Independent Submission
Request for Comments: 8493
Category: Informational
ISSN: 2070-1721

J. Kunze
California Digital Library
J. Littman
Stanford Libraries
E. Madden
Library of Congress
J. Scancella

C. Adams
Library of Congress

October 2018

The BagIt File Packaging Format (V1.0)

Abstract

This document describes BagIt, a set of hierarchical file layout conventions for storage and transfer of arbitrary digital content. A "bag" has just enough structure to enclose descriptive metadata "tags" and a file "payload" but does not require knowledge of the payload's internal semantics. This BagIt format is suitable for reliable storage and transfer.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

1.1. Purpose

BagIt is a set of hierarchical file layout conventions designed to support storage and transfer of arbitrary digital content. A "bag" consists of a directory containing the payload files and other accompanying metadata files known as "tag" files. The "tags" are metadata files intended to facilitate and document the storage and transfer of the bag. Processing a bag does not require any understanding of the payload file contents, and the payload files can be accessed without processing the BagIt metadata.

The name, BagIt, is inspired by the "enclose and deposit" method [ENCDEP], sometimes referred to as "bag it and tag it". BagIt differs from serialized archival formats such as MIME, TAR, or ZIP in two general areas:

1. Strong integrity assurances. The format supports cryptographic-quality hash algorithms (see Section 2.4) and allows for in-place upgrades to add additional manifests using stronger algorithms without breaking backwards compatibility. This provides high levels of confidence against data corruption, but it is not designed to be secure against active attacks.

2. Direct file access. Because BagIt specifies an actual filesystem hierarchy rather than a serialized representation of one, files can be accessed using standard operating system utilities, implementations do not need to process a potentially large archival file to extract a subset of data, and the format imposes no size limits for either individual files or a bag.

BagIt is widely used for preserving digital assets originating from different domains. Organizations involved in digital preservation with BagIt include the Library of Congress, Dryad Data Repository, NSF DataONE, and the Rockefeller Archive Center. Software implementations are available for many languages, including Python, Ruby, Java, Perl, and PHP. It is also used in the libraries of many universities, such as Cornell, Purdue, Stanford, Ghent University, New York University, and the University of California.

1.2. Requirements

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.

Implementers are strongly encouraged to review the interoperability considerations described in Section 6.1.

1.3. Terminology

The following terms have precise definitions as used in this document:

bag: A set of opaque files contained within the structure defined by this document.

bag declaration: The file required to be in all bags conforming to this document. Contains values necessary to process the rest of a bag. See Section 2.1.1.

bag checksum algorithm: The name of a cryptographic checksum algorithm that has been normalized for use in a manifest or tag manifest file name (e.g., "sha512") as described in Section 2.4.

manifest: A tag file that maps filepaths to checksums. A manifest can be a payload manifest (see Section 2.1.3) or a tag manifest (see Section 2.2.1).

payload: The data encapsulated by the bag as a set of named files, which may be organized in subdirectories. The contents of the payload files are opaque to this document, and, with respect to BagIt processing, are always considered as sequences of uninterpreted octets. See Section 2.1.2.

tag directory: A directory that contains one or more tag files.

tag file: A file that contains metadata about the bag or its payload. This document defines the standard BagIt tag files: the bag declaration in "bagit.txt" (see Section 2.1.1), payload manifests (see Section 2.1.3), tag manifests (see Section 2.2.1), bag metadata in "bag-info.txt" (see Section 2.2.2), and remote payload in "fetch.txt" (see Section 2.2.3). This document also allows other arbitrary tag files as described in Section 2.2.4.

complete: A bag that contains every element required by this document, every payload file listed in a manifest, and any optional files that are listed in a tag manifest. See Section 3.

valid: A complete bag where every checksum in every manifest has been successfully verified against the corresponding file.

2. Structure

A bag MUST consist of a base directory containing the following:

1. a set of required and optional tag files (see Section 2.2);

2. a subdirectory named "data", called the payload directory (see Section 2.1.2); and

3. a set of optional tag directories.

