NDIIPP Content Transfer Project	A. Boyko
	Library of Congress
	J. Kunze
	California Digital Library
	J. Littman
	L. Madden
	B. Vargas
	Library of Congress
	November 24, 2008

The BagIt File Package Format (V0.96)

Abstract

This document specifies BagIt, a hierarchical file packaging format for the exchange of generalized digital content. A "bag" has just enough structure to safely enclose a brief descriptive "tag" and a payload but does not require any knowledge of the payload's internal semantics. This BagIt format should be suitable for disk-based or network-based file hierarchy transfer. One important use case is the possibility of eventual safe return of a received bag. Tag information consists of a small number of top-level reserved file names, checksums for transfer validation, and optional small metadata blocks.
An Intenet-Draft of this document is available as a text file at:
http://tools.ietf.org/id/draft-kunze-bagit-03.txt

1. Introduction

BagIt is a hierarchical file packaging format designed to support disk-based or network-based transfer of generalized digital content. A "bag" holds a brief "tag" and an otherwise semantically opaque payload. The name, BagIt, is inspired by the "enclose and deposit" method **[ENCDEP]**, sometimes referred to as "bag it and tag it".

In this document the word "directory" is used interchangeably with the word "folder" and all filesystem examples conform to Unix-based conventions which should translate easily to Windows conventions after substituting the path separator ('\' instead of '/'). On the other hand, only the Unix-based separator ('/') is used inside BagIt manifest and "fetch.txt" files. The BagIt format itself places no limitations on file and path lengths, so implementors thinking about maximal interoperation may wish to consider the issues listed in the Interoperability section of this document.

---

2. BagIt directory (folder) layout

A "bag" consists of a base directory containing a set of top-level files comprising the "tag" and a sub-directory named "data/" that holds the payload. The base directory may have any name and the "data/" directory may contain an arbitrary file hierarchy.

```
<bag_dir>/
|  manifest-<algorithm>.txt
|  bagit.txt
|  [optional additional tag files]
\--- data/
     |  [optional file hierarchy]
```

The "tag" consists of one or more files named "manifest- _algorithm_.txt", a file named "bagit.txt", and zero or more additional files. In top-level text files with ".txt" extension, each line should be terminated by a newline (LF) or carriage return plus newline (CRLF); in practice cautious programmers will also accept a carriage return by itself (CR) as a line terminator. In all such tag files, text is assumed to be Unicode encoded as UTF-8 **[RFC3629]**.

2.1. Declaration that this is a bag: bagit.txt

The "bagit.txt" file should consist of exactly two lines,

```
BagIt-Version: M.N
Tag-File-Character-Encoding: UTF-8
```

where M.N identifies the BagIt major (M) and minor (N) version numbers, and UTF-8 identifies the character set encoding of tag files.

2.2. File manifest: manifest-<algorithm>.txt

A manifest is a top-level file listing payload files that must be present in a complete bag. Every bag must contain one or more manifest files. A manifest file has a name of the form manifest-*algorithm*.txt, where *algorithm* is a string specifying a cryptographic checksum algorithm, such as

```
manifest-md5.txt
manifest-sha1.txt
```

Implementors of tools that create and validate bags are strongly encouraged to support at least two widely implemented checksum algorithms: "md5" **[RFC1321]** and "sha1" **[RFC3174]**. When using other algorithms, the name of the algorithm should be normalized for use in the manifest's filename, by lowercasing the common name of the algorithm, and removing all non-alphanumeric characters.

A manifest contains a complete list of files that must be present in a fully constituted bag. Each line of a file manifest-*algorithm*.txt has the form

```
CHECKSUM FILENAME
```

where FILENAME is the pathname of a file relative to the base directory and CHECKSUM is a hex-encoded checksum calculated according to _algorithm_ over every octet in the file. Only the slash character ('/') may be used as a path separator in FILENAME and one or more linear whitespace characters (spaces and tabs) separate CHECKSUM from FILENAME; to facilitate bag interchange in this regard, implementors are encouraged to read the Interoperability section. As described below, tag (top-level) files should be listed, if listed at all, in a separate tag manifest file.

