SWORDv3 Profile Working Document

V2 - 10th January 2018

This version is now frozen, there's a new version in progress at: https://docs.google.com/document/d/1kZu-LMkZfdgx5-OBJoBaO02sChmBUdru55F0Uej9AGk/edit

This document outlines a proposed specification for SWORDv3. It outlines the protocol operations to be supported in the new version, with all their behaviours, HTTP headers, payload and response bodies, and response codes. It also defines the documents that will form the basis of the REST API that is this new version of SWORD, and outlines some responsibilities and other processes involved.

The proposal is that SWORDv3 move away from the Atom specification, to a pure JSON-LD-based API, using good REST principles. Some of the original flavour of the SWORDv2 profile may be retained, but the overall expression of the documents will be significantly different.

Please see the usage patterns and requirements analysis documentation upon which this new version has been based: https://docs.google.com/spreadsheets/d/1qUiO9Rl_rNoiy7SmTcXnb6T330ENmivP-MTTJIdjqow/edit

There is no pretense that this document is a full specification at this stage (though in this second iteration it is closer) - it exists for review and comment on the essential components of the protocol. A fuller specification will be produced in time, once this document represents the path forward desired by the community.

Please add your comments using the "Insert > Comment" feature of Google Docs, attached to the relevant section of this document.

Definitions

URLs

- Service-Document - the location of the document which describes the server's capabilities for the user
- Deposit-Endpoint - the location where new content can be created
- Object - an Object that exists on the server, probably as a result of a deposit operation
- Metadata - the metadata associated with the Object

- Content - All of the content, in aggregation, associated by the Object - does not include the metadata
- File - a single binary file within the Content of the Object

Document Types

- Service Document - Describes the capabilities of the server with respect to the user
- Metadata - A format for depositing and retrieving object metadata
- Status - A document describing the current status of the object and its content
- Binary File - An opaque binary file
- Packaged Content - A set of files and metadata packaged according to some structure.
- Error - Describes an error that occurred while processing a request.

Protocol Operations

Discover Server Capabilities and List Deposit Endpoints

Request from the server a list of the collections that the client can deposit to. A Deposit Endpoint allows the server to support multiple different deposit conditions - each endpoint may have its own set of rules/workflows behind it; for example, endpoints may be subject-specific, organisational-specific, or process-specific. It is up to the client to determine which is the suitable endpoint for its deposit, based on the information provided by the server. The list of collections may vary depending on the authentication credentials supplied by the client.

Request
GET Service-Document
Headers:

- Authorization (MAY)
- On-Behalf-Of (MAY)

Body: None

Responses

- 200 - The server has responded with a list of deposit endpoints
    - Headers: None
    - Body: Service Document
- 401 - You have not provided authentication information, please do so
- 403 - You are not authorised to access this resource
- 404 - There is no service document available

Deposit new Object with Only Metadata

Create a new object on the server, sending only metadata content (i.e. no binary/file content).

Request
POST Deposit-Endpoint
Headers:

- Content-Type (MUST)
- Content-Length (SHOULD)
- Digest (SHOULD)
- Authorization (MAY)
- On-Behalf-Of (MAY)
- In-Progress (default: false) (MAY)
- Slug (MAY)
- Metadata-Format (default:sword) (SHOULD)
- Content-Disposition (MAY)

Body: Metadata (with optional by-reference deposit files listed)

Responses

- 201 - The server has received your request, and has created the associated resource
    - Headers:
        - Location (MUST)
        - ETag (MAY)
    - Body: Status
- 202 - The server has received your request, and has accepted the item, and queued it for formal creation at some point in the future
    - Headers:
        - Location (MUST)
    - Body: Status
- 400 - The server could not understand your request. Either your headers or content body are wrong or malformed.
- 401 - You have not provided authentication information, please do so
- 403 - You are not authorised to carry out this request
- 404 - The Deposit-Endpoint does not exist
- 412 - There is a problem implementing the request as-is. For example, your checksums may not match, or you may have requested mediated deposit when the server does not support that.
- 413 - Your request body exceeds the size allowed by the server
- 415 - The metadata format is not the same as that identified in Metadata-Format and/or it is not supported by the server

Deposit new Object with File or Package

Create a new object on the server, sending a single binary file which may itself be packaged content.

Request
POST Deposit-Endpoint
Headers:

- Content-Type (MUST)
- Content-Length (SHOULD)
- Digest (SHOULD)
- Authorization (MAY)
- On-Behalf-Of (MAY)
- In-Progress (default: false) (MAY)
- Slug (MAY)
- Content-Disposition (MUST)
- Packaging (default:Binary) (SHOULD)

Body:

- Binary File which may be Packaged Content
- An empty body, if this is a Segment Upload Initialisation request

Responses

- 201 - The server has received your request, and has created the associated resource
    - Headers:
        - Location (MUST)
        - ETag (MAY)
    - Body: Status
- 202 - The server has received your request, and has accepted the item, and queued it for formal creation at some point in the future
    - Headers:
        - Location (MUST)
    - Body: Status
- 400 - The server could not understand your request. Either your headers or content body are wrong or malformed.
- 401 - You have not provided authentication information, please do so
- 403 - You are not authorised to carry out this request
- 404 - The Deposit-Endpoint does not exist
- 412 - There is a problem implementing the request as-is. For example, your checksums may not match, or you may have requested mediated deposit when the server does not support that.
- 413 - Your request body exceeds the size allowed by the server
- 415 - The packaging format is not the same as that identified in Packaging and/or it is not supported by the server

Retrieve Object Information/Status

For an object where you have an Object URL, you may request information about the current state of that resource.

Request
GET Object
Headers:

- Authorization (MAY)
- On-Behalf-Of (MAY)

Body: None

Responses

- 200 - The server has responded with information about the Object
    - Headers:
        - ETag (MAY)
    - Body: Status
- 301, 307 - The Object URL has changed, re-send this request and all future requests to the new URL
    - Headers:
        - Location (MUST)
- 308 - The Object URL has temporarily changed, re-send this request to the new URL
    - Headers
        - Location (MUST)
- 400 - There was a problem with your request parameters
- 401 - You have not provided authentication information, please do so
- 403 - You are not authorised to carry out this request
- 404 - The requested Object does not exist
- 412 - You have requested the resource On-Behalf-Of a user, when the server does not support that.

Retrieve Object Metadata

Retrieve the descriptive metadata document associated with the Object.

Request
GET Metadata
Headers:

- Authorization (MAY)
- On-Behalf-Of (MAY)
- Accept-Metadata-Format (default:sword) (MAY)

Body: None

Responses

- 200 - The server has responded with the Metadata document
    - Headers:
        - ETag (MAY)
        - Metadata-Format (SHOULD)
    - Body: Metadata
- 301, 307 - The Object URL has changed, re-send this request and all future requests to the new URL
    - Headers:
        - Location (MUST)
- 308 - The Object URL has temporarily changed, re-send this request to the new URL
    - Headers
        - Location (MUST)
- 400 - There was a problem with your request parameters
- 401 - You have not provided authentication information, please do so
- 403 - You are not authorised to carry out this request
- 404 - The requested Object does not exist
- 405 - The requested Object does not support metadata retrieval in this context
- 412 - You have requested the resource On-Behalf-Of a user, when the server does not support that.

Retrieve Content as a Package

Retrieve the full object packaged according to some specification.

Request
GET Content
Headers:
- Authorization (MAY)
- On-Behalf-Of (MAY)
- Accept-Packaging (default:sword) (MAY)

Body: None

Responses
- 200 - The server has responded with the packaged content
    - Headers:
        - ETag (MAY)
        - Packaging (SHOULD)
    - Body: Packaged content
- 301, 307 - The Object URL has changed, re-send this request and all future requests to the new URL
    - Headers:
        - Location (MUST)
- 308 - The Object URL has temporarily changed, re-send this request to the new URL
    - Headers
        - Location (MUST)
- 400 - There was a problem with your request parameters

- 401 - You have not provided authentication information, please do so
- 403 - You are not authorised to carry out this request
- 404 - The requested Object does not exist
- 405 - The requested Object does not support package retrieval in this context
- 412 - You have requested the resource On-Behalf-Of a user, when the server does not support that.

