UC3 Merritt

University of California Curation Center

Cutie – A Simple Directory-based Queuing Service

Rev. 0.3 – 2010-04-20

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].

Many Merritt activities are most efficiently and effectively performed on an asynchronous basis, which means it is important to have a mechanism to accumulate and manage pending operations. Cutie (“kew dee”: *Q*ueue *D*irectory) is a flexible directory-based queuing service that can be used to support these activities.

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

Cutie MUST meet the following functional and non-functional requirements:

- The jobs managed in Cutie are opaque to the service; the syntax and semantics of jobs are defined, enforced, and processed external to Cutie, if at all.
- Queue priority is first in, first out (FIFO).
- Jobs pending in the queue can be deleted.

- External processes can examine queued jobs non-consumptively, that is, without affecting their position or status in the queue.

3 Cutie

Cutie is based on the following conceptual entities, each defined in terms of its specific state properties.

- Service.
- Queue.
- Job.

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

3.1 Service

The initial conceptual entity is the Cutie service itself, which provides a mechanism to manage items in a FIFO queue. The global state properties of the service MUST include:

• Service name.	[cut:name]
• Service identifier, assigned to be globally unique among all UC3-controlled instantiations.	[cut:identifier]
• Service implementation version.	[cut:serviceVersion]
• Actionable references to queue states.	[cut:queueStates]
o Actionable reference to queue state.	[cut:queueState]
• Creation date/timestamp.	[cut:created]
• Modification date/timestamp.	[cut:lastModified]
• Service specification and version.	[cut:serviceScheme]
• Base URI for the service method invocations.	[cut:baseURI]
• Support URI for the service.	[cut:supportURI]

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

The Cutie service can be configured to support an arbitrary number of *queues*.

3.2 Queue

A *queue* manages a first-in-first-out-ordered set of jobs. The queue state properties MUST minimally include:

• Queue name.	[que:name]
• Queue identifier, assigned to be locally unique within its service.	[que:identifier]

• Optional queue description.	[que:description]
• Actionable reference to the Cutie service state.	[que:cutieState]
• Number of pending jobs in the queue.	[que:numPendingJobs]
• Number of completed items in the queue.	[que:numCompletedJobs]
• Number of deleted items in the queue.	[que:numDeletedJobs]
• Actionable references to job states.	[que:jobStates]
o Actionable reference to job state.	[que:jobState]
• Processed job culling size threshold.	[que:cullingSizeThreshold]
• Processed job culling time threshold (in hours).	[que:cullingAgeThreshold]
• Pre-get verification status: *true* or *false*.	[que:verifyOnRead]
• Post-submission verification status: *true* or *false*.	[que:verifyOnWrite]
• Creation date/timestamp.	[que:created]
• Last submission date/timestamp.	[que:lastSubmission]

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

Jobs in the *Consumed* and *Deleted* state will be culled, that is, deleted from the file system instantiation of the service, when the number of consumed or deleted jobs exceeds the culling size threshold and the date of completion exceeds the culling age threshold (relative to the current time). Jobs are culled on a first-in, first-out basis. The use of appropriate culling thresholds can facilitate triage of unsuccessful job processing.

A queue can accommodate an arbitrary number of *jobs*.

3.3 Job

A *job* is an opaque object managed in a queue. The syntax and semantics of jobs are defined, enforced, and processed external to Cutie, if at all; Cutie's obligation is only to manage and retrieve jobs for processing one at a time on a first-in, first-out basis. A job encapsulates all information needed by external processes in a payload file and optional textual sidecar data.

The job state properties MUST minimally include:

• Job identifier, assigned to be locally unique within its queue.	[job:identifier]
• Actionable reference to parent queue state.	[job:queueState]
• Submitting user agent.	[job:submitter]
• Payload size (in octets).	[job:size]
• Sidecar data (optional).	[job:sidecar]
• Submission date/timestamp.	[job:submitted]
• Consumption date/timestamp.	[job:consumed]
• Deletion date/timestamp.	[job:deleted]
• Job status: `pending`, `consumed`, `deleted`.	[job:status]

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

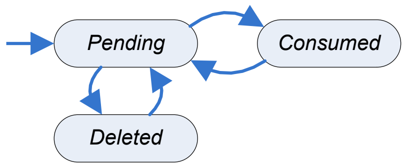
The syntax and semantics of the optional textual sidecar data are opaque to Cutie. Nevertheless, one possible way of representing textual information is as an arbitrary number of semicolon-separated name/value pairs:

```
name=value; name=value; ...
```

Semicolons found in names or values MUST be escaped with a backslash ("\").