3. Example of a basic bag

Here's a very simple bag containing an image and a companion OCR file.
Lines of file content are shown in parentheses beneath the file name.

```
myfirstbag/
|
|   manifest-md5.txt
|     (49afbd86a1ca9f34b677a3f09655eae9 data/27613-h/images/q172.png)
|     (408ad21d50cef31da4df6d9ed81b01a7 data/27613-h/images/q172.txt)
|
|   bagit.txt
|     (BagIt-version: 0.96                     )
|     (Tag-File-Character-Encoding: UTF-8      )
|
\--- data/
      |
      |   27613-h/images/q172.png
      |     (... image bytes ...               )
      |
      |   27613-h/images/q172.txt
      |     (... OCR text ...                  )
      ....
```

4. Completing a bag: 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. An optional top-level file named "fetch.txt", if present, contains such a list. Each line of "fetch.txt" has the form

```
URL LENGTH FILENAME
```

where URL identifies the file to be fetched, LENGTH is the number of octets in the file (or "-", to leave it unspecified), and FILENAME identifies the corresponding payload file. Only the slash character ('/') may be used as a path separator in FILENAME. One or more linear whitespace characters (spaces and tabs) separate these three values, and any such characters in the URL must be percent-encoded **[RFC3986]**.

Because "fetch.txt" lists files that are absent from a sent bag, receivers that are storing completed bags will want some way to record that the bag no longer needs completing, such as renaming this file (e.g., to "fetch-orig.txt") or changing a database flag; if bag return is supported, the "fetch.txt" file will need to be modified as appropriate to support the return transmission method. Receipt of a bag is not final until all absent files are fetched. The receiver of a bag with a "fetch.txt" tag file is expected promptly to complete the bag by fetching all URL-identified components as the sender is not bound to make the absent components available indefinitely.

The "fetch.txt" file essentially 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).

---

5. Tag file manifest: tagmanifest-<algorithm>.txt

Zero or more tag manifest files may be present. A tag manifest is a top-level file with a name of the form "tagmanifest-_algorithm_.txt", where _algorithm_ is a string specifying a cryptographic checksum algorithm. For example, a tag manifest file using SHA1 would have the name

```
tagmanifest-sha1.txt
```

A tag manifest contains a list of tag files that must be present in a fully constituted bag. It has the same form as the file manifest described earlier, but must not list any payload files. As a result, no FILENAME listed in a tag manifest begins "data/...".

---

6. Valid bags and complete bags

A bag is considered *valid* if it is *complete* and if each CHECKSUM in every payload manifest and tag manifest can be verified against the contents of its corresponding FILENAME.

A bag is considered *complete* if every file in every manifest is present, and if every payload file appears in at least one file manifest. A payload file does not need to appear in every file manifest as long as it appears in one file manifest (i.e., it must belong to the "union" of file manifests). In a complete bag containing one or more tag manifests, any tag file may appear in zero or more of those manifests, but every tag file appearing in any tag manifest must be present in the bag.

Because manifests do not list directories specifically, only referencing them indirectly in file pathnames, they cannot account for receipt of an empty directory. Nor can its creation be implied using a "fetch.txt" file. To guarantee receipt of a directory, the sender may wish to include at least one file; it suffices, for example, to include a zero-length file named ".keep".

---

7. Other BagIt metadata: bag-info.txt

Any other tag files are considered to be bag information separate from the payload content. The "data/" directory is the custodial focus of a bag, and the top-level files comprising the tag are intended to facilitate and document the transfer. The tag could also be used to help in returning the bag to its sender at some point in the future.