Retrieve individual File from the Object

Retrieve a single file from inside the deposited object; files available for retrieval are listed in the Status document.

Request
GET File
Headers:
- Authorization (MAY)
- On-Behalf-Of (MAY)

Body: None

Responses
- 200 - The server has responded with the file content
    - Headers:
        - ETag (MAY)
    - Body: File content
- 301, 307 - The File URL has changed, re-send this request and all future requests to the new URL
    - Headers:
        - Location (MUST)
- 308 - The File URL has temporarily changed, re-send this request to the new URL
    - Headers
        - Location (MUST)
- 400 - There was a problem with your request parameters
- 401 - You have not provided authentication information, please do so
- 403 - You are not authorised to carry out this request
- 404 - The requested File does not exist
- 405 - The requested Object does not support file retrieval in this context
- 412 - You have requested the resource On-Behalf-Of a user, when the server does not support that.

Add/Update Object Metadata

Add new metadata or selectively update existing metadata on an item. Metadata provided in this way should be considered to overlay existing metadata, such that any new metadata fields are added to the item, and any existing metadata fields are overwritten, and any other metadata fields held by the server remain untouched.

Request
POST Metadata
Headers:

- Content-Type (MUST)
- Content-Length (SHOULD)
- Digest (SHOULD)
- Authorization (MAY)
- On-Behalf-Of (MAY)
- Metadata-Format (default:sword) (SHOULD)
- If-Match (MAY)

Body: Metadata (no by-reference file links allowed)

Responses

- 204 - The server has received your request, and has updated the associated resource
    - Headers:
        - ETag (MAY)
        - Location (MUST)
- 301, 307 - The Metadata URL has changed, re-send this request and all future requests to the new URL
    - Headers:
        - Location (MUST)
- 308 - The Metadata URL has temporarily changed, re-send this request to the new URL
    - Headers
        - Location (MUST)
- 400 - The server could not understand your request. Either your headers or content body are wrong or malformed.
- 401 - You have not provided authentication information, please do so
- 403 - You are not authorised to carry out this request
- 404 - The Metadata URL does not exist
- 405 - The requested Object does not support metadata update in this context
- 412 - There is a problem implementing the request as-is. For example, your checksums may not match, or you may have requested mediated deposit when the server does not support that, or your If-Match ETag may not match.
- 413 - Your request body exceeds the size allowed by the server
- 415 - The metadata format is not the same as that identified in Metadata-Format and/or it is not supported by the server

Add Packaged Content or other File to Object

Add new content to the Object, on top of existing content which it already contains. Content may be provided as a complex package or as a simple binary file. In the case of packaged content, the package may be unpacked by the server and new file resources added.

Metadata may also be updated, if the package contains actionable metadata. In the case of a binary file, this file will be added as-is to the object's content.

Request
POST Object
Headers:

- Content-Type (MUST)
- Content-Length (SHOULD)
- Digest (SHOULD)
- Authorization (MAY)
- On-Behalf-Of (MAY)
- Content-Disposition (MUST)
- Packaging (default:Binary) (SHOULD)

Body:

- Packaged content or File content
- An empty body if this is a Segment Upload Initialisation request

Responses

- 200 - The server has received your request, and has updated the associated resource
    - Headers:
        - ETag (MAY)
        - Location (MUST)
    - Body: Status
- 202 - The server has received your request, and has accepted the update, and queued it for formal update at some point in the future
    - Headers:
        - Location (MUST)
    - Body: Status
- 301, 307 - The Object URL has changed, re-send this request and all future requests to the new URL
    - Headers:
        - Location (MUST)
- 308 - The Object URL has temporarily changed, re-send this request to the new URL
    - Headers
        - Location (MUST)
- 400 - The server could not understand your request. Either your headers or content body are wrong or malformed.
- 401 - You have not provided authentication information, please do so
- 403 - You are not authorised to carry out this request
- 404 - The Metadata URL does not exist
- 405 - The requested Object does not support content update in this context
- 412 - There is a problem implementing the request as-is. For example, your checksums may not match, or you may have requested mediated deposit when the server does not support that.
- 413 - Your request body exceeds the size allowed by the server

- 415 - The metadata format is not the same as that identified in Metadata-Format and/or it is not supported by the server

Replace Object Metadata

Replace in its entirety the metadata associated with an Object.

Request
PUT Metadata
Headers:
- Content-Type (MUST)
- Content-Length (SHOULD)
- Digest (SHOULD)
- Authorization (MAY)
- On-Behalf-Of (MAY)
- Metadata-Format (default:sword) (SHOULD)
- If-Match (MAY)

Body: Metadata (no by-reference file links allowed)

Responses
- 204 - The server has received your request, and has updated the associated resource
    - Headers:
        - ETag (MAY)
        - Location (MUST)
- 301, 307 - The Metadata URL has changed, re-send this request and all future requests to the new URL
    - Headers:
        - Location (MUST)
- 308 - The Metadata URL has temporarily changed, re-send this request to the new URL
    - Headers
        - Location (MUST)
- 400 - The server could not understand your request. Either your headers or content body are wrong or malformed.
- 401 - You have not provided authentication information, please do so
- 403 - You are not authorised to carry out this request
- 404 - The Metadata URL does not exist
- 405 - The requested Object does not support metadata update in this context
- 412 - There is a problem implementing the request as-is. For example, your checksums may not match, or you may have requested mediated deposit when the server does not support that, or your If-Match ETag may not match.
- 413 - Your request body exceeds the size allowed by the server

- 415 - The metadata format is not the same as that identified in Metadata-Format and/or it is not supported by the server

Replace Object Content

Replace in its entirety the content of the object (not the metadata). All previous files will be removed, and new ones will replace them. The server may or may not keep old versions of the content available.

Request
PUT Content
Headers:
- Content-Type (MUST)
- Content-Length (SHOULD)
- Digest (SHOULD)
- Authorization (MAY)
- On-Behalf-Of (MAY)
- Content-Disposition (MUST)
- Packaging (default:Binary) (SHOULD)
- If-Match (MAY)

Body:
- Packaged content or File content
- An empty body, if this is a Segment Upload Initialisation request

Responses
- 200 - The server has received your request, and has updated the associated resource
    - Headers:
        - ETag (MAY)
        - Location (MUST)
    - Body: Status
- 202 - The server has received your request, and has accepted the update, and queued it for formal update at some point in the future
    - Headers:
        - Location (MUST)
    - Body: Status
- 301, 307 - The Content URL has changed, re-send this request and all future requests to the new URL
    - Headers:
        - Location (MUST)
- 308 - The Content URL has temporarily changed, re-send this request to the new URL
    - Headers
        - Location (MUST)

- 400 - The server could not understand your request. Either your headers or content body are wrong or malformed.
- 401 - You have not provided authentication information, please do so
- 403 - You are not authorised to carry out this request
- 404 - The Metadata URL does not exist
- 405 - The requested Object does not support content update in this context
- 412 - There is a problem implementing the request as-is. For example, your checksums may not match, or you may have requested mediated deposit when the server does not support that, or your If-Match ETag may not match.
- 413 - Your request body exceeds the size allowed by the server

Replace Content File

Replace an existing file in the Object with a new file. The server may keep the old version of the file available.

Request
PUT File
Headers:
- Content-Type (MUST)
- Content-Length (SHOULD)
- Digest (SHOULD)
- Authorization (MAY)
- On-Behalf-Of (MAY)
- If-Match (MAY)

Body:
- File content
- An empty body if this is a Segment Upload Initialisation request

Responses
- 204 - The server has received your request, and has updated the associated resource
    - Headers:
        - ETag (MAY)
        - Location (MUST)
- 301, 307 - The File URL has changed, re-send this request and all future requests to the new URL
    - Headers:
        - Location (MUST)
- 308 - The File URL has temporarily changed, re-send this request to the new URL
    - Headers
        - Location (MUST)
- 400 - The server could not understand your request. Either your headers or content body are wrong or malformed.

