The Year of Content: Learning from Experience Transferring Digital Content across the NDIIPP Network

Michelle Gallinger; Leslie Johnston
Library of Congress (USA)

The Library of Congress and the National Digital Information Infrastructure and Preservation Program (NDIIPP) network partners declared 2008 "The Year of Content." As a part of the Year of Content, the NDIIPP network and the Library focused on a program of digital content transfer. The NDIIPP network comprises libraries, archives, universities, research centers, non-profit and for-profit organizations and professional associations across the United States as well as a wide variety of international partners (Anderson 2008). The NDIIPP partners participated in developing and testing standards, best practices, and technological solutions for digital preservation challenges. A number of NDIIPP projects acquired and preserved at-risk digital content and worked to send a copy of the content to the Library for safekeeping. This effort was accomplished in 2008.

Selecting, packaging, and sending content highlighted the challenges of content transfer as well as the importance of this element of the preservation lifecycle. Many of the partners found that moving digital content to the Library was beneficial to their own work, refining their own stewardship processes and strengthening common standards and practices in the community.

The NDIIPP network gained practical experience moving digital information from one institution to another and, in the process, learned and confirmed digital preservation lessons that inform the ever-changing roles of curators and stewards.

Learning By Doing

Content was moved to the Library primarily by bulk transfers of digital files, using a combination of manual and automated processes that were negotiated between the Library and each partner to establish workflows. Through these experiences we have gained experience in transfer processes, and have been able to simplify the complexity of those initial approaches. As the content packaging, transfer, and inventorying approaches have become increasingly standardized, it has become possible to automate and improve the efficiency of transfers through the development of a suite of tools

Content transferred to the Library includes digital public television programs such as *Nature;* geospatial information ranging from satellite images, historic maps, and transportation infrastructures; captured web sites featuring government and news content; electronic journals, cartoons, and social science datasets.

The first content transferred were collections of Doonesbury and Oliphant comics from the Universal Press Syndicate (UPS). UPS transferred these collections via File Transfer Protocol (FTP). This experience emphasized need for a specification that included standardized methods for dealing with checksums to verify the fixity of the files. One of the files transferred with an error rendering it unusable. We were able to identify the file that had the problem, using the checksums sent by UPS with the files; however, because the checksums and the files were not communicated in a standardized way, the identification was done manually rather than automatically. This worked for an initial transfer but was recognized as being unsustainable for the many transfers expected. A more regularized system was needed.

At the same time, the Library was also working with the California Digital Library (CDL) to transfer its data. The strong interest in developing a more standardized transfer from the Library transfer team was met by an equally strong interest on the part of the CDL team. The groups collaborated on developing a transfer specification that was eventually refined into BagIt (see Appendix A) and adopted for all transfer activities at the Library.

Problems with early transfers that occurred during the development of the specification helped refine it. An early transfer from Stanford University using a preliminary version of BagIt showed an unexpected error. A file did not match the manifest description even though it did match the file on the server it came from. This anomaly helped refine the specification, which now requires that the manifest itself have an accompanying checksum, and also the transfer workflow in which the manifest is transferred and validated first with the content following.

Decisions on which transfer tools to use were also influenced by the actual transfers taking place. Issues that came into play for each partner and set of content included: availability and type of resources, total size and number of files, bag size, and the technical environment (e.g., network options, availability of Internet2 at partner location, etc). Each of the early transfers refined the Library and partner workflow. Each of the tools used offered specific benefits and it was through use of each that the Library was able to evaluate each and determine the most practical tools and guidelines for general use. Many tools were explored. Transfers were made using File Transfer Protocol (FTP), Fast Data Transfer (FDT), Logistical Storage (L-store, www.lstore.org), HTTP, Lots of Copies Keeps Stuff Safe (LOCKSS, www.lockss.org), rsync and hard drives sent through the mail.

Early in the year of content, UPS transferred their numerous but relatively small collection via FTP. This transfer was relatively easy, employing a well-known tool to good effect. However, the speed of transfer was such that FTP was impractical for use by some of the NDIIPP partners with larger collections.