Tag information is optional. If present, tag information at a minimum consists of a "bag-info.txt" file. This is a text file comprised of metadata elements intended primarily for human readability. An element consists of a label, a colon, and a value, where a long value may be folded (continued) onto the next line by inserting a newline (LF) and indenting the next line (spaces and tabs). It is recommended that lines not exceed 79 characters in length. As mentioned earlier, text is assumed to be Unicode encoded as UTF-8.

The "bag-info.txt" file contains metadata elements describing the overall file set. It looks like this.

```
Source-Organization: Spengler University
Organization-Address: 1400 Elm St., Cupertino, California, 95014
Contact-Name: Edna Janssen
Contact-Phone: +1 408-555-1212
Contact-Email: ej@spengler.edu
External-Description: Uncompressed greyscale TIFF images from the
    Yoshimuri papers colle...
Bagging-Date: 2008-01-15
External-Identifier: spengler_yoshimuri_001
Bag-Size: 260 GB
Payload-Oxum: 279164409832.1198
Bag-Group-Identifier: spengler_yoshimuri
Bag-Count: 1 of 15
Internal-Sender-Identifier: /storage/images/yoshimuri
Internal-Sender-Description: Uncompressed greyscale TIFFs created from
```

```
        microfilm and are...
```

All elements are provided as clues to ease handling on the sender and receiver ends. No particular relationship between the sender organization and the payload content is assumed; for example, the sender may be a content aggregator, redistributor, collector, curator, or producer.

Reserved element names are case-insensitive and defined as follows.

Source-Organization
: Organization transferring the content.

Organization-Address
: Mailing address of the 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 delivery.

External-Identifier
: A sender-supplied identifier for the bag. This identifier must be unique across the sender's content, and if recognizable as belonging to a globally unique scheme, the receiver should make an effort to honor reference to it.

Bag-Size
: Size or approximate size of the bag being transferred, followed by an abbreviation such as MB (megabytes), GB, or TB; for example, 42600 MB, 42.6 GB, or .043 TB. Compared to

Payload-Oxum (described next), Bag-Size is more intended for human consumption.

Payload-Oxum

The "octetstream sum" of the payload, namely, a two-part number of 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. Payload-Oxum is easy to compute (e.g., on Unix "wc -lc `find data/-type f`" does the hard part) and should be included in "bag-info.txt" if at all possible. Compared to Bag-Size (above), Payload-Oxum is more intended for machine consumption.

Bag-Group-Identifier

A sender-supplied identifier for the set, if any, of bags to which it logically belongs. This identifier must be unique across the sender's content, and if recognizable as belonging to a globally unique scheme, the receiver should make an effort to honor reference to it.

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). Examples: 1 of 2, 4 of 4, 3 of ?, 89 of 145.

Internal-Sender-Identifier

An alternate sender-specific identifier for the content and/or bag. This value may be useful to senders who may retrieve the content in the future. For instance, it might contain values that are relevant to the re-use of the content at the sender's organization.

Internal-Sender-Description

A sender-local prose description of the contents of the bag, to assist in later use if returned to the sender.

Arbitrary other bag metadata elements may follow these elements. Such elements could be used to describe the payload in ways intended for the sender in case of bag return.

8. Bag serialization

In some scenarios, such as network transfer, it may be convenient for the sender first to serialize the filesystem hierarchy representing the bag (the outermost base directory) into a single-file archive format such as TAR or ZIP. After receiving the resulting aggregate file, which we will call a *serialization*, the receiver deserializes it to recreate the filesystem hierarchy. Several rules govern the serialization of a BagIt bag and apply equally to TAR or ZIP archive files:

1. One and only one bag is contained in one serialization.
2. The serialization should have the same name as the bag's base directory, but with an extension added to identify the format; for example, the receiver of "mybag.tar.gz" expects the corresponding base directory to be created as "mybag".
3. A bag is never serialized from within its base directory, but from the parent of the base directory (where the base directory appears as an entry). Thus, after a bag is deserialized in an empty directory, a listing of that directory shows exactly one entry. For example, deserializing "mybag.zip" in an empty directory causes the creation of the base directory "mybag" and, beneath "mybag", the creation of all payload and tag files.
4. One un-archiving (deserialization) step produces a single base directory bag with the top-level structure as described in this document without requiring an additional un-archiving step. For example, after one un-archiving step it would be an error for the "data/" directory to appear as "data.tar.gz". TAR and ZIP files may appear inside the payload beneath the "data/" directory, where they would be treated as opaquely as any other payload file or directory.

