

The BagIt File Packaging 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 descriptive "tags" 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 storage and transfer.

Table of Contents

- 1.** Introduction
- 2.** BagIt directory layout
 - 2.1.** Tag files
 - 2.2.** Declaration that this is a bag: bagit.txt
 - 2.3.** Payload file manifest: manifest-<algorithm>.txt
 - 2.4.** Tag file manifest: tagmanifest-<algorithm>.txt
 - 2.5.** Example of a basic bag
- 3.** Valid bags and complete bags
- 4.** Other tag files
 - 4.1.** Completing a bag: fetch.txt
 - 4.2.** Other bag metadata: bag-info.txt
 - 4.3.** Another example bag
- 5.** Bag serialization
- 6.** Disk and network transfer
- 7.** Interoperability (non-normative)
 - 7.1.** Checksum tools
 - 7.2.** Windows and Unix file naming
- 8.** Security considerations
 - 8.1.** Special directory characters
 - 8.2.** Control of URLs in fetch.txt
 - 8.3.** File sizes in fetch.txt
- 9.** Acknowledgements
- 10.** References
- Appendix A.** Change history
 - A.1.** Changes from draft-03, 2009.04.11
 - A.2.** Changes from draft-02, 2008.07.11
 - A.3.** Changes from draft-01, 2008.05.30
 - A.4.** Changes from draft-00, 2008.03.24
- §** Authors' Addresses

1. Introduction

BagIt is a hierarchical file packaging format designed to support disk-based or network-based storage and transfer of generalized digital content. A bag consists of a "payload" and "tags". The content of the payload is the custodial focus of the bag and is treated as semantically opaque. The "tags" are metadata files intended to facilitate and document the storage and transfer of the bag. The name, BagIt, is inspired by the "enclose and deposit" method **[ENCDEP]**, sometimes referred to as "bag it and tag it".

Implementors of BagIt tools should consider interoperability between different platforms, operating systems, toolsets, and languages. In particular, differences in path separators, newline characters, reserved file names, and maximum path lengths are all possible barriers to successfully moving bags between different systems. Discussion of these issues may be found in the Interoperability section of this document.

2. BagIt directory layout

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

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

The tags consist of one or more files named "manifest-*algorithm*.txt", a file named "bagit.txt", and zero or more additional files.

2.1. Tag files

In top-level text files with a ".txt" extension, each line should be terminated by a newline (LF), a carriage return (CR), or carriage return plus newline (CRLF). In all such tag files except "bagit.txt", text is assumed to be encoded in the format specified in the "bagit.txt" file.

2.2. 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. The "bagit.txt" file must be encoded in UTF-8 **[RFC3629]**. The appropriate version for a bag that conforms to this version of the specification is "0.96".

2.3. Payload file manifest: manifest-*<algorithm>*.txt

A payload manifest is a tag file listing payload files and corresponding checksums generated using a particular cryptographic checksum algorithm. A payload manifest file has a name of the form manifest-*algorithm*.txt, where *algorithm* is a string specifying a 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 bag may only contain a single payload manifest for a particular checksum algorithm.

Each line of a payload manifest file 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. One or more linear whitespace characters (spaces or tabs) separate CHECKSUM from FILENAME. There is no limitation on the length of a pathname. To facilitate bag interchange, implementors are encouraged to read the Interoperability section. As described below, if listed at all, tag files should be listed in a tag manifest file, not in a payload manifest file.

Payload manifests cannot account for empty directories because they only include the pathnames of files. To account for an empty directory, a bag creator may wish to include at least one file in that directory; it suffices, for example, to include a zero-length file named ".keep".

2.4. Tag file manifest: tagmanifest-*<algorithm>*.txt

A tag manifest is a tag file listing tag files and corresponding checksums generated using a particular cryptographic checksum algorithm. A tag manifest file has a name of the form "tagmanifest-*algorithm*.txt", where *algorithm* is a string specifying a checksum algorithm. For example, a tag manifest file using SHA1 would have the name

```
tagmanifest-sha1.txt
```

A bag may only contain a single tag manifest for a particular checksum algorithm.

A tag manifest file has the same form as the payload file manifest file described earlier, but must not list any payload files. As a result, no FILENAME listed in a tag manifest begins "data/".

2.5. 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
|   (49afbd86alca9f34b677a3f09655eae9 data/27613-h/images/q172.png)
|   (408ad21d50cef31da4df6d9ed81b01a7 data/27613-h/images/q172.txt)
|   tagmanifest-md5.txt
|   (27afbd86alca9f34b677a3f09655eaa4 manifest-md5.txt)
|   (59ad21d50cef31da4df6d9ed81b01b01 bagit.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 ...)
```

3. Valid bags and complete bags

A bag is considered *complete* if every file in every payload manifest and tag manifest is present, and if every payload file appears in at least one payload manifest. A payload file does not need to appear in every payload manifest as long as it appears in one payload manifest (i.e., it must belong to the "union" of payload 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.

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.

Note that tag files (including tag manifest files) can be added to or removed from a bag without impacting the completeness or validity of the bag as long as the tag files do not appear in a tag manifest.

4. Other tag files

4.1. 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, relative to the base directory. Only the slash character ('/') may be used as a path separator in FILENAME. If FILENAME begins with a slash character, the destination must still be treated as relative to the base directory. One or more linear whitespace characters (spaces or tabs) separate these three values, and any such characters in the URL must be percent-encoded [RFC3986]. There is no limitation on the length of any field in "fetch.txt".

The "fetch.txt" 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).

4.2. Other 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 readability. All metadata elements are optional.

A metadata element consists of a label, a colon, and a value. Whitespace after the first non-whitespace in the value is considered part of the value. Long values may be folded (continued) onto the next line by inserting a newline (LF), a carriage return (CR), or carriage return plus newline (CRLF) and indenting the next line (any combination of spaces and tabs). It is recommended that lines not exceed 79 characters in length.

Reserved metadata 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.

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 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`") and should be included in "bag-info.txt" if at all possible. Compared to Bag-Size (above), Payload-Oxum is 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.

Internal-Sender-Description

A sender-local prose description of the contents of the bag.

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

Here is an example "bag-info.txt" file.

