SUIT
Internet-Draft
Intended status: Standards Track
Expires: 18 September 2025

B. Moran
Arm Limited
K. Takayama
SECOM CO., LTD.

17 March 2025

Update Management Extensions for Software Updates for Internet of Things (SUIT) Manifests
draft-ietf-suit-update-management-09

Abstract

This specification describes extensions to the SUIT manifest format. These extensions allow an update author, update distributor or device operator to more precisely control the distribution and installation of updates to devices. These extensions also provide a mechanism to inform a management system of Software Identifier and Software Bill Of Materials information about an updated device.

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 18 September 2025.

Copyright Notice

Copyright (c) 2025 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction

Full management of software updates for unattended, connected devices requires a cooperation between the update author(s) and management, distribution, policy enforcement, and auditing systems. This specification provides the extensions to the SUIT manifest that enable an author to coordinate with these other systems. These extensions enable authors to instruct devices to examine update priority, local update authorisation, update lifetime, and system properties. They also enable devices to report and distributors to collect Software Bill of Materials information.

Extensions in this specification are OPTIONAL to implement and OPTIONAL to include in manifests.

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.

This draft makes use of terminology defined in [RFC9019] and [I-D.ietf-suit-manifest].

3. Extension Metadata

Some additional metadata makes management of SUIT updates easier:

* A semantic version number for the update represented by the manifest
* Concise Software Identifiers (CoSWID) [RFC9393]
* Text descriptions of requirements
* Text description of the current versions of components

3.1. suit-set-version

This metadata encodes a semantic version for the component set that the manifest updates, including any dependencies. This enables version comparisons to be performed on manifests. Non-manifest images encode their versions independently of the manifest.

The version SHOULD be encoded as a semantic version, according to [semver]. There are several restrictions to these composition rules: alphanumeric pre-release indicators are not permitted. Because suit-set-version is a machine-readable parameter for determining compatibility and because [semver] mandates that the build-number is ignored, build numbers SHOULD NOT be included.

The composition of suit-set-version is the same as suit-parameter-version (Section 4.4).

If a build number is desired, it SHOULD be included via text-current-version (Section 3.4).

3.2. suit-coswid

A CoSWID can enable Software Bill of Materials use-cases. Tightly coupling update and attestation ensures that verification infrastructure always knows what software to expect on each device.

suit-coswid is a member of the suit-manifest. It contains a Concise Software Identifier (CoSWID) as defined in [RFC9393]. This element SHOULD be made severable so that it can be discarded by the Recipient or an intermediary if it is not required by the Recipient.

suit-coswid typically requires no processing by the Recipient. However all Recipients MUST NOT fail if a suit-coswid is present.

suit-coswid is RECOMMENDED to implement and RECOMMENDED to include in manifests.

3.3. suit-text-version-required

suit-text-version-required is used to represent a version-based dependency on suit-parameter-version as described in Section 4.4 and Section 5.5. To describe a version dependency, a Manifest Author SHOULD populate the suit-text map with a SUIT_Component_Identifier key for the dependency component, and place in the corresponding map a suit-text-version-required key with a free text expression that is representative of the version constraints placed on the dependency. This text SHOULD be expressive enough that a device operator can be expected to understand the dependency. This is a free text field and there are no specific formatting rules.

By way of example only, to express a dependency on a component "[’x’, ’y’]", where the version should be any v1.x later than v1.2.5, but not v2.0 or above, the author would add the following structure to the suit-text element. Note that this text is in cbor-diag notation.

```
['x','y'] : {
    7 : ">=1.2.5,<2"
}
```

3.4. text-current-version

suit-text-current-version is used to provide human-readable version information equivalent to suit-set-version (Section 3.1). This metadata MAY have a version listed for each or any component. The Manifest Processor MUST NOT consume this version; it is for human readability only.

