Stream:	Internet Engineering Task Force (IETF)
RFC:	10013
Category:	Standards Track
Published:	July 2026
ISSN:	2070-1721
Authors:	S. Frost	T. Fossati	H. Tschofenig	H. Birkholz
---	---	---	---	---
	Arm	*Linaro*	*UniBw M.*	*Fraunhofer SIT*

RFC 10013
Entity Attestation Token (EAT) Measured Component

Abstract

The term "measured component" refers to an object within the attester's target environment whose state can be sampled and typically digested using a cryptographic hash function. Examples of measured components include firmware stored in flash memory, software loaded into memory at start time, data stored in a file system, or values in a CPU register. This document provides the information model for the measured component and two associated data models. This separation is intentional: The JSON and Concise Binary Object Representation (CBOR) serializations, coupled with the media types and associated Constrained Application Protocol (CoAP) Content-Formats, enable the immediate use of the semantics within the Entity Attestation Token (EAT) framework. Meanwhile, the information model can be reused in future specifications to provide additional serializations, for example, using ASN.1.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc10013.

Table of Contents

1. Introduction

Section 4.2.16 of [RFC9711] defines a Measurement claim that:

> [c]ontains descriptions, lists, evidence, or measurements of the software that exists on the entity or on any other measurable subsystem of the entity

This claim allows for different measurement formats, each identified by a different CoAP Content-Format (Section 12.3 of [RFC7252]). Currently, the only specified format is Concise Software Identification (CoSWID) Tags of type "evidence", as per Section 2.9.4 of [RFC9393]. However, CoSWID is not suitable for measurements that cannot be anchored to a file system, such as those in early boot environments. To address this gap, this document introduces a measured component format that can be used with the EAT Measurement claim alongside or instead of CoSWID.

The term "measured component" refers to an object within the attester's target environment whose state can be sampled and typically digested using a cryptographic hash function. For example, this includes the invariant part of a firmware component that is loaded in memory at startup time, a Run-Time Integrity Check (RTIC), a file system object, or a CPU register.

This document provides the information model for the measured component and two associated data models [RFC3444]. This separation is intentional: The JSON and CBOR serializations, coupled with the media types and associated CoAP Content-Formats, enable the immediate use of the semantics within the EAT framework. Meanwhile, the information model can be reused in future specifications to provide additional serializations, for example, using ASN.1. This approach is consistent with the guidance in Section 5.2 of [OPS-MGMT].

2. Conventions and Definitions

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.

In this document, Concise Data Definition Language (CDDL) [RFC8610] [RFC9165] [RFC9741] is used to describe the data formats. This specification uses the following CDDL control operators: `.b64u` defined in Section 2.1 of [RFC9741], `.json` defined in Section 2.4 of [RFC9741], and `.cbor` defined in Section 3.8.4 of [RFC8610].

Examples are folded following the conventions in [RFC8792].

3. Information Model

This section presents the information model of a measured component.

A measured component Information Element (IE) includes the component's sampled state (in digested or raw form) along with metadata that helps in identifying the component. Optionally, any entities responsible for signing the installed component can also be specified.

The IEs that constitute a measured component are described in Table 1.

IE	Description	Requirement Level
Component Name	The name given to the measured component. It is important that this name remains consistent across different releases to allow for better tracking of the same measured item across updates. When combined with a consistent versioning scheme, it enables better signaling from the appraisal procedure to the relying parties.	**REQUIRED**
Component Version	A value representing the specific release or development version of the measured component. Using Semantic Versioning [SEMVER] is **RECOMMENDED**.	**OPTIONAL**
Digested or Raw Value	Either the raw value or the digested value of the measured component.	**REQUIRED**
Digest Algorithm	Hash algorithm used to compute the Digest Value.	**REQUIRED** only if the value is in the digested form
Authorities	One or more entities that can authoritatively identify the component being measured.	**OPTIONAL**

Table 1: Measured Component Information Elements

A data model implementing this information model **SHOULD** allow a limited amount of extensibility to accommodate profile-specific semantics.

4. Data Models

This section presents coordinated JSON and CBOR data models, each of which implements the information model outlined in Section 3.