Queued jobs can transition between three states:

- *Pending*. A new job that has been submitted into a queue.
- *Consumed*. A job removed from the queue for processing by an external process.
- *Deleted*. A pending job removed from the queue before being processed.



Consumed and deleted jobs can be moved back into a pending state for subsequent processing.

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`
JSON	`.json`	`application/json`
RDF/Turtle	`.ttl`	`text/turtle`
XHTML	`.html`	`application/xhtml+xml`
XML	`.xml`	`application/xml`

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.

The default format for state information in command line interfaces is ANVL; the default for web interfaces is XHTML.

RESTful options	*Command line options*		*Function*
`D[=`*`level`*`]`	`-D [`*`level`*`]`	`--debug [`*`level`*`]`	Debug level
`h`	`-h [`*`topic`*`]`	`--help [`*`topic`*`]`	Help
	`-o` *`file`*	`--output` *`file`*	Output to file (rather than standard output)
`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

5 Service Methods

Cutie 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, and a vertical bar "|" separates alternatives.

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[?t=form] HTTP/1.x
UA: Host: cutie.cdlib.org
UA: Accept: response/form
UA:

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

- Command line API

```
% cutie -h [-t form] [-o file]
% cutie 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[?t=form] HTTP/1.x
UA: Host: cutie.cdlib.org
UA: Accept: response/form
UA:

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

- Command line API

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

5.3 Get-Queue-State

METHOD Get-queue-state			[*idempotent*, *safe*]
Queue	Identifier	Mandatory	Queue 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	Queue state.
SIDE EFFECTS	—		
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: GET /state/queue[?t=form] HTTP/1.x
UA: Host: cutie.cdlib.org
UA: Accept: response/form
UA:

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

- Command line API

```
% cutie getQueueState queue [-t form] [-o file]
```

5.4 Get-Job-State

METHOD Get-job-state			[*idempotent*, *safe*]
Queue	Identifier	Mandatory	Queue 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	Queue not found.	
	404	Job not found.	
	415	Unsupported response form.	
	503	Service unavailable.	
	500	Service error.	

- RESTful API

```
UA: GET /state/queue/job[?t=form] 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

```
% cutie getJobState queue job [-t form] [-o file]
```

5.5 Submit Job

METHOD Submit-job			[*non-idempotent*, *unsafe*]
Queue	Identifier	Mandatory	Queue identifier.
Filename	List of String	Mandatory	Job filename(s).
File	List of Octet-stream	Mandatory	If type is *job*, job payload(s); otherwise, manifest of a batch of jobs.
	Checkm		
Type	Enum	Mandatory	File type: *job*, or *batch-manifest*.
Sidecar	Octet-stream	Optional *	Sidecar data.
DigestType	Enum	Optional *	Payload message digest type. The support types SHOULD include Adler-32, CDC-32, MD2, MD5, SHA-1, SHA-256, SHA-384, and SHA-512.
DigestValue	String		Hexadecimal representation of the payload message digest value.
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 (for single job submission) or queue state (for batch submission)..
SIDE EFFECTS	All job payloads defined by the invocation, either directly or by batch manifest, are added to the designated queue with *Pending* status using newly minted job identifiers that are locally unique to the queue. If optional sidecar information is present, it is added to the job file.		
ERRORS	400 Badly-formed request.		
	401 Unauthorized user agent.		
	404 Queue not found.		
	415 Unsupported request form.		
	415 Unsupported response form.		
	503 Service unavailable.		
	500 Service error.		

* Only meaningful for single job submission.

NOTE Optional sidecar information can only be provided using single job submission. Optional message digests can be provided using single job or batch manifest submission.

