Delay-Tolerant Networking
Internet-Draft
Intended status: Standards Track
Expires: 2 July 2026

B. Sipos
JHU/APL

29 December 2025

Bundle Protocol (BP) Manifest Block
draft-sipos-dtn-manifest-block-00

Abstract

This document defines an extension block type for Bundle Protocol version 7 to capture discretionary summary data about the state of a Bundle at the time the extension block is added. This Manifest structure is a general purpose container for information about other blocks, and can be used as a piece-part of a larger security operation.

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 2 July 2026.

Table of Contents

1. Introduction

The structure of data in Bundle Protocol (BP) version 7 [RFC9171] is to divide each independent Bundle into a sequence of blocks. The first "primary" block is mandatory and immutable during the Bundle lifetime, the last "payload" block is mandatory and contains the application data unit (ADU) being transported, and all other blocks in between them are "extension" blocks of various types and orders as needed by the BP network. Each non-primary "canonical" block (Section 4.3.2 of [RFC9171]) of a Bundle contains a block type code point, a unique-to-the-bundle block number, and block-type-specific data (BTSD) containing a byte string encoded and interpreted according to the associated block type.

While the primary and payload blocks are mandatory, each extension block instance (not type) will have a lifecycle controlled by policy and configuration on each node along its path from source to destination (including those two end nodes). Any extension block present at some point along that path will have been created along with or after the Bundle and will be discarded along with or before the Bundle. Some of the extension blocks provide security operations, for integrity or confidentiality, in accordance with Bundle Protocol Security (BPSec) [RFC9172].

The purpose of the Manifest block type specified in this document is to allow a node to capture some or all of the state of a Bundle as that Bundle transits the node (either as the source or an intermediate forwarding node). Each manifest includes metadata about when and why it was created as a way for later users of the block to filter Manifests based on policy. One initial use case for a Manifest, defined in Section 4.2, is to optimize how a security operation targeting one block can be bound to the state of other blocks at the time the operation is added to a Bundle.

1.1. Scope

This document describes the encoding and interpretation of the BTSD content for the Manifest block type, and its initial reason codes and use cases. This document does not address:

* Other potential Manifest reason codes, which can be defined in other specifications and either registered with IANA or use codes from the private use block.

* Policies or mechanisms for when and how other extension blocks are added to, modified in, or discarded from a Bundle.

* Policies about when and how BPSec security operations targeting or covering a Manifest block are added to, modified in, or discarded from a Bundle.

1.2. Use of CDDL

This document defines CBOR structure using the Concise Data Definition Language (CDDL) [RFC8610]. The entire CDDL structure can be extracted from the XML version of this document using the XPath expression:

'//sourcecode[@type="cddl"]'

The following initial fragment defines the top-level symbols of this document's CDDL.

```
start = ext-data-manifest
```

The definitions for the rules dtn-time, and extension-block, block-control-flags, the socket $extension-block, and the generic rule extension-block-use are taken from BP [RFC9171].

1.3. 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.

2. Manifest Block Content

The BTSD content for the Manifest block type consists of a CBOR sequence [RFC8742] of a metadata map followed by an array of block-derived data maps. Each other block referenced by a Manifest is referred to here as "covered" by that Manifest.

Items in the metadata map SHALL be interpreted according to the "Manifest Metadata Items" IANA registry defined in Section 6.2. Its initial contents are defined in Section 2.1 in text and CDDL.

The coverage of blocks by a single Manifest is discretionary to the node creating the Manifest (see Section 3.1). A single Manifest does not not need to cover all blocks present at the time of its creation, and which blocks are covered will depend on the reason (Section 2.1.1) why the Manifest was created. The creator of a Manifest block MAY cover any other block present in the Bundle. A verifier of a Manifest block SHALL NOT assume or rely on a Manifest to cover any or all other blocks in the Bundle.

Items in each block data map SHALL be interpreted according to the "Manifest Data Items" IANA registry defined in Section 6.2. Its initial contents are defined in Section 2.2 in text and CDDL.

A block covered by a Manifest MAY be the target of a BPSec confidentiality operation before or after creation of the Manifest. It is a deployment and usage consideration to ensure that policy controlling Manifest lifecycles is consistent with policy controlling confidentiality lifecycles.