Stanford University and University of California at Santa Barbara (UCSB) attempted to send their National Geospatial Data Archive (NGDA) collections to the Library via FDT. Early experiments with FDT indicated that it offered extremely fast transfer, but calibrating the threads required more time on both

ends of the transfer than either the Library or the NDIIPP partners could really dedicate.

At the same time, Stanford University and UCSB worked with University of Tennessee at Knoxville and Vanderbilt University to transfer data to the Library using L-store. L-store is designed to provide a flexible logistical storage framework for distributed, scalable, and secure access to data for many different users. It is designed to provide a decentralized management system, user controlled replication and striping of data on a file and directory level, a virtual file system interface in both a web and command line form. It also supports the concept of geographical locations for data migration to facilitate quicker access. While providing robust storage options, L-store proved challenging to use for the transfer of NDIIPP data in the period of this transfer project. At that time, L-store was unable to support bags conforming to the emerging BagIt specification. Earlier transfers of data from the Vanderbilt TV archive using L-store were successful but these transfers did not conform to the BagIt specification. The Library decided to continue transferring this video content using L-store because of the previous success. However, we determined that new transfers should conform to the emerging specification. The NGDA data coming from UCSB and Stanford University was ultimately transferred almost exclusively using BagIt.

Many of the NDIIPP partners ultimately transferred their data using rsync, including CDL, Stanford University, UCSB, and WNET of the Preserving Public Television project. Rsync was first released in 1996. It is a well-established software application that synchronizes files and directories from one location to another. An important feature of rsync not found in most similar programs or protocols is that the mirroring takes place with only one transmission in each direction. Rsync can copy or display directory contents and copy files, optionally using compression and recursion. This allowed the transfers to run on multiple threads and speeds the rate of transfer completion.

Rsync was also familiar to most NDIIPP partners. It allowed bags of content to be transferred via the network at the approximate rate of 1 TB/day. This speed has improved as the transfers continued and have been optimized.

The Library and several of the NDIIPP partners have had great success transferring bags via rsync, but it was not practical for all of the partners to transfer their content over the network. Some partners only had connections to the commodity Internet (rather than the higher-speed Internet2), some did not have personnel with time or expertise to set up a software application, and some partners were confronted with security requirements that restricted their use of network-based protocols. Hard disk transfers met the needs of these partners.

University of Maryland transferred a collection of business plans from the Dot-Com Archive via flash drive sent though the mail. North Carolina State University transferred some of the geospatial information that their project has preserved via hard drives also mailed to the Library. Hard drive transfers offered some distinct advantages. It was a well understood technique, employed by the Library in the National Digital Newspaper Project (NDNP), and the NDIIPP partners were equally experienced in transferring content with hard drives. It became a standard second-choice for transfers. However, because of delays in mailing content, the need to return hard drives, and the obvious benefits of increasing network transfer speeds and experience, hard drive transfer will not be a primary mechanism of transfer going forward.

The Library is proposing that HTTP transfer will eventually take the place of some of the hard drive transfers. HTTP transfers will not require the same resources or expertise as rsync transfers. Instead, a well-defined HTTP transfer process will take advantage of the common ability of almost all NDIIPP partners to access web pages for transfer.

Another transfer mechanism that was explored was LOCKSS. Transfers of content from the MetaArchive were made by the Library becoming a member of the MetaArchive Private LOCKSS Network (PLN). This required the Library to set up a LOCKSS box and open it to the private network. A PLN has materials published to the LOCKSS boxes which then collect it for preservation. LOCKSS software continuously compares the copies stored in other LOCKSS boxes in the private network as part of a process to allow repair of lost or corrupted copies. The LOCKSS boxes communicate over the Internet to continually audit the content they are preserving. If the content in one LOCKSS box is damaged or incomplete, that LOCKSS box will receives repairs from the content based on other LOCKSS boxes. This automated cooperation between the LOCKSS boxes avoids the need to back them up individually. It also provides unambiguous reassurance that the system is performing its function and that the correct content is available throughout the network of replicated boxes.

In this way, the MetaArchive collections arrived at the Library, but have never been formally transferred using BagIt. Rather, the Library acts as another member in the PLN. The “transfer” in this instance is not about a complete handoff of data from one institution to another but about the constant update of a collection through a preservation and storage mechanism.

