ECHO DEP Framework for Evaluation of Digital Repository Software

Adaptation of the RLG Checklist for Repository Software (little-r)

The organization of this paper is based upon the repository evaluation core activity checklist of the ECHO DEPository Project at UIUC. The attempt has been made to include the entire evaluation list, marking as such those sections considered out of scope at this time or otherwise not immediately relevant. We use the term "little-r" for repository software applications; this is to distinguish them from the institutional repository itself, or the "big-R." For continuity purpose, the preface to the checklist is quoted below:

Taken in part from the Audit Instrument for the Certification of Trusted Digital Repositories, Draft for Public Comment .

"This checklist is being used for the repository evaluation core activity of the ECHO DEPository Project at UIUC. Not all items from the following checklist are directly applicable to the evaluation of digital repository software systems. Those items which are mostly organizational in nature are shown in italic, and they will not be addressed directly by this evaluation. The remaining items will at least have some relevance to software systems, and will be assessed on a repository by repository basis. Since the original TDR checklist was focused primarily on organizational issues and not necessarily on software, each non-gray checklist item will be followed by one or more bulleted items explaining how the item might apply to repository software, and in some cases the item might also be mapped to related OAIS or PREMIS concepts."

About the annotations

This checklist has been annotated with **questions** that librarians and archivists (effectively, auditors) can ask when deciding whether their chosen repository software application adheres to the checklist item or not. The annotations are intended to give auditors a more complete sense of the ways in which the checklist item pertains to the repository software application. For some checklist items, however, the annotation provided only *suggests* how the checklist item applies to the software. In these instances we preface the annotation accordingly, in italics: "The following are examples of how this may be applied to repository software applications." Such annotations can be considered less definitive types of questions for the auditor to answer when considering a checklist item. Similarly, if a checklist item has more to do with the protocol of the organization where the repository is based, we assert so but also propose how the item may pertain to the software. Also, when an item relates to another one in the checklist, we have inserted cross-referencing phrases such as "See also," preceded by asterisks (****) and italicized.

Finally, since a revised version of the checklist came out in 2007, we have mapped the original checklist items to their equivalents in the new checklist, which is now called "Trustworthy Repositories Audit & Checklist," or TRAC.

Correlations to TRAC requirements	TDR Modified Checklist
	A. The Organization
	A5. Contracts, Licenses and Liabilities
	A5.1 If repository manages, preserves, and/or provides access to digital materials on behalf of another organization, it has and maintains appropriate contracts or deposit agreements.
	The following are examples of how this may be applied to repository software applications:
	a. Does the repository software application have any means to manage, store, or enforce these contracts or deposit agreements? b. Does the repository software application tie specific agreements to individuals or individual items in the repository or to collections of items in the repository?
	A5.3 Repository tracks and manages copyrights and restrictions on use as required by contract or license or deposit agreements. a. Does the repository software application have any capabilities—such as access control lists, Internet address filters, etc.—that can be used to enforce copyright or access restrictions? b. How granular are these access controls? (For example, can different restrictions be applied to different objects in the repository?)
	B. Repository Functions, Processes & Procedures
	B1. Ingest/acquisition of content
	B1.1. Repository identifies properties it will preserve for each class of digital object. a. What kinds of metadata (Representation Information) does the repository software application support out-of-the-box? b. How easy is it to find out what kinds of metadata it supports? c. How easy is it to customize the repository software application to support other kinds of metadata? d. Does the repository software application provide a means of referring to external metadata registries? E.g., the Global Digital Format Registry (GDFR). **** *See also B.3.3.*
	B1.3. Repository has an identifiable, written definition for each SIP or class of information ingested by the repository. *The following are examples of how this may be applied to repository software applications:* a. Are the supported file formats well documented? b. Are the supported metadata formats well documented?

