UC3Merritt

University of California Curation Center

Merritt Ingest Service

Rev. 1.00 – 2016-10-25

1 Introduction

Information technology and resources have become integral and indispensable to the pedagogic mission of the University of California. Members of the UC community routinely produce and utilize a wide variety of digital assets in the course of teaching, learning, and research. These assets represent the intellectual capital of the University; they have inherent enduring value and need to be managed carefully to ensure that they will remain available for use by future scholars. Within the UC system the UC Curation Center (UC3) of the California Digital Library (CDL) has a broad mandate to ensure the long-term usability of the digital assets of the University. UC3 views its mission in terms of *digital curation*, the set of policies and practices aimed at maintaining and adding value to authentic digital assets for use by scholars now and into the indefinite future [Abbott].

In order to meet these obligations UC3 is developing Merritt, an emergent approach to digital curation infrastructure [Merritt]. Merritt devolves infrastructure function into a growing set of granular, orthogonal, but interoperable micro-services embodying curation values and strategies. Since each of the services is small and self-contained, they are collectively easier to develop, deploy, maintain and enhance [Denning]; equally as important, since the level of investment in and commitment to any given service is small, they are more easily replaced when they have outlived their usefulness. Yet at the same time, complex curation functionality can emerge from the strategic combination of individual, atomistic services [Fisher].

The Merritt Ingest service manages the acquisition of new digital content supplied by content producers into a Merritt curation environment.

NOTE The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” are to be interpreted as described in RFC 2119 [RFC2119].

2 Requirements

The Merritt Ingest service MUST meet the following functional and non-functional requirements:

- The service SHALL impose minimal requirements regarding the acceptable structure of submitted digital objects.
    - The simplest acceptable submission SHALL be a single file.
    - The minimal administrative metadata that MUST accompany a submitted object are the identifiers for the submitting user agent and the Storage service and storage node in which the object will be deposited. The Storage services and nodes MAY be constrained to those authorized for the submitting user agent.

    - The minimal descriptive metadata that MUST accompany a submitted object is its primary ARK identifier. This requirement SHALL be fulfilled by specifying either a pre-existing identifier or the noid minter to be used in minting a new identifier.
- It SHALL be possible for multiple digital objects to be submitted in a single operation.
- Submission SHALL be possible without requiring a client-side web server.
- The service SHALL provide interfaces for both synchronous and asynchronous processing. The synchronous interface MAY be restricted to administrative use only.

3 Ingest

The Merritt Ingest service is based on the following conceptual entities, each defined in terms of its specific state properties.

- Service.
- Queue.
- Batch.
- Job.
- Profile.
- Handler.

The Ingest service supports methods that can be used to manipulate and access these entities and their state in useful ways.

All Merritt ingests are relative to one of more *digital objects* [Object], either pre-existing objects or objects newly established as a side-effect of the ingest operation. An ingest operation always results in a new *version* of the relevant object.

3.1 Service

The initial conceptual entity is the Ingest service itself, which provides a mechanism to acquire new digital content in a Merritt curation environment. The global state properties of the service MUST include:

- Service name. [ing:name]
- Service identifier, assigned to be globally unique among all UC3-controlled instantiations. [ing:identifier]
- Service description. [ing:description]
- Service implementation version. [ing:serviceVersion]
- Actionable reference to queue state. [ing:queueState]
- Actionable references to Storage service instantiations known to the service. [ing:storageStates]
    - Actionable reference to Storage service state. [ing:storageState]

- Number of jobs currently in the queue. [ing:numJobs]
- Total number of jobs processed. [ing:numTotalJobs]
- Last submission date/timestamp. [ing:lastSubmission]
- Last consumption date/timestamp. [ing:lastConsumption]
- Creation date/timestamp. [ing:creatied]
- Modification date/timestamp. [ing:lastModified]
- Service specification and version. [ing:serviceScheme]
- Base URI for the service method invocations. [ing:baseURI]
- Support URI for the service. [ing:supportURI]
- Administrative URI for the service. [ing:adminURI

Additional global service properties MAY be defined and managed by the service.

The Ingest service can be configured with knowledge of an arbitrary number of Merritt Storage services, which manage the secure, persistent storage of digital objects [Storage]. Multiple Storage services can be defined to encapsulate different underlying storage technologies, policy regimes, or administrative scope.

Jobs submitted for (asynchronous) processing are managed in a *queue*.

3.2 Queue

The *queue* manages jobs submitted to the service but awaiting processing. Conceptually the queue interacts with two processes: a *submitter* that adds jobs to the queue; and a *consumer* that retrieves jobs from the queue. The queue state properties MUST minimally include:

- Actionable reference to the Ingest service state. [que:ingestState]
- Consumer polling interval (in seconds). [que:pollingInterval]
- Number of consumer threads. [que:numThreads]
- Number of jobs currently in the queue. [que:numJobs]
- Total number of jobs processed. [que:numTotalJobs]
- Creation date/timestamp [que:created]
- Last submission date/timestamp. [que:lastSubmission]
- Last consumption date/timestamp. [que:lastConsumption]
- If the consumer is in a paused state, the date/timestamp at which it was paused. [que:paused]
- Consumer status: *running* or *paused*. [que:status]

Additional queue state properties MAY be defined and managed by the service.

At the conclusion of ingest processing of a given job the consumer's behavior is:

- If the polling mode is *immediate*, poll the queue to see if another pending job is available; if

the mode is *wait*, wait for the polling interval to expire before polling the queue.

- If a job is available, i.e. in *pending* status, at the time of polling, the job is consumed; if not, wait for the polling interval to expire before polling the queue.

Individual jobs are managed explicitly in the queue; batches are managed indirectly, with each queued job knowing the batch of which it is a member. A queue can manage an arbitrary number of *jobs*.

NOTE The queue entity is not strictly necessary; since there is a one-to-one relationship between the service and the queue, all of the queue properties could have been defined as ingest service properties. However, it is useful to have a queue entity so that its state, which may potentially incorporate significant numbers of job references, can be interrogated independently of the global service state, which does not then need to incorporate job references.

3.3 Batch

A *batch* encapsulates an arbitrary number of jobs that are submitted to the Ingest service in a single operation. Batches are defined for administrative convenience, since they permit a single notification to be sent regardless of the number of component jobs. In particular:

- A batch does *not* constitute an indivisible unit of processing; each job in a batch is processed, and succeeds or fails, independent of all other jobs and the batch itself.
- The failure of a given batch job does *not* cause the rollback of previously processed jobs, and does *not* terminate the processing of subsequent jobs.

The batch state properties MUST minimally include:

- Batch identifier, assigned to be locally-unique with the service. [bat:identifier]
- Number of jobs in the batch. [bat:numJobs]
- Number of pending jobs. [bat:numPendingJobs]
- Number of consumed jobs. [bat:numConsumedJobs]
- Number of completed jobs. [bat:numCompletedJobs]
- Actionable reference to the queue state. [bat:queueState]
- Actionable references to job states. [bat:jobStates]
    - Actionable reference to job state. [bat:jobState]
- Submission date/timestamp. [bat:submitted]
- Completion date/timestamp. [bat:completed]
- Status: *pending*, *consumed*, or *completed*. [bat:status]

Additional batch state properties MAY be defined and managed by the service.

A batch is in the *pending* state if all of its jobs are in a *pending* state; in the *consumed* state if any of its jobs are in the *consumed*, *completed*, or *failed* state; and in the *completed* state if all of its jobs are in the *completed* or *failed* state.

A batch encompasses an arbitrary number of *jobs*. Batches are managed by the queue indirectly, with each queued job knowing the batch of which it is a member.

3.4 Job

A *job* encapsulates the ingest processing of a single digital object. The successful processing of a job results in a new *version* of the object managed in an instance of the Merritt Storage service [Storage]. The job state properties MUST minimally include:

- Job identifier, assigned to be locally unique within the batch and queue. [job:identifier]
- Actionable reference to the queue state. [job:queueState]
- Actionable reference to the parent batch state. [job:batchState]
- Actionable reference to the profile state. [job:profileState]
- Batch identifier. [job:batch]
- Submitting user agent. [job:submitter]
- Primary identifier, if supplied as part of the submission or newly minted as part of ingest processing. [job:primaryIdentifier]
- Object creator, if supplied as part of the submission. [job:creator]
- Object title, if supplied as part of the submission. [job.title]
- Object date, if supplied as part of the submission. [job:date]
- Object local identifier(s), if supplied as part of the submission. [job:localIdentifier]
- Object expository note, if supplied as part of the submission. [job:note]
- Submission date/timestamp. [job:submitted]
- Consumption date/timestamp. [job:consumed]
- Completion date/timestamp. [job:completed]
- Optional MetaCat registration status: *success* or *failure*. [job:metacatStatus]
- Job status: *pending*, *consumed*, *completed*, *failed*, or *deleted*. [job:status]

The status transition diagram for jobs is:

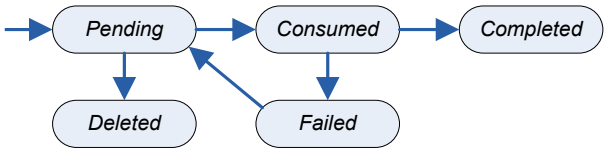

Figure 1 – Job status transitions

Additional job state properties MAY be defined and managed by the service.

In addition to these state properties, a job also has a *payload*: the data and metadata files received from the submitting user agent that collectively define a new object version. A subset of state properties – batch identifier; job identifier; user agent; primary identifier, creator, title, date, local identifier, and note, if supplied; submission, consumption, and completion date/timestamps; and an actionable

reference to the payload – constitute a job's *sidecar* information, which MUST be maintained throughout the job's processing.

The specific processing performed by a job is determined by the object's *profile*.

3.5 Profile

A *profile* declares the administrative categories, content type, and processing disposition associated with an object. The profile state properties MUST minimally include:

- Profile identifier, assigned to be locally unique within the Ingest service. [pro:identifier]
- Optional profile description. [pro:description]
- Object type: *MRT-curatorial* or *MRT-system*.. [pro:type]
- Object role: *MRT-content* or *MRT-class*. [pro:role]
- Optional aggregate type: *MRT-collection*, *MRT-owner*, or *MRT-service-level-agreement*. [pro:aggregate]
- Owner identifier. [pro:owner]
- Collection identifier(s). [pro:collection]
- Queue priority level. [pro:priority]
- Target Storage service identifier. [pro:storageService]
- Target storage node identifier. [pro:storageNode]
- Target Fixity service identifier. [pro:fixityService]
- Identifier scheme: *ARK* or *DOI*. [pro:identifierScheme]
- Identifier namespace. [pro:identifierNamespace]
- Identifier minter URI. [pro:identifierMinter]
- Identifier context. [pro:identifierContext]
- Object content model identifier. [pro:contentModel]
- Notification email address(es). [pro:notification]
- Notification format: *ANVL*, *CSV*, *JSON*, *RDF/Turtle*, *XHTML*, or *XML*. [pro:notificationFormat]
- Notification callback URL (optional). [pro:callbackURL]
- Actionable references to handler states. [pro:handlers]
    - Actionable reference to handler state. [pro:handler]
- Creation date/timestamp. [pro:created]
- Modification date/timestamp. [pro:lastModified]

Additional profile state properties MAY be defined.

The aggregate type property MUST be defined only for profiles used to submit class objects, that is, those for whom the object role is *MRT-class*.

The namespace for the ARK scheme is an ARK *shoulder* of the form: "`ark:/naan/shoulder`"

[EZID]. The namespace for the DOI scheme is a DOI prefix of the form: "`doi:prefix/`" [DOI]. In normal Merritt operations, the identifier scheme is "`ARK`"; the ARK identifier namespace is "`ark:/13030/m5`" for content objects and "`ark:/13030/j2`" for class objects (see [Object] for the distinction between content and class objects); and the identifier minter URI is "`http://ezid.cdlib.org`". The identifier context is a value that is passed to the minter to be associated with the newly minted identifier.

The *primary* identifier for a Merritt object MUST be an ARK. If a submission profile defines the identifier scheme as *DOI*, the minter MUST mint and return both a DOI *and* an ARK identifier.

A profile specifies an arbitrary sequence of *handlers* that will be applied sequentially against all submitted objects.

3.6 Handler

A *handler* encapsulates a specific subset of ingest processing. The handler state properties MUST minimally include:

- Handler name. [han:name]
- Handler identifier, assigned to be locally unique within the Ingest service. [han:identifier]
- Handler implementation version. [han:version]
- Handler description. [han:description]
- Creation date/timestamp. [han:created]
- Modification date/timestamp. [han:lastModified]

Additional handler state properties MAY be defined.

4 Service Interface

All Merritt services are defined in terms of abstract interfaces that can be implemented in various interactive modalities, including a procedural API with various language bindings, a command line API supported in various operating system command shells, and a RESTful API [Fielding].

State information about the various entities managed by the service MAY be requested in the following formats:

Format	*Extension*	*MIME type*
ANVL	`.txt`	`text/anvl`
CSV	`.csv`	`text/csv`
HTML	`.html`	`application/xhtml+xml`
JSON	`.json`	`application/json`
RDF/Turtle	`.ttl`	`text/turtle`
XML	`.xml`	`application/xml`

Table 1 – State formats

NOTE Until such time as a formal MIME types for the ANVL [ANVL] and Turtle [Turtle] formats are established at the IANA registry, the experimental MIME types "`text/x-anvl`" and "`text/x-turtle`" SHOULD be used, respectively.

RESTful options	***Command line options***		***Function***
	`-C` *`context`*	`--context` *`context`*	Object local identifier context
`D[=`*`level`*`]`	`-D [`*`level`*`]`	`--debug [`*`level`*`]`	Debug level.
`h`	`-h [`*`topic`*`]`	`--help [`*`topic`*`]`	Help.
	`-I` *`identifier`*	`--identifier` *`identifier`*	Object local identifier
`M=`*`mode`*	`-M` *`mode`*	`--mode` *`mode`*	Queue consumer mode.
	`-o` *`file`*	`--output` *`file`*	Output to file (rather than standard output).
`S=`*`status`*	`-S` *`status`*	`--status` *`status`*	Queue consumer status.
`Accept:` *`form`*	`-t` *`form`*	`--response-form` *`form`*	Response format.
`Content-type:`	`-T` *`form`*	`--request-form` *`form`*	Request format.
`v`	`-v`	`--verbose`	Verbose response.
`V`	`-V`	`--version`	Version information.

Table 2 – Common API options

5 Service Methods

The Ingest service SHOULD support the following methods. Each method is first defined abstractly and then in terms of RESTful and command shell APIs.