To describe a version, a Manifest Author SHOULD populate the suit-text map with a SUIT_Component_Identifier key for the dependency component, and place in the corresponding map a suit-text-current-version key with a free text version that is representative of the version of the component. This text SHOULD be expressive enough that a device operator can be expected to understand the version. This is a free text field and there are no specific formatting rules.

It is RECOMMENDED that the Manifest Author use a Semantic Version ([semver]) in the free-text field. Unlike suit-set-version (Section 3.1), the full semantic version specification can be used.

4. Extension Parameters

Several parameters are needed to define the behaviour of the commands specified in Extension Commands (Section 5). These parameters follow the same considerations as defined in Section 8.4.8 of [I-D.ietf-suit-manifest].

Name	CDDL Structure	Reference
Use Before	suit-parameter-use-before	Section 4.1
Minimum Battery	suit-parameter-minimum-battery	Section 4.2
Update Priority	suit-parameter-update-priority	Section 4.3
Version	suit-parameter-version	Section 4.4
Wait Info	suit-parameter-wait-info	Section 4.5
Component Metadata	suit-parameter-component-metadata	Section 4.6

Table 1

4.1. suit-parameter-use-before

An expiry date for the use of the manifest encoded as the positive integer number of seconds since 1970-01-01. Implementations that use this parameter MUST use a 64-bit internal representation of the integer. Used with Section 5.1.

4.2. suit-parameter-minimum-battery

This parameter sets the minimum battery level in mWh. This parameter is encoded as a positive integer. Used with suit-condition-minimum-battery (Section 5.3).

4.3. suit-parameter-update-priority

This parameter sets the priority of the update. This parameter is encoded as an integer. It is used along with suit-condition-update-authorized (Section 5.4) to ask an application for permission to initiate an update. This does not constitute a privilege inversion because an explicit request for authorization has been provided by the Update Authority in the form of the suit-condition-update-authorized command.

Applications MAY define their own meanings for the update priority. For example, critical reliability and vulnerability fixes might be given negative numbers, while bug fixes might be given small positive numbers, and feature additions might be given larger positive numbers, which allows an application to make an informed decision about whether and when to allow an update to proceed.

4.4. suit-parameter-version

Indicates allowable versions for the specified component. One version comparison can be made with each suit-parameter-version. This parameter is compared with version asserted by the current component when suit-condition-version (Section 5.5) is invoked. The current component may assert the current version in many ways, including storage in a parameter storage database, in a metadata object, or in a known location within the component itself.

Each suit-parameter-version contains a comparison operator and a version, according to the following CDDL:

```
SUIT_Parameter_Version_Match = [
    suit-condition-version-comparison-type:
        SUIT_Condition_Version_Comparison_Types,
    suit-condition-version-comparison-value:
        SUIT_Condition_Version_Comparison_Value
]
```

The comparison type can be:

* Greater.

* Greater or Equal.

* Equal.

* Lesser or Equal.

* Lesser.

The version comparison value is encoded as a CBOR list of integers. Comparisons are done on each integer in sequence. Comparison stops after all integers in the list defined by the manifest have been consumed OR after an non-equal comparison has occurred. For example, if the manifest defines a comparison, "Equal [1]", then this will match all version sequences starting with 1. If a manifest defines both "Greater or Equal [1,0]" and "Lesser [1,10]", then it will match versions 1.0.x up to, but not including 1.10.

suit-parameter-version is OPTIONAL to implement.

4.4.1. suit-parameter-version Semantic Versioning encoding guidelines

The encoded versions SHOULD be semantic versions (See [semver]). For example,

* 1.2.3 = [1,2,3].
* 1.2-rc.3 = [1,2,-1,3].
* 1.2-beta = [1,2,-2].
* 1.2-alpha = [1,2,-3].
* 1.2.3-alpha.4 = [1,2,3,-3,4].