	c. Can new file formats be added or removed? d. Can the metadata fields or formats be customized?
	B1.4. Repository has a process to ensure that the information is acquired from the expected source. a. Does the repository software application have capabilities that limit who is allowed to submit items? For example, access control lists, internet address filters, etc. b. Does repository software application have capabilities to limit who is allowed to modify or delete Digital Objects or Representation Information in the repository? c. Does repository software application have any means to verify that Digital Objects or Representation Information have not been tampered with from initial receipt to ingestion? For example, through the use of checksums or digitally signed checksums. d. Does repository software application maintain audit logs that identify by whom and when all changes to the Content Information were made?
	B1.5. Repository obtains sufficient physical control over digital objects to preserve them. a. Where does the repository software application store the actual digital objects and their Representation Information? b. Does the storage medium itself provide for sufficient security and reliability such as clear access controls on file systems and database systems?
B1.4 Repository's ingest process verifies each submitted object (i.e., SIP) for completeness and correctness as specified in B1.2. ***Note: In TRAC, B1.6 now reads, "B1.6 Repository provides producer/depositor with appropriate responses at predefined points during the ingest processes." (That is, old B1.7 maps to new B1.6—see row below.)***	**B1.6.** Repository's ingest process verifies each SIP for completeness and correctness. a. Does the repository software application verify the file format against the actual file that was submitted? b. If a SIP does not conform to the accepted formats what happens? c. Are there automated checks of the metadata such as to verify that a date entered into a field really is a date string? How does the repository software application verify that file format metadata is correct? d. Does the repository software application support workflow so that human reviewers can verify data after it is deposited, but before it is 'officially' accepted? e. How does the repository software application deal with incomplete data? With digital objects lacking sufficient or appropriate metadata – such as an XML file that references a DTD or XML Schema, but the DTD or Schema is not available? f. How does the repository software deal with inconsistent information – e.g., byte encodings of the content? For example, a Digital Object that is an XML file might incorrectly list some of its metadata Representation Information, such as its encoding scheme, as in the case of an XML declaration at the top of the file stating the file is encoded as UTF-8, when it is actually encoded as UTF-16.

B1.6	**B1.7.** Repository provides Producer/depositor with appropriate responses at predefined points during the ingest processes. a. How does the repository software application notify a producer/depositor that their submission has been rejected or accepted into the repository? b. Does the repository software application notify a producer/depositor of what is needed to have submission accepted? c. Does the repository software application monitor or guide the workflow?
	B1.8. Repository can demonstrate that all SIPs are either accepted as whole or part of an eventual AIP, or otherwise disposed of in a recorded fashion. How does the repository software application demonstrate that all SIPs are accepted as whole or part of an eventual AIP or otherwise disposed of in a recorded fashion? For example: a. When does a submission become managed by the repository software application itself – if it does? b. Does the repository software application record in an audit log each submission and transition from a SIP to an AIP? If so, is the audit log protected from intentional or inadvertent tampering? c. Does the repository software application support workflows such that a human can formally accept a package into the repository? If so, how does the software record this formal acceptance? d. Does the repository software application periodically send audit log reports to appropriate administrators? e. How does the repository software application keep track of accepted deposits vs. rejected deposits?
B2. Ingest: Creation of the Archival Package	**B2.** Archival storage: management of archived information
	B2.1. Repository has an identifiable, written definition for each AIP or class of information preserved by the repository. a. Does the repository software application have written documentation that describes what data structures it manages (e.g. primary object and alternate versions vs. compound objects with embedded images)? b. Does the repository software application have written documentation that explains the storage models it supports? c. How much control over the data formats, data structures, and storage models does the software package provide? Can new classes of digital objects be added or removed? Can the metadata or digital object formats be customized?
	B2.2. Repository has a definition of each AIP (or class) that is adequate to fit long-term preservation needs.

