

1 Table of Contents

- 1 Table of Contents
- 2 Introduction
- 3 EASY
 - 3.1 Dataset
- 4 SWORD Overview
- 5 Client Implementation
 - 5.1 Java Example
 - 5.1.1 Java: Request Service Document
 - 5.1.2 Java: Deposit Dataset
 - 5.2 Demo Client
 - 5.2.1 Demo Client: Service Document
 - 5.2.2. Demo Client: Data Deposit
 - 5.3 Easy Deposit
 - 5.4 Command-line Example Using curl
 - 5.5 Important Note on the HTTP Expect Header
- 6 SWORD Details
 - 6.1 Request / Response
 - 6.1.1 Requests
 - 6.1.2 Responses
- 7 Packaging Datasets
 - 7.1 Zip Content
 - 7.2 Metadata Format
 - 7.3 Zip Format
- 8 Service Implementation
 - 8.1 Depositor
 - 8.2 PID
- 9 Dataset States from a SWORD Perspective
- 10 Constraints
 - 10.1 Path Names
 - 10.2 Dataset Size
 - 10.3 Premature Visibility

2 Introduction

This document is intended for developers who want to implement applications, which can deposit datasets in EASY (the Electronic

Archiving System of DANS). We shall describe how client applications can submit datasets in EASY by sending compound resources (i.e. zip files) to the auto-ingest service.

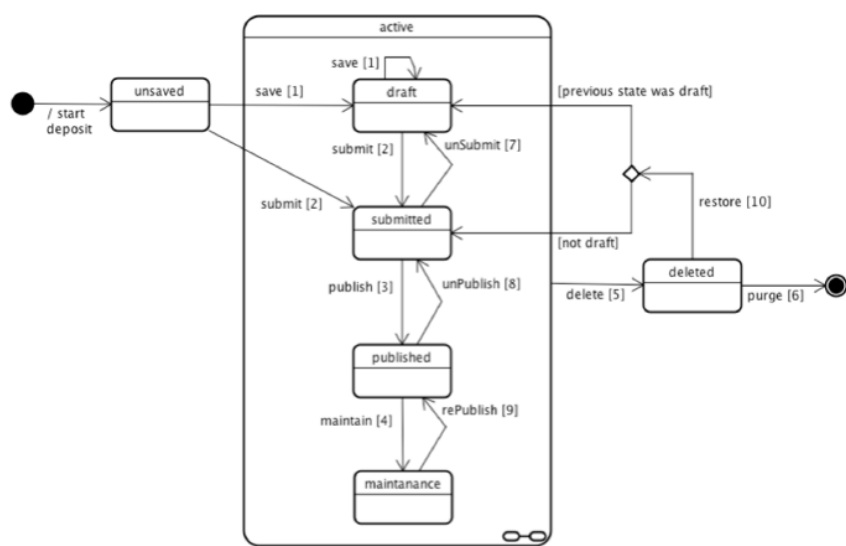
3 EASY

To understand our current implementation of the SWORD auto-ingest service it's best to start with a quick overview of EASY, its datasets, and their workflow. EASY is an online archiving system, which contains thousands of datasets. A dataset is comprised of data (the actual files) and metadata (describing the data). Traditionally, depositors had to go through the web-interface of EASY to manually upload their files and fill in the metadata describing their dataset. Since this facility is not suitable for automation of the depositing process, an auto-ingest interface has been developed for EASY.

3.1 Dataset

This section describes the possible states a dataset can have. A dataset that has been created but not yet submitted has status DRAFT. The depositor can still change the dataset in this state. Once a dataset is submitted, its status is SUBMITTED. At this point a data archivist from DANS will be appointed to the dataset. The archivist will verify the integrity of the dataset. If necessary the dataset will go back to DRAFT state so that the depositor can change certain aspects. If the archivist approves the integrity of the dataset its status can be promoted to PUBLISHED. This means the dataset will be visible on the EASY website. A published dataset cannot be changed by anyone. In order to make changes, the dataset's state must change to either MAINTENANCE or PUBLISHED. In rare cases a dataset can also be given the status DELETED.

The following diagram shows the states (and transitions) a dataset can go through.



4 SWORD Overview

The service that EASY exposes for depositing datasets is based on the SWORD (Simple Web-service Offering Repository Deposit) protocol. Currently, version 1.3 of the SWORD specification (<http://www.swordapp.org/docs/sword-profile-1.3.html>) is implemented.