The exploration of transfer tools, techniques, and mechanisms led to collaboration between the Library and many NDIIPP partners that produced a remarkably simple and valuable specification and a set of automated tools and guidelines that will continue to be leveraged for ongoing preservation activities at the Library and throughout the preservation community.

The Transfer Tools and Specifications

Working with John Kunze of the California Digital Library and Keith Johnson of Stanford University, Andy Boyko, Justin Littman, Liz Madden, and Brian Vargas of the Library produced a generalized version of what had been initially referred to as the "LC Package Specification,. This is now called "**BagIt.**" For the purpose of transferring content, a package is a set of files stored in a file system, which may be a subset of a larger collection of content, to be transferred and managed as a unit. The set of files comprising a package may be transferred as a single file in a container format such as ZIP or tar to be unpacked upon receipt.

BagIt is a hierarchical file package format designed to support the transfer of any digital content. The specification builds on the idea of “bagging and tagging” data, organizing content in a directory and labeling the data in that directory. BagIt organizes files to be transferred and manages them as unit. This eases the transfer process by providing a simple, predictable unit of transfer

that is effective regardless of domain, format, or size of the content being transferred.

The base directory of a Bag contains a **file manifest**, a **content directory**, and an optional **package information** directory. The **content directory (/data)** contains the contents of the package, as determined by the producer of the bag. The file manifest lists the names and checksums of the content files and the package information files—excluding itself and any shipping files. The file manifest is used to provide an inventory and validate the fixity of each of the transferred files. The manifests assist in the transfer, archiving, and preservation of the package as a unit, rather than supplying any description of the content. Neither the file manifest nor the package manifest obviates the need for descriptive metadata being supplied by the package producer or provider.

The content directory may have any name and internal structure. There is no limit on the number of files or directories this directory may contain. The size of a Bag should be based on physical media limitations or expected network transfer rates. The Library recommends 500 Gb as the maximum size (based on practical experience), although Bags as large as 1.8 Tb have been transferred successfully.

A number of tools have been developed for retrieval, receiving and managing of content transfers with the BagIt standard (Johnston 2009). The **Parallel Retriever** implements a simple Python-based wrapper around wget and curl, capturing files and producing a package that meets the BagIt specification when given a file manifest and a fetch.txt file. It was initially built specifically for transfers to the Library via rsync, but has been extended to HTTP and FTP. The Parallel Retriever has been released by the Library as open source on SourceForge (http://sourceforge.net/projects/loc-xferutils/).

The Library has created a **Bag Validator script.** This script checks that all files listed in manifest are in the data directory; there are no files in the data directory that are not listed in manifest; and there are no duplicate entries in the manifest. The **VerifyIt script** is used to verify the checksums of files in a Bag against its manifest. The Bag Validator and VerfiyIt have also been released by the Library as open source on SourceForge (http://sourceforge.net/projects/loc-xferutils/).

A client-side **Bagger** application has been developed for partners engaged in small-scale transfers. Bagger automates the packaging and submission of locally-hosted content without requiring the Library involvement. Ideally, this requires no client-side IT support or infrastructure and can be used by non-technical content curators or producers. This tool will be equally suited for packaging and transferring digital files from fixed media (such as DVDs or CDs) to enterprise transfer and storage environments. It is expected that this can be used within an institution, such as the Library itself**. Bagger** is implemented as a Java Web-Start application for use across platforms, and supports the aggregation of files into Bag packages, including the creation of checksum manifests and Bag information files. This application was, in part, built on top of **BIL**—the BagIt Library—a Java library developed to support Bag services. BIL is also on SourceForge (http://sourceforge.net/projects/loc-xferutils/) released as open source software by the Library of Congress.

In order to support the expanding numbers and types of transfers, the need for additional types of software tools became clear. The **Deposit service** is a web-hosted application for use by transfer partners in registering and negotiating a new transfer. This application will support the registration and initiation of the transfer content both via network transfer and via fixed media, such as hard drives or DVDs. As of early 2009, the **Deposit service** is mid-way through the production implementation process, including review by representatives of multiple digital content acquisition projects.