- 401 - You have not provided authentication information, please do so
- 403 - You are not authorised to carry out this request
- 404 - The File URL does not exist
- 405 - The requested Object does not support file replace in this context
- 412 - There is a problem implementing the request as-is. For example, your checksums may not match, or you may have requested mediated deposit when the server does not support that, or your If-Match ETag may not match.
- 413 - Your request body exceeds the size allowed by the server

Delete Object Content

Remove all the content files from an object. This will leave the object and its metadata intact. The server may keep old versions of the files available.

Request
DELETE Content
Headers:
- Authorization (MAY)
- On-Behalf-Of (MAY)
- If-Match (MAY)

Body: None

Responses
- 204 - The Content of the Object has been deleted.
    - Headers:
        - ETag (MAY)
- 301, 307 - The Content URL has changed, re-send this request and all future requests to the new URL
    - Headers:
        - Location (MUST)
- 308 - The Content URL has temporarily changed, re-send this request to the new URL
    - Headers
        - Location (MUST)
- 400 - The server could not understand your request. Either your headers or content body are wrong or malformed.
- 401 - You have not provided authentication information, please do so
- 403 - You are not authorised to carry out this request
- 404 - The Content URL does not exist
- 405 - The requested Object does not support content delete in this context
- 412 - There is a problem implementing the request as-is. For example, your checksums may not match, or you may have requested mediated deposit when the server does not support that, or your If-Match ETag may not match.

Delete Object

Delete the object in its entirety from the server, along with all metadata and content.

Request
DELETE Object
Headers:
- Authorization (MAY)
- On-Behalf-Of (MAY)
- If-Match (MAY)

Body: None

Responses
- 204 - The Object has been deleted.
- 301, 307 - The Object URL has changed, re-send this request and all future requests to the new URL
    - Headers:
        - Location (MUST)
- 308 - The Object URL has temporarily changed, re-send this request to the new URL
    - Headers
        - Location (MUST)
- 400 - The server could not understand your request. Either your headers or content body are wrong or malformed.
- 401 - You have not provided authentication information, please do so
- 403 - You are not authorised to carry out this request
- 404 - The Object URL does not exist
- 405 - The requested Object does not support delete in this context
- 412 - There is a problem implementing the request as-is. For example, your checksums may not match, or you may have requested mediated deposit when the server does not support that, or your If-Match ETag may not match.

Delete Content File

Delete a single content file from the Object. The server may keep old versions of the file.

Request
DELETE File
Headers:
- Authorization (MAY)
- On-Behalf-Of (MAY)
- If-Match (MAY)

Body: None

Responses

- 204 - The File has been deleted.
- 301, 307 - The File URL has changed, re-send this request and all future requests to the new URL
    - Headers:
        - Location (MUST)
- 308 - The File URL has temporarily changed, re-send this request to the new URL
    - Headers
        - Location (MUST)
- 400 - The server could not understand your request. Either your headers or content body are wrong or malformed.
- 401 - You have not provided authentication information, please do so
- 403 - You are not authorised to carry out this request
- 404 - The File URL does not exist
- 405 - The requested Object does not support file delete in this context
- 412 - There is a problem implementing the request as-is. For example, your checksums may not match, or you may have requested mediated deposit when the server does not support that, or your If-Match ETag may not match.

HTTP Headers

Header	**Default Value**	**Usage**
Authorization		To pass any HTTP authorization headers, such as the content for basic auth
On-Behalf-Of		Username of any user the action is being carried out on behalf of
Content-Type		Mimetype of the content being delivered
Content-Length		Length of the depositing content
Digest		Checksum for the depositing content. Require SHA-256, allow for other formats such as MD5 and SHA-1 if still needed.
In-Progress	false	Whether this operation is part of a larger deposit operation, and the server should expect subsequent related requests before injecting the item into any ingest workflows
Slug		Suggested identifier for the item
Content-Disposition		Used to transmit filename and any file segment upload metadata

Packaging	Binary	URI unambiguously identifying the packaging profile
Metadata-Format	sword	URI unambiguously identifying the metadata format/schema/profile
Accept-Packaging	sword	URI identifying the packaging profile that the client would like to receive
Accept-Metadata-Format	sword	URI identifying the metadata format/schema/profile that the client would like to receive
Location		URI for the location where the requested or deposited content can be found
ETag		Object version identifier, as provided by the server on GET requests and any requests which modify the object and return.
If-Match		The Object version identifier (ETag) for which this request should be considered concurrent. If the supplied ETag does not match, the server should reject the update.

Documents

Service Document

The Service Document defines the capabilities and operational parameters of the server.

```
{
  "@context" : {
    "@version" : "1.1",
    "@vocab" : "http://purl.org/net/sword/3.0/terms/",
    "dcterms" : "http://purl.org/dc/terms/",
    "dc" : "http://purl.org/dc/elements/1.1/",
    "xsd": "http://www.w3.org/2001/XMLSchema#",

    "ServiceDocument" : "http://purl.org/net/sword/3.0/types/ServiceDocument",

    "accept" : { "@type" : "xsd:string" },
    "acceptMetadata" : { "@type" : "xsd:string" },
    "acceptPackaging" : { "@type" : "xsd:string" },
    "authentication" : { "@type" : "xsd:string" },
    "byReference" : { "@type" : "xsd:boolean" },
    "collectionPolicy" : { },
    "collections" : { "@container" : "@list" },
    "concurrencyControl" : { "@type" : "xsd:boolean" },
    "description" : { "@type" : "xsd:string" },
    "digest" : { "@type" : "xsd:string" },
```

```
  "endpoint" : { "@type" : "@id" },
  "href" : { "@type" : "@id" },
  "inProgress" : { "@type" : "xsd:boolean" },
  "maxByReferenceSize" : { "@type" : "xsd:nonNegativeInteger" },
  "maxUploadSize" : { "@type" : "xsd:nonNegativeInteger" },
  "name" : { "@type" : "xsd:string" },
  "onBehalfOf" : { "@type" : "xsd:boolean" },
  "segmentedUpload" : { "@type" : "xsd:boolean" },
  "treatment" : { },
  "version" : { "@type" : "@id" }
},

"@id" : "http://example.com/service-document",
"@type" : "ServiceDocument",

"version": "http://purl.org/net/sword/3.0",
"maxUploadSize" : 16777216,
"maxByReferenceSize" : 30000000000000000,
"name" : "Site Name",

"accept" : ["*/*"],
"acceptPackaging" : ["*"],
"acceptMetadata" : ["*"],

"collectionPolicy" : {
  "href" : "http://www.myorg.ac.uk/collectionpolicy",
  "description" : "...."
},
"treatment" : {
  "href" : "http://www.myorg.ac.uk/treatment",
  "description" : "..."
},

"byReference" : true,
"inProgress" : true,
"digest" : ["md5", "sha-1", "sha-256"],
"authentication": ["Basic", "..."],
"onBehalfOf" : true,
"concurrencyControl" : true,
"segmentedUpload" : true,

"collections" : [
  {
    "endpoint": "http://swordapp.org/col-iri/43",

    "accept" : ["*/*"],
    "acceptPackaging" : ["*"],
    "acceptMetadata" : ["*"],

    "collectionPolicy" : {
      "href" : "http://policy",
      "description" : "...."
    },
    "treatment" : {
```

```
        "href" : "http://treatment",
        "description" : "..."
      },

      "byReference" : true,
      "inProgress" : true,
      "digest" : ["md5", "sha-1", "sha-256"],
      "authentication": ["Basic", "..."],
      "onBehalfOf" : true,
      "concurrencyControl" : true,

      "dcterms:abstract" : "...",

      "collections" : []
    }
  ]
}
```

Properties defined at the base of the document should be considered to apply to all collections unless the collection chooses to override them.