This SWORD protocol based auto-ingest service exposes a Service Document as described in the SWORD specification. The Service Document describes the characteristics of the service. The relevant part is discussed in this document. For an exact description of the various fields in the Service Document we refer to the referenced SWORD specification.

In the most basic terms, a client application that wants to ingest a dataset should send a HTTP POST request that contains some (optional) SWORD headers and a body, which contains a zip file. This zip file must contain a metadata file describing the dataset, as well as the data (files and folders) of the dataset. The “dataset packaging” section describes this in more detail.

If the received dataset passes validation, the dataset will be saved in EASY and its state will be set to SUBMITTED. From this point on, the dataset travels the same path as “regular” datasets deposited through the EASY web interface.

5 Client Implementation

In this section some of the possible approaches will be discussed to illustrate how developers can go about implementing an application that can ingest datasets into EASY. Using the auto-ingest service, client applications can deposit dataset relatively easily. The whole process can be summarized as one POST request containing the zipped dataset in it's body and the authorization header set using HTTP Basic. For security reasons communication with the auto-ingest service goes over a HTTPS connection.

5.1 Java Example

All we need to create a simple client is a way to send HTTP requests. So in principle, we don't need special libraries to implement this. We could simply use the standard HTTP functionalities that Java exposes through its standard libraries. However, to keep the example code short and concise we chose to enlist the help of the [Apache Commons] commons-io and commons-codec libraries.

The complete example code can be requested from DANS as a zip-file. The sections below discuss the essential parts.

5.1.1 Java: Request Service Document

The Service Document can be easily retrieved by sending a GET request to the auto-ingest service. Note that a username and password must be sent along with this request, but the server doesn't check the validity. The example code below retrieves the Service Document and prints it to the standard output.

```
private void getServiceDocument()
    throws MalformedURLException, IOException {
    System.out.println("*** GETTING SERVICE DOCUMENT ***");
    // Open a secure connection to the deposit service
    HttpURLConnection c =
        (HttpURLConnection) new URL(DANS_SWORD_SERVICE_DOCUMENT)
            .openConnection();
    c.setRequestMethod("GET");
    c.setRequestProperty("Authorization", getAuth());
    printResponse(c);
}
```

```

        System.out.println();
    }

```

5.1.2 Java: Deposit Dataset

The Java code below opens a secure connections, sets the headers for a POST request, copies the contents of the ingest packages to the request entity, and prints the response document to the standard output. For this request it is crucial that the supplied username and password are valid. Users can be created through the web-interface of EASY.

```

private void depositPackage()
throws MalformedURLException, IOException {
    System.out.println("*** DEPOSITING PACKAGE ***");
    // Open a secure connection to the deposit service
    HttpURLConnection c =
        (HttpURLConnection) new URL(DANS_SWORD_DEPOSIT_URL)
            .openConnection();
    c.setRequestMethod("POST");
    c.setRequestProperty("Authorization", getAuth());
    c.setRequestProperty("Content-Type", "application/zip");
    c.setRequestProperty("Content-Length",
        Long.toString(depositPackage.length()));
    c.setDoOutput(true);

    // POST the ingest package to the service.
    FileUtils.copyFile(depositPackage, c.getOutputStream());
    printResponse(c);
    System.out.println();
}

```

5.2 Demo Client

There is a demo client included in the SWORD-common library (<http://sourceforge.net/projects/sword-app/files/SWORD%20Java%20Library/>). The demo client is capable of running in GUI mode as well as in CLI mode. By default it runs in GUI mode. In this section we will discuss the usage of the GUI mode only. The demo client requires a service document to allow deposits with a POST command. Therefore, you need the following actions from the menu bar:

- File → Add Service (to add the service document)
- File → POST (to create and send requests)

5.2.1 Demo Client: Service Document

The first step is to add the service by pointing out the location of the Service Document. The following screenshot shows the dialog (File → Add Service) to achieve this.