NOTE The RESTful API is defined in terms of HTTP request and response messages. The notations "`UA`" and "`OS`" are used to distinguish the User Agent request from the Origin Server response. Names in *italics* indicate arbitrary, rather than fixed values. Brackets "[" and "]" enclose optional elements, parentheses "`(`" and "`)`" enclose groups of elements, a vertical bar "|" separates alternatives; and an ellipsis "`...`" indicates the arbitrary repetition of the previous element.

5.1 Help

METHOD Help			[*idempotent*, *safe*]
Method	Enum	Optional	Specific method about which help is requested.
ResponseForm	Enum	Optional	Response form. The supported forms SHOULD include ANVL (default for command line interfaces), JSON, RDF/ Turtle, XHTML (default for web interfaces), and XML.
RETURN	ResponseForm	Mandatory	Help information about the specific method or the service as a whole.
SIDE EFFECTS	—		
ERRORS	400	Badly-formed request.	
	401	Unauthorized user agent.	
	415	Unsupported response form.	
	503	Service unavailable.	
	500	Service error.	

- RESTful API

```
UA: GET /help HTTP/1.x
UA: Host: ingest.cdlib.org
UA: Accept: response/form
UA:

OS: HTTP/1.x 200 OK
OS: Content-type: response/form
OS:
OS: help
```

- Command line API

```
% ingest -h [-t form] [-o file]
% ingest help [-t form] [-o file]
```

5.2 Get Service State

METHOD Get-service-state			[*idempotent*, *safe*]
—			No argument.
ResponseForm	Enum	Optional	Response form. The supported forms SHOULD include ANVL (default for command line interfaces), JSON, RDF/Turtle, XHTML (default for web interfaces), and XML.
RETURN	Response form	Mandatory	Global service state.
SIDE EFFECTS	—		
ERRORS	400 Badly-formed request.		
	401 Unauthorized user agent.		
	415 Unsupported response form.		
	503 Service unavailable.		
	500 Service error.		

- RESTful API

```
UA: GET /state HTTP/1.x
UA: Host: ingest.cdlib.org
UA: Accept: response/form
UA:

OS: HTTP/1.x 200 OK
OS: Content-type: response/form
OS:
OS: state
```

- Command line API

```
% ingest getServiceState [-t form] [-o file]
```

5.3 Get Queue State

METHOD Get-queue-state			[*idempotent*, *safe*]
—			No argument.
ResponseForm	Enum	Optional	Response form. The supported forms SHOULD include ANVL (default for command line interfaces), JSON, RDF/Turtle, XHTML (default for web interfaces), and XML.
RETURN	Response form	Mandatory	Consumer state.
SIDE EFFECTS	—		
ERRORS	400 Badly-formed request.		
	401 Unauthorized user agent.		
	415 Unsupported response form.		
	503 Service unavailable.		
	500 Service error.		

- RESTful API

```
UA: GET /state/queue HTTP/1.x
UA: Host: ingest.cdlib.org
UA: Accept: response/form
UA:

OS: HTTP/1.x 200 OK
OS: Content-type: response/form
OS:
OS: state
```

- Command line API

```
% ingest getQueueState [-t form] [-o file]
```

5.4 Get-Batch State

METHOD Get-batch-state			[*idempotent*, *safe*]
Batch	Identifier	Mandatory	Batch identifier.
ResponseForm	Enum	Optional	Response form. The supported forms SHOULD include ANVL (default for command line interfaces), JSON, RDF/Turtle, XHTML (default for web interfaces), and XML.
RETURN	Response form	Mandatory	Batch state.
SIDE EFFECTS	—		
ERRORS	400 Badly-formed request.		
	401 Unauthorized user agent.		
	404 Consumer not found.		
	404 Batch not found.		
	415 Unsupported response form.		
	503 Service unavailable.		
	500 Service error.		

- RESTful API

```
UA: GET /state/queue/batch HTTP/1.x
UA: Host: ingest.cdlib.org
UA: Accept: response/form
UA:

OS: HTTP/1.x 200 OK
OS: Content-type: response/form
OS:
OS: state
```

- Command line API

```
% ingest getBatchState batch [-t form] [-o file]
```

5.5 Get Job State

METHOD Get-job-state			[*idempotent*, *safe*]
Batch	Identifier	Mandatory	Batch identifier.
Job	Identifier	Mandatory	Job identifier.
ResponseForm	Enum	Optional	Response form. The supported forms SHOULD include ANVL (default for command line interfaces), JSON, RDF/Turtle, XHTML (default for web interfaces), and XML.
RETURN	Response form	Mandatory	Job state.
SIDE EFFECTS	—		
ERRORS	400 Badly-formed request.		
	401 Unauthorized user agent.		
	404 Consumer not found.		
	404 Batch not found.		
	404 Job not found.		
	415 Unsupported response form.		
	503 Service unavailable.		
	500 Service error.		

- RESTful API

```
UA: GET /state/queue/batch/job HTTP/1.x
UA: Host: ingest.cdlib.org
UA: Accept: response/form
UA:

OS: HTTP/1.x 200 OK
OS: Content-type: response/form
OS:
OS: state
```

- Command line API

```
% ingest getBatchState batch job [-t form] [-o file]
```

5.6 Submit

METHOD Submit			[*non-idempotent*, *unsafe*]
Submitter	Agent	Mandatory	Submitting user agent.
Filename	List of String	Mandatory	Submission package filename(s).
File	List of Octet-stream List of Tar | Tgz | Zip List of Checkm	Mandatory	Submission package file(s).
Type	Enum	Optional	Submission package type: *file*, *container*, *object-manifest*, *batch-manifest*, *container-batch-manifest*, or *single-file-batch-manifest*. If type = *file, container*, or *object-manifest* each file in the submission package represents a single object. If type = *file* each object is composed of a single file; if type = *container* each object is composed of the files within the container; if type = *object-manifest* each object is composed of the files described in the Checkm manifest. If type = *batch-manifest*, *container-batch-manifest*, or *single-file-batch-manifest* each Checkm manifest represents a single batch of objects. Each file described in the manifest represents a single object. If type = *batch-manifest* the described files are all *object-manifest*s; if type = *container-batch-manifest* the described files are all containers; and if type = *single-file-manifest* the described files are all individual files that each completely define an object. NOTE: In the absence of an explicit "Type" argument in a *Submit* request, the submission package file type is determined implicitly as follows: • Determine if the package file is a Checkm manifest as evidenced by a "`#%checkm`" structured comment as the first line of the file. • If a Checkm manifest, use the embedded profile identifier specified by the "`#%profile`" structured comment to determine the type: *object-manifest*, *batch-manifest*, *container-batch-manifest*, or *single-file-batch-manifest*.

			• If not a manifest, determine if the file is a Tar, Gzip, or Zip container as evidenced by a "`.tar`", "`.gz`", or "`.zip`" file extension, or if no extension is specified, by the HTTP Content-type header MIME type being one of "application/x-gzip", "application/x-tar", or "application/zip". • If not a container, assume the type is *file*.
Profile	Identifier	Mandatory	Submission profile identifier.
PrimaryIdentifier	Identifier	Optional *	Object ARK identifier.
DigestType	Enum	Optional *	Package message digest type. The supported types SHOULD include: Adler-32, CRC-32, MD2, MD5, SHA-1, SHA-256, SHA-384, and SHA-512.
DigestValue	String		Package hexadecimal message digest value.
Creator	String	Optional *	Person responsible for the object, e.g. creator, contributor, publisher, etc., equivalent to ERC "who".
Title	String	Optional *	Object title, equivalent to ERC "what".
Date	String	Optional *	Meaningful object date, equivalent to ERC "when".
LocalIdentifier	String	Optional *	Locally-meaningful alternative object identifier, equivalent to ERC "where".
RetainTargetURL	Boolean	Optional *	If "true", do not send EZID a Merritt-defined target URL; responsibility for properly setting the target URL rests externally.
DC.contributor	String	Optional *	DC "contributor", an individual, corporate, or automated agent contributing to the object.
DC.coverage	String	Optional *	DC "coverage", a spatial or temporal topic or statement of applicability of jurisdiction.
DC.description	String	Optional *	DC "description", an account of the object.
DC.format	String	Optional *	DC "format", the format, medium, or dimensions of the object.
DC.language	String	Optional *	DC "language", a language used by the object.
DC.publisher	String	Optional *	DC "publisher", an individual or corporate agent publishing the object.
DC.relation	String	Optional *	DC "relation", a related resource.
DC.rights	String	Optional *	DC "rights", a statement of intellectual property rights held in or over the object.
DC.source	String	Optional *	DC "source", a related resource from which the object is derived.
DC.subject	String	Optional *	DC "subject", the topic of the object.
DC.type	String	Optional *	DC "type", the nature or genre of the object.
DataCite.resourceType	Enum	Optional *	DataCite "resourceType", the general type of the object: "Collection", "Dataset", "Event", "Film",

			"Image", "InteractiveResource", "Model", "PhysicalObject", "Service", "Software", "Sound", "Text"
Notification	String	Optional	Notification email address. Multiple addresses can be specified in the string if separated by a semicolon ";" and one or more white space characters.
ResponseForm	Enum	Optional	Response form. The supported forms SHOULD include ANVL (default for command line interfaces), CSV (implied for multi-job batches), XHTML (default for web interfaces), JSON, RDF/Turtle, and XML.
RETURN	*Response form*	Mandatory	Submission notification.
SIDE EFFECTS	A single unique batch identifier is minted and associated with all jobs defined in the submission package, each with its own unique object identifier (either supplied or minted) and *pending* status. All jobs (including their payloads and sidecar information) are placed into the queue for asynchronous processing with the queue priority defined in the profile. In addition to the synchronous submission notification response (documenting all batch jobs), an equivalent asynchronous email message is sent to all email addresses specified in the "notification" parameter and the submission profile (as described in Section § 5.6.5, below).		
ERRORS	400	Badly-formed request.	
	401	User agent not authorized.	
	415	Unsupported package type.	
	413	Submission too large.	
	400	Empty submission.	
	400	Package digest verification failed.	
	404	Profile not found.	
	404	Minter not found	
	404	Storage service not found.	
	404	Storage node not found.	
	415	Invalid resourceType	
	415	Unsupported response form.	
	503	Service unavailable.	
	500	Service error.	
NOTE	Any of the Dublin Kernel or Core elements MAY be repeated.		

* Only meaningful for single file submission.

- RESTful API

NOTE RESTful requests are formatted assuming an underlying HTML form [HTML] and using the "`multipart-form-data`" content type [Multipart].

```
UA: POST /submit HTTP/1.x
```

```
UA: Host: ingest.cdlib.org
UA: Accept: response/form
UA: Content-type: multipart/form-data; boundary=boundary
UA:
UA: --boundary
UA: Content-disposition: form-data; name="submitter"
UA:
UA: user
UA: --boundary
UA: ( Content-disposition: form-data; name="file"; filename="filename"
UA: Content-type: application/octet-stream | application/x-gzip |
UA:               application/x-tar | application/zip | text/checkm
UA:
UA: file ) |
UA: ( Content-disposition: form-data; name="file"
UA: Content-type: multipart/mixed; boundary=boundary2
UA:
UA: -- boundary2
UA: Content-disposition: file; filename="filename"
UA: Content-type: application/octet-stream | application/x-gzip |
UA:               application/x-tar | application/zip | text/checkm
UA:
UA: file
UA: -- boundary2
UA: ...
UA: --boundary2 )
UA: --boundary [
UA: Content-disposition: form-data; name="type"
UA:
UA: type
UA: --boundary ]
UA: Content-disposition: form-data; name="profile"
UA:
UA: profile
UA: --boundary [
UA: Content-disposition: form-data; name="primaryIdentifier"
UA:
UA: identifier
UA: --boundary ] [
UA: Content-disposition: form-data: name="digestType"
UA:
UA: digest-type
UA: --boundary
UA: Content-disposition: form-data; name="digestValue"
UA:
UA: digest-value
UA: --boundary ] [
UA: Content-disposition: form-data; name="creator"
UA:
UA: creator
UA: --boundary ] [
UA: Content-disposition: form-data; name="title"
UA:
UA: title
UA: --boundary ] [
UA: Content-disposition: form-data; name="date"
```

```
UA:
UA: date
UA: --boundary ] [
UA: Content-disposition: form-data: name="localIdentifier"
UA:
UA: identifier
UA: --boundary ] [
UA: Content-disposition: form-data: name="retainTargetURL"
UA:
UA: true | false
UA: --boundary ] [
UA: Content-disposition: form-data: name="DC.contributor"
UA:
UA: contributor
UA: --boundary ] [
UA: Content-disposition: form-data: name="DC.coverage"
UA:
UA: coverage
UA: --boundary ] [
UA: Content-disposition: form-data: name="DC.description"
UA:
UA: description
UA: --boundary ] [
UA: Content-disposition: form-data: name="DC.format"
UA:
UA: format
UA: --boundary ] [
UA: Content-disposition: form-data: name="DC.language"
UA:
UA: language
UA: --boundary ] [
UA: Content-disposition: form-data: name="DC.publisher"
UA:
UA: publisher
UA: --boundary ] [
UA: Content-disposition: form-data: name="DC.relation"
UA:
UA: relation
UA: --boundary ] [
UA: Content-disposition: form-data: name="DC.rights"
UA:
UA: rights
UA: --boundary ] [
UA: Content-disposition: form-data: name="DC.source"
UA:
UA: source
UA: --boundary ] [
UA: Content-disposition: form-data: name="DC.subject"
UA:
UA: subject
UA: --boundary ] [
UA: Content-disposition: form-data: name="DC.type"
UA:
UA: type
UA: --boundary ] [
UA: Content-disposition: form-data: name="DataCite.resourceType"
```

```
UA:
UA: resourceType
UA: --boundary ] [
UA: Content-disposition: form-data: name="notification"
UA:
UA: email; ...
UA: --boundary ]

OS: HTTP/1.x 201 Created
OS: Content-type: response/form
OS: Location: http://ingest.cdlib.org/state/queue/batch
OS:
OS: notification
```

- Command line API

```
% ingest submit filename ... [-T type] [-t form] [-o file]
```

In general, the object primary identifier is left unspecified to indicate that a new object is being submitted (a unique primary identifier will be minted automatically during Ingest processing); the primary identifier is specified to indicate that a new version of a pre-existing object is being submitted. An arbitrary number of additional local identifiers, meaningful in some external curatorial context, can be specified in either case.

Note, however, that a variant submission workflow is also possible. If an object is associated with a local identifier, a new version can be submitted with only a local identifier specified; the Storage service maintains a mapping between object primary and local identifiers that is used by the Ingest service to retrieve the object primary identifier while processing the submission.