A block covered by a Manifest MAY be discarded at any time after creation of the Manifest. One potential future reason for a Manifest would be to record when specific blocks _are_ discarded by a node.

```
; Manifest is encoded in BTSD content as a sequence
; EDITOR NOTE: block type 255 is a placeholder for IANA assignment
$extension-block /=
  extension-block-use<255, (bstr .cborseq ext-data-manifest / bstr)>
ext-data-manifest = [
  metadata-map,
  [* blockdata-map]
]

; General structure of all maps in the manifest
manifest-structure = {
  * int16 => any
}
int16 = -32768 .. 32767

; Metadata item socket
metadata-map = {* $$metadata-item } .within manifest-structure
; Block data item socket
blockdata-map = {* $$blockdata-item } .within manifest-structure
```

Figure 1: Manifest BTSD Content CDDL

2.1. Well-known Metadata Items

The metadata items, and their keys, defined in this section apply only to the first map within the Manifest BTSD content. These items relate to the creation of the Manifest itself and not to any individual blocks covered by the Manifest.

2.1.1. Reason

This item is used to convey the reason code for why the Manifest was created, the audience for its verification, and how it is intended to be used. This item SHALL use map key 0 and value of an int16 interpreted according to the "Manifest Reason Codes" IANA registry defined in Section 6.2.

```
$$metadata-item //= (
  0: int16
)
```

Figure 2: Reason CDDL

2.1.2. Manifest Source

This item is used to represent the source node of the enclosing Manifest block. This item SHALL use map key 2 and value of a bstr-wrapped EID value. This enveloping is in accordance with recommendations in Section 4 of [I-D.ietf-dtn-eid-pattern].

```
$$metadata-item //= (
  2: embed-eid-structure
)
; TODO: copied from [draft-ietf-dtn-eid-pattern] but should be reference
embed-eid-structure = bstr .cbor eid-structure
```

Figure 3: Manifest Source CDDL

2.1.3. Timestamp

The item is used to represent the absolute time of the local timekeeper on the source of the Manifest block when it was created. This item SHALL use map key 3 and value of a dtn-time in accordance with Section 4.2.6 of [RFC9171].

```
$$metadata-item //= (
  3: dtn-time
)
```

Figure 4: Timestamp CDDL

2.2. Common Block Items

The block data items, and their keys, defined in this section apply to all block types equivalently and can be present for any associated block type. These items relate to the BPv7 canonical block structure as defined in Section 4.3.2 of [RFC9171] without concern for the specific block type or content of the BTSD.

2.2.1. Block Identity

This item is used to reference a specific other canonical block in the same Bundle as the Manifest. All other items in the same block data map are about the block referenced by this item. This item SHALL use map key 1 and value of an array containing two items: the block type of the other block as a uint and the block number of the other block as a uint. This array has the same order as the first two items of the canonical block array encoding.

```
$$blockdata-item //= (
  1: block-id
)
block-id = [
  ; From the IANA Bundle Block Types registry
  block-type: uint,
  block-num: uint,
]
```

Figure 5: Block Identity CDDL

A prerequisite to verifying any of the block data items involves first looking up the block referenced from the block number of this item. Verification of this item SHALL consist of matching the block type of the item against block type of the referenced block.

2.2.2. Block Flags

This item is used to convey the "Block processing control flags" value at the time the Manifest is created. This item SHALL use map key 2 and value of a uint representing the flags.

```
$$blockdata-item //= (
  2: block-control-flags
)
```

Figure 6: Block Flags CDDL

Verification of this item SHALL consist of matching the item value against the Block processing control flags of the referenced block.

2.2.3. BTSD Length

This item is used to convey the length of the BTSD content (_i.e._, the argument of the CBOR head of the BTSD item) at the time the Manifest is created. This item SHALL use map key 5 and value of a uint representing a length in bytes.

```
$$blockdata-item //= (
  5: btsd-len
)
btsd-len = uint
```

Figure 7: BTSD Length CDDL

Verification of this item SHALL consist of matching the item value against the length of the BTSD of the referenced block.