The tag files in the base directory consist of one or more files named "manifest-_algorithm_.txt" (see Sections 2.1.3 and 2.4), a file named "bagit.txt" (see Section 2.1.1), and zero or more additional tag files (see Section 2.2). The tag files and directories are in arbitrary file hierarchies and MAY have any name that is not reserved for a file or directory in this document.

The base directory can have any name, as illustrated by the figure below.

```
<base directory>/
|
+-- bagit.txt
|
+-- manifest-<algorithm>.txt
|
+-- [additional tag files]
|
+-- data/
|     |
|     +-- [payload files]
|
+-- [tag directories]/
      |
      +-- [tag files]
```

2.1. Required Elements

2.1.1. Bag Declaration: bagit.txt

The "bagit.txt" tag file MUST consist of exactly two lines in this order:

```
BagIt-Version: M.N
Tag-File-Character-Encoding: ENCODING
```

_M.N_ identifies the BagIt major (M) and minor (N) version numbers. _ENCODING_ identifies the character set encoding used by the remaining tag files. _ENCODING_ SHOULD be "UTF-8", but for backwards compatibility it MAY be any other encoding registered in [cs-registry]. The bag declaration itself MUST be encoded in UTF-8 and MUST NOT contain a Byte Order Mark (BOM) [RFC3629].

The number for this version of BagIt is "1.0".

2.1.2. Payload Directory: data/

The base directory MUST contain a subdirectory named "data".

The payload directory contains the arbitrary digital content within the bag. The files under the payload directory are called payload files, or the payload. Each payload file is treated as an opaque octet stream when verifying file correctness. Payload files MAY be organized in arbitrary subdirectory structures within the payload directory; however, for the purpose of this document, such subdirectory structures and filenames have no given meaning.

2.1.3. Payload Manifest: manifest-algorithm.txt

A payload manifest file provides a complete listing of each payload file name along with a corresponding checksum to permit data integrity checking. A bag can have more than one payload manifest, with each using a different checksum algorithm. Manifest entries MUST satisfy the following constraints:

- Every bag MUST contain at least one payload manifest file and MAY contain more than one.

- Every payload manifest MUST list every payload file name exactly once.

- A payload manifest file MUST have a name of the form "manifest-_algorithm_.txt", where _algorithm_ is a string specifying the checksum algorithm used by that manifest as described in Section 2.4.

Example payload manifest filenames:

```
manifest-sha256.txt
manifest-sha512.txt
```

Each line of a payload manifest file MUST be of the form

checksum filepath

where _filepath_ is the pathname of a file relative to the base directory, and _checksum_ is a hex-encoded checksum calculated by applying _algorithm_ over the file.

- The hex-encoded checksum MAY use uppercase and/or lowercase letters.

- The slash character ('/') MUST be used as a path separator in _filepath_.

- One or more linear whitespace characters (spaces or tabs) MUST separate _checksum_ from _filepath_.

- There is no limitation on the length of a pathname.

- The payload manifest MUST NOT reference files outside the payload directory.

- If a _filepath_ includes a Line Feed (LF), a Carriage Return (CR), a Carriage-Return Line Feed (CRLF), or a percent sign (%), those characters (and only those) MUST be percent-encoded following [RFC3986].

A manifest MUST NOT reference directories. Bag creators who wish to create an otherwise empty directory have typically done so by creating an empty placeholder file with a name such as ".keep".

2.2. Optional Elements

2.2.1. Tag Manifest: tagmanifest-algorithm.txt

A tag manifest is a tag file that lists other tag files and checksums for those tag files generated using a particular bag checksum algorithm.

A bag MAY contain one or more tag manifests, in which case each tag manifest SHOULD list the same set of tag files.

Each tag manifest MUST list every payload manifest. Each tag manifest MUST NOT list any tag manifests but SHOULD list the remaining tag files present in the bag.

A tag manifest file MUST have a name of the form "tagmanifest-_algorithm_.txt", where _algorithm_ is a string following the format described in Section 2.4 that specifies the bag checksum algorithm used in that manifest.

Tag manifests SHOULD use the same algorithms as the payload manifests that are present in the bag.

