Manifest Specification

Introduction

The purposes of the manifest file (manifest.json) are to:

- track all files within the AIP, including file level metadata such as path, size, checksum
- contain actionable access rules based on access directions, copyright permissions and any other pertinent rules
- link access rules to files so that DIPs can be automatically created from AIPs.

The manifest file contains two sections: a list of access rules; and a list of versions.

At the end of this document is a full example of a manifest file.

Access Rules

Access rules are machine actionable rules based on access directions, copyright permissions and any other pertinent rules that control access to digital objects. They contain the following fields.

Label	Description	Mandatory	Value	Example
id	A local reference so that versions and files within the manifest can be linked to the rule.	Yes	blank node references	_:ar0
executeDate	Date from which the rule applies.	Yes	date in W3C format	2017-01-01
scope	Scope of application of the rule. Options are: • root - does not apply to versions or files, just applies to the "root" of the DIP. This is used to generate metadata-only DIPs. • global - applies to all versions and files • local - applies to any versions and files that link to the rule, and also any of their descendants	Yes	enum: root, global, local	global
publish	Access rules can be either publish rules, where the DIP is eligible for online publication, or non-publish rules, where the DIP is generated for offline access only (e.g. reading room access)	Yes	boolean	true
basis	Links to the applicable accessDirection (if there is one) that is the basis for the rule.	Optional (exclude if no access direction)	accessDirection: int; accessDirection: string	accessDirection: 10; accessDescription: "personal information such as age and address"
metadataPatch	Apply the numbered metadata patch file to the metadata.json file when constructing the DIP.	Optional	int	1
fullManifest	Whether a full or redacted (closed files/versions trimmed) version of the manifest should be included with the DIP. Generally this will be true but there may be some cases where the access directions dictate not publishing these details (e.g. because the title of a file itself contains sensitive information).	Optional (default true)	boolean	true
displayTarget	array of references to file(s) that should be presented for display	Optional	array of references to file(s)	[{"@id":"_:v1f1"}]
previewTarget	array of references to file(s) that should be presented as a preview	Optional	array of references to file(s)	[{"@id":"_:v1f2"}]
textTarget	array of references to file(s) that represent the DIP	Optional	array of references to file(s)	[{"@id":"_:v23f0"}]

Example access rule

```
{
      "@id": "_:ar0",
      "repo:executeDate": "2017-01-01",
      "repo:scope": global,
      "repo:publish": true,
      "repo:basis": {
        "repo:accessDirection": "xx",
        "repo:accessDescription": ""
      },
      "repo:metadataPatch": 1,
      "repo:fullManifest": true,
      "repo:displayTarget": [{"@id":"_:v1f1"}],
      "repo:previewTarget": [{"@id":"_:v1f2"}],
      "repo:textTarget": [{"@id":"_:v23f0"}]
  }
```

Versions

Versions represent manifestations of a digital object: these include the original manifestation, any transformations (for access or preservation purposes), and also text/preview manifestations for display online.

Label	Description	Mandatory	Value	Example
@id	id of the version. This corresponds with numbered folders in the AIP structure (0, 1, 2...).	Yes	id	"_:v0"
repo:base	The version path directory relative to the AIP/DIP.	Yes	repo:base	"versions/0"
hasAccessRules	Access rules that are inherited by all files within the version	Optional	array of references to access rules	
ore:aggregates	List of files within the version	Yes	array of files	

Example versions

```
"repo:versions": [
    {

      "@id": "_:v0",
      "repo:base":"versions/92",
      "repo:hasAccessRules": [
        {"@id":"_:ar2"}
      ],
      "ore:aggregates": [
```

Files

Versions can contain any number of files and directories. The schema for the metadata captured for each file may grow but it should be contained to just information relevant at the file level e.g. size, checksum, file type. Information that relates to the object as a whole (e.g. duration of media, creator, etc.) should go in the metadata.json file. Directories aren't described directly in manifest files but can be inferred from the file path used in the manifest. A digital object may contain directories e.g. when a webpage has an index.html file and a img/ directory.

Label	Description	Mandatory	Value	Example
@id	id of the file.	Yes	id	"_:v0f0"
nfo:filename	File name as it appears in AIP/DIP. Directory paths should be represented by a "/"	Yes		`"A3609756.1" or "img/icon.png"`
premis:originalName	File name as it appeared in agency/original system.	No		`"\\771\\A1314771.1"`
nfo:fileSize	File size in bytes	Yes	int	8798742
nfo:hasHash	File checksum	Yes	nfo:hashAlgorithm and nfo:hashValue	`"nfo:hashAlgorithm": "MD5", "nfo:hashValue": "xxxxx"`
premis:format	File format according to either the PRONOM registry or SARA temp registry	Yes	premis:formatRegistry and premis:formatDesignation	"premis:formatRegistry": "pronom", "premis:formatDesignation": "fmt/20"
dc:format	File MIMEType	Yes	string value, taken from IANA MIMEType registry	application/pdf
repo:hasAccessRules	Access rules that apply just to the specific file	Optional	array of references to access rules	