The data model is inspired by the "PSA software component" claim (Section 4.4.1 of [RFC9783]), which has been refactored to take into account the recommendations about the design of new EAT claims described in Appendix E of [RFC9711].

CDDL is used to express rules and constraints of the data model for both JSON and CBOR. These rules must be strictly followed when creating or validating measured component data items. When there is variation between CBOR and JSON, the CDDL generic `JC<>`, defined in Appendix D of [RFC9711], is used.

4.1. Common Types

The following three basic types are used at various places within the measured component data model:

```
bytes-b64u = text .b64u bytes
bytes8 = bytes .size 8
bytes8-b64u = text .b64u bytes8
```

4.2. The `digest` Type

A digest represents the result of a hashing operation together with the hash algorithm used. The type of the digest algorithm identifier can be either `int` or `text` and is interpreted according to the "Named Information Hash Algorithm Registry" IANA registry [IANA.named-information]. Specifically, `int` values are matched against "ID" entries and `text` values are matched against "Hash Name String" entries. Whenever possible, using the `int` encoding is **RECOMMENDED**.

```
digest = [
  alg: (int / text)
  val: digest-value-type
]

digest-value-type = eat.JC<bytes-b64u, bytes>
```

4.3. The `measured-component` Data Item

The `measured-component` data item is as follows:

```
measured-component = {
  component-id-label => component-id
  measurement
  ? authorities-label => [ + authority-id-type ]
  ? flags-label => flags-type
}

measurement //= ( digested-measurement-label => digest )
measurement //= ( raw-measurement-label => bytes )

authority-id-type = eat.JC<bytes-b64u, bytes>
flags-type = eat.JC<bytes8-b64u, bytes8>

component-id-label = eat.JC<"id", 1>
digested-measurement-label = eat.JC<"digested-measurement", 2>
raw-measurement-label = eat.JC<"raw-measurement", 5>
authorities-label = eat.JC<"authorities", 3>
flags-label = eat.JC<"flags", 4>
```

The members of the `measured-component` CBOR map / JSON object are:

`id` (index 1):
: The measured component identifier encoded according to the format described in Section 4.3.1.

`measurements`:
: Either a digest value and digest algorithm (index 2), encoded using the digest format (Section 4.2), or the "raw" measurement (index 5), encoded as a byte string. Note that, while the size of the digested form is constrained by the digest function, the size of the raw form can vary greatly depending on what is being measured (it could be a CPU register or an entire configuration blob, for example). Therefore, a decoder implementation may decide to limit the amount of memory it allocates to this specific field.

`authorities` (index 3):
: One or more authorities, see Section 4.3.2.

`flags` (index 4):
: A 64-bit field with profile-defined semantics, see Section 4.3.3.

4.3.1. Component Identifier

The `component-id` data item is as follows:

```
component-id = [
  name:      text
  ? version: version
]

;# import coswid.$version-scheme from rfc9393 as coswid

version = [
  val:      text
  ? scheme: coswid.$version-scheme
]
```

`name:`
: A string that provides a human-readable identifier for the component in question. Format and adopted conventions depend on the component type.

`version:`
: A compound `version` data item that reuses the encoding and semantics of `sw-version-type` from [RFC9711], extending it to non-software components.

> Note that the complete definition of `sw-version-type` depends on the `$version-scheme` CDDL socket defined in Section 2.2 of [RFC9393].

4.3.2. Authority Identifier

An authority is an entity that can authoritatively identify a given component by digitally signing it. This signature is typically verified during installation (Section 7 of [RFC9019]) or when the measured component is executed by the boot firmware, operating system, or application launcher, as in the case of Unified Extensible Firmware Interface (UEFI) Secure Boot [UEFI2] and Arm Trusted Board Boot [TBBR-CLIENT]. Another example may be the controlling entity in an app store. Note that this signature is in no way related to the attester's signature on the EAT-formatted evidence. By extension, an authority identifier does not, by itself, indicate the signer of the enclosing EAT-formatted evidence.

An authority is identified by its signing public key. It could be an X.509 certificate, a raw public key, a public key thumbprint, or some other identifier that can be uniquely associated with the signing entity. In some cases, multiple parties may need to sign a component to indicate their endorsement or approval. This could include roles such as a firmware update system, fleet owner, or third-party auditor. The specific purpose of each signature may depend on the deployment, and the order of authorities within the array could indicate meaning.