5.6.1 Object manifest

An object manifest is a Checkm manifest [Checkm] listing the location of all of the individual file components that collectively compose the object. The Checkm profile for object manifests (`http://uc3.cdlib.org/registry/ingest/manifest/mrt-ingest-manifest`) is defined as follows:

- The URL and target filename fields MUST be specified.
- The digest algorithm, digest value, and file size fields SHOULD be specified.
- The modification time field SHOULD NOT be specified, and will be ignored if provided.

  NOTE Unspecified Checkm fields MUST be left empty.

- The extension MIME type field SHOULD be specified, if known.
- The first two entries in the manifest MUST be structured comments specifying the Checkm conformance level and profile identifier.
- The next three entries SHOULD be structured comments specifying namespace prefixes and field definitions.
- The last entry in the manifest SHOULD be a structured comment explicitly representing the end of the file.

```
  #%checkm_0.7
  #%profile | http://uc3.cdlib.org/registry/ingest/manifest/mrt-ingest-
                                                                manifest
[ #%prefix  | mrt: | http://uc3.cdlib.org/ontology/mom# ]
[ #%prefix  | nfo: | http://www.semanticdesktop.org/ontologies/2007/03/
                                                              22/nfo# ]
[ #%fields  | nfo:fileUrl  | nfo:hashAlgorithm    | nfo:hashValue |
              nfo:fileSize | nfo:fileLastModified | nfo:fileName  |
              mrt:mimeType ]
  url | [algorithm] | [value] | [size] | | filename [ | mime]
  ...
[ #%eof ]
```

NOTE In the preceding syntax the vertical bar "|" represents the literal field delimiter defined by the Checkm specification and does *not* separate alternatives.

5.6.2 Batch manifest

A batch manifest is an extended Checkm manifest listing the locations of object manifests for all the objects that collectively compose the batch. The Checkm profile for batch manifests (`http://uc3.cdlib.org/registry/ingest/manifest/mrt-batch-manifest`) is defined as follows:

- The URL and target filename fields MUST be specified.
- The digest algorithm, digest value, and file size fields SHOULD be specified.
- The modification time field SHOULD NOT be specified, and will be ignored if provided.
- Two Checkm extension fields are defined for the object's primary and local identifiers. If an object is being initially established by an ingest request, the primary identifier MUST NOT be specified; if a subsequent object version is being created, the primary identifier MUST be specified. In either case, at least one local identifier, meaningful in the object's curatorial context, SHOULD be specified. An arbitrary number of local identifiers MAY be specified using a semicolon-separated list. Any semicolon embedded in a local identifier MUST be represented by the standard ERC escape notation "`%sc`".
- Four Checkm extension fields are defined for the Dublin Kernel "who", "what", "when", and "why" elements, corresponding to the object's creator, title, significant date, and expository note. These fields SHOULD be specified only if they are known.
- Eleven Checkm extension fields are defined for the Dublin Core "contributor", "coverage", "description", "format", "publisher", "relation", "rights", "source", "subject", "type", and "resourceType" elements. These fields are OPTIONAL.
- Multiple values can be specified for any DK or DC element using a semicolon ";" as an internal field separator.
- The first two entries in the manifest MUST be structured comments specifying the Checkm conformance level and profile identifier.
- The next three entries SHOULD be structured comments specifying namespace prefixes and field definitions.
- The last entry in the manifest SHOULD be a structured comment explicitly representing the end of the file.

```
#%checkm_0.7
#%profile | http://uc3.cdlib.org/registry/ingest/manifest/mrt-batch-
                                                          manifest
```

```
[ #%prefix  | mrt: | http://uc3.cdlib.org/ontology/mom# ]
[ #%prefix  | nfo: | http://www.semanticdesktop.org/ontologies/2007/03/
                                                          22/nfo# ]
[ #%fields  | nfo:fileUrl  | nfo:hashAlgorithm    | nfo:hashValue |
              nfo:fileSize | nfo:fileLastModified | nfo:fileName  |
              mrt:primaryIdentifier | mrt:localIdentifier | mrt:creator |
              mrt:title | mrt:date [ | mrt:contributor [ | mrt:coverage [
            | mrt:description [ | mrt:format [ | mrt:language [ |
              mrt:publisher [ | mrt:relation [ | mrt:rights [ |
              mrt:source [ | mrt:subject [ | mrt:type [ |
              mrt:resourceType ]]]]]]]]]]]]]
  @url | [algorithm] | [value] | [size] | | filename | [primary]
                  [ | local] [ | creator [ | title [ | date [ |
                  contributor [ | coverage [ | description [ |
                  format [ | language [ | publisher [ | relation [ |
                  rights [ | source [ | subject [ | type [ |
                  resourceType ]]]]]]]]]]]]]
  ...
[ #%eof ]
```

Note the use of the leading "@" to indicate the URLs of the individual object manifests, which are defined external to the batch manifest.

NOTE This interpretation of the Checkm @inclusion mechanism – that each included URL resolves to an object manifest – is defined by the Ingest service; the Checkm specification itself is silent regarding object semantics.

5.6.3 Container batch manifest

A container batch manifest is an extended Checkm manifest listing the locations of container files for all the objects that collectively compose the batch. The Checkm profile for batch manifests (`http://uc3.cdlib.org/registry/ingest/manifest/mrt-batch-manifest`) is defined as follows:

- The URL and target filename fields MUST be specified.
- The digest algorithm, digest value, and file size fields SHOULD be specified.
- The modification time field SHOULD NOT be specified, and will be ignored if provided.
- Two Checkm extension fields are defined for the object's primary and local identifiers. If an object is being initially established by an ingest request, the primary identifier MUST NOT be specified; if a subsequent object version is being created, the primary identifier MUST be specified. In either case, at least local identifier, meaningful in the object's curatorial context, SHOULD be specified. An arbitrary number of local identifiers MAY be specified using a semicolon-separated list. Any semicolon embedded in a local identifier MUST be represented by the standard ERC escape notation "`%sc`".
- Four Checkm extension fields are defined for the Dublin Kernel "who", "what", "when", and "why" elements, corresponding to the object's creator, title, significant date, and expository note. These fields SHOULD be specified only if they are known.
- Eleven Checkm extension fields are defined for the Dublin Core "contributor", "coverage", "description", "format", "publisher", "relation", "rights", "source", "subject", and "type" elements. These fields are OPTIONAL.
- Multiple values can be specified for any DK or DC element using a semicolon ";" as an internal field separator.

- The first two entries in the manifest MUST be structured comments specifying the Checkm conformance level and profile identifier.
- The next three entries SHOULD be structured comments specifying namespace prefixes and field definitions.
- The last entry in the manifest SHOULD be a structured comment explicitly representing the end of the file.

```
   #%checkm_0.7
   #%profile | http://uc3.cdlib.org/registry/ingest/manifest/mrt-
                                                container-batch-manifest
 [ #%prefix  | mrt: | http://uc3.cdlib.org/ontology/mom# ]
 [ #%prefix  | nfo: | http://www.semanticdesktop.org/ontologies/2007/03/
                                                              22/nfo# ]
 [ #%fields  | nfo:fileUrl  | nfo:hashAlgorithm    | nfo:hashValue |
               nfo:fileSize | nfo:fileLastModified | nfo:fileName  |
               mrt:primaryIdentifier | mrt:localIdentifier | mrt:creator |
               mrt:title | mrt:date [ | mrt:contributor [ | mrt:coverage [
             | mrt:description [ | mrt:format [ | mrt:language [ |
               mrt:publisher [ | mrt:relation [ | mrt:rights [ |
               mrt:source [ | mrt:subject [ | mrt:type [ |
               mrt:resourceType ]]]]]]]]]]]]]
   url | [algorithm] | [value] | [size] | | filename | [primary]
                   [ | local] [ | creator [ | title [ | date [ |
                   contributor [ | coverage [ | description [ |
                   format [ | language [ | publisher [ | relation [ |
                   rights [ | source [ | subject [ | type [ |
                   resourceType ]]]]]]]]]]]]]
   ...
 [ #%eof ]
```

5.6.4 Single file batch manifest

A single file batch manifest is an extended Checkm manifest listing the locations of individual files that define all the objects that collectively compose the batch. The Checkm profile for batch manifests (`http://uc3.cdlib.org/registry/ingest/manifest/mrt-single-file-manifest`) is defined as follows:

- The URL and target filename fields MUST be specified.
- The digest algorithm, digest value, and file size fields SHOULD be specified.
- The modification time field SHOULD NOT be specified, and will be ignored if provided.
- Two Checkm extension fields are defined for the object's primary and local identifiers. If an object is being initially established by an ingest request, the primary identifier MUST NOT be specified; if a subsequent object version is being created, the primary identifier MUST be specified. In either case, at least local identifier, meaningful in the object's curatorial context, SHOULD be specified. An arbitrary number of local identifiers MAY be specified using a semicolon-separated list. Any semicolon embedded in a local identifier MUST be represented by the standard ERC escape notation "`%sc`".
- Four Checkm extension fields are defined for the Dublin Kernel "who", "what", "when", and "why" elements, corresponding to the object's creator, title, significant date, and expository note. These fields SHOULD be specified only if they are known.
- Eleven Checkm extension fields are defined for the Dublin Core "contributor", "coverage", "description", "format", "publisher", "relation", "rights", "source", "subject", and "type"

elements. These fields are OPTIONAL.

- Multiple values can be specified for any DK or DC element using a semicolon ";" as an internal field separator.
- The first two entries in the manifest MUST be structured comments specifying the Checkm conformance level and profile identifier.
- The next three entries SHOULD be structured comments specifying namespace prefixes and field definitions.
- The last entry in the manifest SHOULD be a structured comment explicitly representing the end of the file.

```
#%checkm_0.7
#%profile | http://uc3.cdlib.org/registry/ingest/manifest/mrt-
                                                single-file-batch-manifest
[ #%prefix  | mrt: | http://uc3.cdlib.org/ontology/mom# ]
[ #%prefix  | nfo: | http://www.semanticdesktop.org/ontologies/2007/03/
                                                              22/nfo# ]
[ #%fields  | nfo:fileUrl  | nfo:hashAlgorithm    | nfo:hashValue |
              nfo:fileSize | nfo:fileLastModified | nfo:fileName  |
              mrt:primaryIdentifier | mrt:localIdentifier | mrt:creator |
              mrt:title | mrt:date [ | mrt:contributor [ | mrt:coverage [
            | mrt:description [ | mrt:format [ | mrt:language [ |
              mrt:publisher [ | mrt:relation [ | mrt:rights [ |
              mrt:source [ | mrt:subject [ | mrt:type [ |
              mrt:resourceType ]]]]]]]]]]]]]
  url | [algorithm] | [value] | [size] | | filename | [primary]
                  [ | local] [ | creator [ | title [ | date [ |
                  contributor [ | coverage [ | description [ |
                  format [ | language [ | publisher [ | relation [ |
                  rights [ | source [ | subject [ | type [ |
                  resourceType ]]]]]]]]]]]]]
  ...
[ #%eof ]
```

5.6.5 Notification

The submission notification MUST minimally include the following properties, which can be represented in ANVL, CSV, RDF/Turtle, XHTML, or XML format, as indicated by the ReponseForm parameter. Although these properties are defined on a per-object basis, only a single submission notification is sent per batch, regardless of the number of jobs in that batch.

- Batch identifier. [not:batch]
- Submitting user agent. [not:submitter]
- Filename. [not:filename]
- Submission type: *file*, *container*, or *object-manifest*. [not:type]
- Profile identifier. [not:profile]
- Primary identifier, or "`(:unas)`" if not supplied as part of the submission. [not:primaryIdentifier]
- Object creator, or "`(:unas)`" if not supplied as part of the submission. [not:creator]
- Object title, or "`(:unas)`" if not supplied as part of the [not:title]

submission.

- Object date, or "`(:unas)`" if not supplied as part of the submission. [not:date]
- Object local identifier, or "`(:unas)`" if not supplied as part of the submission. [not:localIdentifier]
- Object expository note, if supplied as part of the submission. [not:note]
- File message digest (*optional*). [not:digest]
  - Digest type: *Adler-32*, *CRC-32*, *MD2*, *MD5*, *SHA-1*, *SHA-384*, or *SHA-512* [not:digestType]
  - Digest hexadecimal value. [not:digestValue]
- Submission date/timestamp. [not:submitted]
- Status: *pending*. [not:status]
- Actionable reference to the batch state. [not:batchState]
- Support URI for the service. [not:supportURI]

To aid in the automated processing of mailed notifications, the subject line of the asynchronous email notification message MUST confirm to the Merritt template:

```
Subject: service [instance]: status -- message: extra; ...
```

where *service* is "`Ingest`", *instance* is the service instance, "`[dev]`" or "`[stg]`" (or not provided, if production), *status* is "`OK`" or "`Fail`"; *message* is "`Submission queued`"; and *extra* is the batch identifier. For example:

```
Subject: Ingest: OK -- Submission queued: bid-batch-uuid
```

The body of the email message MUST be in ANVL format:

```
Submission ID: bid-batch-uuid
Job(s):

  :Number of pending job(s): k
  :Number of completed job(s): m
  :Number of failed job(s): n

User agent: identity/name
Submission date: yyyy-mm-ddThh:mm:ss-hh:mm
Status: QUEUED
```

More detailed notification information is provided in an attached file using the notification format specified in the submission profile, which can be ANVL, CSV, JSON, RDF/Turtle, XHTML, or XML.

All submission notification fields MUST be specified in the CSV notification. Fields for which no values were supplied by the submitting user agent or specified by the Ingest service MUST be assigned the ANVL coded value "`(:unas)`" (unassigned). The first line of a CSV notification MUST define the field labels.

```
Batch,Submitter,Filename,Type,Profile,PrimaryIdentifier,Creator,...
batch,submitter,filename,type,profile,primaryidentifier,creator,...
...
```

5.6.6 HTML interface

The client-side HTML interface for the RESTful API COULD look something like:

Figure 2 – Client HTML interface.