Example tag manifest filenames:

```
tagmanifest-sha256.txt
tagmanifest-sha512.txt
```

A tag manifest file has the same form as the payload manifest file described in Section 2.1.3 but MUST NOT list any payload files. As a result, no _filepath_ listed in a tag manifest begins "data/".

2.2.2. Bag Metadata: bag-info.txt

The "bag-info.txt" file is a tag file that contains metadata elements describing the bag and the payload. The metadata elements contained in the "bag-info.txt" file are intended primarily for human use. All metadata elements are OPTIONAL and MAY be repeated. Because "bag-info.txt" is intended for human reading and editing, ordering MAY be significant and the ordering of metadata elements MUST be preserved.

A metadata element MUST consist of a label, a colon ":", a single linear whitespace character (space or tab), and a value that is terminated with an LF, a CR, or a CRLF.

The label MUST NOT contain a colon (:), LF, or CR. The label MAY contain linear whitespace characters but MUST NOT start or end with whitespace.

It is RECOMMENDED that lines not exceed 79 characters in length. Long values MAY be continued onto the next line by inserting a LF, CR, or CRLF, and then indenting the next line with one or more linear white space characters (spaces or tabs). Except for linebreaks, such padding does not form part of the value.

Implementations wishing to support previous BagIt versions MUST accept multiple linear whitespace characters before and after the colon when the bag version is earlier than 1.0; such whitespace does not form part of the label or value.

The following are reserved metadata elements. The use of these reserved metadata elements is OPTIONAL but encouraged. Reserved metadata element names are case insensitive. Except where indicated otherwise, these metadata element names MAY be repeated to capture multiple values.

Source-Organization: Organization transferring the content.

Organization-Address: Mailing address of the source organization.

Contact-Name: Person at the source organization who is responsible for the content transfer.

Contact-Phone: International format telephone number of person or position responsible.

Contact-Email: Fully qualified email address of person or position responsible.

External-Description: A brief explanation of the contents and provenance.

Bagging-Date: Date (YYYY-MM-DD) that the content was prepared for transfer. This metadata element SHOULD NOT be repeated.

External-Identifier: A sender-supplied identifier for the bag.

Bag-Size: The size or approximate size of the bag being transferred, followed by an abbreviation such as MB (megabytes), GB (gigabytes), or TB (terabytes): for example, 42600 MB, 42.6 GB, or .043 TB. Compared to Payload-Oxum (described next), Bag-Size is intended for human consumption. This metadata element SHOULD NOT be repeated.

Payload-Oxum: The "octetstream sum" of the payload, which is intended for the purpose of quickly detecting incomplete bags before performing checksum validation. This is strictly an optimization, and implementations MUST perform the standard checksum validation process before proclaiming a bag to be valid. This element MUST NOT be present more than once and, if present, MUST be in the form "_OctetCount_._StreamCount_", where _OctetCount_ is the total number of octets (8-bit bytes) across all payload file content and _StreamCount_ is the total number of payload files. This metadata element MUST NOT be repeated.

Bag-Group-Identifier: A sender-supplied identifier for the set, if any, of bags to which it logically belongs. This identifier SHOULD be unique across the sender's content, and if it is recognizable as belonging to a globally unique scheme, the receiver SHOULD make an effort to honor the reference to it. This metadata element SHOULD NOT be repeated.

Bag-Count: Two numbers separated by "of", in particular, "N of T", where T is the total number of bags in a group of bags and N is the ordinal number within the group. If T is not known, specify it as "?" (question mark): for example, 1 of 2, 4 of 4, 3 of ?, 89 of 145. This metadata element SHOULD NOT be repeated. If this metadata element is present, it is RECOMMENDED to also include the Bag-Group-Identifier element.

Internal-Sender-Identifier: An alternate sender-specific identifier for the content and/or bag.

Internal-Sender-Description: A sender-local explanation of the contents and provenance.

In addition to these metadata elements, other arbitrary metadata elements MAY also be present.