Field	**Requirement**	**Default**	**Description**
@context	MUST		The JSON-LD context for this document.
@id	MUST		The URL of the service document you are looking at
@type	MUST		JSON-LD identifier for the document type, in this case “ServiceDocument”.
accept	MUST		“*/*” for any content type, or a list of acceptable content types
acceptMetadata	SHOULD	sword	“*” for any metadata format, or a list of acceptable metadata formats. Acceptable metadata formats SHOULD be an IRI for a known format.
acceptPackaging	SHOULD	Binary	“*” for any packaging format, or a list of acceptable packaging formats. Acceptable packaging formats SHOULD be an IRI for a known format.
authentication	SHOULD		List of authentication schemes supported by the server. If not provided the client MUST assume the server does not support

			authentication.
byReference	SHOULD	false	Does the server support by-reference deposit?
collectionPolicy	MAY		URL and description of the server's collection policy.
collections	MUST		A list of collections, which may in turn be nested. There must be at least one collection at the top level of the service document.
collections/endpoint	MUST		For each collection listed there must be a URL which provides the deposit endpoint for that collection.
collections/collections	MAY		A list of sub-collections, which may in turn be nested.
concurrencyControl	SHOULD	false	Does the server enforce concurrency control. If so, the client MUST take note of the ETag and use the If-Match header in relevant requests.
digest	SHOULD	sha-256	The list of digest formats that the server will accept.
inProgress	SHOULD	false	Does the server support in-progress depositing
maxByReferenceSize	SHOULD	unlimited	Maximum size in kB as an integer for files uploaded by reference.
maxUploadSize	SHOULD	unlimited	Maximum size in kB as an integer for files being uploaded
name	SHOULD		The name of the service.
onBehalfOf	SHOULD	false	Whether the server support deposit on behalf of other users (mediation).
segmentedUpload	SHOULD	false	Whether the server supports deposit of files in segments.
treatment	MAY		URL and description of the treatment content can expect during deposit.
version	MUST		The version of the SWORD protocol this server supports at this endpoint

Metadata

The default sword metadata document allows the deposit of a standard, basic metadata document constructed using the DCMI terms, as well as provides the capacity to specify files to be deposited by-reference.

```
{
  "@context" : {
    "@version" : "1.1",
    "@vocab" : "http://purl.org/net/sword/3.0/terms/",
    "dcterms" : "http://purl.org/dc/terms/",
    "dc" : "http://purl.org/dc/elements/1.1/",
    "xsd": "http://www.w3.org/2001/XMLSchema#",

    "Metadata" : "http://purl.org/net/sword/3.0/types/Metadata",

    "metadata" : { },
    "files" : { "@container" : "@list" },

    "contentDisposition" : { "@type" : "xsd:string" },
    "contentLength" : { "@type" : "xsd:nonNegativeInteger" },
    "contentType" : { "@type" : "xsd:string" },
    "dereference" : { "@type" : "xsd:boolean" },
    "digest" : { "@type" : "xsd:string" },
    "href" : { "@type" : "@id" },
    "packaging" : { "@type" : "xsd:string" },
    "ttl" : { "@type" : "xsd:dateTime" }
  },

  "@id" : "http://example.com/object/1/metadata",
  "@type" : "Metadata",

  "metadata" : {
    "dcterms:abstract" : "....",
    "dcterms:contributor" : "...",
    "etc..." : "...."
  },

  "files" : [
    {
      "href" : "http://www.otherorg.ac.uk/by-reference/file.zip",
      "contentType" : "application/zip",
      "contentLength" : 123456,
      "contentDisposition" : "attachment; filename=file.zip",
      "digest" : "sha-256:....",
      "ttl" : "[timestamp]",
      "dereference" : true,
      "packaging" : "http://purl.org/net/sword/packaging/SimpleZip"
    }
  ]
}
```

Field	Requirement	Default	Description
@context	MUST		The JSON-LD context for this document.
@id	MAY		The Metadata URL for this document, if it was supplied by the Server. Clients MAY provide this URL if updating an existing record, though it is not required. Clients MUST NOT provide this when depositing metadata for the first time.
@type	MUST		JSON-LD identifier for the document type, in this case “Metadata”.
metadata	MAY		The container for the DCMI metadata terms.
files	MAY		A list of files to be deposited by reference.
files/href	MUST		Ever file must provide a URL from which it can be obtained.
files/ttl	MAY	unlimited	A timestamp which indicates when the file will no longer be available (Time To Live). If no date is provided, it is assumed the file will be available indefinitely.
files/dereference	MUST		Should the server dereference the file (i.e. download it and store it locally) or should it simply maintain a link to the external resource. Note that servers may choose to do both, irrespective of the value here, though if “true”, the server should make the external link available to users accessing the resource.
files/contenType	MUST		Content type of the resource being referenced.
files/packaging	SHOULD	Binary	The packaging format of the file, or the Binary file identifier.
files/contentDispositio n	MUST		Content-Disposition as it would have been supplied if this were a by-value file deposit.

files/contentLength	SHOULD		Content-Length as it would have been supplied if this were a by-value deposit.
files/digest	SHOULD		Digest as it would have been supplied if this were a by-value deposit.

