Birkholz

- Oyvind Ronningstad

- Frank Audun Kvamtro

- Krzysztof Chruscinski

- Andrzej Puzdrowski

- Michael Richardson

- David Brown

Finally, we would like to thank the IETF SUIT working group chairs, Dave Thaler, David Waltermire, and Russ Housley, for their support.

14. References

14.1. 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>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/info/rfc4648>.

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

14.2. Informative References

[Architecture] Moran, B., Meriac, M., Tschofenig, H., and D. Brown, "A Firmware Update Architecture for Internet of Things Devices", draft-ietf-suit-architecture-01 (work in progress), July 2018.

[Information]
Moran, B., Tschofenig, H., and H. Birkholz, "Firmware Updates for Internet of Things Devices - An Information Model for Manifests", draft-ietf-suit-information-model-01 (work in progress), July 2018.

14.3. 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@gmx.net