An example of "bag-info.txt" file is as follows:

```
Source-Organization: FOO University
Organization-Address: 1 Main St., Cupertino, California, 11111
Contact-Name: Jane Doe
Contact-Phone: +1 111-111-1111
Contact-Email: example@example.com
External-Description: Uncompressed greyscale TIFF images from the
      FOO papers colle...
Bagging-Date: 2008-01-15
External-Identifier: university_foo_001
Payload-Oxum: 279164409832.1198
Bag-Group-Identifier: university_foo
Bag-Count: 1 of 15
Internal-Sender-Identifier: /storage/images/foo
Internal-Sender-Description: Uncompressed greyscale TIFFs created
      from microfilm and are...
```

2.2.3. Fetch File: fetch.txt

For reasons of efficiency, a bag MAY be sent with a list of files to be fetched and added to the payload before it can meaningfully be checked for completeness. The fetch file allows a bag to be transmitted with "holes" in it, which can be practical for several reasons. For example, it obviates the need for the sender to stage a large serialized copy of the content while the bag is transferred to the receiver. Also, this method allows a sender to construct a bag from components that are either a subset of logically related components (e.g., the localized logical object could be much larger than what is intended for export) or assembled from logically distributed sources (e.g., the object components for export are not stored locally under one filesystem tree). An OPTIONAL tag file, called the fetch file, contains such a list.

The fetch file MUST be named "fetch.txt". Every file listed in the fetch file MUST be listed in every payload manifest. A fetch file MUST NOT list any tag files.

Each line of a fetch file MUST be of the form

```
url length filepath
```

where _url_ identifies the file to be fetched and MUST be an absolute URI as defined in [RFC3986], _length_ is the number of octets in the file (or "-", to leave it unspecified), and _filepath_ identifies the corresponding payload file, relative to the base directory.

The slash character ('/') MUST be used as a path separator in _filepath_. One or more linear whitespace characters (spaces or tabs) MUST separate these three values, and any such characters in the _url_ MUST be percent-encoded [RFC3986]. If _filename_ includes an LF, a CR, a CRLF, or a percent sign (%), those characters (and only those) MUST be percent-encoded as described in [RFC3986]. There is no limitation on the length of any of the fields in the fetch file.

2.2.4. Other Tag Files

A bag MAY contain other tag files that are not defined by this document. Implementations MUST perform standard checksum validation on any tag file that is listed in a tag manifest but MUST otherwise ignore their contents.

2.3. Text Tag File Format

All tag files specifically described in this document MUST adhere to the text tag file format described below. Other tag files MAY adhere to the text tag file format described below.

Text tag files are line oriented, and each line MUST be terminated by an LF, a CR, or a CRLF. It is RECOMMENDED that the last line in a tag file also end with LF, CR, or CRLF. Text tag file names MUST end in the extension ".txt".

In all text tag files except for the bag declaration file, text MUST use the character encoding specified in the "bagit.txt" bag declaration file. Text tag files except for the bag declaration file MAY include a Byte Order Mark (BOM) only if the specified encoding requires it for proper decoding. In accordance with [RFC3629], when "bagit.txt" specifies UTF-8, the tag files MUST NOT begin with a BOM. See Section 2.1.1.

The use of UTF-8 for text tag files is strongly RECOMMENDED. A future version of BagIt may disallow encodings other than UTF-8.

2.4. Bag Checksum Algorithms

The payload manifest and tag manifest permit validating the integrity of the payload and tag files in a bag produced by the checksum algorithms. Checksum values MUST be encoded so as to conform to the manifest format specified in Section 2.1.3. However, the internal details of a checksum are outside the scope of this document.

To avoid future ambiguity, the checksum algorithm SHOULD be registered in IANA's "Named Information Hash Algorithm Registry" [ni-registry] according to [RFC6920] but MAY, for backwards compatibility, also be MD5 [RFC1321] or SHA-1 [RFC3174].