	a. Does the repository software application's implementation of an AIP support the use of data formats, data structures, and storage models that are amenable to long-term preservation needs? b. Does the repository software application's implementation of an AIP support the use of preservation metadata elements? For example, elements from the PREMIS schema. c. Can the repository software application's AIP be easily customized to support long-term preservation? E.g., the addition of new metadata elements.
	B2.3. Repository has a definition of how AIPs are derived from SIPs. a. Does the repository software have a well documented process by which a SIP is ingested into the repository for storage? For example, how and when does the repository software generate fixity data, such as checksums? b. Does the repository software generate or store additional technical metadata derived from the SIPs, such as by using JHOVE?
B2.5 Repository has and uses a naming convention that generates visible, persistent, unique identifiers for all archived objects (i.e., AIPs). ***Note: Here, both old B2.4 and B2.5 map to new [TRAC] B2.5.***	**B2.4.** Repository has and uses a naming convention that can be shown to generate visible, unique identifiers for all AIPs. a. How does the repository software identify or name AIPs? Are the identifiers guaranteed to be unique within the repository, globally unique, including across time? b. How does the repository software itself uniquely identify an object — URIs, URLs, URNs, etc? c. If an AIP is a discrete entity with storage and relational mechanisms, how do you get to the Entity?
	B2.5. If unique identifiers are associated with SIPS before ingest, they are preserved in a way that maintains a persistent association with the resultant AIP. a. Does the repository software preserve pre-existing identifiers for submitted packages?
B2.12	**B2.7.** Repository provides an independent mechanism for audit of the integrity of the repository collection/content. b. Can the repository software application provide a means to validate technical metadata using integrity measures such as checksums? c. Does the repository software provide audit logs of all events that have occurred in the life cycle of a package? d. How does the repository software indicate when a component of an AIP has been corrupted?
B3. Preservation Planning	**B3.** Preservation planning, migration, & other strategies
	B3.1. Repository has documented preservation strategies.

	a. Does the repository software application document basic procedures that can support preservation such as back-up/restore, data integrity checking, etc.
	B3.2. Repository implements/responds to strategies for AIP storage and migration. a. Does the repository software support migration of archival packages? b. Can AIPs be exported from the repository in a standard format such as METS or MPEG-21 DIDL? c. How is an archival package backed up such that all relevant information is preserved and can be easily restored into the same repository software system or a different one? d. Is there a clear policy regarding software upgrades? e. Are older packages guaranteed to be forward compatible, and, if not, do new versions of the software have a clear upgrade mechanism for older packages? f. How many generations removed can a package be from the version of the software that created it before it is no longer supported?
	B3.3. Repository uses appropriate international Representation Information (including format) registries a. What standards does the repository software use to describe file formats, such as Internet MIME Types? b. Does the software support any type of format registry such as PRONOM or Global Digital Format Registry (GDFR)?
	B3.8. Repository has contemporaneous records of actions taken associated with ingest and archival storage processes and those administration processes which are relevant to preservation. a. Does the repository software provide audit logs of events significant to preservation that have occurred in the life cycle of a package? If so, then: 1) What events are logged? 2) What format are the logs in? Are they easily processed by human or machine? 3) Are the logs tamper resistant and can they be kept 'forever'?
B3.2	**B3.9.** Repository has mechanisms in place for monitoring and notification when Representation Information (including formats) approaches obsolescence or is no longer viable. a. Can the repository software application monitor any standard format registries in order to ascertain format obsolescence? b. Can the repository software application support scheduled events, such that a human operator can be notified on a preset schedule to check manually for format obsolescence?
B3.3	**B3.10.** Repository has mechanisms to change its preservation plans as a result of its monitoring activities. *The following are examples of how this may be applied to repository software applications:*

	a. Does the repository software application support the migration of metadata formats or digital object formats? b. Does the repository software application easily support the export of its data? **** *See also B3.2.*
B5. Information Management	**B4.** Data Management
B5.2	**B4.1.** Repository captures or creates minimum descriptive metadata and ensures it is associated with the AIP. a. Can the repository software application infer or derive descriptive metadata from its digital objects? b. How does the repository software application associate descriptive metadata with the AIP? For example, is the descriptive metadata encapsulated within the AIP, or is it referenced from the AIP? Does this create any concern that associations between the descriptive metadata and the AIP may be vulnerable?
B5.3	**B4.2.** Repository can demonstrate that referential integrity is created between all AIPs and associated descriptive information. a. How does the repository software application maintain links between the descriptive metadata and the Digital Objects? b. How are the links maintained for other types of metadata?
B6	**B5.** Access Management
B6.5	**B5.1.** Repository access management system fully implements access policy. a. How does the repository software application provide access to the Content Information? b. How flexible or customizable is the repository software application with regard to access? c. Can the repository software application be set up to allow or deny access based on access rules, such as IP address restrictions, user id and password, or other means?
B6.6	**B5.2.** Repository logs all access management failures, and staff review inappropriate "access denial" incidents. a. Does the repository software application log all access attempts, both successful and unsuccessful? b. Are access denials flagged in any special manner by the software? c. How accessible are the logs to either human- or machine-processing and interpretation?
B6.7. Repository can demonstrate that the process that generates the requested digital	**B5.3.** Repository can demonstrate that the process that generates the DIP is completed in relation to the request.