5.7 Submit Object

METHOD Submit-object			[*non-idempotent*, *unsafe*]
Submitter	Agent	Mandatory	Submitting user agent.
Filename	String	Mandatory	Submission package filename.
File	Octet-stream Tar | Tgz | Zip Checkm	Mandatory	Submission package file.
Type	Enum	Optional	Submission package type: *file*, *container*, or *object-manifest*. NOTE: In the absence of an explicit "Type" argument in a *Submit Object* request, the submission package file type is determined implicitly as follows: • Determine if the package file is a Checkm manifest as evidenced by a "`#%checkm`" structured comment as the first line of the file. • If a Checkm manifest, use the embedded profile identifier specified by the "`#%profile`" structured comment to determine the type: *object-manifest*, *batch-manifest*, *container-batch-manifest*, or *single-file-batch-manifest*. • If not a manifest, determine if the file is a Tar, Gzip, or Zip container as evidenced by a "`.tar`", "`.gz`", or "`.zip`" file extension, or if no file extension is specified, by the HTTP Content-type header MIME type being one of "application/x-gzip", "application/x-tar", or "application/zip". • If not a container, assume the type is *file*.
Profile	Identifier	Mandatory	Submission profile identifier.
Batch	Identifier	Optional	Batch identifier.
PrimaryIdentifier	Identifier	Optional	Object ARK identifier.
DigestType	Enum	Optional	Package message digest type. The supported types SHOULD include: Adler-32, CRC-32, MD2, MD5, SHA-1, SHA-256, SHA-384, and SHA-512.
DigestValue	String		Package hexadecimal message digest value.
Creator	String	Optional	Person responsible for the object, e.g. creator, contributor, publisher, etc., equivalent to ERC "who".
Title	String	Optional	Object title, equivalent to ERC "what".

Date	String	Optional	Meaningful object date, equivalent to ERC "when".
LocalIdentifier	String	Optional	Locally-meaningful alternative object identifier, equivalent to ERC "where".
RetainTargetURL	Boolean	Optional	If "true", do not send EZID a Merritt-defined target URL; responsibility for properly setting the target URL rests externally.
DC.contributor	String	Optional	DC "contributor", an individual, corporate, or automated agent contriburting to the object.
DC.coverage	String	Optional	DC "coverage", a spatial or temporal topic or statement of applicability or jurisdiction.
DC.description	String	Optional	DC "description", an account of the object.
DC.format	String	Optional	DC "format", the format, medium, or dimensions of the object.
DC.language	String	Optional	DC "language", a language used by the object.
DC.publisher	String	Optional	DC "publisher", an individual or corporate agent publhsing the object.
DC.relation	String	Optional	DC "relation", a related resource.
DC.rights	String	Optional	DC "rights", a statement of intellectual property rights hold in or over the object.
DC.source	String	Optional	DC "source", a related resource from which the object is derived.
DC.subject	String	Optional	DC "subject", the topic of the object.
DC.type	String	Optional	DC "type", the nature or genre of the object.
DataCite.resourceType	String	Optional	DataCite "resourceType", the general type of the object.
Notification	String	Optional	Notification email address. Multiple addresses can be specified in the string if separated by a semicolon ";" and one or more white space characters.
ResponseForm	Enum	Optional	Response form. The supported forms SHOULD include ANVL (default for command line interfaces), CSV, XHTML (default for web interfaces), JSON, RDF/Turtle, and XML.
RETURN	*Response form*	Mandatory	Job notification (documenting this single job).
SIDE EFFECTS	If no batch identifier is specified, a unique identifier is minted and associated with the job. The job object is submitted to the Storage service and node specified by the profile via the Storage service's *Add-version* method for synchronous processing. If this is the last job of a batch to complete its processing, an ingest notification (documenting all batch jobs) is sent to all email addresses defined in the "notification" parameter and the submission profile. All of these activities are performed by Ingester handlers, described in Section § 6.4.		
ERRORS	400	Badly-formed request.	
	401	User agent not authorized.	

	415	Unsupported package type.
	413	Submission too large.
	400	Empty submission.
	400	Package digest verification failed.
	404	Batch not found.
	404	Profile not found.
	404	Minter not found
	404	Storage service not found.
	404	Storage node not found.
	415	Invalid resourceType
	415	Unsupported response form.
	503	Service unavailable.
	500	Service error.
NOTE	Any Dublin Kernel or Core element MAY be repeated.	

- RESTful API

NOTE RESTful requests are formatted assuming an underlying HTML form [HTML] and using the "multipart-form-data" content type [Multipart].

```
UA: POST /submit-object HTTP/1.x
UA: Host: ingest.cdlib.org
UA: Accept: response/form
UA: Content-type: multipart/form-data; boundary=boundary
UA:
UA: --boundary
UA: Content-disposition: form-data; name="submitter"
UA:
UA: user
UA: --boundary
UA: Content-disposition: form-data; name="file"; filename="filename"
UA: Content-type: application/octet-stream | application/x-gzip |
UA:               application/x-tar | application/zip | text/checkm
UA:
UA: --boundary [
UA:  Content-disposition: form-data; name="type"
UA:
UA: type
UA: --boundary ]
UA: Content-disposition: form-data; name="profile"
UA:
UA: profile
UA: --boundary [
UA: Content-disposition: form-data; name="batch"
UA:
UA: batch
UA: --boundary ] [
UA: Content-disposition: form-data; name="primaryIdentifier"
UA:
```

```
UA: identifier
UA: --boundary ] [
UA: Content-disposition: form-data: name="digestType"
UA:
UA: digest-type
UA: --boundary
UA: Content-disposition: form-data; name="digestValue"
UA:
UA: digest-value
UA: --boundary ]
UA: Content-disposition: form-data; name="creator"
UA:
UA: creator
UA: --boundary ]
UA: Content-disposition: form-data; name="title"
UA:
UA: title
UA: --boundary ]
UA: Content-disposition: form-data; name="date"
UA:
UA: date
UA: --boundary ]
UA: Content-disposition: form-data: name="localIdentifier"
UA:
UA: identifier
UA: --boundary ] [
UA: Content-disposition: form-data: name="retainTargetURL"
UA:
UA: true | false
UA: --boundary ] [
UA: Content-disposition: form-data: name="DC.contributor"
UA:
UA: contributor
UA: --boundary ] [
UA: Content-disposition: form-data: name="DC.description"
UA:
UA: description
UA: --boundary ] [
UA: Content-disposition: form-data: name="DC.format"
UA:
UA: format
UA: --boundary ] [
UA: Content-disposition: form-data: name="DC.language"
UA:
UA: language
UA: --boundary ] [
UA: Content-disposition: form-data: name="DC.publisher"
UA:
UA: publisher
UA: --boundary ] [
UA: Content-disposition: form-data: name="DC.relation"
UA:
UA: relation
UA: --boundary ] [
UA: Content-disposition: form-data: name="DC.rights"
UA:
```

```
UA: rights
UA: --boundary ] [
UA: Content-disposition: form-data: name="DC.source"
UA:
UA: source
UA: --boundary ] [
UA: Content-disposition: form-data: name="DC.subject"
UA:
UA: subject
UA: --boundary ]
UA: Content-disposition: form-data: name="DC.type"
UA:
UA: type
UA: --boundary ] [
UA: Content-disposition: form-data: name="DataCite.resourceType"
UA:
UA: resourceType
UA: --boundary ] [
UA: Content-disposition: form-data: name="notification"
UA:
UA: email; ...
UA: --boundary ]

OS: HTTP/1.x 201 Created
OS: Content-type: response/form
OS: Location: http://cutie.cdlib.org/state/queue/batch/job
OS:
OS: notification
```

- Command line API

```
% ingest submitObject filename [-T type] [-t form] [-o file]
```

5.7.1 Notification

A job notification MUST minimally include the following properties, which can be represented in ANVL, CSV, RDF/Turtle, XHTML, or XML format, as indicated by the ReponseForm parameter.

- Batch identifier. [not:batch]
- Submitting user agent. [not:submitter]
- Filename. [not:filename]
- Submission type: *file*, *container*, or *object-manifest*. [not:type]
- Profile identifier. [not:profile]
- Primary identifier, or "`(:unas)`" if not supplied as part of the submission. [not:suppliedIdentifier]
- Primary identifier, or "`(:unas)`" if not retrieved during ingest processing. [not:retrievedIdentifier]
- Primary identifier, or "`(:unas)`" if not assigned during ingest processing. [not:assignedIdentifier]
- Persistent URL of object landing page. [not:persistentURL]

- Object creator, or "`(:unas)`" if not supplied as part of the submission. [not:creator]
- Object title, or "`(:unas)`" if not supplied as part of the submission. [not:title]
- Object date, or "`(:unas)`" if not supplied as part of the submission. [not:date]
- Object local identifier(s), or "`(:unas)`" if not supplied as part of the submission). [not:localIdentifier]
- Object expository note, if supplied as part of the submission. [not:note]
- File message digest (*optional*). [not:digest]
    - Digest type: *Adler-32*, *CRC-32*, *MD2*, *MD5*, *SHA-1*, *SHA-384*, or *SHA-512* [not:digestType]
    - Digest hexadecimal value. [not:digestValue]
- Submission date/timestamp. [not:submitted]
- Consumption date/timestamp. [not:consumed]
- Completion date/timestamp. [not:completed]
- Actionable reference to Access service object state, if successful; otherwise "`(:null)`". [not:objectState]
- Status: *pending*, *completed*, or *failed*. [not:status]
- Error message, if "Status" = *failed*. [not:message]
- Support URI for the service. [not:supportURI]

. . .

5.7.2 HTML interface

The client-side interface for the RESTful API COULD look something like:

Figure 3 – Client HTML interface.

5.8 Set Queue Behavior

METHOD Set-queue-behavior			[*idempotent*, *unsafe*]
Status	Enum	Mandatory	Queue consumer status: *restart* or *pause*.
Mode	Enum		Queue consumer mode: *immediate* or *wait*.
ResponseForm	Enum	Optional	Response form. The supported forms SHOULD include ANVL (default for command line interfaces), JSON, RDF/Turtle, XHTML (default for web interfaces), and XML.
RETURN	Response form	Mandatory	Queue state.
SIDE EFFECTS	Queue consumer status is set to the specified value.		
ERRORS	400 Badly-formed request.		
	401 Unauthorized user agent.		
	404 Queue not found.		
	415 Unsupported response form.		
	503 Service unavailable.		
	500 Service error.		

- RESTful API

```
UA: PUT /state/queue?[M=immediate|wait][&][S=restart|pause] HTTP/1.x
UA: Host: ingest.cdlib.org
UA: Accept: response/form
UA:

OS: HTTP/1.x 200 OK
OS: Content-type: response/form
OS:
OS: state
```

- Command line API

```
% ingest setQueueStatus [-M immediate|wait] [-S restart|pause]
  [-t form] [-o file]
```

5.9 Request Identifier

METHOD Request-identifier			[*non-idempotent*, *unsafe*]
Requester	Agent	Mandatory	Requesting user agent.
Profile	Identifier	Mandatory	Submission profile identifier.
ERC	String	Mandatory	Minimal descriptive Electronic Record Citation (ERC) metadata for the identified resource, including "who" (creator), "what" (title), and "when" (date). Any value not known MUST be supplied as "(:unas)".
ResponseForm	Enum	Optional	Response form. The supported forms SHOULD include ANVL (default for command line interfaces), CSV, XHTML (default for web interfaces), JSON, RDF/Turtle, and XML.
RETURN	*Response form*	Mandatory	Minted identifier(s).
SIDE EFFECTS	The requested identifiers (always including an ARK, and OPTIONALLY, a DOI) are minted and bound to the ERC metadata.		
ERRORS	400	Badly-formed request.	
	401	Unauthorized user agent.	
	404	Submission profile not found.	
	415	Unsupported response form.	
	503	Service unavailable.	
	500	Service error.	
NOTE	The form of the ERC metadata		

- RESTful API

```
UA: POST /request-identifier HTTP/1.x
UA: Host: ingest.cdlib.org
UA: Accept: response/form
UA:
UA: --boundary
UA: Content-disposition: form-data; name="profile"
UA:
UA: profile
UA: --boundary
UA: Content-disposition: form-data; name="erc"
UA:
UA: erc: who: creator%0awhat: title%0awhen: date
UA: --boundary
UA: [ Content-disposition: form-data; name="request-form"
UA:
UA: form
UA: --boundary ]

OS: HTTP/1.x 200 OK
OS: Content-type: response/form
OS:
```

```
OS: ark: ark
OS: [ doi: doi ]
```

- Command line API

```
% ingest requestIdentifier profile [-t form] [-o file]
```

6 Implementation

The Ingest service is composed of five independent system components:

- *Submitter*. The Submitter accepts request for asynchronous batch or single object processing and creates a job for each object.
- *Queue*. The Queue manages job sidecar information for jobs awaiting processing.
- *Consumer*. The Consumer polls for the availability of a job at the head of the *Queue*.
- *Ingester*. The Ingester processes a single job by invoking the Storage service *Add-version* method.
- *File system*. The mountable file system manages job payloads.

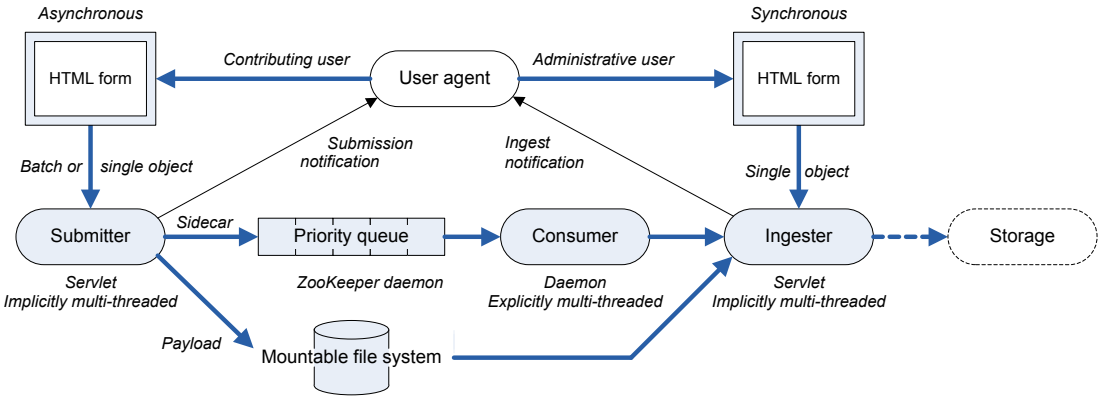

Figure 4 – Ingest architecture

The Submitter and Ingester are implemented as servlets; the Consumer is a multi-threaded daemon, with a thread assigned to each consumed job. The Submitter handles asynchronous Ingest requests for batches and single objects from Ingest users; the Ingester handles synchronous Ingest requests for single objects, either directly from administratively-privileged users or from the Consumer.

6.1 Submitter

The processing of a batch is performed as a configurable sequence of *handlers*, whose responsibilities are to push job sidecar metadata onto the queue for each object specified in the batch, write job payload data into the shared file system, and send the submission notification.

6.2 Queue

The priority queue is implemented using ZooKeeper [ZooKeeper].

6.3 Consumer

At the conclusion of ingest processing of a given job the Consumer's behavior is:

- If the polling mode is *immediate*, poll the queue to see if another pending job is available; if

the mode is *wait*, wait for the polling interval to expire before polling the queue.

- If polling indicates that a job is available, consume it; otherwise, wait for the polling interval to expire before polling the queue.