The **Inventory Tool** implements a domain model for packages, a suite of command line inventory tools, and a reporting web application. The **Inventory Tool** collects and maintains data by recording life cycle events on a package level and on a file level. Examples of Package Events include “Package Received Events,” which are recorded when a project receives a package; and “Package Accepted Events,” which are recorded when a project accepts curatorial responsibility for a package. Examples of File Location Events include “File Copy Events,” which are recorded when a package is copied from one File Location to another; and “Quality Review Events,” which are recorded when quality review is performed.

All the tools can be tied together into any of a number of project-based **Workflow systems** (see Appendix B for a generic project-based data flow diagram). The underlying workflow process steps are formalized with a graphical interface on top of them. With this graphical user interface, users can identify lists of tasks to be performed, initiate, monitor and administer processes; and notify the workflow engine of the outcome of manual tasks, including task completion. Workflow tasks can include transfer, validation by project-specific validation applications such as the one in production for the NDNP (Littman 2009), manual quality review inspection, and file copying to archival storage and production storage. Both the **Inventory Tool** and all workflows are built on top of the **BIL** Java Library.

In late 2008 the Library for the first time released its own open source software. The first of the Transfer tools—the Parallel Retriever, the Bag Validator, VerifyIt, and BIL—have all been released by the Library as open source on SourceForge (http://sourceforge.net/projects/loc-xferutils/). The Library plans to release additional tools as part of a suite of solutions and software development resources as they are completed over time, and expects to continue to maintain and refine the tools with more experience.

NDIIPP Partner Workflow

The transfer tools and specifications that the Library and various NDIIPP partners developed have transformed digital preservation workflow into a true network with well-defined interfaces. Several NDIIPP partners have adopted "Bagged" transfer and several of the tools developed by the Library of Congress in their own preservation practices and for use with other projects and partners.

The content transfers from Stanford University to the Library have inspired changes to the Stanford Digital Repository (SDR) output. The SDR is now outputting all content that is output from the repository in the form of Bags.

New York University (NYU), the technical partner of the Preserving Public Television (PTV) project (Pawletko 2008), is using Bags for general content transfer within their project. Bags

are employed when transferring content from the NYU digitizing stations to the NYU storage servers, from outside vendors to the NYU repository, to transfer submission information packages from content producers to the NYU repository, and also to transfer content from NYU to other repositories. The PTV project and NYU are developing scripts of their own to automate transfer processes including a source-node script that 'bags' content and a target-node script to validate a bag. NYU is developing its own system for handling receipts and acknowledgements of transferred bags. NYU is also exploring fetch.txt as a way of facilitating content processing at the target node.

The North Carolina Geospatial Data Archiving Project (NCGDAP) has been working with very large geospatial data collections originating from small, local agencies (Morris 2008); such as hundreds of Gb of orthophotos (aerial photographs that have been corrected to have uniform scale, a photo-map) from a single county. The small, local government agencies have limited network capacity and require simple network transfer solutions. Therefore the project transfers data from these agencies via external hard drives. This orthophoto "sneakernet" between agencies has been a good transfer method but has lacked data integrity management in the disk transfers. Bags have offered a way for NCGDAP to provide the data integrity management they needed without adding to the burden of the local agencies.

Conclusion

These transfer tools and processes are important advances for digital stewardship. Our initial interest came from the need to better manage transfers from NDIIPP partners to the Library, at the same time that the transfer and transport of files within the organization for the purpose of archiving, transformation, and delivery is an increasingly large part of daily operations. Developing tools to manage such transfer tasks reduce the number of tasks performed and tracked by humans, and automatically provides for the validation and verification of files with each transfer event.

After much experimentation, the best transfer practices that emerged during the Year of Content relied upon established, reliable tools; well-defined transfer specifications; and good communication between content provider and content receiver. Each transfer provided insight into the developing content transfer best practices and each exchange brought more expertise. The digital preservation community continues to engage with transfer best practices, helping these practices to evolve.

The digital preservation community not only helped develop transfer tools, standards, and best practices, but is applying them to their own preservation lifecycles in innovative ways. The community is always refining digital preservation practices based on the newest developments, and transfer related activities are no exception. Digital preservation practitioners have adopted these transfer best practices to ingest, backup, export, and other activities. They have come to believe, as Justin Mathena of UCSB has commented, "it is **all** transfer."