object(s) (i.e., DIP) is completed in relation to the request.	Can the repository software application communicate to the user the status of their request, including indicating whether the process that generates the DIP is complete or incomplete in relation to the request? a. Does repository software application ever deliberately return a partial response to a request and if so how does it notify the user? b. How does repository software notify user when it is unable to respond to a request?
B6.8. Repository can demonstrate that the process that generates the requested digital object(s) (i.e., DIP) is correct in relation to the request.	**B5.4.** Repository can demonstrate that the process that generates the DIP is correct in relation to the request. Does the recipient of the DIP have any means at their disposal to verify the correctness of a DIP? E.g., checksums to verify the integrity of the bitstreams received; schema that could be used to programmatically validate the correctness of a DIP.
B6.9	**B5.5.** Repository demonstrates that all access requests result in a response of acceptance or rejection. a. How does the repository software application notify the end-user that an access request is accepted or rejected? b. Does the repository software application log all access requests?
B6.10	**B5.6.** Repository enables the dissemination of authentic copies of the original or objects traceable to originals. a. Does the repository software application support validation of the original document — for example, by using checksums, digital signatures, file comparisons, etc? b. Does the repository software application support provenance metadata, and make it accessible in some manner to users? c. Does the DIP contain provenance information for the digital object — e.g., does it track file transformations? *** *See ref to B5.3.*
In TRAC, Section C is now "Technologies, Technical Infrastructure, and Security." What was in Section C in the TDR Audit Checklist (right-hand column) now has been integrated into the rest of TRAC.	**C.** Designated Community and the Usability of Information
	C3. Use and Usability
	C3.2. Repository has implemented a policy for recording all access actions (includes requests, orders, etc.) that meet the requirements of the repository and information Producers/depositors.

	a. Does the repository software have an event log? What events are logged?
	C3.3. Repository ensures that agreements applicable to access conditions are adhered to. b. What sort of access restrictions and authentication mechanisms can the repository software application support? c. How does the repository software application document and implement access restrictions?
	D. Technologies & Technical Infrastructure
C1	**D1.** System infrastructure
C1.1	**D1.1.** Repository functions on well-supported operating systems and other core infrastructural software. a. Are the systems and services required by the repository software application still current and supported by the developer or manufacturer? (For example, these might include the operating system, web servers, database servers, and particular computer language run-time environments.) 1) A related consideration: If a key software component fails, can it be replaced? For example, if there is a security hole in the database, are there patches available? Or can the software component be switched for a different database application?
C1.2 Repository ensures that it has adequate hardware and software support for backup functionality sufficient for the repository's services and for the data held, e.g., metadata associated with access controls, repository main content.	**D1.2.** Repository ensures that all platforms have a backup function, sufficient for the repository's services and for the data held, e.g., metadata associated with access controls, repository main content, etc. a. What sorts of backup strategies does the architecture of the repository software application afford? b. Does the repository software application explicitly require any particular backup strategy, or does it just rely on system-level backup plans, like periodic disk backups to tape? c. In the event of a disaster, how would the repository software application be restored?
C1.3. Repository manages the number and location of copies of all digital objects.	**D1.3.** Repository stipulates the number and location of copies of all digital objects. For a basic, non-distributed repository systems this should be part of basic functionality, but for a more complex distributed architectures the management of multiple copies of digital object may be a concern, for example you should consider the following questions: a. Does the repository software application have any explicit support for parallel operations, for example multiple duplicate copies of the service running on different servers, to support data redundancy in the event of a disaster, or for load balancing in high usage scenarios? b. Does the repository software have any explicit support for distributed operations such that different parts of the system are running on different servers, either in terms of data or in terms of functionality? For example, packages belonging to collection A are stored