6.4 Ingester

The processing of a job is performed as a configurable sequence of *handlers*. The canonical sequence of handler steps is:

NOTE All pathnames used below are defined relative to a staging area specific to the submission batch and job, "*ingest_home*/queue/*batch*/*job*". An arbitrary submission package is represented by "*package*"; an arbitrary package file or component is represented by "*component*".

1. *Initialize*. [Mandatory]

   (1) Create a Merritt ingest metadata file, "mrt-ingest.txt", in ANVL syntax, documenting the parameters of the *Submit-object* method invocation and relevant properties of the controlling submission profile in the staging area "system" directory.

   (2) Create a Merritt membership file, "mrt-membership.txt", containing the collection identifiers retrieved from the submission profile, in the staging area "system" directory.

   (3) Create a Merritt object model (MOM) metadata file [MOM], "mrt-mom.txt", in ANVL syntax, specifying the object's primary identifier, if known, type, role, and optionally, aggregate and local identifiers, if known, in the starting area "system" directory.

   (4) Create a stub Merritt object resource map, "mrt-object-map.ttl", in Turtle syntax, in the staging area "system" directory. The resource map defines the object initially as an aggregation ("ore:aggregates") of the ingest, membership, MOM, and owner metadata files as well as the resource map itself. Each metadata file is associated to the object with a "mrt:hasMetadata" relationship, and is the subject of "mrt:metadataSchema" and "mrt:mimeType" relationships.

   (5) Create a Merritt owner file, "mrt-owner.txt", containing the owner identifier retrieved from the submission profile, in the staging area "system" directory.

./system/mrt-ingest.txt

```
ingest: name
submissionDate: yyyy-mm-ddThh:mm:ss±zz:zz
batch: batch
job: job
userAgent: user
file: filename
type: type
profile: profileid
queuePriority: priority
storageService: service
storageNode: node
notification: email[; ...]
suppliedIdentifier: objectid | (:unas)
```

```
digestType: type | (:unas)
digestValue: value | (:unas)
creator: creator | (:unas)
title: title | (:unas)
date: date | (:unas)
localIdentifier: localid[; localid[; ...]] | (:unas)
note: note | (:unas)
notification: email[; ...]
```

NOTE Multiple notification email addresses are separated by semicolons (“;”) to conform to ANVL syntax.

./system/mrt-membership.txt

```
collectionid
...
```

./system/mrt-mom.txt

```
[ primaryIdentifier: objectid ]
type: type
role: role
[ aggregate: aggregate ]
[ localIdentifier: localid[; localid[; ...]]]
```

NOTE The primary and local identifiers are the two most significant properties of an ingested object. The primary identifier is the basis for all manipulation and reporting for the object. The local identifier is the mechanism for associating an object in a Merritt curation environment with external discovery and administrative metadata. The primary and local identifiers MUST be documented in the MOM file at this stage *only if they are known*. If the identifiers were *not* specified as *Submit* or *Submit-object* request parameters, the “primaryIdentifier” and/or “localidentifier” elements MUST not be written. The primary identifier will always be available by the end of processing of Step 8 [*Mint*], below.

NOTE Multiple local identifiers MUST be expressed as a semicolon-separated list. Any semicolon embedded in an identifier MUST be represented in the standard ERC escape notation “%sc”.

./system/mrt-object-map.ttl

```
@prefix msc: <http://uc3.cdlib.org/ontology/schema#>.
@prefix mrt: <http://uc3.cdlib.org/ontology/mom#>.
@prefix ore: <http://www.openarchives.org/ore/terms/>.
@prefix n2t: <http://n2t.net/>.
n2t:objectid
  ore:aggregates
    <http://merrritt.cdlib.org/d/objectid/0/system%2Fmrt-
ingest.txt> ,
```

```
    <http://merrritt.cdlib.org/d/objectid/0/system%2Fmrt-
membership.txt> ,
    <http://merrritt.cdlib.org/d/objectid/0/system%2Fmrt-
mom.txt ,
    <http://merrritt.cdlib.org/d/objectid/0/system%2Fmrt-
owner.txt> ,
    <http://merrritt.cdlib.org/d/objectid/0/system%2Fmrt-
object-map.ttl> ;
  mrt:hasMetadata
    <http://merrritt.cdlib.org/d/objectid/0/system%2Fmrt-
ingest.txt> ,
    <http://merrritt.cdlib.org/d/objectid/0/system%2Fmrt-
membership.txt> ,
    <http://merrritt.cdlib.org/d/objectid/0/system%2Fmrt-
mom.txt> ,
    <http://merrritt.cdlib.org/d/objectid/0/system%2Fmrt-
owner.txt> ,
    <http://merrritt.cdlib.org/d/objectid/0/system%2Fmrt-
object-map.ttl> .
<http://merrritt.cdlib.org/d/objectid/0/system%2Fmrt-
ingest.txt>
  mrt:metadataSchema
    msc:MRT-ingest ;
  mrt:mimeType
    <http://purl.org/NET/mediatypes/text/x-anvl> .
<http://merrritt.cdlib.org/d/objectid/0/system%2Fmrt-
membership.txt>
  mrt:metadataSchema
    msc:MRT-membership ;
  mrt:mimeType
    <http://purl.org/NET/mediatypes/text/plain> .
<http://merrritt.cdlib.org/d/objectid/0/system%2Fmrt-
mom.txt>
  mrt:metadataSchema
    msc:MRT-MOM ;
  mrt:mimeType
    <http://purl.org/NET/mediatypes/text/plain> .
<http://merrritt.cdlib.org/d/objectid/0/system%2Fmrt-
owner.txt>
  mrt:metadataSchema
    msc:MRT-owner ;
  mrt:mimeType
    <http://purl.org/NET/mediatypes/text/plain> .
<http://merrritt.cdlib.org/d/objectid/0/system%2Fmrt-
object-map.ttl>
  mrt:metadataSchema
    msc:MRT-ORE ;
  mrt:mimeType
    <http://purl.org/NET/mediatypes/text/turtle> ;
  ore:describes
    n2t:objectid .
```

./system/mrt-owner.txt

```
ownerid
```

2. *Accept*. [Mandatory]

   (1) Copy the submission package to the staging area "`producer`" directory.

   NOTE Merritt submission packages *always* take the form of a single file, which may be a single component object, a container file encapsulating an arbitrary number of object components, or a manifest file referencing an arbitrary number of object components.

   (2) If "Type" is *file* (as opposed to *container* or *manifest*), define an "`ore:aggregates`" relationship between the object and the single file component, i.e., the submission package file, in the system object resource map "`mrt-object-map.ttl`"

```
./producer/packageid                # Submission package
./system/mrt-ingest.txt             # Ingest metadata
./system/mrt-membership.txt         # Membership metadata
./system/mrt-mom.txt                # MOM metadata
./system/mrt-object-map.ttl         # Object resource map
```

```
...
n2t:objectid
  ore:aggregates
    <http://merritt.cdlib.orgd/d/objectid/0/producer%2f
packageid> .
```

```
./system/mrt-owner.txt              # Owner metadata
```

3. *Verify*. [Optional, if "DigestType" and "DigestValue" are available]

   (1) Compute and compare the message digest for the submission package, adding the results ("`verified`" or "`failed`") to the ingest metadata component.

```
./producer/package
./system/mrt-ingest.txt
```

```
...
packageIntegrity: verified | failed
```

```
./system/mrt-membership.txt         # Membership metadata
./system/mrt-mom.txt                # MOM metadata
./system/mrt-object-map.ttl         # Object resource map
./system/mrt-owner.txt              # Owner metadata
```

4. *Disaggregate*. [Conditional, if "Type" = *container*]

   (1) Validate the container according to its format, adding the result ("`valid`" or "`invalid`") to the ingest metadata component.

(2) Disaggregate the contents of the container in the staging area "producer" directory

(3) Define an "ore:aggregates" relationship between the object and each disaggregated file component in the system object resource map "mrt-object-map.ttl".

(4) Delete the container file.

```
[ ./producer/container ]              # Deleted after processing
[ ,/producer/mrt-dc.xml ]             # Optional DC metadata
[ ./producer/mrt-erc.txt ]            # Optional ERC metadata
[ ./producer/mrt-manifest.txt ]       # Optional Checkm manifest
  ./producer/fileid                   # Disggregated file components
  ...
  ./system/mrt-ingest.txt             # Ingest metadata
```

```
...
containerValidity: valid | invalid
```

```
  ./system/mrt-membership.txt         # Membership metadata
  ./system/mrt-mom.txt                # MOM metadata
  ./system/mrt-object-map.ttl         # Object resource map
```

```
...
n2t:objectid
  ore:aggregates
    <http://merritt.cdlib.org/d/objectid/0/producer%2F
fileid> ,
    ...
    <http://merritt.cdlib.org/d/objectid/0/producer%2F
fileid> .
```

```
  ./system/mrt-owner.txt              # Owner metadata
```

NOTE Container-based submission packages MAY contain a Checkm manifest component with the reserved filename "mrt-manifest.txt" and/or a DC component with the reserved filename "mrt-dc.xml" and/or an ERC component with the reserved filename "mrt-erc.txt".

5. *Retrieve*. [Conditional, if "Type" = *object-manifest*]

(1) Validate the manifest according to its format, adding the result ("valid" or "invalid") to the ingest metadata component.

(2) Read the manifest and perform a (possibly parallel) retrieval of all referenced file components, placing them in the staging area "producer" directory.

(3) Define an "ore:aggregates" relationship between the object and each referenced file component in the system object resource map "mrt-object-map.ttl".

```
  ./producer/manifestid               # Object manifest
[ ,/producer/mrt-dc.xml ]             # Optional DC metadata
[ ./producer/mrt-erc.txt ]            # Optional ERC metadata
```

```
./producer/fileid                    # Referenced file components
...
./system/mrt-ingest.txt              # Ingest metadata
```

```
...
manifestValidity: valid | invalid
```

```
./system/mrt-membership.txt          # Membership metadata
./system/mrt-mom.txt                 # MOM metadata
./system/mrt-object-map.ttl          # Object resource map
```

```
...
n2t:objectid
  ore:aggregates
    <http://merritt.cdlib.org/d/objectid/0/producer%2F
fileid> ,
    ...
    <http://merritt.cdlib.org/d/objectid/0/producer%2F
fileid> .
```

```
./system/mrt-owner.txt               # Owner metadata
```

NOTE Manifest-based submission packages MAY contain an ERC component with the reserved filename "`mrt-erc.txt`".

6. *Corroborate.* [Conditional, if "Type" = *container* and a manifest component (`mrt-manifest.txt`) is available]

   (1) Validate the manifest according to its format, adding the results ("`valid`" or "`invalid`") to the ingest metadata component.

   (2) Read the manifest and compute and compare file size and digest values, adding the results ("`verified`" or "`failed`") to the ingest metadata component. All files in the manifest MUST be found in the staging area; all files in the staging area, except the manifest itself, MUST be referenced in the manifest.

```
[ ./producer/manifestid ]            # Submission manifest
[ ./producer/mrt-dc.xml ]            # Optional DC metadata
[ ./producer/mrt-erc.txt ]           # Optional ERC metadata
[ ./producer/mrt-manifest.txt ]      # Optional container manifest
  ./producer/fileid                  # File components
  ...
  ./system/mrt-ingest.txt            # Ingest metadata
```

```
...
manifestFile: manifestid
manifestValidity: valid | invalid
manifestIntegrity: verified | failed
```

```
./system/mrt-membership.txt          # Membership metadata
```

```
./system/mrt-mom.txt                # MOM metadata
./system/mrt-object-map.ttl         # Object resource map
./system/mrt-owner.txt              # Owner metadata
```

NOTE For manifest-based submission packages a manifest will *always* be available as the package file; for container-based packages a Checkm manifest, "`mrt-manifest.txt`", *may* be available.

7. *Characterize*. [Optional]

   (1) Invoke JHOVE2 against each component file in the staging area "`producer`" directory, copying the results "`mrt-jhove2.xml`" to the staging area "`system`" directory.

   (2) Define a "mrt:`hasMetadata`" relationship between the component file and its JHOVE2 output, a "mrt:`metadataSchema`" relationship between the JHOVE2 output and "`JHOVE2`", and a "mrt:`mimeType`" relationship between the JHOVE2 output and "`text/xml`" in the system object resource map "`mrt-object-map.ttl`".

```
[ ./producer/manifestid ]           # Submission manifest
[ ./producer/mrt-dc.xml ]           # Optional DC metadata
[ ./producer/mrt-erc.txt ]          # Optional ERC metadata
[ ./producer/mrt-manifest.txt ]     # Optional container manifest
  ./producer/fileid                 # File components
  ...
  ./system/mrt-ingest.txt           # Ingest metadata
  ./system/mrt-jhove2.xml           # JHOVE2 metadata
  ./system/mrt-membership.txt       # Membership metadata
  ./system/mrt-mom.txt              # MOM metadata
  ./system/mrt-object-map.ttl       # Object resource map
```

```
...
<http://merritt.cdlib.org/d/objectid/0/producer%2Ffileid>
  mrt:hasMetadata
    <http://merritt.cdlib.org/d/objectid/0/system%2Fmrt-
jhove2.xml> .
<http://merritt.cdlib.org/d/objectid/0/system%2Fmrt-
jhove2.xml>
  mrt:metadataSchema
    msc:JHOVE2 ;
  mrt:mimeType
    <http://purl.org/NET/mediatypes/text/xml> .
```

```
./system/mrt-owner.txt              # Owner metadata
```

8. *Mint*. [Mandatory]

   (1) Define the Dublin Kernel "who", "what", "when", "where" elements by preferring the "who", "what", "when", and "where" values from the optional ERC metadata component "`mrt-erc.txt`" in the staging area "`producer`" directory over the analogous values in a batch manifest, which in turn are preferred over any "`creator`", "`title`" (or "`alternative`"), "`date`"(or "`available`", "`created`", "`issued`", "`modified`", or

"valid"), or "identifier" elements from the optional Dublin Core (DC) metadata component "mrt-dc.xml" in the staging area "producer" directory, which in turn are preferred over the analogous values in a batch manifest, which in turn are preferred over the optional Creator, Title, Date, and LocalIdentifier values submitted as part of the *Submit* or *Submit-object* request.

NOTE The Kernel "who", "what", "when", and "where" elements MUST be defined; if no value is available, the string "(:unas)" MUST be used. Multiple values for a single element MUST be concatenated using a semicolon ("**;**") as the separator. Any semicolon embedded in a value MUST be represented in the standard ERC escape notation "%sc".

NOTE Any ARK identifier that is the value of a "where" element found in the ERC component MUST be assumed to define the object primary identifier; any non-ARK identifiers MUST be assumed to be local identifiers.