When preparing a bag in an archive file format, care must be taken to ensure that the format's restrictions on file naming, such as allowable characters, length, or character encoding, will support the receiver's requirements.

9. Disk and network transfer

When recording a bag on physical media (such as hard disk, CD-ROM, or DVD), the sender will need to select and format the media in a manner compatible with both the content requirements (e.g., file names and sizes) and the receiver's technical infrastructure. If the receiver's infrastructure is not known or the media needs to be compatible with a range of potential receivers, consideration should be given to portability and common usage. For example, a "lowest common denominator" for some content and potential receivers could be USB disk drives formatted with the FAT32 filesystem.

During network-based transfer, while overall bag size is unlimited in principle, it may be useful to split a large bag into several smaller bags if there are practical constraints on the amount of bag data that a receiver can stage at one time.

Transmitting a whole bag in serialized form as a single file will tend to be the most straightforward mode of transfer. When throughput is a priority, use of "fetch.txt" lends itself to an easy, application-level parallelism in which the list of URL-addressed items to fetch is divided among multiple processes. The mechanics of sending and receiving bags over networks is otherwise out of scope of the present document and may be facilitated by protocols such as **[GRABIT]** and **[SWORD]**.

---

10. Another example bag

Here's a bag of material resulting from a hypothetical web harvest. As before, lines of file content are shown in parentheses beneath the file name, with long lines continued indented on subsequent lines. This bag is not completely retrieved, of course, until every component listed in the "fetch.txt" file is retrieved.

```
mysecondbag/
|
|   manifest-md5.txt
|    (93c53193ef96732c76e00b3fdd8f9dd3 data/Collection Overview.txt        )
|    (e9c5753d65b1ef5aeb281c0bb880c6c8 data/Seed List.txt                  )
|    (61c96810788283dc7be157b340e4eff4 data/gov-20060601-oth-
050019.arc.gz )
|    (55c7c80c6635d5a4c8fe76a940bf353e data/gov-20060601-img-
100002.arc.gz )
|
|   fetch.txt
|    (http://WB20.Stanford.Edu/gov-06-2006-ARC/gov-20060601-oth-
050019.arc.gz
|        26583985 data/gov-20060601-oth-050019.arc.gz                    )
|    (http://WB20.Stanford.Edu/gov-06-2006-ARC/gov-20060601-img-
100002.arc.gz
|        99509720 data/gov-20060601-img-100002.arc.gz                    )
|    ( ................................................................ )
|
|   bag-info.txt
|   (Source-organization: California Digital Library                    )
|   (Organization-address: 415 20th Street, 4th Floor, Oakland, CA. 94612 )
|   (Contact-name: A. E. Newman                                         )
|   (Contact-phone: +1 510-555-1234                                      )
|   (Contact-email: alfred@ucop.edu                                     )
|   (External-Description: The collection "Local Davis Flood Control      )
|     Collection" includes captured California State and local websites   )
|     containing information on flood control resources for the Davis and )
|     Sacramento area.  Sites were captured by UC Davis curator Wrigley   )
|     Spyder using the Web Archiving Service in February 2007 and        )
|     October 2007.                                              )
|   (Bagging-date: 2008.04.15                                    )
|   (External-identifier: ark:/13030/fk4jm2bcp          )
|   (Bag-size: about 22Gb                               )
|   (Payload-Oxum: 21836794142.831                      )
|   (Internal-sender-identifier: UCDL                           )
|   (Internal-sender-description: University of California Davis Libraries)
|
|  bagit.txt
|   (BagIt-version: 0.96                                 )
|   (Tag-File-Character-Encoding: UTF-8                          )
|
\--- data/
    |
    |  Collection Overview.txt
    |   (... narrative description ...                         )
    |
    |  Seed List.txt
    |   (... list of crawler starting point URLs ...                  )
    ....
```

---

11. Interoperability (non-normative)

This section is not part of the BagIt specification. It describes some practical considerations for bag creators and receivers circa 2008.