2.2.4. BTSD Hash

This item is used to convey a concise summary of BTSD content (_i.e._, the byte string within the BTSD item) without revealing the content itself. This item SHALL use map key 6 and value of an array containing a sequence of pairs of items. Each pair SHALL consist of a "COSE Algorithms" value [IANA-COSE] representing the hash algorithm used, and the hash as a bstr. The input to the hash SHALL be the content of the BTSD of the referenced block, excluding any CBOR head associated with the BTSD item itself.

```
$$blockdata-item //= (
  ; One or more pair of hash outputs
  6: [+ btsd-hash]
)
btsd-hash = (
  ; From the IANA COSE Algorithms registry
  alg: tstr / int,
  value: bstr
)
```

Figure 8: BTSD Hash CDDL

Verification of this item SHALL consist of matching at least one of the BTSD Hash values included in the Manifest against a locally computed hash of the content of the BTSD of the referenced block.

It is expected that truncated hash algorithms (_e.g._, SHA-256/64) can be used to trade hash output size for security strength. The choice of which hash algorithm(s) to use for any covered block is an implementation decision (and discussed in Section 5.2).

2.3. Security Block Items

The block items, and their keys, defined in this section SHALL apply only to block types for Block Integrity (11) and Block Confidentiality (12). These items relate to the abstract security block (ASB) structure as defined in Section 3.6 of [RFC9172] within its BTSD content.

2.3.1. Security Targets

This item is used to convey the target block numbers present in the security block at the time the Manifest is created. This item SHALL use map key -1 and value of an array of uint representing block numbers. The order of the target block numbers in this item SHOULD be the same as the original security block.

```
$$blockdata-item //= (
  -1: bpsec-targets
)
bpsec-targets = [+ uint ]
```

Figure 9: Security Targets CDDL

Verification of this item SHALL consist of matching the targets array in this item against the ASB field of the referenced block. It is not relevant to this verification as to whether or not the referenced target blocks themselves are present.

2.3.2. Security Context ID

This item is used to convey the Context ID field at the time the Manifest is created. This item SHALL use map key -2 and value of an int16 representing one of the "BPSec Security Context Identifiers" [IANA-BUNDLE] from the security block.

```
$$blockdata-item //= (
  -2: int16
)
```

Figure 10: Security Context CDDL

Verification of this item SHALL consist of matching the value of this item against the ASB field of the referenced block. It is not relevant to this verification as to whether or not the security context is known to the verifying node.

3. Manifest Processing

There are two processing activities associated with the Manifest block type: creating the manifest and verifying it. Those are defined in the following subsections.

A Manifest is intended to be a snapshot of part of the Bundle state at its time of creation. There is no defined concept of a Manifest being manipulated by an intermediate node; any manipulation would be the result of bad configuration or an on-path attack (Section 5.4).

3.1. Creation

The decision of when and why a Manifest is created and added to a Bundle is a local policy decision on a node.

The Reason item SHALL be present in all metadata maps. This metadata item is needed by other nodes to filter Manifest blocks and to interpret their semantics. The definition of a reason code can refine and require specific metadata or block data items to be present (see Section 4).

The selection of which blocks to cover by a Manifest is also a local policy decision and likely depends on the Reason code. Each covered block will correspond with a separate block data map.

The Block Identity item SHALL be present in each block data map. All of the other items in the block data map depend on its block number to create and verify, and some (specifically negative keys) depend on its block type.

3.2. Verification

The general Manifest verification procedure checks items in each block data map against the actual block data. Because blocks can be discarded after a Manifest has been created, one possible outcome for an individual block verification is that the block is missing; a missing covered block is a valid outcome and does not affect the verification of other blocks referenced by the same Manifest. It is a policy decision for how to act upon the outcome of verification, including cases of missing covered blocks.

The verification of a Manifest SHALL include checking whether the Manifest covers a minimum set of blocks, and if each block data map contains a minimum set of items. It is a matter of local policy on a verifier (within constraints for the associated Reason) to determine the minimum acceptable coverage for any particular Manifest.

The verification of a Manifest SHALL include verification of each item contained in each block data map based on the individual item definitions. It is an implementation matter to determine whether verification of a single block needs to stop on the first failure or continue to verify all items. The definition of a reason code can refine and require specific metadata or block data items to succeed in verification (see Section 4).

4. Initial Manifest Reasons

This section defines the semantics and constraints of manifest reason codes for the initial registry assignments in Table 3.