The result should look like the following:

Collection Summary	
Collection location	http://localhost:8083/deposit
Collection title	EASY
Abstract	Electronic Archive System, accepts deposits by users registered on EASY .
Collection Policy	THIS IS A TEST SERVER! No guarantee of service, or that deposits will be retained for any length of time.
Treatment	DANS will store deposited data according to the guidelines of the international Data Seal of Approval . After several workdays the data will be available to other researchers, in accordance with the conditions indicated by you and DANS. At the DANS website you'll find more information on depositing data .
Mediation	false
Nested Service Document	Not defined
Accepts	application/zip
Accepts Packaging	http://eef12.dans.knaw.nl/schemas/md/emd/2012/easymetadata.xsd (1.0), http://eef12.dans.knaw.nl/schemas/docs/emd/emd.html (1.0), http://eef12.dans.knaw.nl/schemas/md/dataset/2012/dans-dataset-md.xsd (0.0), http://eef12.dans.knaw.nl/schemas/docs/ddm/dans-dataset-md.html (0.0)

```

status: Requesting the document from http://localhost:8083/servicedocument
<?xml version="1.0" encoding="UTF-8"?>
<app:service xmlns:atom="http://www.w3.org/2005/Atom" xmlns:app="http://www.w3.org/2007/app" xmlns:sword="http://purl.org/net/sword" xmlns:d="http://purl.org/dc/terms" version="1.3">
  <atom:generator uri="http://localhost:8083/servicedocument" version="1.3">Easy SWORD Server</atom:generator>
  <app:workspace>
    <atom:title type="text">DANS sword interface</atom:title>
    <app:collection href="http://localhost:8083/deposit">
      <atom:title type="text">EASY</atom:title>
      <app:accept type="application/zip">
        <app:acceptPackaging q="1.0">http://eef12.dans.knaw.nl/schemas/md/emd/2012/easymetadata.xsd</app:acceptPackaging>
        <app:acceptPackaging q="1.0">http://eef12.dans.knaw.nl/schemas/docs/emd/emd.html</app:acceptPackaging>
        <app:acceptPackaging q="0.0">http://eef12.dans.knaw.nl/schemas/md/dataset/2012/dans-dataset-md.xsd</app:acceptPackaging>
        <app:acceptPackaging q="0.0">http://eef12.dans.knaw.nl/schemas/docs/ddm/dans-dataset-md.html</app:acceptPackaging>
      </app:accept>
      <atom:collectionPolicy><div>THIS IS A TEST SERVER! No guarantee of service, or that deposits will be retained for any length of time.</div></atom:collectionPolicy>
      <atom:abstract><div>Electronic Archive System, accepts deposits by users registered on <a href="http://localhost:8083">EASY</a>.</div></atom:abstract>
      <atom:mediation type="text">false</atom:mediation>
    </app:collection>
  </app:workspace>
</app:service>

```

status: Data received for location: http://localhost:8083/servicedocument

There are two notes to make about the result:

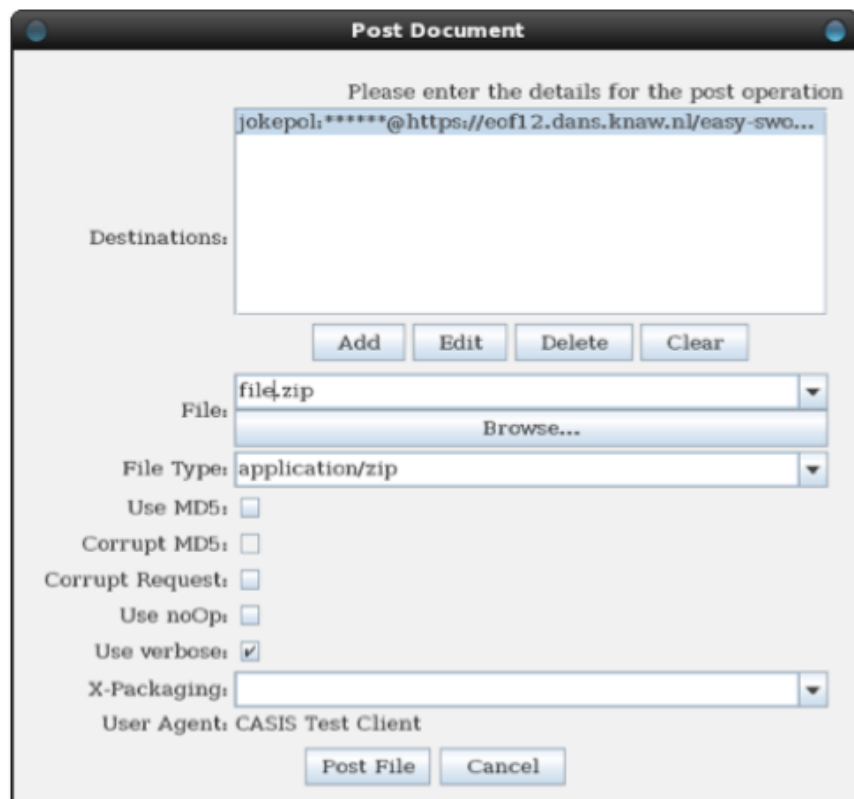
- The client shows a warning on the “accept packaging” field in the warning tab. This is because the URI in that field is not

one of the types specified in the SWORD types specification (<http://www.swordapp.org/docs/sword-type-1.0.html>) . The field rather describes the format of the metadata document that should be included in the zip-file.