	on server X, but packages belonging to collection B are stored on server Y, or metadata are stored on server X, but Digital Objects are stored on server Y. *****See also D1.4.*
C1.4	**D1.4.** Repository has mechanisms in place to insure any/multiple copies of digital objects are synchronized. Consider: a. In cases of distributed or parallel systems, how is synchronization of the data ensured? What sorts of latencies are involved when objects are changed? b. How does the repository software handle duplicates? For example, what happens if the same submission package is submitted multiple times? *****See D1.3 above.*
C1.5	**D1.5.** Repository has effective mechanisms to detect data corruption or loss. a. What techniques are used to detect data corruption or loss? Possible techniques include (from less to more effective) comparison of files sizes, use of digest algorithms, comparisons of secondary copies of files, and digitally signed files. b. Related to detection is the issue of repair. Can the repository offer any means to correct corrupt files? Certain architectures may be more amenable to file repair than others, for example architectures where multiple copies of files are distributed. *****See D1.6*
C1.6	**D1.6.** Repository reports to its administration all incidents of data corruption or loss, and steps taken to repair/replace corrupt or lost data. a. Does the repository software application log events that occur on a package, specifically the detection of data corruption or loss? If so, how are these errors logged? b. Can administrators be automatically notified of these types of events, for example by email or RSS feeds? *****See D1.6*
C1.7	**D1.7.** Repository has defined processes for storage media migration. How can the repository software application support migration? For example, you may want to consider the following: a. Does the repository software support migration of archival packages? Can they be exported from the repository in a standard format such as METS or MPEG-21 DIDL? How is an archival package backed up such that all relevant information is preserved and can be easily restored into the same repository software system or a different one? b. Can the repository software ingest its own dissemination packages without data loss? c. Does the repository software documentation provide recommendations for how to execute storage media migration—such as dumping data to a temporary storage space while swapping out disks?

C1.9	**D1.9.** Repository has a process for testing the effect of critical changes to the system. This is primarily an organizational issue. Consideration related to the repository software application might include: a. How does the group or development community responsible for the repository software application test new releases, patches, fixes, etc., that are relevant to the core system as it's generally distributed?
C1.10. Repository has a process to react to the availability of new software security updates based on a risk-benefit assessment.	**D1.10.** Repository has a process to stay current with the latest operating system security fixes. *This is primarily an organizational issue. Consideration related to the repository software application might include:* a. Does the group or development community responsible for the repository software have a process for dealing with security issues either related to the operating systems or any dependent software that may affect the repository software? b. Does the repository software application documentation report what operating systems and dependent software it requires?
C.2	**D2.** Appropriate technologies
C2.1	**D2.1.** Repository has hardware technologies appropriate to the services it provides to its designated communities and has procedures in place to monitor and receive notifications when hardware technology changes are needed. *This is primarily an organizational issue. Consideration related to the repository software application might include:* a. What are the hardware requirements for the repository software application? b. How are implementers and managers of the repository software notified when new repository software application versions are available that may have different hardware requirements?
C2.2	**D2.2.** Repository has software technologies appropriate to the services it provides to its designated communities and has procedures in place to monitor and receive notifications when software technology changes are needed. *This is not primarily a repository software application issue. However, considerations related to the repository software application might include:* a. Is the repository software application amenable to modification if needed? For example, do the repository software application developers have a mechanism for receiving feedback from implementers requesting bug fixes and additional functionality–such as surveys, focus groups, etc.? b. How are implementers and managers of the repository software notified when new repository software application versions are available that may have different software requirements?
C3	**D3.** Security
C3.2. Repository has implemented controls	**D3.2.** Repository has implemented mechanisms (processes) to adequately address each of the

to adequately address each of the defined security needs.	defined security needs. a. Does the repository software application support different levels of access for different types of users or different types of objects? How does it provide security? b. Does the repository software application afford the appropriate levels of security required for your user community?