- RESTful API

```
UA: POST /content/queue[?t=form] HTTP/1.x
UA: Host: cutie.cdlib.org
UA: Accept: response/form
UA: Content-type: multipart/form-data; boundary=boundary
UA:
UA: --boundary
```

```
UA: ( Content-disposition: form-data; name="file"; filename="filename"
UA: Content-type: application/octet-stream | text/checkm
UA:
UA: file | manifest ) |
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
UA:
UA: file
UA: --boundary2
UA: ...
UA: --boundary2)
UA: --boundary
UA: [ Content-disposition: form-data; name="type"
UA: Content-type: text/plain
UA:
UA: type
UA: --boundary ]
UA: [ Content-disposition: form-data; name="sidecar"
UA: Content-type: application/octet-stream
UA:
UA: sidecar
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 ]

OS: HTTP/1.x 201 CREATED
OS: Content-type: response/form
OS: Location: http://cutie.cdlib.org/state/queue[/job]
OS:
OS: state
```

- Command line API

```
% cutie submitJob queue filename ... -T mode [-t form][ -o file]
```

A batch manifest is a Checkm manifest [Checkm] containing URL references to one or more job

payload files. Message digests SHOULD be provided for each payload.

```
        url [ digestType digestValue ] - - filename
        ...
```

5.6 Get Next Job

METHOD Get-next-job			*[non-idempotent, unsafe]*
Queue	Identifier	Mandatory	Queue identifier
RETURN	Octet-stream	Mandatory	Job payload and sidecar information, if defined, for the job at the head of the queue.
SIDE EFFECTS	The job status is changed from *Pending* to *Consumed*.		
ERRORS	400	Badly-formed request.	
	401	Unauthorized user agent.	
	404	Queue not found.	
	503	Service not available.	
	500	Service error.	

- RESTful API

```
UA: POST /consume/queue HTTP/1.x
UA: Host: cutie.cdlib.org
UA:

OS: HTTP/1.x 200 OK | 204 No Content
OS: [ Content-type: multipart/form-data; boundary=boundary
OS:
OS: --boundary
OS: Content-disposition: form-data; name="payload"
OS: Content-type: application/octet-stream
OS:
OS: payload
OS: --boundary
OS: [ Content-disposition: form-data; name="sidecar"
OS: Content-type: application/octet-stream
OS:
OS: sidecar
OS: --boundary ]]
```

NOTE In order for the transactional semantics of the method (non-idempotent, unsafe) to conform to RESTful principles, the HTTP POST method is used instead of GET.

- Command line API

```
% cutie getNextJob queue [-o file]
```

NOTE If no jobs are pending on the queue at the time a job is requested, the method MUST return a 204 "No Content". Invoking user agents MUST be prepared to accept this as an appropriate, not an exceptional, response.

NOTE This method MUST be implemented carefully in order to be thread-safe in concurrent operation.

5.7 Peek at Job

METHOD Peek-at-job			*[idempotent, safe]*
Queue	Identifier	Mandatory	Queue identifier
Job	Identifier	Optional	Job identifier. If not specified, the job at the head of the queue is assumed.
RETURN	Octet-stream	Mandatory	Job payload and sidecar information, if defined.
SIDE EFFECTS	—		
ERRORS	400	Badly-formed request.	
	401	Unauthorized user agent.	
	404	Queue not found.	
	404	Job not found.	
	503	Service not available.	
	500	Service error.	

- RESTful API

```
UA: GET /content/queue[/job] HTTP/1.x
UA: Host: cutie.cdlib.org
UA:

OS: HTTP/1.x 200 OK | 204 No Content
OS: [ Content-type: multipart/form-data; boundary=boundary
OS: --boundary
OS: Content-disposition: form-data; name="payload"
OS: Content-type: application/octet-stream
OS:
OS: payload
OS: --boundary
OS: [ Content-disposition: form-data; name="sidecare"
OS: Content-type: application/octet-stream
OS:
OS: sidecar
OS: --boundary ]]
```

- Command line API

```
% cutie getNextJob queue [-o file]
```

NOTE *This method is non-consumptive*. The job being peeked at remains in the *Pending* status with no change to its state.

NOTE If no jobs are pending on the queue at the time a job is requested, the method MUST return a 204 “No Content”. Invoking user agents MUST be prepared to accept this as an appropriate, not an exceptional, response.

5.8 Delete Job

METHOD Delete-job			[*idempotent*, *unsafe*]
Queue	Identifier	Mandatory	Queue 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 of the deleted job.
SIDE EFFECTS	The job is deleted from the queue and its status is set to *Deleted*.		
ERRORS	400	Badly-formed request.	
	401	Unauthorized user agent.	
	404	Queue not found.	
	404	Item not found.	
	415	Unsupported response form.	
	503	Service unavailable.	
	500	Service error.	