If an EAT profile (Section 6 of [RFC9711]) uses measured components, it **MUST** specify whether the `authorities` field is used. If it is used, the profile **MUST** also specify what each of the entries in the `authorities` array represents and how to interpret the corresponding `authority-id-type`.

The `authority-id-type` is defined as follows:

```
authority-id-type = eat.JC<bytes-b64u, bytes>
```

4.3.3. Profile-Specific Flags

This optional field can contain up to 64 bits of profile-defined semantics, enabling a profile of this specification to encode additional information and extend the base type. It can be used to carry information in fixed-size chunks, such as a bit mask or a single value within a predetermined set of codepoints. Regardless of its internal structure, the size of this field is exactly 8 bytes.

The `flags-type` is defined as follows:

```
flags-type = eat.JC<bytes8-b64u, bytes8>
```

If an EAT profile (Section 6 of [RFC9711]) uses measured components, it **MUST** specify whether the `flags` field is used. If it is used, the profile **MUST** also specify how to interpret the 64 bits.

4.4. EAT `measurements-format` Extensions

The CDDL in Figure 1 extends the `$measurements-body-cbor` and `$measurements-body-json` EAT sockets to add support for `measured-components` to the Measurement claim.

```
mc-cbor = bytes .cbor measured-component
mc-json = text .json measured-component

; EAT CBOR (`.feature "cbor"`)
$measurements-body-cbor /= mc-cbor ; homogeneous
$measurements-body-cbor /= mc-json ; tunnel

; EAT JSON (`.feature "json"`)
$measurements-body-json /= mc-json             ; homogeneous
$measurements-body-json /= text .b64u mc-cbor ; tunnel
```

Figure 1: EAT `measurements-format` Extensions

Each socket is extended with two new types: a "homogeneous" representation that is used when `measured-component` and the EAT have the same serialization (e.g., they are both CBOR) and a "tunnel" representation that is used when the serializations differ.

4.4.1. `measurements-format` for CBOR EAT

The entries in Table 2 are the allowed `content-type`/`content-format` pairs when the `measured-component` is carried in a CBOR EAT.

Note the use of the "homogeneous" and "tunnel" formats from Figure 1 and how the associated CoAP Content-Format is used to describe the original serialization.

content-type (CoAP C-F equivalent)	content-format
application/measured-component+cbor	mc-cbor
application/measured-component+json	mc-json

Table 2: `measurements-format` for EAT CBOR Web Token (CWT)

4.4.2. `measurements-format` for JSON EAT

Table 3 is the equivalent of Table 2 for JSON-serialized EAT.

content-type (CoAP C-F equivalent)	content-format
application/measured-component+json	mc-json
application/measured-component+cbor	tstr .b64u mc-cbor

Table 3: `measurements-format` for EAT JSON Web Token (JWT)

4.5. EAT Profiles and Measured Components

The semantics of the `authorities` and profile `flags` fields are defined by the applicable EAT profile, i.e., the profile of the wrapping EAT.

If the profile of the EAT is not known to the consumer and one or more measured components within that EAT include `authorities` and/or profile `flags`, the consumer **MUST** reject the EAT.

4.6. Examples

The example in Figure 2 is a digested measured component with all the fields populated.

```
{
  / id / 1: [
    / name / "boot loader X",
    / version / [
      "1.2.3rc2",
      16384 / semver /
    ]
  ],
  / measurement / 2: [
    / alg / "sha-256",
    / val / h'3996003d486fb91ffb056f7d03f2b2992b215b31dbe7af4b37
              3431fc7d319da3'
  ],
  / authorities / 3: [
    h'492e9b676c21f6012b1ceeb9032feb4141a880797355f6675015ec59c5
      1ca1ec',
    h'4277bb97ba7b51577a0d38151d3e08b40bdf946753f5b5bdeb814d6ff5
      7a8a5e'
  ],
  / flags / 4: h'0000000000000101'
}
```

Figure 2: Complete Measured Component

The example depicted in Figure 3 is the same measured component as above but used as the format of a Measurement claim in an EAT claims-set.