4.1. Preserve Extensions

When bundles are being forwarded through one or more transit networks on their path from source to destination, it is useful to be able to mark at ingress which extension blocks need to be preserved during the transit across those networks. This marking could be created by the edge nodes themselves or by a routing node on either side of the ingress hop.

The Manifest reason "Preserve Extensions" (code 2) SHALL be used to indicate that the Manifest block is intended to mark one or more blocks as needing to be kept in the Bundle until the Manifest itself is discarded.

When this reason is present, the block itself SHALL be marked with block processing control flags having bit 2 set, indicating "Delete bundle if block can’t be processed."

When this reason is present in the metadata map, each block data map SHALL contain a Block Identity (key 1). When this reason is present in the metadata map, each block data map MAY contain a BTSD Length (key 5) and/or BTSD Hash (key 6).

When verifying for this reason, if any block is missing or has a mismatch with the Manifest the entire verification SHALL be considered to have failed.

4.2. Security AAD Hashing

The definition of external additional authenticated data (AAD) in the BPSec COSE context [I-D.ietf-dtn-bpsec-cose] allows an arbitrary number of other blocks (both canonical block metadata and BTSD) to be bound to a security operation on a single target block. Using external AAD coverage allows a security verifier or acceptor to assert that (a chosen subset of) the state of those other blocks at

the time of verification matches their state at the time the operation was added to the Bundle. The use of direct BTSD-in-external-AAD is depicted in Figure 11.

A side effect of AAD directly covering the BTSD of other blocks is the need to serialize that BTSD as input to the COSE message processing, at the security source and for all verifiers and acceptors. For cases where the AAD covers large BTSD, including that of the payload block, this can impose a large data handling burden on the security processing entity (specifically its cryptographic element).

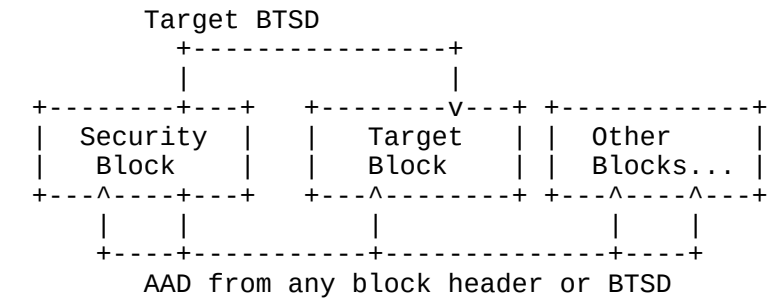

Figure 11: Security AAD Using Other Blocks Directly

A common way to mitigate this need for cryptographic processing of large data is distill the data using a hash algorithm and then use the hash value as input to the cryptographic function. This preimage hashing is available for the target of a security operation using a COSE Hash Envelope [I-D.ietf-cose-hash-envelope] but is not available for its external AAD. A straightforward use of a Manifest block, defined later in this section, is to capture the hash values of other blocks for external AAD coverage and thus reduce the size of that external AAD needed as cryptographic input.

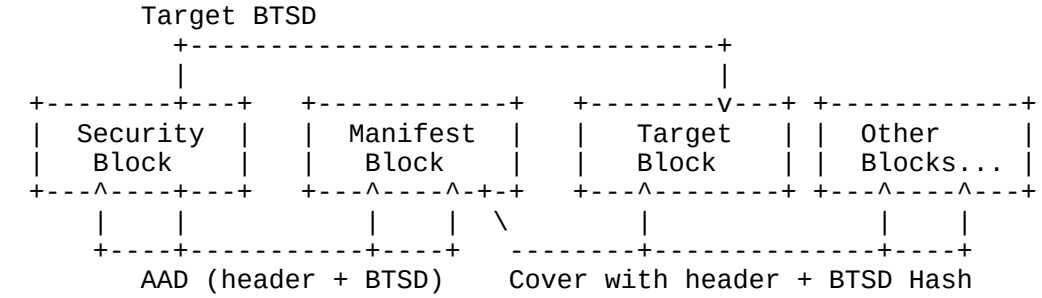

Figure 12: Security AAD Using a Manifest

Because some block types will necessarily have a small expected BTSD length, this use case involves a trade between using a hashing manifest, to guarantee external AAD size, and just using the other block BTSD in the AAD scope directly. For example, when using an overall security strength needing a SHA-384 hash a BTSD Hash item will contain a bstr value of 48 bytes. If the other block being included is a Hop Count type (Section 4.4.3 of [RFC9171]) it will never have a BTSD larger than 19 bytes (or more realistically, for hop limits below 64k, no larger than 7 bytes). It is an implementation matter by the manifest and security source to determine how and when to apply BTSD hashing for external AAD.