- RESTful API

```
UA: DELETE /content/queue/job[?t=form] HTTP/1.x
UA: Host: cutie.cdlib.org
UA: Accept: response/form
UA:

OS: HTTP/1.x 202 Accepted
OS: Content-type: response/form
OS:
OS: state
```

- Command line API

```
% cutie deleteItem queue item [-t form] [-o file]
```

6 Implementation

Cutie is instantiated in a file system as:

```
<cutie_home>/
             0=cutie_0.3
```

```
                    cutie-info.txt
                    log/
                    queues/
```

Within the file system hierarchy rooted at a Cutie home directory, all file names starting with "`cutie`", "`cut`", "`merrit`", or "`mrt`", on a case-insensitive basis, are reserved.

6.1 Namaste Tag (0=cutie_*version*)

The home directory MUST contain a file named "`0=cutie_0.2`" that is the service's Namaste tag [Namaste. The tag file MUST contain the Cutie specification name and version:

```
Cutie/0.2
```

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

6.2 Global Service Properties (cutie-info.txt)

The home directory MUST contain a file named "`cutie-info.txt`" that specifies the global properties of the service, for example:

```
name: Cutie01
identifier: uc3:cutie01
description: UC3 queuing service
serviceScheme: Cutie/0.3/0.1
baseURI: http://cutie.cdlib.org/
supportURI: mailto:merritt-support@cdlib.org
```

Within a properties file all property names starting with "`cutie`", "`cut`", "`merrit`", or "`mrt`", on a case-insensitive basis, are reserved.

6.3 Logs (log/)

The home directory MUST contain a sub-directory named "log" that holds the Cutie logs:

```
log/
     ...
```

6.4 Queues (queue/ and queue-info.txt)

The home directory MUST contain a sub-directory named "queues" that is the parent for all queues:

```
queue/
      queueid/
              consumed/
```

```
                [ jobid
                  ...     ]
        deleted/
                [ jobid
                  ...     ]
        payload/
                [ jobid
                  ...     ]
        pending/
                [ jobid
                  ...     ]
        queue-info.txt
...
```

There MUST be a queue directory to correspond with each queue identifier defined in the "`queues.txt`" file. A queue directory MUST contain a file named "`queue-info.txt`" that defines the queue state properties:

```
name: Queue01
identifier: uc3:cutie/01/queue/01
description: UC3 primary ingest queue
cullingSizeThreshold: 100
cullingAgeThreshold: 720
```

Within a properties file all property names starting with "`cutie`", "`cut`", "`merrit`", or "`mrt`", on a case-insensitive basis, are reserved.

6.5 Jobs

A job is represented by two files:

- *Job* file. Job state information defined by Cutie and optional opaque sidecar information defined by external processes.
- *Payload* file. All job information needed for processing by an external process.

The status of job determines the placement of its job file in either the "`pending`", "`consumed`", or "`deleted`" directory. When a new job is accepted into the queue (via the *Submit-job* method):

- A job identifier unique to the queue is minted ("*`jobid`*").
- The job payload is stored in a payload file "*`jobid`*" in the "`payload`" directory.
- A job file "*`jobid`*" is created in the "`pending`" directory.

Job identifiers are minted in a strict monotonically-increasing lexicographic collating order. Thus, the job at the head of the queue is always the first in a lexicographic sort of job files.

A job file represents job state in ANVL form:

```
      identifier: jobid
      submitter: useragent
      size: size
    [ sidecar: sidecar ]
      submitted: date
      status: pending
```

When a job is processed (via the *Get-next-job* method):

- The job file is moved from the "`pending`" to the "`consumed`" `directory`.
- The payload file remains in place in the "`payload`" directory.

```
      ...
      submitted: date
      consumed: date
      status: consumed
```

If a pending job is deleted (via the *Delete-job* method):

- The job file is moved from the "`pending`" to the "`deleted`" directory.
- The payload file remains in place in the "`payload`" directory.

```
      ...
      submitted: date
      consumed: date
      deleted: date
      status: deleted
```

If the number of completed jobs exceeds the culling size threshold, the payload and job files for all completed jobs exceeding the age threshold are deleted from the "`payload`" and "`consumed`" directories. If the age threshold is 0 then the necessary number of jobs are deleted, on a oldest-first basis, to conform to the size threshold. This culling mechanism implies some sort of daemon process that periodically examines queue directories and evaluates whether jobs should be culled.

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)*, Februrary 14, 2005 <http://www.cdlib.org/inside/diglib/ark/anvlspec.pdf>.

[Checkm] UC3, *Checkm: A Checksum-based Manifest Format*, 2010.

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

[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>.

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

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

[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>.

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