This example uses 295 as the `content-type` value of the `measurements-format` entry.

Note that the array contains only one measured component, but additional entries could be added if the measured Trusted Computing Base (TCB) is made of multiple individually measured components.

```
{
  273: [
    [
      295, / measured-component+cbor /
      <<
        {
          / id / 1: [
            / name / "boot loader X",
            / version / [
              "1.2.3rc2",
              16384 / semver /
            ]
          ],
          / measurement / 2: [
            / alg / "sha-256",
            / val / h'3996003d486fb91ffb056f7d03f2b2992b215b31db
                      e7af4b373431fc7d319da3'
          ],
          / authorities / 3: [
            h'492e9b676c21f6012b1ceeb9032feb4141a880797355f66750
              15ec59c51ca1ec',
            h'4277bb97ba7b51577a0d38151d3e08b40bdf946753f5b5bdeb
              814d6ff57a8a5e'
          ]
        }
      >>
    ]
  ]
}
```

Figure 3: EAT Measurements Claim Using a Measured Component (CBOR)

The example in Figure 4 illustrates the inclusion of a JSON measured component inside a JSON EAT.

This example uses 296 as the `content-type` value of the `measurements-format` entry.

```
=============== NOTE: '\' line wrapping per RFC 8792 ================

{
  "measurements": [
    [
      296,
      "{ \"id\": [ \"boot loader X\", [ \"1.2.3rc2\", 16384 ] ], \"\
digested-measurement\": [ \"sha-256\", \"\
OZYAPUhvuR_7BW99A_KymSshWzHb569LNzQx_H0xnaM\" ], \"authorities\": [ \
\"SS6bZ2wh9gErHO65Ay_rQUGogHlzVfZnUBXsWcUcoew\", \"\
                  Qne7l7p7UVd6DTgVHT4ItAvflGdT9bW964FNb_V6il4\" ] }"
    ]
  ]
}
```

Figure 4: EAT Measurements Claim Using a Measured Component (JSON)

The example shown in Figure 5 is a measured component representing a boot loader identified by its path name:

```
{
  / id / 1: [
    / name / "/boot/loader.bin"
  ],
  / measurement / 2: [
    / alg / "sha-384",
    / val / h'66ec2fb4e02d8c8b3eee320e750d9389d66c52c51db11cc6
              9cc5e410816283ed60ba573795f5fcc85e513af57b3f6def'
  ],
  / flags / 4: h'0000000000000101'
}
```

Figure 5: Digested Measured Component Using a File Path as an Identifier

The example in Figure 6 is a raw measured component.

```
{
  / id / 1: [
    / name / "hardware-config"
  ],
  / measurement / 5: h'4f6d616861'
}
```

Figure 6: Raw Measured Component

5. Security Considerations

The considerations discussed in Sections 9.1 (Claim Trustworthiness), 9.4 (Multiple EAT Consumers), and 9.5 (Detached EAT Bundle Digest Security Considerations) of [RFC9711] apply to this document as well. Note that similar security considerations may apply when the measured component information model is serialized using different data models than the ones specified in this document.

The Component Name and Component Version can give an attacker detailed information about the software running on a device and its configuration settings. This information could offer an attacker valuable insight.

Any textual fields (e.g., Component Name and Component Version) that are stored in a file, inserted into a database, or displayed to humans must be properly sanitized to prevent attacks and undesirable behavior. Further discussion and references on this topic can be found in Section 7 of [RFC9839].

If the component measurement is digested, the digest must be computed using a strong cryptographic hash function.

6. Privacy Considerations

The differential encryption considerations discussed in Section 9.4 (Multiple EAT Consumers) of [RFC9711] also apply to this document.

The Component Name and Component Version may reveal private information about a device and its owner.

Additionally, the stability requirement of the Component Name may enable tracking.

7. IANA Considerations

7.1. Media Type Registrations

IANA has added the following media types to the "Media Types" registry [IANA.media-types].