The Manifest reason "Security AAD Hashing" (code 1) SHALL be used to indicate that the Manifest block is intended to distill the BTSD of one or more blocks in the Bundle to be included in some security scope.

When this reason is present, the block itself SHALL be marked with block processing control flags having bit 2 set, indicating "Delete bundle if block can’t be processed."
// TBD: is this adequate to handle middle-boxes that don’t care about
// the Manifest (or any other) block content?

For this use case the Manifest Source will be identical to the Security Source of the security block using this Manifest in its AAD scope. When this reason is present in the metadata map, each block data map SHALL contain the following items:

* Block Identity (key 1)

* Block Flags (key 2)

* BTSD Hash (key 6)

When this reason is present in the metadata map, each block data map MAY contain a BTSD Length (key 5).

When verifying for this reason, if any block is missing or has a mismatch with the Manifest the entire verification SHALL be considered to have failed.

When a Manifest with this reason is included in the target or AAD scope of a security operation, the full verification of that operation SHALL include verifying the Manifest. This logic does not need to be implemented as part of BPSec processing, but can be done as part of larger security policy alongside BPSec proper. Although successful verification now requires both BPSec verification and Manifest verification, the order in which these two occur can be done

in either order (or even concurrently). Choosing a desirable order allows an implementation to optimize further for expected success and failure modes.

4.3. Security Sourcing

When a BPSec security operation is created by a node, as its security source, the security target and its associated security results are embodied within the ASB structure of its parent security block. Because the ASB retains no history of security actions, later updates in the lifecycle of the security operation can cause the parent security block to be modified or even discarded (_e.g._, if the operation is accepted before the destination node). The reason defined in this section allows security sourcing to be recorded for downstream auditing [I-D.tian-dtn-sbam].

The Manifest reason "Security Sourcing" (code 3) SHALL be used to indicate that the Manifest block is intended to record the actions of a security source as defined by BPSec [RFC9172].

When this reason is present in the metadata map, each block data map SHALL contain the following items:

* Block Identity (key 1) having a covered block type of either 11 or 12 and having a Security Source field from the same node as the Manifest Source

* Security Targets (key -1)

* Security Context ID (key -2)

* // Additional data about key identifiers?

Multiple block data maps MAY be present as needed to indicate all sourcing actions performed by the node across any security blocks in the Bundle.

Because this reason is meant to be used separately from the covered security blocks themselves, with a likely longer block lifecycle, the containing Manifest is not meant to be verified (in the sense of Section 3.2) by any node. This reason can be coupled with Security Acceptance manifest(s) to audit the lifecycles of those covered security blocks.

4.4. Security Acceptance

When a BPSec security operation is accepted by a node, the security target and its associated security results are removed from the ASB structure of its parent security block. Because the ASB retains no history of security actions, a removal due to acceptance is not visible downstream from the acceptor node any differently than malicious removal by an on-path attacker. The reason defined in this section allows security acceptance to be recorded for downstream auditing [I-D.tian-dtn-sbam].

The Manifest reason "Security Acceptance" (code 4) SHALL be used to indicate that the Manifest block is intended to record the actions of a security acceptor as defined by BPSec [RFC9172].

When this reason is present in the metadata map, each block data map SHALL contain the following items:

* Block Identity (key 1) having a covered block type of either 11 or 12

* Security Targets (key -1)

* Security Context ID (key -2)

* // Additional data about key identifiers?

Multiple block data maps MAY be present as needed to indicate all acceptance actions performed by the node across any security blocks in the Bundle.

Because this reason is meant to be used when the referenced security operations (and their security targets) are removed from their parent security blocks, the containing Manifest is not meant to be verified (in the sense of Section 3.2) by any node. It is a recording of what _was_ present before the security acceptance and meant to be coupled with either Security Sourcing manifest(s) or out-of-band information about expected security operations.