Status

The status document is provided in response to deposit operations on the Collection or the Object, and tells the client detailed information about the content and current state of the item.

```
{
  "@context" : {
    "@version" : "1.1",
    "@vocab" : "http://purl.org/net/sword/3.0/terms/",
    "dcterms" : "http://purl.org/dc/terms/",
    "dc" : "http://purl.org/dc/elements/1.1/",
    "xsd": "http://www.w3.org/2001/XMLSchema#",

    "Status" : "http://purl.org/net/sword/3.0/types/Status",

    "actions" : "@nest",
    "addContent" : { "@type" : "xsd:boolean" },
    "byReference" : { "@type" : "@id" },
    "contentEndpoint" : { "@type" : "@id" },
    "contentPackaging" : { "@type" : "xsd:string" },
    "contentType" : { "@type" : "xsd:string" },
    "deleteContent" : { "@type" : "xsd:boolean" },
    "deleteObject" : { "@type" : "xsd:boolean" },
    "deleteFile" : { "@type" : "xsd:boolean" },
    "depositedBy" : { "@type" : "xsd:string" },
    "depositedOn" : { "@type" : "xsd:dateTime" },
    "depositedOnBehalfOf" : { "@type" : "xsd:string" },
    "derivedFrom" : { "@type" : "@id" },
    "description" : { "@type" : "xsd:string" },
    "expecting" : { "@type" : "xsd:positiveInteger" },
    "forwarding" : { },
    "getContent" : { "@type" : "xsd:boolean" },
    "getFile" : { "@type" : "xsd:boolean" },
    "getMetadata" : { "@type" : "xsd:boolean" },
    "href" : { "@type" : "@id" },
    "lastAction" : { },
    "links" : { },
    "log" : { "@type" : "xsd:string" },
    "metadata" : { },
    "metadataEndpoint" : { "@type" : "@id" },
    "metadataFormat" : { "@type" : "xsd:string" },
```

```
  "metadataFormats" : { "@type" : "xsd:string" },
  "objectEndpoint" : { "@type" : "@id" },
  "packaging" : { "@type" : "xsd:string" },
  "received" : { "@type" : "xsd:positiveInteger" },
  "rel" : { "@type" : "xsd:string" },
  "replaceContent" : { "@type" : "xsd:boolean" },
  "replaceFile" : { "@type" : "xsd:boolean" },
  "replaceMetadata" : { "@type" : "xsd:boolean" },
  "segments" : { },
  "sequenceId" : { "@type" : "xsd:string" },
  "size" : {"@type" : "xsd:positiveInteger" },
  "state" : { },
  "status" : { "@type" : "xsd:string" },
  "timestamp" : { "@type" : "xsd:dateTime" },
  "treatment" : { },
  "updateMetadata" : { "@type" : "xsd:boolean" },
  "versionReplaced" : { "@type" : "xsd:dateTime" },

  "dcterms:relation" : { "@type" : "@id" },
  "dcterms:replaces" : { "@type" : "@id" },
  "dcterms:isReplacedBy" : { "@type" : "@id" }
},

"@id" : "http://example.com/object/1",
"@type" : "Status",

"objectEndpoint" : "http://www.myorg.ac.uk/sword3/object1",

"metadataEndpoint" : "http://www.myorg.ac.uk/sword3/object1/metadata",
"metadataFormats" : ["sword", "mods", "..."],

"contentEndpoint" : "http://www.myorg.ac.uk/sword3/object1/content",
"contentPackaging" : ["Binary", "SimpleZip", "SWORDBagIt", "..."],

"metadata" : {
  "dcterms:abstract" : "....",
  "etc..." : "..."
},

"lastAction" : {
  "timestamp" : "[xsd:dateTime]",
  "log" : "description of the event that occurred, with any verbose information",
  "treatment" : {
    "href" : "http://www.myorg.ac.uk/treatment",
    "description" : "treatment description"
  }
},

"links" : [
  {
    "@id" : "http://www.myorg.ac.uk/col1/mydeposit.html",
    "rel" : ["alternate"],
    "contentType" : "text/html"
  },
```

```
{
  "@id" : "http://www.myorg.ac.uk/sword3/object1/package.zip",
  "rel" : ["http://purl.org/net/sword/3.0/terms/originalDeposit"],
  "contentType" : "application/zip",
  "packaging" : "http://purl.org/net/sword/3.0/packaging/SimpleZip",
  "depositedOn" : "[timestamp]",
  "depositedBy" : "[user identifier]",
  "depositedOnBehalfOf" : "[user identifier]",
  "byReference" : "http://www.otherorg.ac.uk/by-reference/file.zip",
  "status" : "pending|downloading|unpacking|error|ingested",
  "log" : "[any information associated with the deposit that the client should know]"
},
   {
  "@id" : "http://www.myorg.ac.uk/sword3/object1/file1.pdf",
  "rel" : ["http://purl.org/net/sword/3.0/terms/derivedResource"],
  "contentType" : "application/pdf",
  "derivedFrom" : "http://www.myorg.ac.uk/sword3/object1/package.zip",
  "dcterms:relation" : "http://www.myorg.ac.uk/repo/123456789/file1.pdf",
  "dcterms:replaces" : "http://www.myorg.ac.uk/sword3/object1/versions/file1.1.pdf"
},
{
  "@id" : "http://www.myorg.ac.uk/sword3/object1/package.1.zip",
  "rel" : ["http://purl.org/net/sword/terms/packagedContent"],
  "contentType" : "application/zip",
  "packaging" : "http://purl.org/net/sword/3.0/packaging/SimpleZip"
},
{
  "@id" : "http://www.swordserver.ac.uk/col1/mydeposit/metadata.xml",
  "rel" : ["http://purl.org/net/sword/3.0/terms/formattedMetadata"],
  "contentType" : "text/json",
  "metadataFormat" : "http://purl.org/net/sword/3.0/metadata/SWORD"
},
{
  "@id" : "http://www.myorg.ac.uk/sword3/object1/versions/file1.1.pdf",
  "rel" : ["http://purl.org/net/sword/3.0/terms/derivedResource"],
  "contentType" : "application/pdf",
  "dcterms:isReplacedBy" : "http://www.myorg.ac.uk/sword3/object1/file1.pdf",
  "versionReplaced" : "[xsd:dateTime]"
},
{
  "@id" : "http://www.myorg.ac.uk/sword3/object1/big.zip",
  "rel" : [
    "http://purl.org/net/sword/3.0/terms/segmentedDeposit",
    "http://purl.org/net/sword/3.0/terms/originalDeposit",
    "http://purl.org/net/sword/3.0/terms/contentReplacement"
  ],
  "contentType" : "application/zip",
  "packaging" : "http://purl.org/net/sword/3.0/packaging/SwordBagIt",
  "depositedOn" : "[xsd:dateTime]",
  "depositedBy" : "[user identifier]",
  "depositedOnBehalfOf" : "[user identifier]",
  "segments" : {
    "sequenceId" : "1234",
    "received" : [1,2,4],
```

```
        "expecting" : [3,5],
        "size" : 10000000
      }
    },
    {
      "@id" : "http://www.myorg.ac.uk/sword3/object1/reference.zip",
      "rel" : [
        "http://purl.org/net/sword/3.0/terms/referenceDeposit",
        "http://purl.org/net/sword/3.0/terms/originalDeposit"
      ],
      "status" : "pending|downloading|unpacking|error|ingested",
      "byReference" : "http://www.otherorg.ac.uk/by-reference/file2.zip",
      "log" : "Any information on the download, especially if it failed"
    }
  ],

  "state" : [
    {
      "@id" : "http://purl.org/net/sword/3.0/state/inProgress",
      "description" : "the item is currently inProgress"
    }
  ],

  "actions" : {
    "replaceMetadata" : true,
    "replaceContent" : true,
    "deleteContent" : true,
    "deleteFile" : true,
    "replaceFile" : true,
    "addContent" : true,
    "updateMetadata" : true,
    "deleteObject" : true,
    "getContent" : true,
    "getMetadata" : true,
    "getFile" : true
  },

  "forwarding" : [
    {
      "objectEndpoint" : "http://www.otherorg.ac.uk/sword3/object12",
      "links" : [
        {
          "@id" : "http://www.otherorg.ac.uk/col2/yourdeposit.html",
          "rel" : ["alternate"],
          "contentType" : "text/html"
        }
      ]
    }
  ]
}
```

Field	Requirement	Default	Description

@context	MUST		The JSON-LD context for this document.
@id	MAY		The Object URL for this document.
@type	MUST		JSON-LD identifier for the document type, in this case “Status”.
actions	MUST		Container for the list of actions that are available against the object for the client.
actions/replaceMetadata	MUST		Whether the client can issue a request to replace the item metadata
actions/replaceContent	MUST		Whether the client can issue a request to replace the item content
actions/deleteContent	MUST		Whether the client can issue a request to delete the content of the item
actions/deleteFile	MUST		Whether the client can issue a request to delete an individual file
actions/replaceFile	MUST		Whether the client can issue a request to replace an individual file.
actions/addContent	MUST		Whether the client can issue a request to add content to the object
actions/updateMetadata	MUST		Whether the client can issue a request to add/update existing metadata.
actions/deleteObject	MUST		Whether the client can issue a request to delete the entire object.
actions/getContent	MUST		Whether the client can issue a request to retrieve the entire content of the item
actions/getMetadata	MUST		Whether the client can issue a request to retrieve the metadata of an item
actions/getFile	MUST		Whether the client can issue a request to retrieve an individual file.
contentEndpoint	MUST		The Content URL for the Object
contentPackaging	SHOULD		The list of packaging formats that this item can be exported as.
lastAction	SHOULD		Container for information about the last action taken on the object by the

			client.
lastAction/timestamp	SHOULD		When the last action was taken by the client
lastAction/log	MAY		Detailed log information about the last action
lastAction/treatment	MAY		Container for information about the treatment the item received in the last action
lastAction/treatment/href	MAY		URL for information about the treatment the item received
lastAction/treatment/description	MAY		Description of the treatment the item received.
links	SHOULD		List of link objects referring to the various files, both content and metadata, available on the object
links/@id	MUST		The URL of the resource
links/rel	MUST		The relationship between the resource and the object. Note that multiple relationships are supported.
links/contentType	SHOULD		Content type of the resource
links/packaging	See Below		The package format identifier if the resource is a package.
links/depositedOn	See Below		Timestamp of when the deposit happened
links/depositedBy	See Below		Identifier for the user that deposited the item
links/depositedOnBehalfOf	See Below		Identifier for the user that the item was deposited on behalf of.
links/byReference	See Below		The external URL of the location a by-reference deposit was retrieved from
links/log	See Below		Any information associated with the deposit that the client should know. Especially if there are asynchronous errors with things like unpacking or download.
links/status	See Below	ingested	The status of the resource, with regard

			to ingest. For example, packaged resources which are still being unpacked and ingested may announce their status here. Likewise, by-reference deposits may do the same. MUST be one of pending|downloading|unpacking|error|ingested. Any associated information to go along with the status, especially if the status is an error, should be in link/log.
link/derivedFrom	See Below		Reference to URL of resource from which the current resource was derived, in particular a single file from a package.
link/dcterms:relation	MAY		URL to a non-sword access point to the file. For example, the URL from which an end-user would download the file via the website. This related URL does not need to support any of the SWORD protocol operations, and indeed may even be on a server or application which has no sword support. Primary use case is to redirect the user to the web front end for the repository.
links/dcterms:replaces	SHOULD		URL to an older version of the file in the same Object, if this is also present as a resource.
links/metadataFormat	See Below		Identifier for a metadata format that the resource conforms to
links/dcterms:isReplacedBy	SHOULD		URL to a newer version of the file in the same Object, if this is present as a resource
links/versionReplaced	SHOULD		Date that the current resource was replaced by a newer resource
links/segments	See Below		Container for information on segmented upload files
links/segments/sequenceId	See Below		The client-supplied sequence ID for this segmented upload
links/segments/received	See Below		The list of integers identifying the segments that have been successfully

			uploaded so far.
links/segments/expecting	See Below		This list of integers identifying the segments which are expected and that have not yet been depoisted
links/segments/size	See Below		The size in bytes of the final resulting assembled file.
state	MUST		List of states that the item is in on the server. At least one state MUST be present, using the SWORD state vocabulary. Other states using server-specific vocabularies may also be used alongside.
state/@id	MUST		Identifier for the state. At least one such identifier MUST be from the SWORD state vocabulary.
state/description	MAY		Human readable description of the state the item is in
forwarding	MAY		List of other locations where the object is available. The inner structure of the forwarding element is to have an objectEndpoint element for the new system (MAY) and zero or more link fields for the object in the new location.