Versions SHOULD be composed of:

1. A release version encoded as a sequence of 1 to 3 positive integers
2. An optional pre-release indicator encoded as a negative integer, followed by zero or more positive integers

While [semver] allows a build number, it mandates that the build number is ignored. Because suit-parameter-version exists solely to enable the Manifest Processor to make a decision about version compatibility, build numbers SHOULD NOT be included.

In [semver],

1. The first integer represents the major number. This indicates breaking changes to the component.
2. The second integer represents the minor number. This is typically reserved for new features or large, non-breaking changes.
3. The third integer is the patch version. This is typically reserved for bug fixes.

The pre-release indicator SHOULD NOT appear as element 0. The pre-release indicator is encoded as:

* -1: Release Candidate
* -2: Beta

* -3: Alpha

This allows these releases to compare correctly with final releases. For example, Version 2.0, RC1 should be lower than Version 2.0.0 and higher than any Version 1.x. By encoding RC as -1, this works correctly: [2,0,-1,1] compares as lower than [2,0,0]. Similarly, beta (-2) is lower than RC and alpha (-3) is lower than RC.

4.5. suit-parameter-wait-info

suit-directive-wait (Section 5.6) directs the manifest processor to pause until a specified event occurs. The suit-parameter-wait-info encodes the parameters needed for the directive.

The exact implementation of the pause is implementation-defined. For example, this could be done by blocking on a semaphore, registering an event handler and suspending the manifest processor, polling for a notification, or aborting the update entirely, then restarting when a notification is received.

suit-parameter-wait-info is encoded as a map of wait events. When ALL wait events are satisfied, the Manifest Processor continues. The wait events currently defined are described in the following table.

Name	Encoding	Description
suit-wait-event-authorization	int	Same as suit-parameter-update-priority
suit-wait-event-power	int	Wait until power state
suit-wait-event-network	int	Wait until network state
suit-wait-event-other-device-version	See below	Wait for other device to match version
suit-wait-event-time	uint	Wait until time (seconds since 1970-01-01)
suit-wait-event-time-of-day	uint	Wait until seconds since 00:00:00 Local Time
suit-wait-event-time-of-day-utc	uint	Wait until seconds since 00:00:00 UTC
suit-wait-event-day-of-week	uint	Wait until days since Sunday Local Time
suit-wait-event-day-of-week-utc	uint	Wait until days since Sunday UTC

Table 2

suit-wait-event-other-device-version reuses the encoding of suit-parameter-version-match. It is encoded as a sequence that contains an implementation-defined bstr identifier for the other device, and a list of one or more SUIT_Parameter_Version_Match.

4.6. suit-parameter-component-metadata

In some instances, a system may need to know the file metadata for a component. This metadata can include:

* creator
* creation time
* modification time

* default permissions (rwx)

* a map of user/permission pairs

* a map of role/permission pairs

* a map of group/permission pairs

* file type

Component metadata is applied at time of fetch, copy, or write; see [I-D.ietf-suit-manifest], sections 8.4.10.4, 8.4.10.5, 8.4.10.6. Therefore, the component metadata parameter must be set in advance of the component being fetched, copied into, or written.

4.6.1. Creator

Sometimes, management of file systems requires that the creator of each file is correctly recorded. Because the default creator of files will be the update agent, this can obscure the actual creator of each file. The Creator metadata element allows overriding the default behaviour and setting the correct creator.

The creator is defined as follows:

```
SUIT_meta_actor_id = UUID_Tagged / bstr / str / int
UUID_Tagged = #6.37(bstr)
```

The actor ID can be whatever is most appropriate for any given system. For example, the actor ID might be a string (e.g., username), integer (e.g., POSIX userid), or UUID (e.g., TEEP TA UUID).