5. Security Considerations

This section separates security considerations into threat categories based on guidance of BCP 72 [RFC3552].

5.1. Threat: Passive Leak of Bundle Data

The use of a Manifest block does not cause any additional visibility of data at the time the Manifest is added to a Bundle. All of its metadata and block data is derived from other block items already visible to any handler of the Bundle.

It is possible that a block covered by a Manifest is later made the target of a confidentiality operation and its BTSD becomes ciphertext. Even in this case, the block data items defined by this specification in Section 2 do not directly leak the BTSD, only information _derived from_ the BTSD is present in the Manifest. The strength of the BTSD Hash item is a separate consideration discussed in Section 5.2.

Future block data item definitions are RECOMMENDED to consider if and how they reveal information from the BTSD of covered blocks.

5.2. Threat: Weak Hash Algorithms

When used as defined in Section 4.2 the hashing function(s) become part of the overall security verification. A weak strength hash algorithm in a Manifest will lower the cryptographic binding to the associated BTSD down to the strength of that hashing.

When used as BPSec AAD, the choice of BTSD hash algorithm(s) in a Manifest SHALL be at least as strong as the overall security strength of the associated integrity or confidentiality algorithm(s).

5.3. Threat: Untrusted Manifest

The mere presence of a Manifest block in a Bundle does not grant any implied trust for the metadata or block data contained in the Manifest. A BPA can compare the block data within a Manifest with the actual blocks of a Bundle before any related security processing to catch mismatches early, but the results of any such comparison need to be treated as conditional on trust of the Manifest block itself.

Beyond the specific use in Section 4.2, each Manifest block SHOULD be the target or AAD of an associated BPSec integrity operation.

5.4. Threat: Manipulated Manifest

Any Manifest block which is kept in plaintext (_i.e._, is not the target of a BPSec confidentially operation) is vulnerable to on-path manipulation of its block content, either in-transit or at-rest.

The verification requirements in Section 3.2 ensure that entities processing a Manifest block verify not just the contents of the Manifest itself but also verify that the contained block data matches the actual Bundle contents.

6. IANA Considerations

This section provides guidance to the Internet Assigned Numbers Authority (IANA) regarding registration of code points in existing registries and creation of new registries in accordance with BCP 26 [RFC8126].

6.1. BPv7 Block Type

IANA has registered, under the "Bundle Protocol" registry group [IANA-BUNDLE], an entry in the registry "Bundle Block Types" with the following information.

Bundle Protocol Version	Value	Description	Reference
7	// TBA1	Manifest	[This specification]

Table 1: Bundle Block Types

IANA NOTE: The
// TBA1 requested value is the lowest unassigned value (13).

6.2. Manifest Registries

EDITOR NOTE: registries to-be-created upon publication of this specification.

For all of the registries defined in this section, expert(s) are encouraged to be biased towards approving registrations unless they are abusive, frivolous, or actively harmful (not merely aesthetically displeasing, or architecturally dubious).

Specifications of new metadata items need to define the CBOR item structure of the map value as well as the interpretation of that value by entities processing the Manifest. Entities will ignore items with an unknown key, and that behavior needs to be considered by new items.

IANA has created, under the "Bundle Protocol" registry group [IANA-BUNDLE], a registry titled "Manifest Metadata Items" and initialize it with the contents of Table 2. For positive code points the registration procedure is Specification Required. Negative code points are reserved for use on private networks for functions not published to the IANA.

Key	Name	Value Type	References
-32768 to -32641	Reserved for Experimental Use		[This specification]
-32640 to -1	Reserved for Private Use		[This specification]
0	Reason	int16	[This specification]
2	Manifest Source	embed-eid-structure	[This specification]
3	Timestamp	dtn-time	[This specification]
4 to 32767	Unassigned		

Table 2: Manifest Metadata Items

IANA has created, under the "Bundle Protocol" registry group [IANA-BUNDLE], a registry titled "Manifest Reason Codes" and initialize it with the contents of Table 3. For positive code points the registration procedure is Specification Required. Negative code points are reserved for use on private networks for functions not published to the IANA.