Available “rel” types and their meanings

alternate
An alternate, non-SWORD URL which will allow the user to access the same object. For example, this could be the URL of the landing page in the repository for the item.

http://purl.org/net/sword/3.0/terms/originalDeposit
The resource (file or package) was explicitly deposited via some deposit operation. The properties of the link section for any resource with this rel must be (beyond those already defined in the table above):

Field	**Conditions**
packaging	MAY include this if the original deposit was a packaged item
depositedOn	SHOULD

depositedBy	SHOULD
depositedOnBehalfOf	SHOULD if this was an OnBehalfOf deposit
byReference	MAY if this was a by-reference deposit
status	MUST
log	SHOULD if the status is “error”
dcterms:relation	MAY
dcterms:replaces	SHOULD if this file replaces another file. Principally this is of use for individual file deposits.
dcterms:isReplacedBy	SHOULD if this file is replaced by another file. Principally this is of use for individual file deposits.
versionReplaced	SHOULD if this file is replaced by another file. Principally this is of use for individual file deposits.
segments	MUST if also a segmentedDeposit (see below)

http://purl.org/net/sword/3.0/terms/derivedResource
A file which was unpacked or otherwise derived from another deposited resource, and which itself was not explicitly deposited through some deposit operation. The main usage would be to identify files which were extracted from a deposited zip file.

The properties of the link section for any resource with this rel must be (beyond those already defined in the table above):

Field	**Conditions**
derivedFrom	SHOULD
dcterms:relation	MAY
dcterms:replaces	SHOULD if this file replaces another file. Principally this is of use for individual file deposits.
dcterms:isReplacedBy	SHOULD if this file is replaced by another file. Principally this is of use for individual file deposits.
versionReplaced	SHOULD if this file is replaced by another file. Principally this is of use for individual file deposits.

http://purl.org/net/sword/terms/packagedContent

A resource which makes this object available packaged in the specified package format on HTTP GET. This is not a resource which has been deposited or derived (though it may be very similar to an originally deposited package), it is one which the server makes available as a service to the client. Packages may be pre-built or assembled on the fly - that responsibility rests with the server.

The properties of the link section for any resource with this rel must be (beyond those already defined in the table above):

Field	**Conditions**
packaging	MUST

http://purl.org/net/sword/3.0/terms/formattedMetadata
A resource which makes this object's metadata available, serialised in the specified metadata format on HTTP GET. This is not a resource which has been deposited or derived (though it may be very similar to the originally deposited metadata), it is one which the server makes available as a service to the client. Metadata documents may be pre-built or assembled on the fly - that responsibility rests with the server.

The properties of the link section for any resource with this rel must be (beyond those already defined in the table above):

Field	**Conditions**
metadataFormat	MUST

http://purl.org/net/sword/3.0/terms/segmentedDeposit
A file which is actively being deposited in segments. Often will also have the rel for originalDeposit, and once all segments have been uploaded the segmentedDeposit rel can be removed.

The properties of the link section for any resource with this rel must be (beyond those already defined in the table above, and the originalDeposit table):

Field	**Conditions**
segments	MUST
segments/sequenceId	MUST
segments/received	MUST
segments/expecting	MUST

segments/size	MUST

http://purl.org/net/sword/3.0/terms/referenceDeposit
A file which is currently being downloaded from an external reference. Often will also have the rel for originalDeposit, and once all segments have been uploaded the referenceDeposit rel can be removed.

This rel does not require any additional fields than those already defined for originalDeposit.

http://purl.org/net/sword/3.0/terms/contentReplacement
A segmentedDeposit or referenceDeposit which is also going to replace all of the current Object's content when it is completely uploaded/downloaded, reassembled (if needed) and unpacked (if needed). This value MUST only appear alongside a segmentedDeposit rel or referenceDeposit rel.

Required SWORD State Information

state/@id MUST contain one of:

- http://purl.org/net/sword/3.0/state/accepted - for records accepted for processing but not yet created
- http://purl.org/net/sword/3.0/state/inProgress - for records that have been deposited, but for which the deposit has not yet completed
- http://purl.org/net/sword/3.0/state/inWorkflow - for records that are in the server's ingest workflow
- http://purl.org/net/sword/3.0/state/inArchive - for records that are in the server's archive state, whatever that might mean (e.g. published to the web)
- http://purl.org/net/sword/3.0/state/rejected - for records that have been rejected from the server's workflow
- http://purl.org/net/sword/3.0/state/deleted - for tombstone records

The state field is a list, so it may also contain other states that are server-specific in addition to the sword ones.

Error

An error document is returned at any point that a synchronous operation fails.

```
{
  "@context" : {
    "@version" : "1.1",
    "@vocab" : "http://purl.org/net/sword/3.0/terms/",
    "dcterms" : "http://purl.org/dc/terms/",
```

```
    "dc" : "http://purl.org/dc/elements/1.1/",
    "xsd": "http://www.w3.org/2001/XMLSchema#",

    "ContentError" : "http://purl.org/net/sword/3.0/error/ContentError",
    "ChecksumMismatch" : "http://purl.org/net/sword/3.0/error/ChecksumMismatch",
    "BadRequest" : "http://purl.org/net/sword/3.0/error/BadRequest",
    "OnBehalfOfNotAllowed" : "http://purl.org/net/sword/3.0/error/OnBehalfOfNotAllowed",
    "MethodNotAllowed" : "http://purl.org/net/sword/3.0/error/MethodNotAllowed",
    "MaxUploadSizeExceeded" : "http://purl.org/net/sword/3.0/error/MaxUploadSizeExceeded",

    "error" : { "@type" : "xsd:string" },
    "log" : { "@type" : "xsd:string" },
    "timestamp" : { "@type" : "xsd:dateTime" }
  },

  "@type" : "BadRequest",

  "timestamp" : "[xsd:dateTime]",
  "error" : "error summary",
  "log" : "text log of any debug information for the client"
}
```

Field	Requirement	Default	Description
@context	MUST		The JSON-LD context for this document.
@type	MUST		JSON-LD identifier for the document type. There are a variety of Error Document types, which can be provided, depending on the nature of the error.
error	MUST		A short summary/title for the error
log	SHOULD		Some detail as to the error, with any information that might help resolve it.
timestamp	MUST		When the error occurred.

Authentication and Authorisation

It is strongly RECOMMENDED that SWORD servers support authentication and authorisation for requests.

SWORD servers are not restricted in the forms of authentication that they employ, and there is no minimum requirement or default supported approach.

Servers SHOULD enumerate the authentication schemes that they support in the Service Document, in the field “authentication”, and MUST draw from the IANA registry of HTTP auth scheme names where one is available:
https://www.iana.org/assignments/http-authschemes/http-authschemes.xhtml

For example, a Server which supports Basic, Digest and OAuth authentication could indicate as follows:

```
{
  …
  “authentication” : [“Basic”, “Digest”, “OAuth”]
  …
}
```

Where an authentication scheme is in use by the server which is not covered by the IANA registry - such as a custom API-token-based approach, the server MAY indicate this in whatever way seems most appropriate.