The name of the checksum algorithm MUST be normalized for use in the manifest's filename by lowercasing the common name of the algorithm and removing all non-alphanumeric characters. Following is a partial list that maps common algorithm names to normalized names:

- MD5: md5
- SHA-1: sha1
- sha-256: sha256
- sha-512: sha512

Starting with BagIt 1.0, bag creation and validation tools MUST support the SHA-256 and SHA-512 algorithms [RFC6234] and SHOULD enable SHA-512 by default when creating new bags. For backwards compatibility, implementers SHOULD support MD5 [RFC1321] and SHA-1 [RFC3174]. Implementers are encouraged to simplify the process of adding additional manifests using new algorithms to streamline the process of in-place upgrades.

3. Complete and Valid Bags

A _complete_ bag MUST meet the following requirements:

1. Every required element MUST be present (see Section 2.1).

2. Every file listed in every tag manifest MUST be present.

3. Every file listed in every payload manifest MUST be present.

4. For BagIt 1.0, every payload file MUST be listed in every payload manifest. Note that older versions of BagIt allowed payload files to be listed in just one of the manifests.

5. Every element present MUST conform to BagIt 1.0.

A _valid_ bag MUST meet the following requirements:

1. The bag MUST be _complete_.

2. Every checksum in every payload manifest and tag manifest has been successfully verified against the contents of the corresponding file.

4. Examples

4.1. Example of a Basic Bag

This is the layout of a basic bag containing an image and a companion Optical Character Recognition (OCR) file. Lines of file content are shown with added parentheses to indicate each complete line. For brevity, this example uses MD5 rather than the recommended SHA-512.

```
myfirstbag/
|
|   manifest-md5.txt
|    (49afbd86a1ca9f34b677a3f09655eae9 data/27613-h/images/q172.png)
|    (408ad21d50cef31da4df6d9ed81b01a7 data/27613-h/images/q172.txt)
|
|   bagit.txt
|    (BagIt-version: 1.0                                           )
|    (Tag-File-Character-Encoding: UTF-8                           )
|
\--- data/
     |
     |   27613-h/images/q172.png
     |    (... image bytes ...                                     )
     |
     |   27613-h/images/q172.txt
     |    (... OCR text ...                                        )
     ....
```

4.2. Example Bag Using fetch.txt

This is the layout of a bag that expects the receiver to download the files listed in the payload manifests prior to validation. Lines of file content are shown with added parentheses to indicate each complete line. For brevity, this example uses MD5 rather than the recommended SHA-512.

```
highsmith-tahoe/
|
|   manifest-md5.txt
|    (102b0e6effe208ef9b29864946de9e22 data/23364a.tif              )
|
|    fetch.txt
|     (https://cdn.loc.gov/master/pnp/highsm/23300/23364a.tif
|         216951362 data/23364a.tif                                )
|
|   bagit.txt
|    (BagIt-version: 1.0                                           )
|    (Tag-File-Character-Encoding: UTF-8                           )
|
|   bag-info.txt
|    (Internal-Sender-Description: Download link found at          )
|    (  https://www.loc.gov/resource/highsm.23364/                 )
```

5. Security Considerations

5.1. Special Directory Characters

The paths specified in the payload manifests, tag manifests, and fetch files do not prohibit special directory characters that have special meaning on some operating systems. Implementers MUST ensure that files outside the bag directory structure are not accessed when reading or writing files based on paths specified in a bag.

All implementations SHOULD have a test suite to guard against special directory characters.

For example, a maliciously crafted "tagmanifest-sha512.txt" file might contain entries that begin with a path character such as "/", "..", or a "~username" home directory reference in an attempt to cause a naive implementation to leak or overwrite targeted files on a POSIX operating system.

Windows implementations SHOULD test their implementations to ensure that safety checks prevent use of drive letters and the less commonly used namespace sequences (e.g., "\\?\C:\...") described in [MSFNAM].

To assist implementers, the Library of Congress conformance suite [LC-CONFORMANCE-SUITE] has some tests for invalid bags that are expected to fail on POSIX or Windows clients.

5.2. Control of URLs in fetch.txt

Implementers of tools that complete bags by retrieving URLs listed in a fetch file need to be aware that some of those URLs might point to hosts, intentionally or unintentionally, that are not under control of the bag’s sender. Moreover, older checksum algorithms, even if reasonable for detecting corruption during transit, may not offer strong cryptographic protection against intentional spoofing.