```
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...
```

4.3. Another example bag

The following example demonstrates a partially "holey" bag that includes a "fetch.txt" file and a "bag-info.txt" file. 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 complete until every file 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-20060601-050019.arc.gz)
|   (55c7c80c6635d5a4c8fe76a940bf353e data/gov-20060601-100002.arc.gz)
|
|   fetch.txt
|   (http://foo.example.com/gov-06-2006/gov-20060601-050019.arc.gz
|       26583985 data/gov-20060601-050019.arc.gz)
|   (http://foo.example.com/gov-06-2006/gov-20060601-100002.arc.gz
|       99509720 data/gov-20060601-100002.arc.gz)
|
|   bag-info.txt
|   (Source-organization: California Digital Library)
|   (Organization-address: 415 20th St, 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.)
|   (Bag-date: 2008.04.15)
|   (External-identifier: ark:/13030/fk4jm2bcp)
|   (Bag-size: about 22Gb)
|   (Payload-Oxum: 21836794142.4)
|   (Internal-sender-identifier: UC DL)
|   (Internal-sender-description: UC 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 ...)
```

5. Bag serialization

In some scenarios, it may be convenient to serialize the bag's base directory (i.e., the filesystem hierarchy representing the bag) into a single-file archive format such as TAR or ZIP (which might involve a compression step). The resulting *serialization* may later be deserialized (which might involve an uncompression step) to recreate the filesystem hierarchy. Several rules govern the serialization of a bag and apply equally to all types of archive files:

1. One and only one bag appears in the top-level directory of a 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 serializing a bag, 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 requirements of the systems on which it will be used. See the section on Interoperability.

6. 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]**.

7. Interoperability (non-normative)

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

7.1. 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. Implementors creating or processing bags with md5sum should be aware of these subtle differences, and ensure compliance with the manifest specification in this document. Implementors creating and processing bags with other tools may wish to be tolerant of asterisks found in the manifests.

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.

7.2. 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.

8. Security considerations

8.1. Special directory characters

The paths specified in the payload file manifest, tag file manifest, and "fetch.txt" file do not prohibit special directory characters which might be significant on implementing systems. Implementors should take care that files outside the bag directory structure are not accessed when reading or writing files based on paths specified in a bag (cf Unix chroot).

For example, path characters such as "." or "~" in a maliciously crafted "fetch.txt" file might cause a naive implementation to overwrite critical system files.

8.2. Control of URLs in fetch.txt

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. Checksums are not a guarantee against corruption and spoofing in bag transfer.

8.3. File sizes in fetch.txt

The size of a file, as optionally reported in the "fetch.txt" file, cannot be guaranteed to match the actual size of the file to be downloaded. Implementors should not use the file size in "fetch.txt" for critical resource allocation, such as buffer sizing or storage requisitioning.

9. Acknowledgements

BagIt owes much to many thoughtful contributors and reviewers, including Stephen Abrams, Mike Ashenfelder, Dan Chudnov, Brad Hards, Scott Fisher, Keith Johnson, Erik Hetzner, Leslie Johnston, David Loy, Mark Phillips, Tracy Seneca, Brian Tingle, Adam Turoff, and Jim Tuttle.

10. 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 File 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 Repository Deposit \(SWORD\)](#)," 2008 ([HTML](#)).
-

Appendix A. Change history

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

A.1. Changes from draft-03, 2009.04.11

Tweaked "tags" to leave as metaphor and added definition of the previously undefined term, "tag file(s)". Added implicit definition of "serialization". [Mental note: the sentence, "Tag information is optional." no longer appears in this spec.] (John)

Re-worded interoperability statement in the Introduction. (Justin)

Added statements regarding no limitations on various paths, URI, and other lengths.

Clarified that the bag directory may not contain any other directories except for the "data"

directory.

A soel carriage return character is now explicitly allowed as a valid line separator.

Tag file encoding requirements are now required to be as-stated in the "bagit.txt". The "bagit.txt" file is explicitly required to be in UTF-8.

Wording cleanup, clarifying payload file manifests and tag file manifests.

Tags in "bag-info.txt" no longer have any ordering requirement.

Tag formatting now explicitly states where significant whitespace begins in the tag.

After some consideration, added some security considerations.

Made it clear that a bag may contain other bags, re: serialization.

Re-worded interoperability to concerns to require creators to be spec-compliant, and readers to be tolerant of known potential issues.

Specificity to the FILENAME element in "fetch.txt" is relative to the bag root, and to make sure to treat leading slashes as relative.

Updated acknowledgements.

Various other minor edits for clarity and readability.

A.2. 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.3. 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.4. 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