For example:

```
{
  …
  “authentication” : [“Basic”, “APIKey”]
  …
}
```

When carrying out authenticated requests, Authorization headers MUST be sent with every request to the server - the server is not responsible for maintaining state for the client. The server is responsible for authenticating and authorising every request individually. Clients may choose also to send Cookie headers, and servers may support these, but support for Cookies is explicitly outside this specification.

Servers MAY choose to support On-Behalf-Of deposit, which means that the authenticating user is providing content to the server, as if another user were actually carrying out this request. A use case for this would be when a known third-party deposit tool is sending content to a server and has been authorised by another user to add content on their behalf.

If a server supports On-Behalf-Of deposit, it SHOULD indicate this in the Service Document with the field “onBehalfOf” set to true. If this field is not present clients MUST assume that the server does not support On-Behalf-Of deposit.

```
{
  …
  “onBehalfOf” : true
  …
}
```

When an On-Behalf-Of deposit is received, the server MUST ensure that the user identified in that header is valid with respect to the associated Authorization header. For example, when using OAuth2, the On-Behalf-Of user MUST match the user for which the token in the Authorization header was granted.

Transport Security

It is strongly RECOMMENDED that servers implement modern transport layer security in any case. If you are carrying out authenticated protocol operations you MUST implement TLS.

Server Responsibilities

- Deposited files should be available for retrieval by the original depositor immediately (this may be limited to the actual deposited bitstreams, so the contents of a package may not be available individually, but the package itself should be)
- File replacements should be available for retrieval by the original depositor as the latest version of that file immediately (see R-030)

File Segment Upload

Announcing Support for File Segment Upload

Servers MAY support segmented file upload. If a server supports segmented upload it SHOULD indicate this in the Service Document using the field “segmentedUpload”:

```
{
  …
  “segmentedUpload” : true
  …
}
```

Outline of Process for Segmented Upload

Segmented upload MUST follow the pattern:

1. Create an object on the server first. The object may be created in a number of ways:
    a. As a metadata-only object
    b. Via some previous deposit operation of metadata or content
    c. With a “Segment Upload Initialisation” request (see below)

2. The client should receive, or obtain with a GET to the Object URL, the Status document.
    a. If the Object was not created with a “Segment Upload Initialisation” request, then the client should now make such a request to the Object URL (POST) or Content URL (PUT) in line with the usual protocol operations
    b. Once a “Segment Upload Initialisation” request has completed, there will be a File record for the upload with the “rel” “segmentedUpload”, and this provices you with a File URL for which to send file segments.
3. The client may now send file segments in any order, and in parallel if so desired, to the File URL, in line with the specification below.

Segment Upload Initialisation

Before sending any segments to the server, the client must initialise the process. Initialisation requests can be sent to the following locations:

- Deposit-Endpoint (POST) - this will create a new object, and at the same time initialise a segmented upload. In this case, In-Progress will default to true on the server.
- Object (POST) - this will create a File which will ultimately be added to the object as a new file, as per the protocol operation “Add Packaged Content or other File to Object”
- Content (PUT) - this will create a File which will ultimately replace all the existing content in the Object, as per the protocol operation “Replace Object Content”. Note that existing content MUST NOT be removed by the server until all file segments have been uploaded and validated.
- File (PUT) - this will create a new File which will ultimately replace the existing file, as per the protocol operation “Replace Content File”. Note that existing content MUST NOT be removed by the server until all file segments have been uploaded and validated.

In the case of all the above requests, the following conditions on the deposit MUST be met:

Header	**Requirement**	**Description**
Content-Type	MUST	application/octet-stream
Content-Disposition	MUST	The Segment Upload Initialisation parameters

The Content-Disposition header contains the following parameters:

Property	**Requirement**	**Description**
sequence_id	MUST	The client-supplied sequence ID. MUST be a UUID4.

		The server will use this to tie subsequent requests to this segmented upload.
size	MUST	The total size of the final file. This MUST be sent so that the server can determine when all the bytes of the file have been uploaded.
digest	SHOULD	The Digest information for the resulting file as a whole, after assembly. This MUST be in the same form as if it were the HTTP header you would use if depositing this file as a whole.
segment_count	MUST	The total number of segments that will be sent to this sequence. Later, any segment uploads with segment_number greater than this number MUST be rejected by the server.
content_type	MUST	The content type of the resulting object
packaging	SHOULD	The packaging format of the resulting object
filename	SHOULD	The filename of the resulting object. If not provided, the server will assign a filename of its choosing.

The Content-Disposition header can be expressed as follows:

```
Content-Disposition: attachment; sequence_id=<uuid4>; size=<s>; digest=<d>;
segment_count=<N>; content_type=<mime>; packaging=<packaging>; filename=<filename>
```

The body of the request MUST be empty. This, in combination with the Content-Disposition header, will alert the server to prepare for a segmented upload, and respond accordingly in the Status document.

A full example of a request to create an Object with a Segment Upload Initialisation request is as follows:

Request:

```
POST Deposit-Endpoint
Content-Type: application/octet-stream
Content-Disposition: attachment; sequence_id=51747c4a49ac4063a4c14a88b7d67612;
size=10000000; digest=SHA256=skdfskdafjkasdhfakjshdfa; segment_count=5;
content_type=application/zip; packaging=http://purl.org/net/sword/package/SimpleZip;
filename=file.zip
Content-Length: 0
```

Response:

```
HTTP 1.1  201 Created
Content-Type: application/json

{
  "@context" : "...",
  "@id" : "http://example.com/object/1",
  "@type" : "Status",

  "objectEndpoint" : "http://www.myorg.ac.uk/sword3/object1",
  "metadataEndpoint" : "http://www.myorg.ac.uk/sword3/object1/metadata",
  "metadataFormats" : ["sword", "mods", "..."],
  "contentEndpoint" : "http://www.myorg.ac.uk/sword3/object1/content",
  "contentPackaging" : ["Binary", "SimpleZip", "SWORDBagIt", "..."],

  "metadata" : "..."

  "lastAction" : {
    "timestamp" : "2018-01-01T00:00:00Z",
    "log" : "object created; segment upload started",
    "treatment" : {
      "href" : "http://www.myorg.ac.uk/treatment",
      "description" : "treatment description"
    }
  },

  "links" : [
    {
      "@id" : "http://www.myorg.ac.uk/sword3/object1/big.zip",
      "rel" : [
        "http://purl.org/net/sword/3.0/terms/segmentedDeposit",
        "http://purl.org/net/sword/3.0/terms/originalDeposit",
        "http://purl.org/net/sword/3.0/terms/contentReplacement"
      ],
      "contentType" : "application/zip",
      "packaging" : "http://purl.org/net/sword/package/SimpleZip",
      "depositedOn" : "2018-01-01T00:00:00Z",
      "depositedBy" : "sword",
      "segments" : {
        "sequenceId" : "51747c4a49ac4063a4c14a88b7d67612",
        "received" : [],
        "expected" : [1,2,3,4,5],
        "size" : 10000000
      }
    }
  ],

  "state" : "...",
  "actions" : "..."
}
```

Now segments themselves can be uploaded to the link at
`http://www.myorg.ac.uk/sword3/object1/big.zip`

Uploading Segments

Segments may be uploaded in any order and may also be parallelised.

Each request MUST be POSTed to the File URL. The following HTTP headers are needed:

Header	**Requirements**	**Description**
Content-Disposition	MUST	See below
Content-Length	MUST	The size of the current segment being uploaded
Digest	SHOULD	The Digest of the segment being uploaded
Content-Type	MUST	application/octet-stream

The properties of the Content-Disposition header required when uploading a segment are:

Property	**Requirement**	**Description**
sequence_id	MUST	The client-supplied sequence ID. MUST be the UUID4 provided in the initialisation request.
segment_number	MUST	The position in the full sequence of this segment. MUST be an integer, MUST start counting at 1. Full list of segments MUST be a sequential list of integers.