- The quality values for the “accept packaging”, 1.0 and 0.0, mean “preferred” respectively “not yet implemented”.

5.2.2. Demo Client: Data Deposit

The POST dialog (File -> POST) can be used to create a deposit request by uploading the compressed dataset (note: EASY only accepts zip files) and selecting the specific features that are applicable. The following screenshot shows this dialog.



5.3 Easy Deposit

Instead of writing the client code by hand a client developer may also choose to build a web interface with EasyDeposit (<http://easydeposit.swordapp.org/>) , an open source SWORD client creation toolkit (<http://sourceforge.net/projects/sword-app/files/SWORD%20Java%20Library/>) . Note that EasyDeposit has no relation to the EASY archive.

5.4 Command-line Example Using curl

Curl (<http://curl.haxx.se/>) is an open source command line tool for transferring data with URL syntax. For scripting purposes or simple task automation it can be useful to utilize curl for ingesting datasets into EASY.

Example to retrieve a service document:

```
curl -u USER:PASSWORD \  
https://act.easy.dans.knaw.nl/sword/servicedocument
```

Example to create a deposit request:

```
curl -H "Content-Type: application/zip" -v -i -u USER:PASSWORD \  
--data-binary @example-deposit.zip \  
https://act.easy.dans.knaw.nl/sword/deposit
```

Note the following:

- The example uses the acceptance test server.
- The strings “USER” and “PASSWORD” need to be substituted with actual EASY user credentials.
- A backslash denotes a continuing line.
- Do *not* use the -k option to ignore the server certificate. The acceptance test and production servers have valid certificates
- “example-deposit.zip” should be replaced with the name of an actual zipped ingest package. The layout of an ingest package is described in 7 Packaging datasets.

5.5 Important Note on the HTTP Expect Header

Note that clients are expected to support and use the http expect header (<http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.20>) . When sending a POST-request the client must begin by sending only the headers, including a “Expect: 100-continue”, to indicate that it awaits a go-ahead by the server. Failure to use this header may result in a buffer overflow on the server and a “413 Request Entity Too Large” status code.

Note that the JRE `HttpsURLConnection` did not support the `Expect`-header prior to Java 7. From Java 7 onwards the header

seems to be sent by default. For other client libraries it may be necessary to send the header explicitly (e.g. Apache http client 4).

6 SWORD Details

Simply put, the SWORD protocol is an AtomPub (<http://www.ietf.org/rfc/rfc5023.txt>) profile, which provides both a set of extensions and constraint relaxations, and enforcements with regards to the AtomPub protocol. This section will discuss the details of our SWORD implementation in the contrast of the specifications.

6.1 Request / Response

We will start our description by covering the types of requests that clients can send to the auto-ingest service. Also, the various responses the client can receive will be covered.

6.1.1 Requests

The following table shows the various requests that a client can send to the service. For each request the client can and/or must include certain headers, these are shown in the table as well. While developing a client the X-No-Op header can be very useful for testing purposes. When this header is set to true the submission will be treated as a dry run, i.e. no actual deposit will be made.

Method	Headers	Notes
GET	Authorization	Required. Value of username and password does not matter.
	X-On-Behalf-Of	Invalid. The presence of this header results in response code <i>412 - pre-condition failed</i> .
POST	Authorization	Required. Must be valid EASY username/password
	MD5	Optional.
	X-On-Behalf-Of	Invalid. The presence of this header results in

Method	Headers	Notes
	X-No-Op	<p>response code <i>412 - pre-condition failed</i>.</p> <p>Optional. Default: <i>false</i>. A value other than <i>"true"</i> or <i>"false"</i> results in response code <i>400 – bad request</i>. The value <i>"true"</i> will prevent any changes in the repository. This header is meant for testing purposes while developing a client. A response code <i>202 – Accepted</i> predicts a high probability of success if the request is repeated for real. The notification e-mail will not have a license document attached, although the body of the message says so.</p>
	X-Verbose	<p>Optional. Default: <i>false</i>. A value other than <i>"true"</i> or <i>"false"</i> results in response code <i>400 – bad request</i>. The value <i>"true"</i> will generate more information in the response body.</p>
	X-Packaging	Currently ignored.
Other		Results in response code <i>405 – method not allowed</i> .