5.3. File Sizes in fetch.txt

The size of files, as optionally reported in the fetch file, cannot be guaranteed to match the actual file size to be downloaded. Implementers SHOULD take steps to monitor and abort transfer when the received file size exceeds the file size reported in the fetch file. Implementers SHOULD NOT use the file size in the fetch file for critical resource allocation, such as buffer sizing or storage requisitioning.

5.4. Attacks on Payload File Content

The integrity assurance provided by manifests is designed to provide high levels of confidence against data corruption but is not designed to be secure against active attacks. Organizations that need to secure bags against such threats SHOULD agree on additional measures, such as digital signatures, that are out of scope for this specification.

6. Practical Considerations (Non-normative)

6.1. Interoperability

This section lists practical considerations for implementers and users. None of the points below are required, but they are recommended for general-purpose usage.

Upon discovering errors in bags, an implementation is free to take action (for example, logging or reporting) in an application-specific manner. This document does not mandate any particular action.

The Library of Congress conformance suite [LC-CONFORMANCE-SUITE] is provided as a public resource to test new implementations for compatibility and error handling.

6.1.1. Filename Normalization

This section provides background information on various challenges caused by differences in how operating systems, filesystems, and common tools handle filenames. This section is followed by a list of recommendations for implementers in Section 6.1.1.3.

6.1.1.1. Case Sensitivity

There are three challenges for interoperability related to filename case:

- Filesystems such as File Allocation Table (FAT) or Extended File Allocation Table (EXFAT) always convert filenames to uppercase: "example.txt" will be stored as "EXAMPLE.TXT".

- Many Unix filesystems save filenames exactly as provided, which allows multiple files that differ only in case: "example.txt" and "Example.txt" are separate files.

- New Technology File System (NTFS) and Apple's Hierarchical File System (HFS) Plus usually preserve case when storing files but are case insensitive when retrieving them. A file saved as "Example.txt" will be retrieved by that name but will also be retrieved as "EXAMPLE.TXT", "example.txt", etc.

6.1.1.2. Unicode Normalization

The Unicode specification has common cases where different character sequences produce the same human-meaningful text. These are referred to as "canonically equivalent" and the Unicode specification defines different normalization forms - see [UNICODE-TR15] for the full details.

The example below shows the common surname "Nunez" normalized in different forms.

Normalization Form D (Decomposition)

Char	UTF8 Hex	Name
N	4e	LATIN CAPITAL LETTER N
u	75	LATIN SMALL LETTER U
\u0301	cc81	COMBINING ACUTE ACCENT
n	6e	LATIN SMALL LETTER N
\u0303	cc83	COMBINING TILDE
e	65	LATIN SMALL LETTER E
z	7a	LATIN SMALL LETTER Z

Normalization Form C (Canonical Composition)

Char	UTF8 Hex	Name
N	4e	LATIN CAPITAL LETTER N
u	c3ba	LATIN SMALL LETTER U WITH ACUTE
n	c3b1	LATIN SMALL LETTER N WITH TILDE
e	65	LATIN SMALL LETTER E
z	7a	LATIN SMALL LETTER Z

Unicode normalization is relevant to BagIt implementors because different systems have different standards for normalization:

- Apple’s HFS Plus filesystem always normalizes filenames to a fully decomposed form based on the Unicode 2.0 specification (see [TN1150]).
- Windows treats filenames as opaque character sequences (see [MSFNAM]) and will store and return the encoded bytes exactly as provided.
- Linux and other common Unix systems are generally similar to Windows in storing and returning opaque byte streams, but this behavior is technically dependent on the filesystem.
- Utilities used for file management, transfer, and archiving may ignore this issue, apply an arbitrary normalization form, or allow the user to control how normalization is applied.

In practice, this means that the encoded filename stored in a manifest may fail a simple file existence check because the filename’s normalization was changed at some point after the manifest was written. This situation is very confusing for users because the filenames are visually indistinguishable, and the "missing" file is obviously present in the payload directory.