(2) If the submission profile identifier scheme is "DOI" and there is a DataCite metadata file, "mrt-datacite.xml", in the staging area "producer" directory, create an escaped version of the file contents, with all percent signs, "%", converted to "%25"; all newlines, "\n" (U+000A), converted to "%0A"; all carriage returns, "\r" (U+000D), converted to "%0D"; and all colons, ":", converted to "%3A". (Refer to the EZID API document [EZID] for more information.)

(3) If the submission profile identifier scheme is "DOI" but there is *not* a "mrt-datacite.xml" file is defined in the staging area "producer" directory, define the DataCite "publicationdate" element by Dublin Kernel "when" element of the "mrt-erc.txt" component in the "system" directory. If the Dublin Kernel value is "(:unas)" set the "publicationdate" to the current year.

Also define the Data Cite "resourceType" element by preferring the Dublin Core.resourceType value submitted as part of a *Submit* or *Submit-object* request over the "dc:format" value from the Dublin Core metadata component "mrt-dc.xml" in the staging area "producer" directory, which in turn is preferred over the analogous value in a batch manifest, which in turn is preferred over a Dublin Core.format value submitted as part of the *Submit* or *Submit-object* request. If the "resourceType" value is not a valid DataCite resourceType but is a MIME type, map it to a valid "resourceType" as follows (where an asterisk "*" indicates wildcard matching):

"application/*" ⇒ "Dataset"
"audio/*" ⇒ "Sound"
"example" ⇒ "Text"
"image/*" ⇒ "Image"
"message/*" ⇒ "Text"
"model/*" ⇒ "Model"
"multipart/*" ⇒ "Collection"
"text/*" ⇒ "Text"
"video/*" ⇒ "Film"

NOTE The "example" MIME type does *not* have any defined subtypes, so there will not be any text after the final "e".

(4) If the primary identifier ARK is null and the local identifier list is not null:

i. Retrieve the storage node from the submission profile.

ii. Iteratively retrieve the object primary identifier ARK associated with the profile and each local identifier in the Storage service, if defined.

```
UA: GET /primary/node/profile/localid HTTP/1.x
UA: Host: store.cdlib.org
UA:

OS: HTTP/1.x 200 OK
OS: Content-type: text/plain
OS:
OS: objectid
```

iii. This will result in either: (a) no primary identifier ARK associated with any local identifier; (b) a single primary identifier ARK associated with all or some local identifiers; or (c) multiple primary identifier ARKs associated with various local identifiers. This last condition (c) MUST be considered an error and SHALL terminate further ingest processing.

(5) If the primary identifier ARK is null and is *not* retrievable via a local identifier as described in Step (4) and the local identifier is a DOI of the form "doi:*doi*":

(a) Retrieve the metadata associated with the DOI from EZID:

```
UA: GET /id/doi:doi HTTP/1.x
UA: Host: ezid.cdlib.org
UA:

OS: HTTP/1.x 200 OK
OS: Content-type: text/plain
OS:
OS: success: doi:doi
OS: ...
OS: [ _shadowedby: ark ]
OS: ...
```

(b) If a shadow ARK "*ark*" is returned, use it as the primary identifier and continue to Step (7).

(c) If a shadow ARK "*ark*" is *not* returned, continue to Step (6),

(d) If a 400 Bad Request response code is returned, continue to Step (6).

(6) If the primary identifier ARK is null and is *not* retrievable via a local identifier as described in Step (4) and is *not* retrievable as a shadow ARK as described in Step (5):

i. Retrieve the identifier scheme, namespace, minter, and context from the submission profile.

(a) The ARK-only minter SHOULD be "`ark:/13030/fm5`".

(b) The DOI minter SHOULD be of the form "`doi:`*`prefix`*".

ii. Invoke the minter, passing the namespace and context to mint a new object primary identifier in the identifier scheme and namespace and associated with the context, adding the newly minted primary identifier as the value of the "`assignedIdentifier`" element in the ingest metadata component "`mrt-ingest.txt`"and as the value of the "`primaryIdentifier`" element in the MOM metadata component "`mrt-mom.txt`", both in the staging area "`system`" directory.

(a) If the submission profile identifier scheme is "ARK", a single ARK identifier is returned.

```
UA: PUT /shoulder/minter HTTP/1.x
UA: Host: ezid.cdlib.org
UA: Content-type: text/plain
UA:
UA: _group: context

OS: HTTP/1.x 201 Created
OS: Content-type: text/plain
OS:
OS: success: ark
```

(b) If the submission profile identifier scheme is "DOI", both a DOI *and* an ARK identifier are returned. The DOI MUST be used subsequently only as a local identifier (in addition to any other local identifiers introduced through the Kernel "where" element); the object primary identifier MUST be the ARK.

```
UA: PUT /shoulder/minter HTTP/1.x
UA: Host: ezid.cdlib.org
UA: Context-type: text/plain
UA:
UA: _group: context

OS: HTTP/1.x 201 Created
OS: Content-type: text/plain
OS:
OS: success: doi ; ark
```

(7) If the primary identifier is not null or is retrievable via a local identifier as described in Step (4) or is retrievable as a shadow ARK as described in Step (5):

i. Invoke EZID for the object primary ID ARK, passing the object's landing page URL (if the "RetainTargetURL" parameter of the "Submit" or "Submit Object" methods is "false" or unspecified) and the Dublin Kernel "who", "what", "when", and "where" elements, and, if the submission profile identifier scheme is "DOI", the entire contents of the DataCite "mrt-datacite.xml" file in escaped form (see Step (2)), if defined, or if not, the Data Cite "resourceType" element, to update the metadata associated with the primary identifier. If the primary identifier was retrieved (and not originally supplied as a *Submit* or *Submit-object* request parameter), add it as the value of the "`retreivedIdentifier`" element in the ingest metadata component "`mrt-ingest.txt`" and as the value of the "`primaryIdentifier`" element in the MOM metadata component "`mrt-mom.txt`", both in the staging area "`system`" directory.

```
UA: POST /id/ark HTTP/1.x
UA: Host: ezid.cdlib.org
UA: Content-type: text/plain
UA:
UA: [ _target: http://merritt.cdlib.org/?object=objectid ]
UA: erc: who: who%0awhat: title%0awhen: when%0a where: where%0a
UA: [ datacite: datacite-metadata ]
UA: [ datacite.publicationdate: date
UA: datacite.resourcetype: resourceType ]

OS: HTTP/1.x 200 OK
OS: Content-type: text/plain
```

```
  OS:
  OS: success: objectid

[ ./producer/manifestid ]              # Submission manifest
[ ./producer/mrt-datacite.xml ]        # Optional DataCite metadata
[ ./producer/mrt-dc.xml ]              # Optional DC metadata
[ ./producer/mrt-erc.txt ]             # Optional ERC metadata
[ ./producer/mrt-manifest.txt ]        # Optional container manifest
  ./producer/fileid                    # File components
  ...
  ./system/mrt-ingest.txt              # Ingest metadata
```

```
...
retrievedIdentifier: retrievedid | (:unas)
assignedIdentifier: assignedid | (:unas)
[ localIdentifier: localid]
```

```
  ./system/mrt-jhove2.xml              # JHOVE2 metadata
  ./system/mrt-membership.txt          # Membership metadata
  ./system/mrt-mom.txt                 # MOM metadata
```

```
[ primaryIdentifier: objectid ]
```

```
  ./system/mrt-object-map.ttl
  ./system/mrt-owner.txt             # Owner metadata
```

NOTE The primary identifier is written to the MOM metadata component *only* if it is *not* supplied as a *Submit* or *Submit-object* request parameter and is minted in Step 8(3)ii or retrieved using the localIdentifier in Step 8(4)i. If the primary identifier *is* supplied as a request parameter it will already have been written to the MOM component in Step 1(3). The local identifier is always written to the MOM component in Step 1(3).

9. *Describe*. [Mandatory]

   (1) Copy the Dublin Kernel elements Creator, Title, Date, LocalIdentifier, PrimaryIdentifier, whether supplied, retrieved, or newly minted in Step 8(2), to an ERC metadata component, "`mrt-erc.txt`", in ANVL syntax, in the staging area "`system`" directory. Elements with no submitter-assigned value MUST be specified with the Dublin Kernel "`(:unas)`" code.

   (2) Define an "`ore:aggregates`" relationship between the object and the ERC metadata file, "`mrt-erc.txt`" in the "system" directory; a "`mrt:hasMetadata`" relationship between the object and the ERC metadata, a "`mrt:metadataSchema`" relationship between the ERC metadata and "`ERC`", and a "`mrt:mimeType`" relationship between the ERC metadata and "`text/anvl`" in the system object resource map "`mrt-object-map.ttl`".

   (3) If an ERC metadata component "`mrt-erc.txt`" exists in the "`producer`" directory, define a "`mrt:hasMetadata`" relationship between the object and the producer ERC metadata, a "`mrt:metadataSchema`" relationship between the ERC metadata and "`ERC`",

and a "mrt:mimeType" relationship between the ERC metadata and "text/anvl" in the system object resource map "mrt-object-map.ttl".

NOTE The producer ERC metadata will already have been added to the object aggregation, along with all other producer-supplied files, in Step 2(2), 4(3), or 5(3).

(4) If any of the non-ERC-equivalent metadata elements, that is, DC.ontributer, DC.coverage, DC.description, DC.format, DC.language, DC.publisher, DC.relation, DC.rights, DC.source, DC.subject, DC.type, or DataCite.resourceType, are defined as *Submit* request parameters or fields in a batch manifest, then copy them to a DC metadata component, "mrt-dc.xml", in XML syntax, in the staging area "system" directory. Any undefined DC elements MUST NOT be added as empty tags to the DC component. If a DC element is defined *both* as a *Submit* parameter and a batch manifest field, the batch manifest field is preferred. If defined, the value of DC.resourceType MUST be expressed as the Dublin Core "format" element.

NOTE The "system" DC metadata component is created *only* if at least one non-ERC-equivalent element is passed as a *Submit* parameter or is defined in a batch manifest.

NOTE The *Submit* parameters MAY be repeated. The elements in a batch manifest MAY be repeated as part of a semi-colon separated list in the appropriate field. Each individual instance of a repeated element MUST be added to the DC metadata component as a separate tag. Each repeated value of a given element MUST be added to the DC metadata component as a separate tag.

(5) If a DC metadata component, "mrt-dc.xml", was created in the "system" directory in Step 9(4), define an "ore:aggregates" relationship between the object and the DC metadata file; a "mrt:hasMetadata" relationship between the object and the DC metadata, a "mrt:metadataSchema" relationship between the DC metadata and "DC", and a "mrt:mimeType" relationship between the DC metadata and "text/xml" in the system object resource map "mrt-object-map.ttl".

(6) If a DC metadata component, "mrt-dc.xml", exists in the "producer" directory, define a "mrt:hasMetadata" relationship between the object and the producer DC metadata, a "mrt:metadataSchema" relationship between the DC metadata and "DC", and a "mrt:mimeType" relationship between the DC metadata and "text/xml" in the system object resource map "mrt-object-map.ttl".

NOTE The producer DC metadata will already have been added to the object aggregation, along with all other producer-supplied files, in Step 2(2), 4(3), or 5(3).

(7) If a data use agreement (DUA) metadata component, "mrt-dua.txt" exists in the "producer" directory, define a "mrt:hasMetadata" relationship between the object and the producer DUA metadata, a "mrt:metadataSchema" relationship between the DUA metadata and "MRT_DUA", and a "mrt:mimeType" relationship between the DUA metadata and "text/anvl" in the system object resource map "mrt-object-map.ttl".

(8) If a DataCite metadata component, "mrt-datacite.xml", exists in the "producer" directory, define a "mrt:hasMetadata" relationshtip between the object and the producer DataCite metadata, a "mrt:metdataSchema" relationship between the DataCite metadata and "DataCite", and a "mrt:mimeType" relationship between the DataCite metadata and "text/xml" in the system object resource map "mrt-object-map.ttl".

(9) If an EML metadata component, "mrt-eml.xml", exists in the "producer" directory,

define a "mrt:hasMetadata" relationshtip between the object and the producer EML metadata, a "mrt:metdataSchema" relationship between the EML metadata and "EML", and a "mrt:mimeType" relationship between the DataCite metadata and "text/xml" in the system object resource map "mrt-object-map.ttl".

```
[ ./producer/manifestid ]              # Submission manifest
[ ./producer/mrt-datacite.xml ]        # Optional DataCite metadata
[ ./producer/mrt-dc.xml ]              # Optional DC metadata
[ ./producer/mrt-eml.xml ]             # Optional EML metadata
[ ./producer/mrt-erc.txt ]             # Optional ERC metadata
[ ./producer/mrt-manifest.txt ]        # Optional container manifest
  ./producer/fileid                    # File components
  ...
[ ./system/mrt-dc.xml ]                # Optional DC metadata
```

```
<?xml version="1.0" encoding="UTF-8"?>
<DublinCore xmlns:xsi="http://www.w3.org/2001/XMLSchema-
                                                    instance"
            xmlns:dc="http://purl.org/dc/elements/1.1/">
[ <dc:creator>creator</dc:creator> ]
[ <dc:title>title</dc:tile> ]
[ <dc:date>date</dc:date> ]
  <dc:identifier>objectid</dc:identifier>
[ <dc:identifier>localid</dc:identifier> ]
[ <dc:contributor>contributor</dc:contributor> ]
[ <dc:coverage>coverage</dc:coverage> ]
[ <dc:description>description</dc:description> ]
[ <dc:format>format</dc:format> ]
[ <dc:language>language</dc:language> ]
[ <dc:publisher>publisher</dc:publisher> ]
[ <dc:relation>relation</dc:relation> ]
[ <dc:rights>rights</dc:rights> ]
[ <dc:source>source</dc:source> ]
[ <dc:subject>subject</dc:subject> ]
[ <dc:type>type</dc:type> ]
</DublinCore>
```

```
[ ./system/mrt-erc.txt ]               # System ERC metadata
```

```
erc:
who: creator | (:unas)
what: title | (:unas)
when: date | (:unas)
where: objectid
where: localid | (:unas)
```

```
  ./system/mrt-ingest.txt              # Ingest metadata
  ./system/mrt-jhove2.xml              # JHOVE2 metadata
  ./system/mrt-membership.txt          # Membership metadata
  ./system/mrt-mom.txt                 # MOM metadata
  ./system/mrt-object-map.ttl          # Object resource map
```

```
...
n2t:objectid
  ore:aggregates
    <http://merritt.cdlib.org/d/objectid/0/system%2Fmrt-
erc.txt> ;
  mrt:hasMetadata
    <http://merritt.cdlib.org/d/objectid/system%2Fmrt-
erc.txt> .
<http://merritt.cdlib.org/d/objectid/0/system%2Fmrt-
erc.txt>
  mrt:metadataSchema
    msc:ERC ;
  mrt:mimeType
    <http://purl.org/NET/mediatypes/text/x-anvl> . [
n2t:objectid
  mrt:hasMetadata
    producer-erc .
<http://merritt.cdlib.org/d/objectid/0/producer%2Fmrt-
erc.txt>
  mrt:metadataSchema
    msc:ERC ;
  mrt:mimeType
    <http://purl.org/NET/mediatypes/text/x-anvl> . ] [
n2t:objectid
  ore:aggregates
    <http://merritt.cdlib.org/d/objectid/0/system%2Fmrt-
dc.xml> ;
  mrt:hasMetadata
    <http://merritt.cdlib.org/d/objectid/0/system%2Fmrt-
dc.xml> .
<http://merritt.cdlib.org/d/objectid/0/system%2Fmrt-
dc.xml>
  mrt:metadataSchema
    msc:DC ;
  mrt:mimeType
    <http://purl.org/NET/mediatypes/text/xml> . ] [
n2t:objectid
  mrt:hasMetadata
    <http://merritt.cdlib.org/d/objectid/0/producer%2F
mrt-dc.xml> .
<http://merritt.cdlib.org/d/objectid/0/producer%2Fmrt-
dc.xml>
  mrt:metadataSchema
    msc:DC ;
  mrt:mimeType
    <http://purl.org/NET/mediatypes/text/xml> . ] [
n2t:objectid
  mrt:hasMetadata
    dua-metadata-url .
<http://merritt.cdlib.org/d/objectid/0/dua-metadata-url
  mrt:metadataSchema
    msc:MRT-DUA ;
  mrt:mimeType
    <http://purl.org/NET/mediatypes/text/x-anvl> . ] [
```

```
n2t:objectid
  mrt:hasMetadata
    http://merritt.cdlib.org/d/objectid/0/producer %2F
mrt-datacite.xml> .
<http://merritt.cdlib.org/d/objectid/0/producer%2Fmrt-
datacite.xml>
  mrt:metadataSchema
    msc:DataCite ;
  mrt:mimeType
    <http://purl.org/NET/mediatypes/text/xml> . ] [
n2t:objectid
  mrt:hasMetadata
    <http://merritt.cdlib.org/d/objectid/0/producer%2
Fmrt-eml> .
<http://merritt.cdlib.org/d/objectid/0/producer%2Fmrt-
eml.xml>
  mrt:metadataSchema
    msc:EML ;
  mrt:mimeType
    <http://purl.org/NET/mediatypes/text/xml> . ]
```

```
./system/mrt-owner.txt              # Owner metadata
```