4.6.2. Creation & Modification Time

The creation and modification times are defined by CBOR time types. These are defined in [RFC8949], Section 3.4.2. The CBOR tag is REQUIRED when either creation or modification time are provided.

```
suit-meta-modification-time => #6.1(uint)
suit-meta-creation-time => #6.1(uint)
```

4.6.3. Component Default Permissions

Typical permissions management systems require read, write, and execute permissions that are applied to all users who do not have their own explicit permissions. These are the default permissions for the current component. Default permissions are described by the following CDDL:

```
SUIT_meta_permissions = uint .bits SUIT_meta_permission_bits
SUIT_meta_permission_bits = &(
    r: 2, w: 1, x: 0,
    * $$SUIT_meta_permission_bits_extensions
)
```

4.6.4. User, Role, Group permissions

Many filesystems have users and groups. Additionally some have roles. Actors that have these associations can have specific permissions associated with them for each component. Each of these sets of permissions is defined the same way: with a map of actor identifiers to permissions.

```
SUIT_meta_permission_map = {
    + SUIT_meta_actor_id => SUIT_meta_permissions
}
```

The SUIT_meta_actor_id is the same as defined for Creator, Section 4.6.1.

4.6.5. File Type

File Type typically identifies whether a file is a directory, regular file, or symbolic link. If not specified, File Type defaults to regular file.

This enables specific management operations for SUIT command sequences:

* To create a directory
  - Set the Component Index to the Component Identifier of the directory to be created
  - Set the Component metadata, including the file type for directory
  - Set suit-parameter-content to an empty bstr

- Invoke suit-directive-write

* To create a symbolic link

- Set the Component Index to the Component Identifier of the link to be created
- Set the Component metadata, including the file type for symbolic link
- Set suit-parameter-content to the link target
- Invoke suit-directive-write

For example, the following Payload Fetch & Install sequences will create a new /usr/local/bin directory, download https://cdn.example/example3.bin into a new file: /usr/local/bin/example3, then create a symlink at /usr/bin/example that points to /usr/local/bin/example3.

* Common has components for:

- /usr/bin/example
- /usr/local/bin
- /usr/local/bin/example3

* Payload fetch:

- set component index = 1
- set parameters:
  - content = h''
  - metadata = {file-type: directory}
- write
- set component index = 2
- set URI = "https://cdn.example/example3.bin"
- fetch
- condition image digest

* Install:

- set component index = 0
- set parameters:
  - o content = "/usr/local/bin/example3"
  - o metadata = {file-type: symlink}
- write

5. Extension Commands

The following table defines the semantics of the commands defined in this specification in the same way as in the Abstract Machine Description, Section 6.4, of [I-D.ietf-suit-manifest].

Command Name	CDDL Identifier	Semantic of the Operation
Use Before	suit-condition-use-before	assert(now() < current.params[use-before])
Check Image Not Match	suit-condition-image-not-match	assert(not binary-match(digest(current), current.params[digest]))
Check Minimum Battery	suit-condition-minimum-battery	assert(battery >= current.params[minimum-battery])
Check Update Authorized	suit-condition-update-authorized	assert(isAuthorized(current.params[priority]))
Check Version	suit-condition-version	assert(version_check(current, current.params[version]))
Wait For Event	suit-directive-wait	until event(arg), wait
Override Multiple	suit-directive-override-multiple	components[i].params[k] := v for-each k,v in d for-each i,d in arg
Copy Params	suit-directive-copy-params	current.params[k] = components[i].params[k] for k in l for i,l in arg

Table 3

5.1. 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. The recipient evaluates the current time against the suit-parameter-use-before parameter (Section 4.1), which must have already been set as a parameter, encoded as seconds after 1970-01-01 00:00:00 UTC. Timestamp conditions MUST be evaluated in 64 bits, regardless of encoded CBOR size. suit-condition-use-before is OPTIONAL to implement.

5.2. suit-condition-image-not-match

Verify that the current component does not match the suit-parameter-image-digest (Section 8.4.8.6 of [I-D.ietf-suit-manifest]). If no digest is specified, the condition fails. suit-condition-image-not-match is OPTIONAL to implement.