6.1.2 Responses

Code	Message	Body	Notes
------	---------	------	-------

Code	Message	Body	Notes
200	OK	XML	Returned after a successful GET request. Content of the body is shown in the demo client example.
201	Created		Currently, not to be expected as a deposit requires manual steps by an archivist of DANS to publish the dataset.
202	Accepted	XML	Returned after a successful POST request. The dataset is created with status “ <i>submitted</i> ” and a confirmation e-mail is sent to the depositor (the authenticated user). Content of the body is shown in the demo client example.
400	Bad request	XML	The request was formulated incorrectly.
401	Unauthorized	XML	Either the user is invalid or unauthorized to conduct the specified action.
403	Forbidden		Not to be expected, as currently, the X-On-Behalf-Of header is not implemented.
405	Not supported	HTML	The requested action is not supported by the service.
412	Precondition failed	XML	Can have several reasons, e.g. in response to an X-On-Behalf-Of header
415	Unsupported media	XML	The specified Media Type is not supported by the

Code	Message	Body	Notes
	type	service.	

7 Packaging Datasets

We have discussed the types of requests and the headers that can be issued to the auto-ingest service. This paragraph describes how datasets should be packaged. The body of a POST request must be a zip-file that meets the requirements described below.

7.1 Zip Content

The zip file that is sent within the body of a deposit request should contain the following:

- A file containing metadata with the name:
DansDatasetMetadata.xml
- A folder named “data” containing the data files.

Complete details, including examples, of the XML metadata file are available on the EASY website. The referenced link is also returned by the service document in the field

<sword:acceptPackaging q=”1.0”>, the quality attribute (q) value of 1.0 indicates the preferred format. Details about the formats of the data files can also be found on the DANS website

(http://www.dans.knaw.nl/en/deposit/information-about-depositing-data?set_language=en) .

7.2 Metadata Format

The preferred metadata format for datasets is DDM (DANS Dataset Metadata). The above-mentioned documentation regarding the metadata format gives a minimal valid example, which is shown below. Naturally, more fields should be added to actually describe the dataset.

```
<?xml version="1.0" encoding="UTF-8"?>
<ddm:DDM xmlns:dcx=http://easy.dans.knaw.nl/schemas/dcx/
          xmlns:dc=http://purl.org/dc/elements/1.1/
          xmlns:ddm=http://easy.dans.knaw.nl/schemas/md/ddm/
          xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
          xsi:schemaLocation="http://easy.dans.knaw.nl/schemas/md/
```

```
http://easy.dans.knaw.nl/schemas/md/2012/11/ddm.xsd">
<ddm:profile>
  <dc:title>My title</dc:title>
  <dc:description>My description</dc:description>
  <dc:creator>John Doe</dc:creator>
  <ddm:created>2012</ddm:created>
  <ddm:audience>D32000</ddm:audience>
  <ddm:accessRights>OPEN_ACCESS</ddm:accessRights>
</ddm:profile>
</ddm:DDM>
```

Please note:

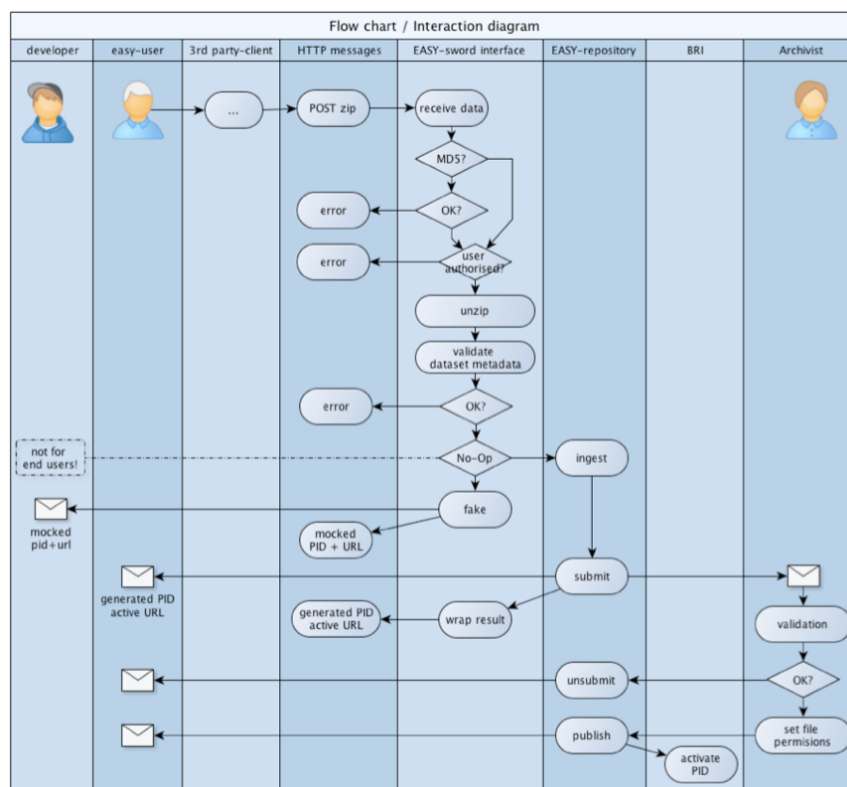
- This concrete example doesn't contain all the required fields for the metadata to validate. Since a complete example is too large to show here, we refer to the examples shown on: <http://easy.dans.knaw.nl/schemas>
- Empty fields are regarded as non-existent, thus having empty fields for required elements will not pass validation in the auto-ingest. In this case, the response will report the empty fields as missing.