Example file

```
{
   "@id": "_:v0f0",
      "nfo:fileName": "A3609756.1",
      "premis:originalName": "E:\\771\\A1314771.1",
      "nfo:fileSize": 37463,
      "nfo:hasHash": {
          "nfo:hashAlgorithm": "MD5",
          "nfo:hashValue": "xxx"
      },
      "premis:format": {
          "premis:formatRegistry": "pronom",
          "premis:formatDesignation": "fmt/20"
      },
      "dc:format": "application/pdf",
      "repo:hasAccessRules": [
          {"@id":"_:ar3"}
      ]
}
```

DIP Generation Algorithm

Step 1. Selecting the set of Active Rules

The DIP generation algorithm takes two inputs, a date (YYYY-MM-DD) and a publish value (boolean).

Given these inputs, we apply the following following two tests to the set of rules. Only rules that pass both tests pass and are deemed **active rules**.

A) Testing a rule against a date value

If the rule's execute date matches or follows the provided date value, then the rule passes.

If the rule's execute date precedes the provided date value, than the rule fails.

Getting fancy

This can be simplified as:

A >= B, where A is the provided date, and B is the rule's execute date.

B) Testing a rule against a publish argument

If the publish argument is true, and the rule's publish variable is true, then the rule passes.

If the publish argument is true, and the rule's publish variable is false, then the rule fails.

If the publish argument is false, and the rule's publish variable is true, then the rule passes.

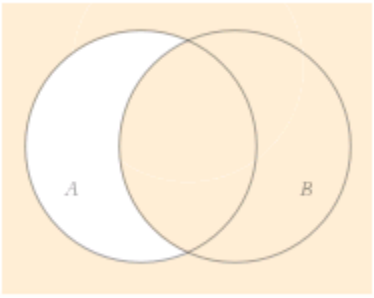
If the publish argument is false, and the rule's publish variable is false, then the rule passes.

Getting fancy

This can be simplified as:

(A XOR B) NAND A, where A is the rule's publish variable and B is the publish argument.

Or as a venn diagram:



Step 2. Applying the Active Rules and Determining the Primary Rule

The Active Rules and the Primary Rule

The active rules determine which versions and files within an AIP are copied into the DIP.

Of the active rules, we identify a single **primary rule** that is used to determines actions relating to metadata patching, manifest filtering, and text and display targets. The MostOpen and MostClosed functions allow us to identify the primary rule.

MostOpen and MostClosed Functions

The "MostOpen" function applies these tests sequentially to a list of active rules:

1. if the list scope=root rules as well as other active rules with scope=global or scope=local, then discard the scope=root rules.
2. if any of the rules have publish=true, then exclude all others. Otherwise keep all.
3. from the remaining rules, select the rule with the latest executeDate value.

The "MostClosed" function applies these tests to *two* active rules:

1. if the publish argument supplied at DIP generation is false, and if one of the rules has publish=true, and if the other has publish=false then return the publish=false rule
2. otherwise, return the rule with the earliest executeDate value.

Traversing the manifest

Let's define two variables, the **parent rule** and the **primary rule**.

The "root" of the manifest is the set of active rules that has scope=global or scope=local. If the root has members, filter it through the MostOpen function and assign the result to both the **parent rule** and **primary rule**.

We now iterate through the versions:

1. add the parent rule to any active rules that are referenced from the version and apply the MostOpen function to this list. *Within the scope of each version*, set the parent rule to the result of that function.
2. apply the MostClosed function to the result of that function and the primary rule. Set the primary rule to the result.
3. Now iterate through each file in the version and:
    a. if there is a parent rule, or any active rules referenced from the file, include the file in the DIP output.
    b. add the parent rule to any active rules that are referenced from the file and apply the MostOpen function. Apply the MostClosed f unction to the result of that function and to the primary rule. Set the primary rule to the result.

Step 3. Creating the DIP

As a result of this traversal, we should have a list of files for inclusion in the DIP and a primary access rule. The files should now be copied into the DIP and the primary rule invoked (to set display targets, metadata patching, manifest redacting etc.).

Steps in this process are:

1. copy files into the DIP and create directories as needed (i.e. version directories and directories within versions)
2. apply any metadataPatch specified in the primary rule and copy metadata.json file into the DIP
3. if the primary rule has fullManifest:true, copy the manifest.json file into the DIP, otherwise redact all file entries that aren't in the list of files copied into the DIP (remove the file entries but leave version entries intact and empty)
4. if the primary rule has publish=true, create a display.json file in the DIP that has the display and text targets (if any) listed in the primary rule. This display.json file is used by the DA viewer. An example follows.

Example display.json

```
{
   "repo:textTarget": [{"@id":"_:v23f0"}],
      "repo:displayTarget": [{"@id":"_:v1f1"}],
      "repo:previewTarget": [{"@id":"_:v1f2"}]
}
```

A full example

manifest.json