5.3. suit-condition-minimum-battery

suit-condition-minimum-battery provides a mechanism to test a Recipient’s battery level before installing an update. This condition is primarily for use in primary-cell applications, where the battery is only ever discharged. For batteries that are charged, suit-directive-wait 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. suit-condition-minimum-battery consumes suit-parameter-minimum-battery (Section 4.2).

5.4. suit-condition-update-authorized

Request authorization from the application and fail if not authorized. This can allow a user to decline an update. suit-parameter-update-priority (Section 4.3) provides an integer priority level that the application can use to determine whether or not to authorize the update. Priorities are application defined. suit-condition-update-authorized is OPTIONAL to implement.

5.5. suit-condition-version

suit-condition-version allows comparing versions of firmware. Verifying image digests is preferred to version checks because digests are more precise. suit-condition-version examines a component’s version against the version info specified in suit-parameter-version (Section 4.4).

5.6. suit-directive-wait

suit-directive-wait directs the manifest processor to pause until a specified event occurs. Some possible events include:

1. Authorization
2. External power
3. Network availability
4. Other device firmware version

5. Time

6. Time of day

7. Day of week

5.7. suit-directive-override-multiple

This directive enables setting parameters for multiple components at the same time. This allows a small reduction in encoding overhead:

* without override-multiple, the encoding for each component consists of:
  - set-component-index (2 bytes)
  - override-parameters (1 byte + parameter map)
* with override-multiple, the encoding for each component consists of:
  - the component index key (1 byte)
  - the parameter map

Override-multiple requires the command (1-2 bytes) and one additional map to hold the parameter sets (1 byte). For one component, there is no savings. For multiple components, there is an encoding savings of 2 bytes per component.

Proper structuring of code should ensure that override-multiple follows a code-path nearly identical to set-component-index + override-parameters.

This command is purely an encoding alias for set-component-index and override-parameters. The component index is set to the last component listed in the override-multiple argument when override-multiple completes.

The following CDDL defines the argument for suit-directive-override-multiple:

```
CDDL SUIT_Override_Mult_Arg = { uint => {+ $$SUIT_Parameters} }
```

5.8. suit-directive-copy-params

suit-directive-copy-params enables a manifest author to specify one or more components to copy parameters from, and a list of parameters to copy from each specified source component.

The behaviour is exactly the same as override parameters, but with parameter values defined in existing components. Parameters are only copied between identical keys (no copying from URI to digest, for example).

For each entry in the map, the manifest processor sets the source component to be the component identified by the index contained in the map key. For each parameter identified in the copy list, the manifest processor copies the parameter from the source component to the current component.

The following CDDL defines the argument for suit-directive-copy-params:

```
CDDL SUIT_Directive_Copy_Params = { uint => [+ int] }
```

6. IANA Considerations

IANA is requested to:

* allocate key 14 in the SUIT Envelope registry for suit-coswid
* allocate key 14 in the SUIT Manifest registry for suit-coswid
* allocate key 7 in the SUIT Component Text registry for suit-text-version-required
* allocate the commands and parameters as shown in the following tables

6.1. SUIT Commands

Label	Name	Reference
4	Use Before	Section 5.1
25	Image Not Match	Section 5.2
26	Minimum Battery	Section 5.3
27	Update Authorized	Section 5.4

28	Version	Section 5.5
29	Wait For Event	Section 5.6
34	Override Multiple	Section 5.7
35	Copy Params	Section 5.8

Table 4

6.2. SUIT Parameters

Label	Name	Reference
4	Use Before	Section 4.1
26	Minimum Battery	Section 4.2
27	Update Priority	Section 4.3
28	Version	Section 4.4
29	Wait Info	Section 4.5

Table 5

7. Security Considerations

This document extends the SUIT manifest specification. A detailed security treatment can be found in the architecture [RFC9019] and in the information model [I-D.ietf-suit-information-model] documents.