Ultimately, the practices and tools that arose from the Year of Content focused on not just on transfer optimization, but on ways in which to improve the communication between submitter and receiver. The most important part of transfer is not the technical connection but the exchange of information. Communicating what is coming, when it will arrive, what form it will take and making the process predictable and flexible has been the most valuable result of the 2008 efforts to bring NDIIPP partner content to the Library of Congress.

These transfer developments are important for the digital preservation community beyond the NDIIPP network. Information creators have ever-growing data volumes that require transfer to a long-term stewardship entity, and these tools will be of great help. Archives, libraries, and other content stewards also can use the tools to distribute one or more copies of digital content among different locations to protect it from catastrophic loss.

References:

[1] Anderson, Martha. 2008. "Evolving a Network of Networks: The Experience of Partnerships in the National Digital Information Infrastructure and Preservation Program." *The International Journal of Digital Curation* (July 2008: Volume 3, Issue 1). http://www.ijdc.net/ijdc/article/view/59/60.

[2] Johnston, Leslie, 2009. "Identifying and Implementing Modular Repository Services." DigCCurr 2009: Digital Curation Practice, Promise and Prospects.

[3] Littman, Justin. 2009. "A Set of Transfer-Related Services." *D-Lib Magazine* (January/February 2009: Volume 15, Number 1/2). http://dlib.org/dlib/january09/littman/01littman.html.

[4] Morris, Steve. "The NC Geospatial Data Project Archiving Experience." Digital Library Federation Fall 2008 Forum. http://www.diglib.org/forums/fall2008/presentations/Morris.pdf

[5] Pawletko, Joseph. "NDIIPP: Preserving Digital Public Television." Digital Library Federation Fall 2008 Forum. http://www.diglib.org/forums/fall2008/presentations/Pawletko.pdf

Appendix A: BagIt Specification

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, sometimes referred to as "bag it and tag it".

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.

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.

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. A manifest contains a complete list of files that must be present in a fully constituted bag.

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. 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.

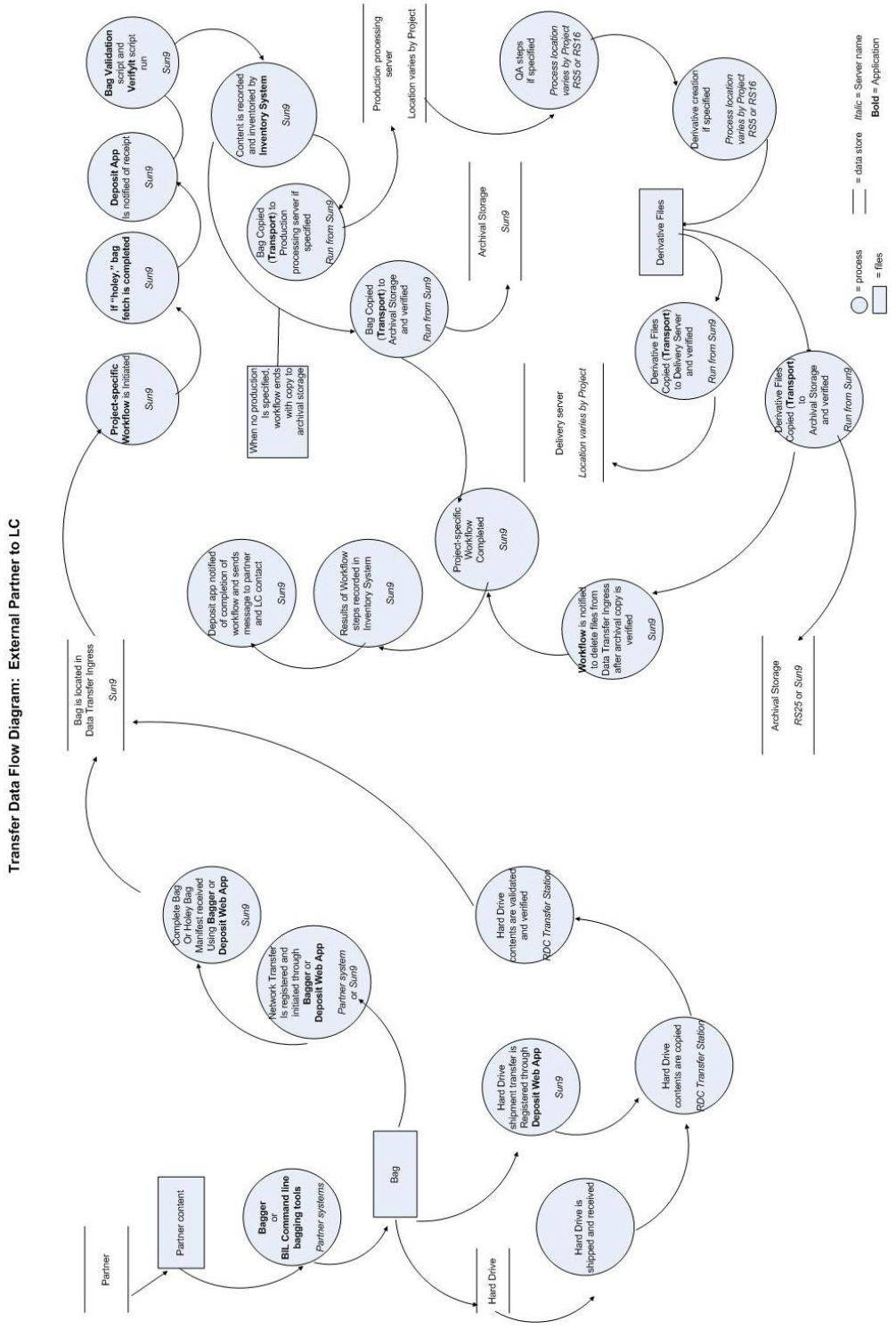
The "fetch.txt" file essentially allows a bag to be transmitted with "holes" in it, which can be practical for several reasons. For example, it obviates the need for the sender to stage a large serialized copy of the content while the bag is transferred to the receiver. Also, this method allows a sender 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).