The Content-Disposition header can be expressed as follows:

```
Content-Disposition: attachment; sequence_id=<uuid4>; segment_number=<n>
```

For example:

```
POST File
Content-Type: application/octet-stream
Content-Disposition: attachment; sequence_id=51747c4a49ac4063a4c14a88b7d67612;
segment_number=2
Digest: SHA256=lskdfaioerqwjfkwqjfeqjwefijqwf
Content-Length: 10000

[Binary Content]
```

Aborting an Upload

If, part way through a segmented upload you wish to abort, you can send an HTTP DELETE request to the URL of the segmentedDeposit, which can be retrieved from the Status document.

Incomplete Upload Retention

Servers MAY delete incomplete segmented uploads after an unspecified amount of time, if they are not finalised with all segments within a reasonable amount of time. Clients should retrieve the Status document for the object to review the current state of their segmented uploads.

Errors

Servers MUST respond with Error documents under the following circumstances:

1. More bytes have been sent than indicated in the total_size field
2. A request is sent after the total_size has been reached
3. A request is sent after the segment_count has been reached.

If any other errors occur asynchronously, such as in reassembling or unpacking the resulting file, servers MUST provide an error “status” field and suitable “log” information in the link record in the Status document.

By Reference Deposit

Announcing Support for By Reference Deposit

Servers MAY support by-reference deposit. If a server supports by-reference it SHOULD indicate this in the Service Document using the field “byReference”:

```
{
  …
  “byReference” : true
  …
}
```

Options By Reference Deposit

There are 4 ways that by-reference deposits may be used in SWORD:

1. Create the Object (POST to Deposit-Endpoint) and pass the references at the same time. References and metadata can be sent together in the Metadata document. In this case In-Progress will default to true.
2. Add new by-reference files to an existing object by sending (POST) them to the Object URL
3. Replace the entire content of an Object with a by-reference deposit to the Content URL (PUT)
4. Replace an individual file in an Object with a by-reference deposit to the File URL (PUT)

In each of the above requests, the following conditions on the deposit MUST be met:

Header	**Requirement**	**Description**
Content-Type	MUST	application/json
Content-Disposition	MUST	attachment; by-reference=true

For example:

Request:

```
POST Deposit-Endpoint
Content-Type: application/json
Metadata-Format: sword
Content-Disposition: attachment; by-reference:true

{
   “metadata” : “....”
   “files” : [
    {
      "href" : "http://www.otherorg.ac.uk/by-reference/file.zip",
      "contentType" : "application/zip",
      "contentLength" : 123456,
      "contentDisposition" : "attachment; filename=file.zip",
      "digest" : "SHA256=....",
      "ttl" : "2019-01-01T00:00:00Z",
      "dereference" : true,
      "packaging" : "http://purl.org/net/sword/packaging/SimpleZip"
    }
  ]
}
```

Response:

```
{
  "@context" : "...",

  "@id" : "http://example.com/object/1",
  "@type" : "Status",

  "objectEndpoint" : "http://www.myorg.ac.uk/sword3/object1",
```

```
  "metadataEndpoint" : "http://www.myorg.ac.uk/sword3/object1/metadata",
  "metadataFormats" : ["sword", "mods", "..."],
  "contentEndpoint" : "http://www.myorg.ac.uk/sword3/object1/content",
  "contentPackaging" : ["Binary", "SimpleZip", "SWORDBagIt", "..."],

  "metadata" : "....",

  "lastAction" : {
    "timestamp" : "2018-01-01T00:00:00Z",
    "log" : "metadata and files deposited by reference",
    "treatment" : {
      "href" : "http://www.myorg.ac.uk/treatment",
      "description" : "treatment description"
    }
  },

  "links" : [
    {
      "@id" : "http://www.myorg.ac.uk/sword3/object1/reference.zip",
      "rel" : [
        "http://purl.org/net/sword/3.0/terms/referenceDeposit",
        "http://purl.org/net/sword/3.0/terms/originalDeposit",
        "http://purl.org/net/sword/3.0/terms/contentReplacement"
      ],
      "status" : "pending",
      "byReference" : "http://www.otherorg.ac.uk/by-reference/file2.zip"
    }
  ],

  "state" : "...",
  "actions" : "...",
}
```

Server-Side Processing of By Reference Deposits

1. The server receives a by-reference deposit document with one or more files listed
2. It creates records for each of these files that it plans to dereference, which are visible in the Status document. Files marked by the client not to be dereferenced are considered metadata, and MAY NOT appear in the Status document. These Files all have the status “pending”.
3. At its own pace, taking into account the ttl of the files, the server downloads all the files that are marked for dereference. During the download the server SHOULD set the status to “downloading”. The server SHOULD be able to resume and interrupted download.
4. Once the Files are downloaded and processed, the server MUST set the status to “ingested”. If the files need unpacking first, the server SHOULD set the status to “unpacking”. The server MUST also remove the referenceDeposit and contentReplacement rels, as needed.

5. If there is an error in downloading or otherwise processing the file, the server MUST set the status to “error” and SHOULD provide a meaningful “log” message.
6. The server MAY continue to record the original URL of the file if desired.

Responsibilities of the client/reference server

To provide deposit by reference, the reference server SHOULD:

1. Support resumable downloads
2. Hold the file for long enough for the repository to retrieve it

To use by reference, the client SHOULD:

1. Follow up on the deposit to determine if the dereference of the file has been successful
2. Be able to take suitable onward action if there is an error

Packaging Formats

There are 3 packaging formats the all SWORD implementations MUST support.

Binary

URI: http://purl.org/net/sword/3.0/package/Binary

This format indicates that the package should be interpreted as an opaque blob, and the server SHOULD NOT attempt to extract any content from it. This is typically for use when depositing single files, which do not need unpacking of any kind.

Servers MAY choose, nonetheless, to extract content from Binary packages, if they have the capabilities, such as metadata from images, structural information from text documents, etc.

SimpleZip

URI: http://purl.org/net/sword/3.0/package/SimpleZip

This format indicates that the package is a compressed set of one or more files in an arbitrary directory structure. The nature of the compression and the structure of the compressed content is not specified.

Servers MAY choose to extract the content from SimpleZip packages, and present the individual file components as “derivedResource”s, if desired.

SWORDBagIt

URI: http://purl.org/net/sword/3.0/package/SWORDBagIt

This format is a profile of the BagIt package structure, which has in turn been compressed. The nature of the compression is not specified.

```
SwordBagIt
| -- bag-info.txt
| -- bagit.txt
| -- data
| -- | -- bitstreams ...
|    \ -- directories ...
|         \ bitstreams ...
| -- manifest-sha256.txt
| -- metadata
|     \-- sword.json
\ -- tagmanifest-sha256.txt
```

This allows us to represent the item as a combination of an arbitrary structure of bitstreams in the data directory (similar to SimpleZip), and the metadata in the sword default format in metadata/metadata.json. A manifest (and tagmanifest) of sha-256 checksums is required, as well as the bagit.txt file and a bag-info.txt file.

The content of metadata.json is exactly as defined in the Metadata file, including the ability to supply by-reference files during deposit in this way. Note that use of fetch.txt is not supported here.

The server SHOULD unpack this file, and action at least the metadata. The contents of the data directory MAY be unpackaged into “derivedResource”s if the server desires. It is RECOMMENDED that the contents of the data directory be a flat file structure, to aid mutual comprehension by servers/clients.

TODO: Create a profile for this format using https://github.com/ruebot/bagit-profiles

Depositing Other Metadata Formats

In addition to the standard SWORD metadata format described above, SWORD can support the deposit of arbitrary metadata schemas and serialisations. For example:

```
POST Deposit-Endpoint
Content-Type: application/xml
```

```
Metadata-Format: http://www.loc.gov/mods/v3

<mods xmlns:mods="http://www.loc.gov/mods/v3">
  <originInfo>
    <place>
      <placeTerm type="code" authority="marccountry">nyu</placeTerm>
      <placeTerm type="text">Ithaca, NY</placeTerm>
    </place>
    <publisher>Cornell University Press</publisher>
    <copyrightDate>1999</copyrightDate>
  </originInfo>
</mods>
```

If the server supports the MODS Metadata-Format, then it will be able to create a new object from this XML document, and populate the metadata from the data therein.

TODO List

A list of observations about the specification which we could do to make it better, and which are not directly related to the actual specification itself:

1. All SWORD terms should resolve to a web page which describes the term.