NOTE It is possible for a single submission to define ERC metadata both within the package itself (i.e. "`./producer/mrt-erc.txt`") and by method arguments (i.e., Creator, Title, Date, LocalIdentifier, used to define "`./system/mrt-erc.txt`").

10. *Member node*. [Optional, if interoperation with DataONE [DataONE] is desired]

(1) If the Merritt DateONE manifest file "`mrt-dataone-manifest.txt`" does exist in the staging area "`producer`" directory:

i. Define a "`mrt-hasMetadata`" relationship between the object and the producer DataONE manifest file, a "`mrt:metadataSchema`" relationship between the producer DataONE manifest file and "`DataONE-manifest`", and a "`mrt:mimeType`" relationship between the producer DataONE manifest file and "`text/anvl`", in the system object resource map, "`mrt-object-map.ttl`".

The DataONE manifest file conforms to the general format:

```
#%dataonem_0.1
#%profile | http://uc3.cdlib.org/registry/ingest/manifest/
                                              mrt-dataone-manifest
#%prefix  | dom: | http://uc3.cdlib.org/ontology/dataonem#
#%prefix  | mrt: | http://uc3.cdlib.org/ontology/mom#
#%fields  | dom:scienceMetadataFile | dom:scienceMetadataFormat |
                                dom:scienceDataFile | mrt:mimeType
metadata-file | metadata-format | data-file | mime/type
...
#%eof
```

The list of defined DataONE science metadata formats is available at <http://mule1.dataone.org/ArchitectureDocs-current/design/WhatIsData.html#metadata-types>.

(2) If the file "mrt-dataone-manifest.txt" does not exist in the staging area "producer" directory:

i. Create the file "mrt-dataone-manifest.txt" in the staging area "system" directory, conforming to the Merritt DataONE manifest format, with a separate line associating the system ERC file "mrt-erc.txt" with each file in the "producer" directory:

```
#%dataonem_0.1
#%profile | http://uc3.cdlib.org/registry/ingest/manifest/
                                                mrt-dataone-manifest
#%prefix  | dom: | http://uc3.cdlib.org/ontology/dataonem#
#%prefix  | mrt: | http://uc3.cdlib.org/ontology/mom#
#%fields  | dom:scienceMetadataFile | dom:scienceMetadataFormat |
                                  dom:scienceDataFile | mrt:mimeType
mrt-erc.txt | ERC | data-file | mime/type
...
#%eof
```

ii. Define an "ore:aggregates" relationship between the object and the system DataONE manifest file, a "mrt:hasMetadata" relationship between the object and the system DataONE manifest file, a "mrt:metadataSchema" relationship between the system DataONE manifest file and "DataONE-manifest", and a "mrt:mimeType" relationship between the system DataONE manifest file and "text/anvl", in the system object resource map, "mrt-object-map.ttl".

(3) Use the file "mrt-dataone-manifest.txt" found in the staging area "producer" directory, or the file newly created in the "system" directory in step 10(1)(i) to create a DataONE resource map [ORE] with the filename "mrt-dataone-map.xml" in the staging area "system" directory:

NOTE The abstract structure and the specific serialization format of the DataONE resource map are not yet well defined and stable, and are thus not defined here.

(4) Define an "ore:aggregates" relationship between the object and the DataONE resource map, a "mrt:hasMetadata" relationship between the object and the DataONE resource map, a "mrt:metadataSchema" relationship between the DataONE resource map and "DataONE-map", and a "mrt:mimeType" relationship between the DataONE resource map and "text/turtle", in the system object resource map, "mrt-object-map.ttl".

(5) Register the object with Metacat [Metacat], the central metadata store for DataONE member nodes.

i. Invoke the Metacat API to register the object.

ii. Add the registration status, "success" or "failure", as the value of the "metacatRegistration" element of the ingest metadata component, "mrt-ingest.txt" in the staging area "system" directory.

NOTE The failure of Metacat registration does *not* imply the failure and rollback of the Merritt ingest. The success or failure of the two processes The success or failure of Merritt ingest is completely independent of the success or

failure of Metacat registration.

```
[ ./producer/mrt-dataone-manifest.txt]# Optional DataONE manifest
[ ./producer/manifestid ]             # Submission manifest
[ ./producer/mrt-datacite.xml ]       # Optional DataCite metadata
[ ./producer/mrt-dc.xml ]             # Optional DC metadata
[ ./producer/mrt-eml.xml ]            # Optional EML metadata
[ ./producer/mrt-erc.txt ]            # Optional ERC metadata
[ ./producer/mrt-manifest.txt ]       # Optional container manifest
  ./producer/fileid
  ...
[ ./system/mrt-dataone-manifest.txt ] # Optional DataONE manifest
  ./system/mrt-dataone-map.ttl        # DataONE resource map
[ ./system/mrt-dc.xml ]               # Optional DC metadata
[ ./system/mrt-erc.txt ]              # System ERC metadata
  ./system/mrt-ingest.txt             # Ingest metadata
```

```
...
metacatRegistration: success | failure
```

```
  ./system/mrt-jhove2.xml             # JHOVE2 metadata
  ./system/mrt-membership.txt         # Membership metadata
  ./system/mrt-mom.txt                # MOM metadata
  ./system/mrt-object-map.ttl         # Object resource map
```

```
...
n2t:objectid
  ore: aggregates
   [ <http://merritt.cdlib.org/d/objectid/0/system%2Fmrt-
dataone-manifest.txt> , ]
    <http://merritt.cdlib.org/d/objectid/0/producer%2F
mrt-dataone-map.ttl> ;
  mrt:hasMetadata
    <http://merritt.cdlib.org/d/objectid/0/system%2Fmrt-
dataone-manifest.txt> ,
    <http://merritt.cdlib.org/d/objectid/0/producer%2F
mrt-dataone-map.ttl> .
<http://merritt.cdlib.org/d/objectid/0/system%2Fmrt-
dataone-manifest.txt>
  mrt:metadataSchema
    msc:DataONE-manifest ;
  mrt:mimeType
    <http://purl.org/NET/mediatypes/text/x-anvl> .
<http://merritt.cdlib.org/d/objectid/0/producer%2F mrt-
dataone-map.ttl>
  mrt:metadataSchema
    msc:DataONE-map ;
  mrt:mimeType
    <http://purl.org/NET/mediatypes/text/turtle> . ]
```

NOTE The "`ore:aggregates`" relationship for the DataONE manifest file only applies if the manifest is created in Step 10(2)(i); if the manifest was supplied by the producer, it is already a member of the object aggregation by virtue of Steps 4(1), 5(3), or 6(3).

```
  ./system/mrt-owner.txt              # Owner metadata
```

11. *Data use agreement*. [Optional, if submission profile "type" = *MRT-class* and "aggregate" = *MRT-collection*]

    (1) If a data use agreement (DUA) metadata component, "`mrt-dua.txt`" exists in the "`producer`" directory, ….

```
[ ./producer/manifestid ]             # Submission manifest
[ ./producer/mrt-datacite.xml ]       # Optional DataCite metadata
[ ./producer/mrt-dc.xml ]             # Optional DC metadata
[ ./producer/mrt-dua.xml ]            # Optional DUA metadata
[ ./producer/mrt-eml.xml ]            # Optional EML metadata
[ ./producer/mrt-erc.txt ]            # Optional ERC metadata
[ ./producer/mrt-manifest.txt ]       # Optional container manifest
  ./producer/fileid
  ...
[ ./system/mrt-dc.xml ]               # Optional DC metadata
[ ./system/mrt-erc.txt ]              # System ERC metadata
  ./system/mrt-ingest.txt
  ./system/mrt-jhove2.xml             # JHOVE2 metadata
  ./system/mrt-membership.txt         # Membership metadata
  ./system/mrt-mom.txt                # MOM metadata
  ./system/mrt-object-map.ttl         # Object resource map
  ./system/mrt-owner.txt              # Owner metadata
```

12. *Document*. [Mandatory]

    (1) Document the sequence of handlers, including those in subsequent steps, in the ingest metadata component "`mrt-ingest.txt`" in the staging area "`system`" directory.

```
[ ./producer/manifestid ]             # Submission manifest
[ ./producer/mrt-datacite.xml ]       # Optional DataCite metadata
[ ./producer/mrt-dc.xml ]             # Optional DC metadata
[ ./producer/mrt-dua.xml ]            # Optional DUA metadata
[ ./producer/mrt-eml.xml ]            # Optional EML metadata
[ ./producer/mrt-erc.txt ]            # Optional ERC metadata
[ ./producer/mrt-manifest.txt ]       # Optional container manifest
  ./producer/fileid
  ...
[ ./system/mrt-dc.xml ]               # Optional DC metadata
[ ./system/mrt-erc.txt ]              # System ERC metadata
  ./system/mrt-ingest.txt
```

```
...
Handlers: handler/version; ...
```

```
  ./system/mrt-jhove2.xml             # JHOVE2 metadata
  ./system/mrt-membership.txt         # Membership metadata
  ./system/mrt-mom.txt                # MOM metadata
  ./system/mrt-object-map.ttl         # Object resource map
```

```
./system/mrt-owner.txt                # Owner metadata
```

13. *Digest.* [Mandatory]

(1) Create a Checkm manifest "`mrt-manifest.txt`" in the "`system`" directory that references the URL, SHA-256 digest type and value, file size, and target filename for all files in the "`producer`" and "`system`" directories, except the manifest itself.

(2) Compute an SHA-256 message digest for the manifest.

```
[ ./producer/manifestid ]             # Submission manifest
[ ./producer/mrt-manifest.txt ]       # Optional container manifest
[ ./producer/mrt-datacite.xml ]       # Optional DataCite metadata
[ ./producer/mrt-dc.xml ]             # Optional DC metadata
[ ./producer/mrt-dua.xml ]            # Optional DUA metadata
[ ./producer/mrt-eml.xml ]            # Optional EML metadata
[ ./producer/mrt-erc.txt ]            # Optional ERC metadata
  ./producer/fileid                   # File components
  ...
[ ./system/mrt-dc.xml ]               # Optional DC metadata
[ ./system/mrt-erc.txt ]              # System ERC metadata
  ./system/mrt-ingest.txt             # Ingest metadata
  ./system/mrt-jhove2.xml             # JHOVE2 metadata
  ./system/mrt-manifest.txt           # AIP manifest
```

```
#%checkm_0.7
#%profile | http://uc3.cdlib.org/registry/store/mrt-add-m
                                                    anifest
#%prefix  | mrt: | http://uc3.cdlib.org/ontology/mom#
#%prefix  | nfo: | http://www.semanticdesktop.org/ontolog
                                          ies/2007/03/22/nfo#
#%fields  | nfo:fileurl   | nfo:hashAlgorithm |
            nfo:hashValue | nfo:fileLastModified |
            nfo:fileName
url | sha256 | digest | size | | filename
...
#%eof
```

```
./system/mrt-membership.txt           # Membership metadata
./system/mrt-mom.txt                  # MOM metadata
./system/mrt-object-map.ttl           # Object resource map
./system/mrt-owner.txt                # Owner metadata
```

13. *Transfer.* [Mandatory]

(1) Retrieve the Storage service and storage node identifiers from the profile.