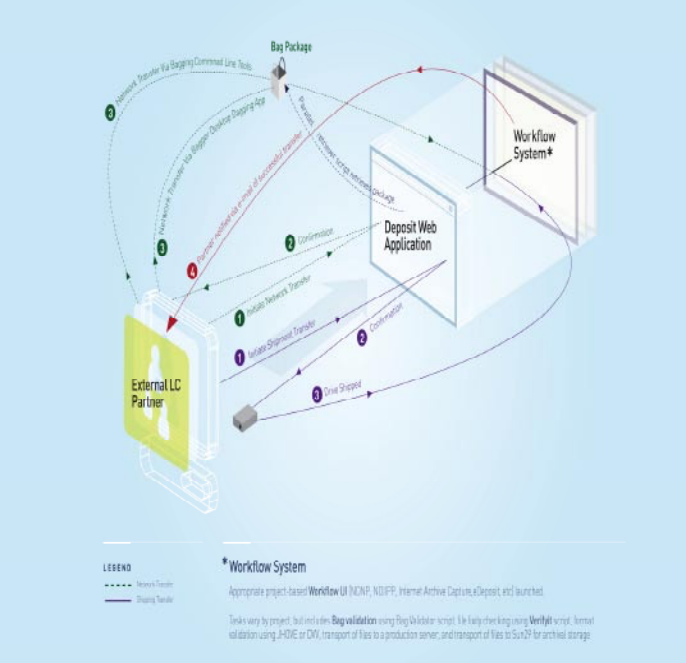
In some scenarios, such as network transfer, it may be convenient for the sender first to serialize the filesystem hierarchy representing the bag (the outermost base directory) into a single-file archive format such as TAR or ZIP. After receiving the resulting aggregate file, which we will call a *serialization*,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

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.

The most current version of the BagIt specification is available at http://www.digitalpreservation.gov/library/resources/tools/docs/bagitspec.pdf.