Value	Name	References
-32768 to -32641	Reserved for Experimental Use	[This specification]
-32640 to -1	Reserved for Private Use	[This specification]
0	Reserved	[This specification]
1	Security AAD Hashing	[This specification]
2	Preserve Extensions	[This specification]
3	Security Sourcing	[This specification]
4	Security Acceptance	[This specification]
5 to 32767	Unassigned	

Table 3: Manifest Reason Codes

IANA has created, under the "Bundle Protocol" registry group [IANA-BUNDLE], a registry titled "Manifest Data Items" and initialize it with the contents of Table 4. For non-negative code points the registration procedure is Specification Required and the Block Type column is empty. For negative code points the registration procedure is Specification Required and the Block Type column references a type from the "Bundle Block Types" registry in the same group.

Block Type	Key	Name	Value Type	References
	0	Reserved		[This specification]
	1	Block Identity	array of uint	[This specification]
	2	Block Flags	uint	[This specification]
	5	BTSD Length	uint	[This specification]
	6	BTSD Hash	btsd-hash	[This specification]
	7 to 31614	Unassigned		
	31615 to 32639	Reserved for Private Use		[This specification]
	32640 to 32767	Reserved for Experimental Use		[This specification]
Block Type Specific items follow				
11,12	-1	Security Targets	array of uint	[This specification]
11,12	-2	Security Context ID	int16	[This specification]
	-31616 to -32640	Reserved for Private Use		[This specification]
	-32641 to -32768	Reserved for Experimental Use		[This specification]

Table 4: Manifest Data Items

7. References

7.1. Normative References

[IANA-BUNDLE]
IANA, "Bundle Protocol",
<https://www.iana.org/assignments/bundle/>.

[IANA-COSE]
IANA, "CBOR Object Signing and Encryption (COSE)",
<https://www.iana.org/assignments/cose/>.

[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>.

[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>.

[RFC8742] Bormann, C., "Concise Binary Object Representation (CBOR) Sequences", RFC 8742, DOI 10.17487/RFC8742, February 2020, <https://www.rfc-editor.org/info/rfc8742>.

[RFC9171] Burleigh, S., Fall, K., and E. Birrane, III, "Bundle Protocol Version 7", RFC 9171, DOI 10.17487/RFC9171, January 2022, <https://www.rfc-editor.org/info/rfc9171>.

[RFC9172] Birrane, III, E. and K. McKeever, "Bundle Protocol Security (BPSec)", RFC 9172, DOI 10.17487/RFC9172, January 2022, <https://www.rfc-editor.org/info/rfc9172>.

7.2. Informative References

[I-D.ietf-cose-hash-envelope]
Steele, O., Lasker, S., and H. Birkholz, "COSE Hash Envelope", Work in Progress, Internet-Draft, draft-ietf-cose-hash-envelope-10, 15 November 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-cose-hash-envelope-10>.

[I-D.ietf-dtn-bpsec-cose]
Sipos, B., "Bundle Protocol Security (BPSec) COSE Context", Work in Progress, Internet-Draft, draft-ietf-dtn-bpsec-cose-13, 21 November 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-dtn-bpsec-cose-13>.

[I-D.ietf-dtn-eid-pattern]
Sipos, B., "Bundle Protocol Endpoint ID Patterns", Work in Progress, Internet-Draft, draft-ietf-dtn-eid-pattern-05, 15 December 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-dtn-eid-pattern-05>.

[I-D.tian-dtn-sbam]
Dowling, B., Hale, B., Tian, X., and B. Wimalasiri, "Securing BPSec Against Arbitrary Packet Dropping", Work in Progress, Internet-Draft, draft-tian-dtn-sbam-00, 2 July 2025, <https://datatracker.ietf.org/doc/html/draft-tian-dtn-sbam-00>.

[RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, DOI 10.17487/RFC3552, July 2003, <https://www.rfc-editor.org/info/rfc3552>.

[RFC5050] Scott, K. and S. Burleigh, "Bundle Protocol Specification", RFC 5050, DOI 10.17487/RFC5050, November 2007, <https://www.rfc-editor.org/info/rfc5050>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

Acknowledgments

Author’s Address

Brian Sipos
The Johns Hopkins University Applied Physics Laboratory
11100 Johns Hopkins Rd.
Laurel, MD 20723
United States of America
Email: brian.sipos+ietf@gmail.com