11.1. Checksum tools

Some cautions regarding bag interchange arise in regard to the commonly available checksum tools distributed with the GNU Coreutils package (md5sum, sha1sum, etc.), collectively referred to here as "md5sum". First, md5sum can be run in binary or text mode, where text mode sometimes normalizes line-endings. While these modes appear to produce the same checksums under Unix-like systems, they can produce different checksums under Windows. If using md5sum, it is therefore safest run it in binary mode, with one caveat: a side- effect of binary mode is that md5sum requires a space and an asterisk ('*'), compared to two spaces in text mode, between the CHECKSUM and FILENAME in its manifest format.

Due to the widespread use of md5sum (and its relatives), it is not unexpected for bag receivers to see manifests in which CHECKSUM and FILENAME are separated by a space followed by an asterisk. Defensive programmers may wish to pre-process such invalid manifests to remove the asterisk. A final note about md5sum-generated manifests is that for a FILENAME containing a backslash ('\'), the manifest line will have a backslash inserted in front of the CHECKSUM and, under Windows, the backslashes inside FILENAME may be doubled.

11.2. Windows and Unix file naming

As mentioned previously, only the Unix-based path separator ('/') may be used inside filenames listed in BagIt manifests and "fetch.txt" files. When bags are exchanged between Windows and Unix platforms, care should be taken to translate the path separator 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: CON, PRN, AUX, NUL, COM1, COM2, COM3, COM4, COM5, COM6, COM7, COM8, COM9, LPT1, LPT2, LPT3, LPT4, LPT5, LPT6, LPT7, LPT8, and LPT9. See **[MSFNAM]** for more information.

---

12. Security considerations

The BagIt file hierarchy format poses no direct risk to computers and networks. Implementors of tools that complete bags by retrieving URLs listed in a "fetch.txt" file need to be aware that some of those URLs may point to hosts, intentionally or unintentionally, that are not under control of the bag's sender. Checksum algorithms are designed to protect against corruption and spoofing in bag transfer, but they are not a guarantee.

13. Acknowledgements

BagIt owes much to many thoughtful contributers and reviewers, including Stephen Abrams, Mike Ashenfelder, Scott Fisher, Erik Hetzner, David Loy, Mark Phillips, Tracy Seneca, Adam Turoff, and Jim Tuttle.

14. References

[ENCDEP] Tabata, K., "**A Collaboration Model between Archival Systems to Enhance the Reliability of Preservation by an Enclose-and-Deposit Method**," 2005 (**PDF**).

[GRABIT] NDIIPP/CDL, "**The GrabIt Package Exchange Protocol**," 2008 (**HTML**).

[MSFNAM] Microsoft, "**Naming a File**," 2008 (**HTML**).

[RFC1321] **Rivest, R.**, "**The MD5 Message-Digest Algorithm**," RFC 1321, April 1992.

[RFC3174] Eastlake, D. and P. Jones, "**US Secure Hash Algorithm 1 (SHA1)**," RFC 3174, September 2001.

[RFC3629] Yergeau, F., "**UTF-8, a transformation format of ISO 10646**," STD 63, RFC 3629, November 2003.

[RFC3986] **Berners-Lee, T.**, **Fielding, R.**, and **L. Masinter**, "**Uniform Resource Identifier (URI): Generic Syntax**," STD 66, RFC 3986, January 2005 (**TXT**, **HTML**, **XML**).

[SWORD] UKOLN/JISC CETIS, "**Simple Web-service Offering Repsitory Deposit (SWORD)**," 2008.