8. References

8.1. Normative References

[I-D.ietf-suit-manifest]
Moran, B., Tschofenig, H., Birkholz, H., Zandberg, K., and O. R淡nningstad, "A Concise Binary Object Representation (CBOR)-based Serialization Format for the Software Updates for Internet of Things (SUIT) Manifest", Work in Progress, Internet-Draft, draft-ietf-suit-manifest-33, 24 February 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-suit-manifest-33>.

[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/rfc/rfc2119>.

[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/rfc/rfc8174>.

[RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/rfc/rfc8949>.

[RFC9393] Birkholz, H., Fitzgerald-McKay, J., Schmidt, C., and D. Waltermire, "Concise Software Identification Tags", RFC 9393, DOI 10.17487/RFC9393, June 2023, <https://www.rfc-editor.org/rfc/rfc9393>.

[semver] "Semantic Versioning 2.0.0", 18 June 2013, <https://semver.org>.

8.2. Informative References

[I-D.ietf-suit-information-model] Moran, B., Tschofenig, H., and H. Birkholz, "A Manifest Information Model for Firmware Updates in Internet of Things (IoT) Devices", Work in Progress, Internet-Draft, draft-ietf-suit-information-model-13, 8 July 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-suit-information-model-13>.

[RFC9019] Moran, B., Tschofenig, H., Brown, D., and M. Meriac, "A Firmware Update Architecture for Internet of Things", RFC 9019, DOI 10.17487/RFC9019, April 2021, <https://www.rfc-editor.org/rfc/rfc9019>.