6.1.1.3. Recommendations

- Implementations SHOULD discourage the creation of bags containing files that differ only in case.
- Implementations SHOULD prevent the creation of bags containing files that differ only in normalization form.
- BagIt implementations SHOULD tolerate differences in normalization form by comparing both the list of filesystem and manifest names after applying the same normalization form to both.

- Implementations SHOULD issue a warning when multiple manifests are present that differ only in case or normalization form.

6.1.2. Windows and Unix File Naming

As specified above, only the Unix-based path separator ('/') may be used inside filenames listed in BagIt manifest and fetch.txt files. When bags are exchanged between Windows and Unix platforms, the path separator SHOULD be translated as needed. Receivers of bags on physical media SHOULD be prepared for filesystems created under either Windows or Unix. Besides the fundamental difference between path separators ('\' and '/'), generally, Windows filesystems have more limitations than Unix filesystems.

Windows path names have a maximum of 255 characters, and none of these characters may be used in a path component:

```
< > : " / | ? *
```

Windows also reserves the following names, with or without a file extension:

```
CON, PRN, AUX, NUL
COM1, COM2, COM3, COM4, COM5, COM6, COM7, COM8, COM9
LPT1, LPT2, LPT3, LPT4, LPT5, LPT6, LPT7, LPT8, LPT9
```

See [MSFNAM] for more information and possible alternatives.

6.1.3. Legacy Checksum Tools

Some bags have been manually assembled using checksum utilities such as those contained in the GNU Coreutils package (md5sum, sha1sum, etc.), collectively referred to here as "md5sum". Implementers who desire wide support of legacy content should be aware of some known quirks of these tools.

md5sum can be run in "text mode", which causes it to normalize line endings on some operating systems. On Unix-like systems, both modes will usually produce the same results; on systems like Windows, they can produce different results based on the file contents. The md5sum output format has two characters between the checksum and the filepath: the first is always a space, and the second is an asterisk ("*") for binary mode and a space for text mode.

A final note about md5sum-generated manifests is that, for a _filepath_ containing a backslash ('\'), the manifest line will have a backslash inserted in front of the _checksum_ and, under Windows, the backslashes inside _filepath_ can be doubled.

Implementers MAY wish to accept this format by ignoring a leading asterisk or handling differences in line termination gracefully but, if so, implementations MUST warn the user that the bag in question will fail strict validation. In such cases, it is RECOMMENDED that tools provide an easy option to update the bag with valid manifests.

7. Augmented Backus-Naur Form (Non-normative)

The Augmented Backus-Naur Form (ABNF) rules provided below are non-normative. If there is a discrepancy between requirements in the normative sections and the ABNF, the requirements in the normative sections prevail. Some definitions use the core rules (e.g., DIGIT, HEXDIG, etc) as defined in [RFC5234].

7.1. Bag Declaration: bagit.txt

bagit.txt ABNF rules:

```
bagit-txt = "BagIt-Version: " 1*DIGIT "." 1*DIGIT ending
            "Tag-File-Character-Encoding: " encoding ending
encoding  = 1*CHAR
ending    = CR / LF / CRLF
```

7.2. Payload Manifest: manifest-algorithm.txt

Payload Manifest ABNF rules:

```
payload-manifest      = 1*payload-manifest-line
payload-manifest-line = checksum 1*WSP filepath ending
checksum              = 1*case-hexdig
case-hexdig           = DIGIT / "A" / "a" / "B" / "b" / "C" / "c" /
                        "D" / "d" / "E"/ "e"/ "F" / "f"
filepath              = "data/"
                        1*( unreserved / pct-encoded / sub-delims )
unreserved            = ALPHA / DIGIT / "-" / "." / "_" / "˜"
sub-delims            = "!" / "$" / "&" / DQUOTE / "’" / "(" / ")" /
                        "*" / "+" / "," / ";" / "=" / "/"
pct-encoded           = "%0D" / "%0d" / "%0A" / "%0a" / "%25"
ending                = CR / LF / CRLF
```