(2) Invoke the Storage server *Add-version* method passing the storage node identifier, primary identifier, manifest, manifest size, manifest SHA-256 message digest algorithm and value, and if supplied, the local identifier list and profile identifier, and add the result (`success` or `failure`) to the ingest metadata component "`mrt-ingest.txt`" in the staging area "`system`" directory.

```
UA: POST /content/node/object HTTP/1.x
UA: Host: store.cdlib.org
UA: Accept: response/form
UA: Content-type: multipart/form-data; boundary=boundary
UA:
UA: --boundary
UA: Content-disposition: form-data; name="manifest"
UA: Content-type: text/checkm
UA:
UA: manifestid
UA: --boundary
UA: Content-disposition: form-data; name="size"
UA: Content-type: text/plain
UA:
UA: size
UA: --boundary
UA: Content-disposition: form-data; name="digest-type"
UA: Content-type: text/plain
UA:
UA: type
UA: --boundary
UA: Content-disposition: form-data; name="digest-value"
UA: Content-type: text/plain
UA:
UA: value
UA: --boundary
UA: [ Content-disposition: form-data; name="local-context"
UA: Content-type: text/plain
UA:
UA: profileid
UA: --boundary
UA: Content-disposition: form-data; name="local-identifier"
UA: Content-type: text/plain
UA:
UA: localid[; localid[; ...]]
UA: --boundary ]
UA: Content-disposition: form-data; name="response-form"
UA: Content-type: text/plain
UA:
UA: form
UA: --boundary

OS: HTTP/1.x 201 CREATED
OS: Content-type: response/form
OS: Location: http://store.cdlib.org/state/node/object/version
OS:
OS: state
```

```
[ ./producer/manifestid ]              # Submission manifest
[ ./producer/mrt-manifest.txt ]        # Optional container manifest
[ ./producer/mrt-datacite.xml ]        # Optional DataCite metadata
[ ./producer/mrt-dc.xml ]              # Optional DC metadata
[ ./producer/mrt-dua.xml ]             # Optional DUA metadata
[ ./producer/mrt-eml.xml ]             # Optional EML metadata
[ ./producer/mrt-erc.txt ]             # Optional ERC metadata
  ./producer/fileid                    # File components
  ...
[ ./system/mrt-dc.xml ]                # Optional DC metadata
[ ./system/mrt-erc.txt ]               # System ERC metadata
  ./system/mrt-ingest.txt              # Ingest metadata
```

```
...
storageResult: success | failure
```

```
  ./system/mrt-jhove2.xml              # JHOVE2 metadata
  ./system/mrt-manifest.txt            # AIP manifest
  ./system/mrt-membership.txt          # Membership metadata
  ./system/mrt-mom.txt                 # MOM metadata
  ./system/mrt-object-map.ttl          # Object resource map
  ./system/mrt-owner.txt               # Owner metadata
```

14. *Fixity*. [Conditional, if Transfer was successful]

    (1) Retrieve the Fixity service, owner, and collection identifiers from the profile.

    (2) For each object component file, invoke the Fixity service *Add-Item* method, passing the component's Storage service URL, size (in octets), SHA-256 message digest, and contexts for the object's owner, collections, and primary identifier, and add the result (number of successful adds, number of failed adds, URLs of the individual failed items) to the ingest metadata component "`mrt-ingest.txt`" in the staging area "`system`" directory.

```
UA: POST /add HTTP/1.x
UA: Host: fixity.cdlib.org
UA: Content-type: multipart/form-data; boundary=boundary
      UA:
      UA: --boundary
      UA: Content-disposition: form-data; name="url"
      UA:
      UA: url
      UA: --boundary
      UA: Content-disposition: form-data; name="source"
      UA:
      UA: merritt
      UA: --boundary
      UA: Content-disposition: form-data; name="size"
      UA:
      UA: size
      UA: --boundary
      UA: Content-disposition: form-data; name="digest-type"
      UA:
      UA: sha-256
      UA: --boundary
```

```
UA: Content-disposition: form-data; name="digest-value"
UA:
UA: value
UA: --boundary
UA: Content-disposition: form-data; name="context"
UA:
UA: ownerid
UA: collectionid
UA: ...
UA: objectid
UA: --boundary
UA: Content-disposition: form-data; name="response-form"
UA:
UA: xml
UA: --boundary

OS: HTTP/1.x 201 Created
OS: Content-type: response/form
OS: Location: http://fixity.cdlib.org/state/url
OS:
OS: state
```

(3) Add the counts of *Add-item* successes and failures to the the ingest metadata component "mrt-ingest.txt" in the staging area "system" directory.

```
[ ./producer/manifestid ]              # Submission manifest
[ ./producer/mrt-manifest.txt ]        # Optional container manifest
[ ./producer/mrt-datacite.xml ]        # Optional DataCite metadata
[ ./producer/mrt-dc.xml ]              # Optional DC metadata
[ ./producer/mrt-dua.xml ]             # Optional DUA metadata
[ ./producer/mrt-eml.xml ]             # Optional EML metadata
[ ./producer/mrt-erc.txt ]             # Optional ERC metadata
  ./producer/fileid                    # File components
  ...
[ ./system/mrt-dc.xml ]                # Optional DC metadata
[ ./system/mrt-erc.txt ]               # System ERC metadata
  ./system/mrt-ingest.txt              # Ingest metadata
```

```
...
[ fixityAddFailure: item
... ]
fixityAddSucesses: n
fixityAddFailures: m
```

```
./system/mrt-jhove2.xml                # JHOVE2 metadata
./system/mrt-manifest.txt              # AIP manifest
./system/mrt-membership.txt            # Membership metadata
./system/mrt-mom.txt                   # MOM metadata
./system/mrt-object-map.ttl            # Object resource map
./system/mrt-owner.txt                 # Owner metadata
```

15. *Notify*. [Mandatory]

(1) Retrieve the notification email address(es) from the submission request and the

address(es) and format from the submission profile.

(2) Email to each address notification of Ingest results, formatted as indicated by the notification format in the submission profile. Only a single notification email to each address is generated for each batch.

a. The notification message MUST minimally include the following information for the entire submission (batch or single object):

- Submitting user agent.
- Submission package filename.
- Submission date/timestamp.
- Completion date/timestamp.
- Ingest status.

The notification message MUST minimally include the following information for each object processed in the submission:

- Primary identifier.
- Local identifier, if defined.
- Version identifier.
- Creator, title, and date, if defined (either as request parameters or in an included ERC metadata component, "`mrt-erc.txt`").
- Checksum algorithm and value.
- Submission date/timestamp.
- Completion date/timestamp.
- Metacat registration status, if the *Member Node* handler was invoked in Step 10.
- Merritt Ingest status.

Additional information MAY also be included in the notification message.

To aid in the automatic processing of the emailed notification, the subject line of the email message MUST conform to the Merritt template:

```
Subject: service [instance]: status -- message: extra; ...
```

where *service* is "Ingest"; *instance* is "dev" or "stg" (or not provided, if production); *status* is "OK" or "fail"; *message* is "Submission processed"; and *extra* is the batch identifier.

```
Subject: Ingest: OK -- Submission processed: bid-batch-
uuid
```

The body of the email message MUST be in ANVL format:

```
Submission ID: bid-batch-uuid
Job(s):

  :Number of pending job(s): k
  :Number of completed job(s): m
  :Number of failed job(s): n

User agent: identity/name
```

```
Queue priority: nn
Submission date: yyyy-mm-ddThh:mm:ss-hh:mm
Completion date: yyyy-mm-ddThh:mm:ss-hh:mm
Status: COMPLETED
```

More detailed notification information is provided in an attached file using the notification format specified in the submission profile, which can be ANVL, CSV, JSON, RDF/Turtle, XHTML, or XML.

(3) If an error condition is raised, send email notification to the service administrative address.

16. *Callback*. [Optional, if callback URL specified in profile]

(1) Retrieve the callback URL from the submission profile.

(2) Format the notification message as specified in the profile.

(3) Send the notification to the callback URL in the body of an HTTP POST request:

```
POST /callback/path HTTP/1.x
Host: callback.host
Content-type: application/x-www-form-urlencoded

jobstate=notification
```

(4) Receive an HTTP response:

```
HTTP/1.x 200 OK
```

17. *Cleanup*. [Mandatory]

(1) Delete the staging area.

6.5 File System Instantiation

The Ingest service is instantiated in a file system as:

```
<ingest_home>/
              0=ingest_0.28
              ingest-info.txt
              log/
              profiles.txt
              profiles/
              queue/
              stores.txt
```

Within the file system hierarchy rooted at an Ingest home directory, all file and directory names starting with "`ingest`", "`ing`", "`merritt`", or "`mrt`", on a case-insensitive basis, are reserved.

6.5.1 Namaste Tag (0=ingest_*version*)

The home directory MUST contain a file named "`0=ingest_0.28`" that is the service's Namaste tag [Namaste]. The tag file MUST contain the Ingest service specification name and version:

```
Ingest/0.28
```

A Namaste tag fulfills the same function for a directory that a magic number does for a file.

6.5.2 Global Service Properties (ingest-info.txt)

The home directory MUST contain a file named "`ingest-info.txt`" that specifies the global properties of the service.

```
name: Ingest01
identifier: ingest.cdlib.org/ingest01
description: UC3 curation repository ingest service
serviceScheme: Ingest/0.28/0.6
baseURI: http://ingest.cdlib.org/
supportURI: mailto:merritt-support@cdlib.org
adminURI: mailto:merritt-admin@cdlib.org
```

Within an Ingest properties file all property names starting with "`ingest`", "`ing`", "`merritt`", or "`mrt`", on a case-insensitive basis, are reserved.

6.5.3 Profiles (*profile*.txt and profiles/)

The home directory MUST contain a file "`profiles.txt`".

```
profiles.txt
```

This file contains a list of identifiers for all registered profiles.

```
profileid
...
```

Each profile MUST be defined by file "*profiles*.txt" in a sub-directory named "`profiles`".

```
profiles/
        profileid.txt
        ...
```

Each profile is defined by the ANVL file "*profile*.txt"

```
  identifier: profileid
  description: description
  type: type
  role: role
[ aggregate: aggregate ]
  owner: ownerid
  collection: collectionid[; ...]
  queuePriority: priority
  storageService: service
  storageNode: node
  contentModel: model
  identifierScheme: scheme
  identifierNamespace: namespace
  identifierMinter: minter
  handler: handler[; ...]
  ...
  notification: email[; ...]
  notificationFormat: anvl | csv | json | rdf | xhtml | xml
[ callbackURL: url ]
```

Note The Ingest service should determine which profiles are registered by reading the file "`profiles.txt`", not by looking for files "*`profile`*`.txt`" in the "`profiles/`" sub-directory. This permits partially completed profile definitions to be present in the directory without making them actionable.

6.5.4 Queue (queue/)

The home directory MUST contain a sub-directory named "queue" that holds all batches and jobs in the queue.

```
queue/
      batch/
            job/
            ...
      ...
```

6.5.5 Storage Services (stores.txt)

The home directory MUST contain a file named "`stores.txt`" that defines the Storage and Access services known to the Ingest service in terms of their base URIs.

```
store.1: http://store1.cdlib.org/
access.1: http://access1.cdlib.org/
...
store.n: http://storen.cdlib.org/
access.n: http://accessn.cdlib.org/
```

The association between a Storage service and its Access service is made via the numeric suffix on the "`store.n`" and "`access.n`" property names.

NOTE There is always a one-to-one relationship between given Storage and Access service.

References

[Abbott] Daisy Abbott, *What is Digital Curation?* April 3, 2008 <http://www.dcc.ac.uk/resource/briefing-papers/what-is-digital-curation/>.

[ANVL] J. Kunze, B. Kahle, J. Masanes, and G. Mohr, *A Name-Value Language (ANVL)*, February 14, 2005 <http://www.cdlib.org/inside/diglib/ark/anvlspec.pdf>.

[Checkm] J. Kunze, *Checkm: A Checksum-based Manifest Format*, 2009.

[DataONE] DataONE, "OAI-ORE." Welcome to Data Observation Network for Earth (DataONE) <http://www.dataone.org/>.

[DC] Dublin Core Metadata Initiative, *Dublin Core Metadata Element Set, Version 1.1*, November 11, 2010 <http://dublincore.org/documents/dces/>.

[Denning] Peter J. Denning, Chris Gunderson, and Rich Hayes-Roth, "Evolutionary system development," *Communications of the ACM* 51:17 (December 2008): 29-31

[Dflat] UC3, *Dflat: A Simple File System Convention for Digital Object Storage*, 2010.

[DOI] International DOI Foundation, *The DOI Handbook*, February 1, 2010 <doi:10.1000/186>.

[ERC] J. Kunze and A. Turner, *Kernel Metadata and Electronic Resource Citations (ERCs)*, April 28, 2009 <http://dublincore.org/kernelwiki/FrontPage?action=AttachFile&do=get&target=ercspec1_3.html>.

[EZID] UC3, *The EZID API*, Version 2, September 21, 2010 <http://n2t.net/ezid/docs/apidoc.html>.

[Fielding] Roy Fielding and Richard Taylor, "Principled design of the modern web architecture," *ACM Transactions on Internet Technology* 2:2 (May 2002): 115-150 <doi:10.1145/514183.514185>.

[Fisher] David A. Fisher, *An Emergent Perspective on Interoperation in Systems of Systems*, CMU/SEI-2006-TR-003, ESC-TR-2006-003, March 2006 <http://www.sei.cmu.edu/pub/documents/06.reports/pdf/06tr003.pdf>.

[HTML] Dave Ragget, Arnaud Le Hors, and Ian Jacobs, *HTML 4.01 Specification*, W3C Recommendation, December 24, 1999 <http://www.w3.org/TR/html401/>.

[Kernel] J. Kunze and A. Turner, *Kernel Metadata/ERC Application Profile,* Draft V1.4A, February 20, 2008 <http://dublincore.org/kernelwiki/FrontPage?action=AttachFile&do=get&target=KernelMetadataERCApplicationProfiles1_4a.htm>.

[Merritt] UC3, *Merritt: An Emergent Approach to Digital Curation Infrastructure*, 2010.

[Metacat] Knowledge Network for Biocomplexity, *Metacat: Metadata and Data Management System* <http://knb.ecoinformatics.org/software/metacat/>.

[MOM] UC3, *Merritt Object Modeling*, 2011.

[Multipart] L. Masinter, *Returning Values from Forms: multipart/form-data*, RFC 2388, August 1989 <http://www.ietf.org/rfc/rfc2388.txt>.

[Namaste] UC3, *Name-as-Text (Namaste)*, 2009.

[Object] UC3, *Merritt Object Modeling*. 2010.

[ORE] DataONE, "OAI-ORE." *Data Packaging* <http://mule1.dataone.org/ArchitectureDocs-current/design/DataPackage.html#id12>.

[RFC2119] S. Bradner, *Key Words for Use in RFCs to Indicate Requirement Levels*, BCP 14, RFC 2119, March 1997 <http://www.ietf.org/rfc/rfc2119.txt>.

[RFC4180] Y. Shafranovich, *Common Format and MIME Type for Comma-Separated Values (CSV) Files*, RFC 4180, October 2005 <http://www.ietf.org/rfc/rfc4180.txt>.

[Storage] UC3, *Merritt Storage Service*, 2010.

[Turtle] David Beckett and Tim Berners-Lee, *Turtle – Terse RDF Trtple Language*, January 14, 2008, <http://www.w3.org/TeamSubmission/turtle/>.

[ZooKeeper] Apache*, Welcome to Apache ZooKeeper!* <http://hadoop.apache.org/zookeeper>.