Appendix B: Generic Project-Based Transfer Tool Data Flow Diagram





Appendix C: "FetchIt" Parallel Retriever

The Parallel Retriever (alternately referred to as "FetchIt") implements a Python-based wrapper around wget and rsync to run a parallelized multi-threaded process for efficient file transfers. When given a "file manifest" and a "fetch.txt" file it completes a bag that conforms to the BagIt specification. It is used to transfer content from rsync, FTP, and HTTP servers. The Parallel Retriever relies on two operating scripts:

- parallelretriever.py
  - Used for rsync and FTP transfers of complete and holey bags.
    - Implements a Python-based wrapper around wget and rsync to run a parallelized multi-threaded process for efficient file transfers.
  - Python script invoked from the command line.
  - Takes as input:
    - The file manifest
    - The retrieval order
    - A local identifier I for the package.
    - The fetch.txt if it is a Holey Bag
  - Takes as configuration:
    - A reasonable number of parallel processes based on local environment, general rules of thumb. (i.e. "16")
    - A destination directory into which to place the retrieved package, a means of notifying a coordinating system that the retrieval completed, and its success or failure (e.g. a 'messaging' destination)
  - Outputs a text log for success, failures, errors.
- fetch-it-http-ranges.sh
  - Used only for parallel transfers of large serialized Bags via HTTP 1.1 byte ranges.
  - Bash shell script invoked from the command line.
  - Takes as input:
    - The file manifest
    - The retrieval order
    - A local identifier I for the package.
  - Takes as configuration:
    - A reasonable number of parallel processes based on local environment, general rules of thumb. (i.e. "16")
    - A destination directory into which to place the retrieved package, a means of notifying a coordinating system that the retrieval completed, and its success or failure (e.g. a 'messaging' destination)
  - Outputs a text log for success, failures, errors.

Appendix C: GrabIt and SWORD

GrabIt is a proposed HTTP-based protocol for systematically transferring lists of digital packages. A package is an arbitrary sequence of octets (e.g., a single TAR or ZIP archive) that can come with an associated content type indicating that extra steps (not central to GrabIt) may need to be taken to complete and validate a package before successful receipt can be reported.

A GrabIt job is initiated by a HTTP "post" request with a list URLs corresponding to packages, sent to a base URL that a receiver has made known to a sender by a pre-arranged process outside the scope of GrabIt. Package lists and package report lists are text files. It is assumed that the receiver maintains a drop box at that location and that the sender has the right to "post" to it.

The receiver responds to a GrabIt request by promptly returning a text file that is the initial transfer status report. The report includes a job URL in an HTTP "Location" header that can be probed periodically to retrieve transfer status reports. There is no requirement that the transfer be finished and immediate reporting is encouraged. Transfer operations implied by GrabIt jobs often take minutes to hours to complete and reporting early and often is normal. There are no GrabIt responses that are not transfer status reports.

Authentication is not addressed by GrabIt. It is assumed that senders and receivers will make their own arrangements and take measures (certificates, SSL, passwords, IP address filtering, etc.) appropriate to their transfer security requirements.

The GrabIt specification is available at: http://dot.ucop.edu/home/jak/grabitspec.html.

SWORD is a profile of the Atom Publishing Protocol (APP), which is an application-level protocol for publishing and editing Web resources. The APP is based on HTTP transfer of Atom-formatted representations. The SWORD Profile specifies a subset of elements from the APP for use in depositing content into information systems, such as repositories. The SWORD profile is concerned only with the deposit (POST) of data files or packages and defines a mechanism for POST. The SWORD Profile also specifies a number of element extensions to APP. This profile also makes use of the Atom Syndication Format as used in APP, with extensions. The Profile was produced to support the work of the JISC-funded SWORD ((Simple Web-service Offering Repository Deposit) project (http://www.ukoln.ac.uk/repositories/digirep/index/SWORD_Project).

APP works by issuing HTTP requests (GET, POST). GET Service Document (explain/discover). POST ATOM document or file to collection URI. HTTP response and ATOM document is returned. HTTP basic authentication is required.

SWORD does not at this time deal in update/delete, and is POST only. Only binary files can be posted, and SWORD does not specify how to post Atom documents. APP uses the MIME mechanism to describe the data encoding for resources. This mechanism is inadequate where compound types are involved, such as tar archive files, METS packages, SCORM packages, MPEG21 DIDL packages, etc. To this end, SWORD extends APP, adding the sword:acceptPackaging element, which is used in a similar fashion to app:accept elements inside a app:collection element within a Service Document to indicate that a SWORD collection will accept deposits of a particular packaging format. Atom Categories are not used.

The most current SWORD Profile is available at http://www.swordapp.org/docs/sword-profile-1.3.html . SWORD 2 is under development.