7.3. Bag Metadata: bag-info.txt

bag-info.txt ABNF rules:

```
metadata      = 1*metadata-line
metadata-line = key ":" WSP value ending *(continuation ending)
key           = 1*non-reserved
value         = 1*non-reserved
continuation  = WSP 1*non-reserved
non-reserved  = VCHAR / WSP
                ; any valid character for the specific encoding
                ; except those that match "ending"
ending        = CR / LF / CRLF
```

7.4. Fetch File: fetch.txt

fetch.txt ABNF rules:

```
fetch      = 1*fetch-line
fetch-line = url 1*WSP length 1*WSP filepath ending
url        = <absolute-URI, see [RFC3986], Section 4.3>
length     = 1*DIGIT / "-"
filepath   = ("data/"
               1*( unreserved / pct-encoded / sub-delims ))
ending     = CR / LF / CRLF
```

8. IANA Considerations

This document has no IANA actions.

9. References

9.1. Normative References

[cs-registry]
IANA, "Character Set",
<https://www.iana.org/assignments/character-sets>.

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

[RFC1321] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321,
DOI 10.17487/RFC1321, April 1992,
<https://www.rfc-editor.org/info/rfc1321>.

[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>.

[RFC3174] Eastlake 3rd, D. and P. Jones, "US Secure Hash Algorithm 1 (SHA1)", RFC 3174, DOI 10.17487/RFC3174, September 2001, <https://www.rfc-editor.org/info/rfc3174>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <https://www.rfc-editor.org/info/rfc3629>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <https://www.rfc-editor.org/info/rfc6234>.

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

[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>.

9.2. Informative References

[ENCDEP] Tabata, K., Okada, T., Nagamori, M., Sakaguchi, T., and S. Sugimoto, "A Collaboration Model between Archival Systems to Enhance the Reliability of Preservation by an Enclose-and-Deposit Method", 2005, <https://web.archive.org/web/20060508015635/http://www.iwaw.net/05/papers/iwaw05-tabata.pdf>.

[LC-CONFORMANCE-SUITE] The Library of Congress, "Test cases for validating Bagit Implementations", commit 43bcbdf, November 2017, <https://github.com/LibraryOfCongress/bagit-conformance-suite/>.

[MSFNAM] Microsoft, Inc., "Naming Files, Paths, and Namespaces", May 2018, <http://msdn2.microsoft.com/en-us/library/aa365247.aspx>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

[TN1150] Apple Inc., "Technical Note TN1150: HFS Plus Volume Format", March 2004, <https://developer.apple.com/legacy/library/technotes/tn/tn1150.html>.

[UNICODE-TR15]
Unicode Consortium, "Unicode Standard Annex #15: Unicode Normalization Forms", Technical Report, Unicode 11.0.0, May 2018, <http://www.unicode.org/reports/tr15/>.

Acknowledgements

BagIt benefitted from the thoughtful assistance of Stephen Abrams, Mike Ashenfelder, Dan Chudnov, Dave Crocker, Scott Fisher, Brad Hards, Erik Hetzner, Keith Johnson, Leslie Johnston, David Loy, Mark Phillips, Tracy Seneca, Stian Soiland-Reyes, Brian Tingle, Adam Turoff, and Jim Tuttle.

Contributors

Additional contributors to the authoring of BagIt are Andy Boyko, David Brunton, Rosie Storey, Ed Summers, Brian Vargas, and Kate Zwaard.

Authors' Addresses

John A. Kunze
California Digital Library
415 20th St, 4th Floor
Oakland, CA 94612
United States of America

Email: jak@ucop.edu

Justin Littman
Stanford Libraries
518 Memorial Way
Stanford, CA 94305
United States of America

Email: justinlittman@stanford.edu

Liz Madden
Library of Congress
101 Independence Avenue SE
Washington, DC 20540
United States of America

Email: emad@loc.gov

John Scancella

Email: john.scancella@gmail.com

Chris Adams
Library of Congress
101 Independence Avenue SE
Washington, DC 20540
United States of America

Email: cadams@loc.gov