Name	Template	Reference
`measured-component+cbor`	`application/measured-component+cbor`	RFC 10013
`measured-component+json`	`application/measured-component+json`	RFC 10013

Table 4: Measured Component Media Types

7.1.1. `application/measured-component+cbor`

Type name: application

Subtype name: measured-component+cbor

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: binary (CBOR)

Security considerations: Section 5 of RFC 10013

Interoperability considerations: N/A

Published specification: RFC 10013

Applications that use this media type: Attesters, Verifiers, and Relying Parties

Fragment identifier considerations: The syntax and semantics of fragment identifiers are as specified for "application/cbor". (No fragment identification syntax is currently defined for "application/cbor".)

Person & email address to contact for further information: RATS WG mailing list (rats@ietf.org)

Intended usage: COMMON

Restrictions on usage: none

Author/Change controller: IETF

7.1.2. `application/measured-component+json`

Type name: application

Subtype name: measured-component+json

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: binary (JSON is UTF-8-encoded text)

Security considerations: Section 5 of RFC 10013

Interoperability considerations: N/A

Published specification: RFC 10013

Applications that use this media type: Attesters, Verifiers, and Relying Parties

Fragment identifier considerations: The syntax and semantics of fragment identifiers are as specified for "application/json". (No fragment identification syntax is currently defined for "application/json".)

Person & email address to contact for further information: RATS WG mailing list (rats@ietf.org)

Intended usage: COMMON

Restrictions on usage: none

Author/Change controller: IETF

7.2. Measured Component Content-Format Registrations

IANA has registered these two Content-Format numbers in the "CoAP Content-Formats" registry within the "Constrained RESTful Environments (CoRE) Parameters" registry group [IANA.core-parameters] as follows:

Content Type	Content Coding	ID	Reference
application/measured-component+cbor	-	295	RFC 10013
application/measured-component+json	-	296	RFC 10013

Table 5: Content-Format Number Registrations

8. References

8.1. Normative References

[IANA.named-information] IANA, "Named Information Hash Algorithm Registry", <https://www.iana.org/assignments/named-information>.

[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>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <https://www.rfc-editor.org/info/rfc7252>.

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

[RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <https://www.rfc-editor.org/info/rfc8610>.

[RFC8792] Watsen, K., Auerswald, E., Farrel, A., and Q. Wu, "Handling Long Lines in Content of Internet-Drafts and RFCs", RFC 8792, DOI 10.17487/RFC8792, June 2020, <https://www.rfc-editor.org/info/rfc8792>.

[RFC9165] Bormann, C., "Additional Control Operators for the Concise Data Definition Language (CDDL)", RFC 9165, DOI 10.17487/RFC9165, December 2021, <https://www.rfc-editor.org/info/rfc9165>.

[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/info/rfc9393>.

[RFC9711] Lundblade, L., Mandyam, G., O'Donoghue, J., and C. Wallace, "The Entity Attestation Token (EAT)", RFC 9711, DOI 10.17487/RFC9711, April 2025, <https://www.rfc-editor.org/info/rfc9711>.

[RFC9741] Bormann, C., "Concise Data Definition Language (CDDL): Additional Control Operators for the Conversion and Processing of Text", RFC 9741, DOI 10.17487/RFC9741, March 2025, <https://www.rfc-editor.org/info/rfc9741>.

[SEMVER] "Semantic Versioning 2.0.0", <https://semver.org/spec/v2.0.0.html>.

8.2. Informative References

[IANA.core-parameters] IANA, "Constrained RESTful Environments (CoRE) Parameters", <https://www.iana.org/assignments/core-parameters>.

[IANA.media-types] IANA, "Media Types", <https://www.iana.org/assignments/media-types>.

[OPS-MGMT] Claise, B., Clarke, J., Farrel, A., Barguil, S., Pignataro, C., and R. Chen, "Guidelines for Considering Operations and Management in IETF Specifications", Work in Progress, Internet-Draft, draft-ietf-opsawg-rfc5706bis-05, 26 June 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-opsawg-rfc5706bis-05>.

[RFC3444] Pras, A. and J. Schoenwaelder, "On the Difference between Information Models and Data Models", RFC 3444, DOI 10.17487/RFC3444, January 2003, <https://www.rfc-editor.org/info/rfc3444>.

[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/info/rfc9019>.