7.3 Zip Format

The metadata XML file and the data folder should be in the root of the zip file. When creating the zip file, please pay close attention to hidden files and folders (e.g. .DS_Store files on OS X). These should not be included in the zip file, as they will show up in EASY otherwise.

8 Service Implementation

The following diagram shows the interactions between various systems starting from a client application initiating the deposit until the actual publishing of the dataset in EASY.



The process starts with the client application sending a POST request, containing the packaged dataset in its body, to the auto-ingest interface. If the request contains a Content-MD5 header, the received content will be checked against the hash. Note that the MD5 hash *must* be encoded as a hexadecimal number in ASCII (so 32 characters from the set 0-9 and a-f representing a 128 bit number). According to RFC1864 (<http://www.ietf.org/rfc/rfc1864.txt>) it should actually be encoded as either base 64 or quoted-printable (http://www.w3.org/Protocols/rfc1341/5_Content-Transfer-Encoding.html) , so the SWORD service deviates from the standard here. This will be fixed in a later release. If the received and computed MD5 hashes differ the server will respond with status code 412 Precondition Failed and an error message. Otherwise the supplied user will be authenticated, returning an error in case this fails. If successful, the zip file is decompressed and the metadata will be validated. Again, returning an error if this fails. Then the service will check whether the client requested a dry run. If so, mocked results are returned. Otherwise the dataset will be submitted into EASY. The dataset has status SUBMITTED. From this point, the regular steps for depositing datasets mentioned earlier are in effect.

8.1 Depositor

The depositor (EASY user) of the data should have registered on the web interface of EASY . This account should be used for authenticating deposits through the SWORD interface. The e-mail address of that account is used for any communication regarding the dataset.

It's advised to verify the correctness of the username and password through the web interface before trying a large deposit. Since the authentication can only be conducted when the complete request has been received, in case of invalid authentication, the client would need to re-transmit the complete deposit. Currently, it's not practical to execute dry runs with small datasets attached to check whether the user details are correct since EASY will send e-mail to this address regarding the “fake” deposit.

8.2 PID

Every dataset submitted in EASY receives a PID (<http://www.dans.knaw.nl/nl/deponeren/toelichting-data-deponeren/persistent-identifiers>) (persistent identifier). This PID uniquely identifies a dataset. DANS provides a resolver service to link PID's to their actual locations. The most important thing to note is that, as the deposit confirmation e-mail states, the assigned PID will not be functional until one day after publishing the dataset. This is because the resolver has to harvest the new PID.

9 Dataset States from a SWORD Perspective

The various states a dataset have been described earlier. For completeness we will describe these states from the perspective of the SWORD auto-ingest service. The following interaction diagram gives a complete overview of the states, and their transitions, that an ingested dataset can go through.



10.1 Path Names

DansDatasetMetadata.xml
data/foo/bar/img.png

10.2 Dataset Size

16

to deposit larger datasets please contact DANS.

10.3 Premature Visibility

Depositing a dataset consist of several non-atomic steps that are not guarded by a transactional system. Before all deposit steps are completed, the dataset may become visible to the depositor under the “my datasets” section in the web interface. Also, an interrupted deposit will retain its DRAFT status. New attempts at depositing the dataset will result in a new dataset, keeping the previous attempt as a DRAFT.