Appendix A. Full CDDL

To be valid, the following CDDL MUST be appended to the SUIT Manifest CDDL. The SUIT CDDL is defined in Appendix A of [I-D.ietf-suit-manifest].

```
   $$unseverable-manifest-member-extensions //= (
       suit-current-version => \
           bstr .cbor SUIT_Condition_Version_Comparison_Value
   )
   $$SUIT_severable-members-extensions //= (
       suit-coswid => bstr)
   ;    suit-coswid => bstr .cbor concise-swid-tag)

   $$severable-manifest-members-choice-extensions //= (
       suit-coswid => bstr .cbor SUIT_Command_Sequence / SUIT_Digest
   )

   SUIT_Condition //= (
       suit-condition-image-not-match,   SUIT_Rep_Policy)
   SUIT_Condition //= (
       suit-condition-use-before,        SUIT_Rep_Policy)
   SUIT_Condition //= (
       suit-condition-minimum-battery,   SUIT_Rep_Policy)
   SUIT_Condition //= (
       suit-condition-update-authorized, SUIT_Rep_Policy)
   SUIT_Condition //= (
       suit-condition-version,           SUIT_Rep_Policy)

   SUIT_Directive //= (
       suit-directive-wait,              SUIT_Rep_Policy)

   SUIT_Directive //= (
       suit-directive-override-multiple, SUIT_Override_Mult_Arg)
   SUIT_Directive //=(
       suit-directive-copy-params,       SUIT_Directive_Copy_Params)


   SUIT_Override_Mult_Arg = {
       + uint => {+ $$SUIT_Parameters}
   }
   SUIT_Directive_Copy_Params = {
       + uint => [+ int]
   }

   SUIT_Wait_Event = { + SUIT_Wait_Events }

   SUIT_Wait_Events //= (suit-wait-event-authorization => int)
   SUIT_Wait_Events //= (suit-wait-event-power => int)
   SUIT_Wait_Events //= (suit-wait-event-network => int)
   SUIT_Wait_Events //= (suit-wait-event-other-device-version
       => SUIT_Wait_Event_Argument_Other_Device_Version)
   SUIT_Wait_Events //= (suit-wait-event-time => uint); Timestamp
   SUIT_Wait_Events //= (suit-wait-event-time-of-day
```

```
        => uint); Time of Day (seconds since 00:00:00)
    SUIT_Wait_Events //= (suit-wait-event-day-of-week
        => uint); Days since Sunday

    SUIT_Wait_Event_Argument_Other_Device_Version = [
        other-device: bstr,
        other-device-version: [ + SUIT_Parameter_Version_Match ]
    ]

    SUIT_Parameters //= (suit-parameter-use-before => uint)
    SUIT_Parameters //= (suit-parameter-minimum-battery => uint)
    SUIT_Parameters //= (suit-parameter-update-priority => int)
    SUIT_Parameters //= (suit-parameter-version =>
        bstr .cbor SUIT_Parameter_Version_Match)
    SUIT_Parameters //= (suit-parameter-wait-info =>
        bstr .cbor SUIT_Wait_Event)
    SUIT_Parameters //= (suit-parameter-component-metadata =>
        bstr .cbor SUIT_Component_Metadata)

    SUIT_Parameter_Version_Match = [
        suit-condition-version-comparison-type:
            SUIT_Condition_Version_Comparison_Types,
        suit-condition-version-comparison-value:
            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_Component_Metadata = {
        ? suit-meta-default-permissions => SUIT_meta_permissions,
        ? suit-meta-user-permissions => SUIT_meta_permission_map,
```

```
        ? suit-meta-group-permissions => SUIT_meta_permission_map,
        ? suit-meta-role-permissions => SUIT_meta_permission_map,
        ? suit-meta-file-type => SUIT_Filetype,
        ? suit-meta-modification-time => CBOR_Datetime,
        ? suit-meta-creation-time => CBOR_Datetime,
        ? suit-meta-creator => SUIT_meta_actor_id,
        * $$SUIT_Component_Metadata_Extensions
    }

    SUIT_meta_permissions = uint .bits SUIT_meta_permission_bits
    SUIT_meta_permission_bits = &(
        write_attr_ex: 13,
        read_attr_ex: 12,
        sync: 11,
        delete: 10,
        recurse_delete: 9,
        write_attr: 8,
        change_owner: 7,
        change_perm: 6,
        read_perm: 5,
        read_attr: 4,
        creatdir_append: 3,
        list_read: 2,
        create_write: 1,
        traverse_exec: 0,
        * $$SUIT_meta_permission_bits_extensions
    )

    SUIT_meta_permission_map = {
        + SUIT_meta_actor_id => SUIT_meta_permissions
    }

    SUIT_meta_actor_id = UUID_Tagged / bstr / str / int
    UUID_Tagged = #6.37(bstr)



    $$suit-text-component-key-extensions //= (
        suit-text-version-required => tstr)
    $$suit-text-component-key-extensions //= (
        suit-text-current-version => tstr)

    suit-set-version = 6
    suit-coswid = 14
    suit-condition-use-before        = 4
    suit-condition-image-not-match          = 25
    suit-condition-minimum-battery          = 26
    suit-condition-update-authorized        = 27
```

```
   suit-condition-version                  = 28

   suit-directive-wait                     = 29
   suit-directive-override-multiple        = 34
   suit-directive-copy-params              = 35

   suit-wait-event-authorization        = 1
   suit-wait-event-power                = 2
   suit-wait-event-network              = 3
   suit-wait-event-other-device-version = 4
   suit-wait-event-time                 = 5
   suit-wait-event-time-of-day          = 6
   suit-wait-event-day-of-week          = 7

   suit-parameter-use-before        = 4
   suit-parameter-minimum-battery   = 26
   suit-parameter-update-priority   = 27
   suit-parameter-version           = 28
   suit-parameter-wait-info         = 29

   suit-text-version-required      = 7
   suit-text-current-version       = 8
```

Authors' Addresses

Brendan Moran
Arm Limited
Email: Brendan.Moran.ietf@gmail.com

Ken Takayama
SECOM CO., LTD.
Email: ken.takayama.ietf@gmail.com