[RFC9783] Tschofenig, H., Frost, S., Brossard, M., Shaw, A., and T. Fossati, "Arm's Platform Security Architecture (PSA) Attestation Token", RFC 9783, DOI 10.17487/RFC9783, June 2025, <https://www.rfc-editor.org/info/rfc9783>.

[RFC9839] Bray, T. and P. Hoffman, "Unicode Character Repertoire Subsets", RFC 9839, DOI 10.17487/RFC9839, August 2025, <https://www.rfc-editor.org/info/rfc9839>.

[TBBR-CLIENT] Arm Ltd, "Trusted Board Boot Requirements Client (TBBR-CLIENT) Armv8-A", ARM DEN0006D, September 2018, <https://developer.arm.com/documentation/den0006>.

[UEFI2] UEFI Forum, Inc., "Unified Extensible Firmware Interface (UEFI) Specification", Release 2.10, August 2022, <https://uefi.org/sites/default/files/resources/UEFI_Spec_2_10_Aug29.pdf>.

Appendix A. Collated CDDL

This appendix contains all the CDDL definitions included in this specification.

```
=============== NOTE: '\' line wrapping per RFC 8792 ================

measured-component = {
  component-id-label => component-id,
  measurement,
  ? authorities-label => [+ authority-id-type],
  ? flags-label => flags-type,
}
measurement //= (digested-measurement-label => digest // raw-\
                                              measurement-label => bytes)
authority-id-type = eat.JC<bytes-b64u, bytes>
flags-type = eat.JC<bytes8-b64u, bytes8>
component-id = [
  name: text,
  ? version: version,
]
version = [
  val: text,
  ? scheme: coswid.$version-scheme,
]
digest = [
  alg: int / text,
  val: digest-value-type,
]
digest-value-type = eat.JC<bytes-b64u, bytes>
bytes-b64u = text .b64u bytes
bytes8 = bytes .size 8
bytes8-b64u = text .b64u bytes8
component-id-label = eat.JC<"id", 1>
digested-measurement-label = eat.JC<"digested-measurement", 2>
raw-measurement-label = eat.JC<"raw-measurement", 5>
authorities-label = eat.JC<"authorities", 3>
flags-label = eat.JC<"flags", 4>
mc-cbor = bytes .cbor measured-component
mc-json = text .json measured-component
$measurements-body-cbor /= mc-cbor / mc-json
$measurements-body-json /= mc-json / text .b64u mc-cbor
eat.JSON-ONLY<J> = J .feature "json"
eat.CBOR-ONLY<C> = C .feature "cbor"
eat.JC<J, C> = eat.JSON-ONLY<J> / eat.CBOR-ONLY<C>
coswid.$version-scheme /= coswid.multipartnumeric / coswid.\
multipartnumeric-suffix / coswid.alphanumeric / coswid.decimal / \
                                               coswid.semver / int / text
coswid.multipartnumeric = 1
coswid.multipartnumeric-suffix = 2
coswid.alphanumeric = 3
coswid.decimal = 4
coswid.semver = 16384
```

Acknowledgments

The authors would like to thank Carl Wallace, Carsten Bormann, Charles Nicas, Deb Cooley, Dionna Glaze, Esko Dijk, Giridhar Mandyam, Gorry Fairhurst, Henry Thompson, Houda Labiod, Ionuț Mihalcea, Joe Salowey, Jun Zhang, Laurence Lundblade, Mahesh Jethanandani, Michael Richardson, Mohamed Boucadair, Muhammad Usama Sardar, and Yogesh Deshpande for providing comments, reviews, and suggestions that greatly improved this document.

The authors would also like to thank Ken Takayama for providing an implementation of this specification in the veraison/eat package.

Authors' Addresses

Simon Frost
Arm
Email: Simon.Frost@arm.com

Thomas Fossati
Linaro
Email: Thomas.Fossati@linaro.org

Hannes Tschofenig
University of the Bundeswehr Munich
Institute of Distributed Intelligent Systems
Werner-Heisenberg-Weg 39
85577 Neubiberg
Germany
Email: Hannes.Tschofenig@gmx.net

Henk Birkholz
Fraunhofer SIT
Email: henk.birkholz@ietf.contact