Appendix A. Change history

(This appendix to be removed in the final draft.)

A.1. Changes from draft-02, 2008.07.11

Added language to require the slash ('/') as path separator, regardless of the platform where the bag was created. Added an extra co-author and an Acknowledgements section.

Deleted the unnecessary "(optional)" from four of the metadata elements, since all metadata elements are optional. Softened the equivalence of the serialization name and name of the contained bag base directory. Replaced the reference to RFC2822 with an inline description of the simpler bag-info.txt format.

Changed to a variable linear whitespace separator in the description of manifest layout and in manifest examples. Added two paragraphs under a new "Checksum tools" subsection of the Interoperability section to describe some of the peculiarities of dealing with the widely used GNU Coreutils checksum tools.

With the new version, 0.96, there is an important and incompatible change of file name (package-info.txt -> bag-info.txt), metadata element names (Package-Size -> Bag-Size, Packing-Date -> Bagging- Date), and descriptive language to replace the noun "package" with "bag" throughout the spec. This was to reduce unnecessary synonymy and free up the noun "package" to name the physical container (e.g., a mailing carton) used to transfer hard disks.

In section 7, another important change is the introduction of the Payload-Oxum ("octetstream sum") metadata element to convey precise, machine-readable payload size information for capacity planning (especially useful when preparing to receive files listed in fetch.txt). The Bag-size definition was adjusted to steer it more towards human consumption.

In section 2.2 the spec now requires exactly two spaces between checksum and filename in manifests. This results from the experience that as of 2008, not all widely available validation tools are flexible in the kind of separating whitespace recognized. The examples have been updated to include use the two-space form as well.

Comment added that while overall bag size is unlimited, practical limitations on the amount of data that a receiver can stage may warrant splitting a large bag into several smaller bags.

Added a reference to the SWORD protocol.

Minor edits for scanning and reformatting to cut down line length for some figures that exceeded 72 chars (limit for Internet-Drafts).

A.2. Changes from draft-01, 2008.05.30

Added mention of preserving empty directories.

Simplified function of "tag checksum file" to "tag manifest", having same format as payload manifest. The tag manifest is optional and need not include every tag file.

Loosened interpretation of payload manifest to "union" concept: every payload file must be listed in at least one manifest but need not be listed in every manifest.

Shortened the Introduction's first paragraph to be less duplicative of text in the Abstract.

Changed Delivery-Date to Packing-Date.

Correctly sorted the author list and clarification of deserialization wording.

A.3. Changes from draft-00, 2008.03.24

Author address corrections and miscellaneous stylistic edits.

Added some mention of physical media-based transfers, preferred characteristics of transfer filesystems, and network transfer issues.

Added basic bag example early and changed the narrative to more clearly delineate component files.

Wording changes under fetch.txt, and note that fetch.txt will need to be modified before bag return.

Fixed checksum encoding reference to base64 rather than hex. (B. Vargas)

Described simple normalization approach for checksum algorithm names. (B. Vargas)

In the example bag, add the ARC files found in the fetch.txt to the manifest as well (A. Turoff)

Authors' Addresses

Andy Boyko
Library of Congress
101 Independence Avenue SE
Washington, DC 20540
USA
Email: andy@boyko.net

John A. Kunze
California Digital Library
415 20th St, 4th Floor
Oakland, CA 94612
US
Fax: +1 510-893-5212
Email: **jak@ucop.edu**

Justin Littman
Library of Congress
101 Independence Avenue SE
Washington, DC 20540
USA
Fax: +1 202-707-1957
Email: **jlit@loc.gov**

Liz Madden
Library of Congress
101 Independence Avenue SE
Washington, DC 20540
USA
Fax: +1 202-707-1957
Email: **emad@loc.gov**

Brian Vargas
Library of Congress
101 Independence Avenue SE
Washington, DC 20540
USA
Fax: +1 202-707-1957
Email: **brian@ardvaark.net**